POLITECNICO DI MILANO

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

A DEVELOPMENT EFFORT AND SIZE ESTIMATION METHOD FOR PARTIALLY AND FULLY SPECIFIED VHDL PROJECTS

Relatore: Prof. Fabio Salice

Correlatore: Prof. William Fornaciari

Daniele Paolo Scarpazza

Matricola n. 630655

Anno Accademico 2001 - 2002

Sommario

UNA METODOLOGIA PER LA STIMA DELLA DIMENSIONE E DELLO SFORZO DI SVILUPPO PER PROGETTI VHDL PARZIALMENTE E COMPLETAMENTE SPECIFICATI.

La risorsa oggi più preziosa a disposizione di chi gestisce la progettazione di un sistema embedded è senz'altro il tempo umano, ed è anche la risorsa più costosa e più scarsa, se confrontata con le altre coinvolte nel processo di progetto e realizzazione. Tuttavia, mentre esistono teoria e strumenti accurati ed esaustivi per la stima dei ritardi, del consumo di potenza, dell'area di silicio occupata e di numerosi altri parametri di realizzazione, sorprendentemente, non sono disponibili né teoria né strumenti per la stima del tempo umano di sviluppo umano del progetto.

La diffusione dei componenti di proprietà intellettuale messi a disposizione da fornitori specializzati rende virtualmente possibile acquisire ogni tipo di componente standard richiesto dall'industria dei sistemi embedded, a un prezzo che è ben noto e, cosa ancora più importante, noto in anticipo. Nonostante ciò, non esistono metodologie formali per la stima del costo di un componente sviluppato internamente, rendendo così ogni comparazione di tipo *make or buy* virtualmente impossibile. Tali costi non saranno noti prima del completamento del progetto, e in quel momento tutti i costi avranno già avuto luogo e tutte le decisioni saranno già state prese.

L'obiettivo primario di questa tesi è fornire una metodologia di stima per il numero di linee di codice in cui un progetto VHDL può svilupparsi (da cui è facile ricavare il costo e il tempo atteso di sviluppo), date le specifiche del progetto, a qualunque stadio dello sviluppo (idealmente pensato per raffinamenti successivi) esse si trovino, e con qualunque livello di dettaglio esse vengano fornite. È inoltre desiderabile che tale metodologia sia soggetta al restringimento degli intervalli di confidenza degli errori di stima, a mano a mano che la quantità di informazioni disponibili sul progetto cresce; d'altra parte desidereremmo un errore che degradi in maniera non esageratamente brusca quando parte di tali informazioni vengono sottratte alla metodologia di stima.

Per raggiungere tale traguardo, abbiamo raccolto una base di progetti molto significativa (in totale 60 progetti, comprendenti 610 000 linee di codice VHDL, per un totale di 28 M di codice puro) e, dopo aver realizzato un insieme completo di strumenti atti ad estrarre complessi e altamente strutturati insiemi di informazioni sintattiche riguardanti tali progetti, abbiamo provveduto all'identificazione di insiemi-tipo di informazioni che il progettista ha a disposizione al momento della specifica ad alto e medio livello, a riguardo dei blocchi funzionali in cui il progetto risulterà composto.

Quindi abbiamo creato un cospicuo numero di modelli, che fossero in grado di predire la dimensione di quegli oggetti di natura sintattica (di varie granularità, la cui disamina

non appare adeguato introdurre ora) in cui i progetti risultano partizionabili. Abbiamo sottoposto tali modelli a validazione, integrandoli fra loro mediante regole di applicazione che conferiscono al loro apparentemente disaggregato *corpus* lo *status* di metodologia.

Tale metodologia si dimostra nel contempo accurata e robusta. L'accuratezza è comprovata da elevati coefficienti di correlazione fra dati effettivi e stimati (pari a 0.8627 nella validazione interna e a 0.8713 in quella esterna) e da una accettabie varianza dell'errore di stima (scarto quadratico pari a 1400.412 linee di codice nella validazione interna e a 3034.134 linee per l'esterna; si tenga presente che in più dell'80% dei casi l'errore di stima cade nell'intervallo pari a $\pm$ lo scarto quadratico). La robustezza è dimostrata dalla inesistente degradazione dei coefficienti di correlazione al passaggio da validazione interna a esterna, e da un tollerabile aumento nella varianza dell'errore di stima nel medesimo passaggio.

Nonostante i risultati conseguiti siano incoraggianti, la tesi non è priva di numerosi spunti che lasciano aperte interessanti possibilità di sviluppo, ampiamente descritte nel testo ogniqualvolta vengano individuate.

Al fine di ottenere risultati migliori di quelli da noi conseguiti, auspichiamo l'analisi di un archivio di sorgenti VHDL di numerosità ancora superiore a quello da noi costituito. Nutriamo infatti la ragionevole convinzione che i progetti liberamente disponibili su Internet (la cui ampia maggioranza è stata sfruttata proprio in questa tesi) non siano sufficienti per la ottimale identificazione di taluni dei modelli qui proposti, e pensiamo che una cooperazione fra aziende che si occupano della progettazione di sistemi embedded in VHDL su larga scala possa giungere ben oltre i risultati raggiunti da una tesi elaborata da un singolo individuo, chiaramente senza alcun tipo di finanziamenti.

Contents

Chapter 1

The structure of this thesis

The object of this thesis is to provide a method for estimating the resulting size, and therefore the development cost, of an embedded system project designed using the VHDL language, given its high-level specifications.

This thesis is divided in five parts: problem, tools, data, models and conclusions:

- in the first part ("the problem") we will take care of all the definitional aspects of the problem; we will define some useful formalisms and show their properties. In this part you will find definitions, graphs, theorems and proofs;
- in the second part ("the tools") we will thoroughly describe the set of software tools we designed in order to automate the data collection on the problem. In this part you will find BNF grammar productions, SQL queries and table structures, and descriptions of dedicated software, written in flex, bison, C++ and Tcl/Tk;
- in the third part ("the data") we will describe a large set of VHDL source code projects which we assembled by searching among multiple sources on the Internet, in order to create an heterogeneous and sizable project base. All our hypotheses will be checked against these data and all our models will be validated on them. In this part you will find project details, syntax graphs, block diagrams and various reports;
- the fourth part ("the method") is clearly the most important and innovative part of the thesis, the one in which hypotheses on how to deal with the problem are formulated, models are created, tuned and validated. In this part you will find correlation studies, model descriptions and error plots. This part also summarizes the results and findings of the thesis and proposes the use of described methodology.

Part I

The Problem

Chapter 2

The Aim of this Thesis

Human time is currently the most valuable resource that is available to a manager in charge of an embedded system design and realization project. Furthermore, it is also the most expensive and the scarcest, if compared to the other resources involved in the design and realization process.

Nevertheless, though accurate and exhaustive theory and tools have been designed to estimate the number of gates used, or the occupied silicon area (and therefore the project's cost), the timing features (and therefore each component's speed), and the power consumption, surprisingly, when it comes to estimating the development effort, i.e. the cost of the human resources, no such theory and tools are available.

The widespread availability of intellectual property components provided by dedicated suppliers, makes now possible to virtually acquire any type of standardized components required by the embedded system industry, at a price which is well-known and, most important, known in advance. Despite this, no formal methodology is available to estimate the cost of an internally developed component, thus being any *make or buy* comparison virtually impossible. That cost will not be known before the project is completed, and at that time, all the costs will be entirely occurred and all the decisions taken.

The object of this thesis is to provide a method for estimating the number of lines of VHDL code in which an embedded system design project will evolve –and therefore its expected development cost and time, according to relationships clearly delineated in [1]– given its specifications. Specifications will not be provided in a distinct specification language but in VHDL itself: the designer will just start the development of the project by writing a set of VHDL files containing a draft of the architectures and packages he plan to use. He can proceed in a *development by refinement* way, and as the draft approaches the completed project, estimates will converge to the actual value of the completed project size in lines of code. The basic idea is that every estimate comes with a confidence interval, and any additional information available supplied by the designer reduces its the width.

That confidence interval represents the uncertainty due to two causes:

- the lack of complete knowledge about the project's specifications and internal details, (*gnoseological uncertainty*): the developer himself could not have a clear idea of all of the project details a given time, moreover the language used to provide specifications (declarative VHDL enriched with metatags) cannot represent all of the available knowledge. Gnoseological uncertainty decreases as development completion advances and possibly reaches zero when the structure of everything is known;

- the different project implementations resulting from the same specifications (*ontolog-*

ical uncertainty); e.g., the same specifications can be satisfied by a number of structurally different projects, with different features (and, most important, different sizes), implemented in a number of ways depending on a plethora of variables such as the use of standard implementation methodologies, the way of splitting the main objective in sub-objectives and the main unit in connected subunits, the amount of generality/reusability of the implementation, the developer's acquaintance with the language, his experience in treating similar problems, his writing style and personal preferences, and so on. Ontological uncertainty is more difficult to treat than gnoseological, since there is no general way to synthetically describe a structure/algorithm. For example the only general way to describe an algorithm is (trivial solution) to write it down, and it is not synthetical, i.e., it is not possible to give a description of something without fully implementing it, and at that point the description is useless from a decisional point of view, since the object is already implemented;

Our objective is to design an estimation method capable of accepting as input any VHDL project in any possible completion state it is (yet it must be valid VHDL) and return an estimate exploiting as much as possible the available information. During the *development by refinement* process, each of the object in the project can be known by the developer (thus by the estimation algorithm) in several ways:

- the object is *completely known*, i.e., all the parts composing it are known and finished (and not necessarily the objects referenced by it); e.g. a behavioral architecture whose processes are all completely written; the bodies of the functions and procedures used in those processes needs not to be completely known; a project in which all the architectures are implemented. When an object is completely known, all the objects that compose it are completely known. No estimation is to be generated: the object is completed and it is possible to know the actual number of lines composing it;

- the object is *externally known*, i.e., we know how the object talks to the rest of the project (the entity port signals for an architecture, the parameter number and type for a function) but no internal details are given. Nothing is known about the objects it contains (e.g., if it is a behavioral architecture we know nothing about its processes) nor about the object it references (e.g., if it is a structural architecture we know nothing about the architectures it instances). The developer could provide some meta-information describing the object in a qualitative way ("this object is a ROM", "this object will be implemented as a structural architecture"). Again, some meta-information describing in a quantitative way some properties of the dependence graph rooted at the object (what the object will be composed of / what the object will reference) could be available. Estimates are generated on the basis of its interface information and meta-information.

- the object is *internally known*, i.e., it is qualifies to be externally known plus, for every contained element, declarative information is available. The object will simply contain the declarative statements (e.g. the signal and variable declarations, processes with their sensitivity lists but with empty bodies, full component instantiation information) but no more than that;

- the object is *partially known*, when it qualifies to be externally known and one or more of the following cases happen:

  - there is at least one couple of contained objects of which one is externally known and one is completely unknown;

  - there is at least one couple of contained objects of which one is completely known and one is completely unknown;
  - there is at least one couple of referenced objects of which one is completely unknown and one is externally known.

  Intuitively, the partially known state stays somewhat in between being externally, internally and completely known, i.e. only some of the desired elements have their declaration or expansion, and some have not been mentioned at all. The partially known condition wants to express an *unbalanced* state. The estimation algorithm prefers to avoid partially known states, since if we take into account only internal elements which are at least externally known (obviously we cannot take into account completely unknown objects), we are underestimating object's complexity; on the other hand if we estimate the object's complexity only from an external point of view (i.e, taking into account only information required to qualify for externally known objects) we are neglecting a large amount of useful information;

- the object is *virtually completely known*, in the sense that its details are not specified at this time, but its cost is already known because of one of the following reasons:
  - component is bought from a third-party intellectual property provider, therefore its cost is predetermined and equal to the actual monetary purchase cost;
  - component is reused from a previous project, therefore their cost is zero;

  Virtually completely known objects will be practically instanced by declaring them externally and specifying that their cost is predetermined. This thesis will not take care of virtually completely known objects;

- the object is *completely unknown* so far, in the sense that even the possibility of its existence is unknown; (for example the object will be referenced by an architecture which is only *externally known* so far). Completely unknown objects are obviously not declared at all. Size estimates for completely unknown objects cannot be generated; instead, estimates for externally known objects must take into account the possibility of existence of completely unknown objects.

The following table schematically represents what must be known of a given object in order to qualify for each one of the "knowledge *statai*" described above (U stands for completely unknown, E for externally known, P for partially known, I for internally known, C for completely known and V for virtually completely known).

Knowledge:	U	E	P	I	C	V
Object's external interface is present	·	✓	✓	✓	✓	✓
Declaration of contained elements are present	·	·	✓	✓	✓	·
Body of contained elements is present	·	·	?	·	✓	·
Declaration of referred elements is present	·	·	?	✓	✓	·
Declaration of internal connectivity/variables is present	·	·	?	✓	✓	·
Cost can be estimated	·	✓	✓	✓	✓	✓
Cost is known	·	·	·	·	✓	✓

(Note: the '?' marks in the P column intuitively stand for "sometimes yes, sometimes not", refer to definitions above for formal requisites)

In this thesis we will propose a methodology, composed by a set of models and of rules on how to apply them according to the knowledge status of each element, which will be able to estimate the size of a given project with a confidence interval which becomes narrower and narrower as the knowledge status of each element in the specification of the project evolves from a status with few information to a status with more information.

Chapter 3

Theoretical foundations

In this section we lay the foundations of the development effort estimation method we are going to describe in this thesis, by introducing a number of formal definitions. These definitions are required to create a language that we will exploit throughout all the rest of this thesis to describe things, to express hypotheses and theses, and to detail algorithms. Such a language allows us to express any idea in quasi-natural language, while preserving anytime the freedom from ambiguity, which is required in order to make the whole construction actually implementable.

Remarks and examples are provided not only for the sake of clarity, but also in order to expose to the reader our inspiring principles. In the following pages, a basic competence on VHDL is assumed, otherwise the reader could find a good introduction to both syntax and usage of VHDL by reading text [3].

3.1 Entities

> Definition: an entity is a portion of VHDL code that, when parsed, can be reduced to the `entity_declaration` symbol of the VHDL language standard[1].

Remarks: the primary use of an entity declaration is to declare the existence of an object, whose internal details will be given forward (possibly more than once), and to specify which are the ways for this object to talk to the rest of the world, i.e. its interface. The interface is a collection of port declarations, where each port is a variable that ideally tries to model as much as possible the behavior of a pin of those integrated circuit that can be found in digital electronics. In all except a few cases, an entity declaration reduces to the above elements. But in fact it is possible to specify a list of concurrent statements inside an entity declaration, as long as they are *passive*, that is, they must not affect the operation of the entity in any way. In practice, passive statements allow to include assertion tests and similar instructions that allow to monitor or trace the behavior the given entity.

[1] We are referring to the VHDL-93 language, specified in the IEEE standard 1076-1993. This document is reported in full in chapter 7. Grammar details could slightly differ from VHDL-87 (IEEE standard 1076-1987), but our cost considerations could be applied to the old syntax as well.

Listing 3.1: Example of structural architecture

```
architecture struct of reg4 is
  signal int_clk : bit;
begin
  bit0 : entity work.d_latch(basic)
         port map (d0, int_clk, q0);
  bit1 : entity work.d_latch(basic)
         port map (d1, int_clk, q1);
  bit2 : entity work.d_latch(basic)
         port map (d2, int_clk, q2);
  bit3 : entity work.d_latch(basic)
         port map (d3, int_clk, q3);
  gate : entity work.and2(basic)
         port map (en, clk, int_clk);
end architecture struct;
```

3.2 Architectures

Definition: an architecture is a portion of VHDL code that, when parsed, can be reduced to the `architecture_body` symbol of the VHDL language standard.

Remarks: An architecture can be thought of as an implementation (the contents) of an entity, whereas an entity can be thought of the description of the boundaries of one or more architectures. The above formal definition can be expressed in a more intuitive way: an architecture is that piece of code that begins with "`architecture` arch-name `of` entity-name `is`" and ends with "`end architecture`". Each architecture is described in one of the following ways (or in a mixture of them):

- in terms of the subsystems and interconnections of which it is composed. In this case the architecture is said to be **structural**. Its VHDL code will be typically composed of a number of signal declarations and the instantiation of two or more subsystems, done via a `entity ... port map` construct or via a `component` construct. The code in listing 3.1 illustrates an example of a structural architecture[2].

  The cost of such an architecture is composed mainly by three contributions:

  1. the **subsystem declaration cost**, which expresses the number of lines of source code required to declare components to be used in the architecture body. It is clear that the subsystem instantiation cost strictly depends on the number of declared components and on the complexity of their interface;
  2. the **subsystem instantiation cost**, which expresses the number of lines of source code required to instantiate entities with the `port map` construct (if any) plus the number of lines of source code required to instantiate components with the `component` construct. We believe it is self-evident that the subsystem instantiation cost strictly depends on the number of subsystems components or entities) and the number of port signals which are to be connected;
  3. the **signal declaration cost**, which depends on the number of signals we are using to bring around information among components;

[2]This example is taken from [3]

Listing 3.2: Example of behavioral architecture

```
architecture behavioral of clk_gen is
  constant T_CLK : time := 195 ns;
  signal CLK : std_logic  := '0';
begin
  nRESET <= '0' after 2 ns, '1' after 335 ns;
  process
  begin
    wait for T_CLK;
    CLK <= not(CLK);
    CLK_OUT <= CLK;
  end process;
end behavioral;
```

If there are subsystems instantiated with the `component` construct, in line of principle we have to take into account the **configuration cost**, which expresses the number of lines of source code required to provide a specialization for all the components declared in the architecture body. In this thesis we will safely ignore the cost contribution given by configuration declarations, since their impact on overall project size is negligible.

- in terms of the functions it carries out. In this case the architecture is said to be **behavioral**. Its VHDL code will be typically composed of a number of **processes**, which in turn contain sequential statements. While subsystem instances contained in a structural architecture inherently enforce parallelism, sequential statements belonging to a process, are intended to be executed one after the other, like in a conventional programming language.

  The code in listing 3.2 illustrates an example of a behavioral architecture[3].

  The cost of such an architecture is composed mainly by the **process cost**, which expresses the number of lines of source code included in each process;

- in terms of how input directly influences output. "Directly" means "without passing through any subsystem or process". In this case the architecture is said to be **data-flow** A data flow description makes use of only VHDL built-in operators and concurrent statements.

  The code in listing 3.3 illustrates an example of a data transfer architecture[4].

As said before, an architecture can be a mixture of the above models, as in the example listing 3.1 [5].

3.3 Processes

Definition: a process is portion of VHDL code that, when parsed, can be reduced to the `process_statement` symbol of the VHDL language standard.

[3]This example is taken from Xilinx Application Note XAPP363.
[4]This example is taken from RISC8 core sources, by AnsLab.
[5]This example is taken from [3].

Listing 3.3: Example of data-flow architecture

```
architecture rtl_mux_alua of mux_alua is
begin
  alua  <= fout when (opcode_subwf = '1') or
                     (opcode_decf  = '1') or
                     (opcode_movf  = '1') or
                     (opcode_comf  = '1') or
                     (opcode_incf  = '1') or
                     (opcode_decfsz = '1') or
                     (opcode_rrf   = '1') or
                     (opcode_rlf   = '1') or
                     (opcode_swapf = '1') or
                     (opcode_incfsz = '1') or
                     (opcode_bcf   = '1') or
                     (opcode_bsf   = '1') or
                     (opcode_btfsc = '1') or
                     (opcode_btfss = '1') else
           k     when (opcode_movlw = '1') or
                     (opcode_iorlw = '1') or
                     (opcode_andlw = '1') or
                     (opcode_xorlw = '1') or
                     (opcode_retlw = '1') else
           w;

end rtl_mux_alua;
```

Listing 3.4: Example of mixed model architecture

```
entity multiplier is
  port ( clk, reset : in bit;
         multiplicand, multiplier : in integer;
         product : out integer );
end entity multiplier;
--------------------------------
architecture mixed of multiplier is
  signal partial_product, full_product : integer;
  signal arith_control, result_en, mult_bit, mult_load : bit;
begin -- mixed
  arith_unit : entity work.shift_adder(behavior)
    port map ( addend => multiplicand, augend => full_product,
               sum => partial_product,
               add_control => arith_control);
  result : entity work.reg(behavior)
    port map ( d => partial_product, q => full_product,
               en => result_en, reset => reset);
  multiplier_sr : entity work.shift_reg(behavior)
    port map ( d => multiplier, q => mult_bit,
               load => mult_load, clk => clk);
  product <= full_product;

  control_section : process is
    -- variable declarations for control_section
    -- ...
  begin -- control section
    -- sequential statements to assign values to control signals
    -- ...
    wait on clk, reset;
  end process control_section;
end architecture mixed;
```

Remarks: a process is an ordered collection of sequential statements, used to describe the behavior of an architecture or a part of it. Again, the above formal definition can be expressed in a more intuitive way: a process is that piece of code that begins with "`process`" and ends with "`end process`". Whereas all the statements directly inserted in an architecture body are concurrent (that is, their effects take place at the same time and their order of specification is uninfluent), a process is composed by sequential statements, which are executed one at a time, such that the second statement does not begin execution before the first one is completely carried out. A process resembles a program in a traditional programming language in the sense that variables are available in order to represent the process state, and the control-flow abstraction can be applied, together with all the instructions required to manipulate on it (`if ... then ... elsif ... else ...` instructions, `case ... is ... when ...` instructions, `for ... in ... loop ...` instructions, `while ... loop ...` instructions, procedure calls, etc.). As far as cost is concerned, processes act as a sort of monolithic black box, where it is difficult to operate further subdivisions which are sensible and have a good predictive power at the same time. Our studies show that there is an easy answer to the question "is there an indicator which is highly correlated with process sizes?", and we will show what this indicator should be too, but the data required to calculate this indicator will be unavailable in practically all application cases;

3.4 Subprograms

Definition: a subprogram is portion of VHDL code that, when parsed, can be reduced to the `subprogram_body` symbol of the VHDL language standard.

Remarks: like in the Pascal programming language, from a taxonomical point of view, subprograms are called either functions or procedures, depending on whether they have or not a return value respectively.

3.5 Top-level architecture

Definition: an architecture v_i is said to be top-level iff $\nexists v_j$ such that $v_j \rightarrow v_i$.

Remarks: the $\rightarrow$ symbol represents the *references* relation and will be defined later.

Examples: typically the architectures of the test-bench entity (if there are any) are our only top-level architecture.

3.6 Projects

Definition: a project is a collection of entities and top-level subprograms.

Remarks: the project is our universal superset. It has no special role or properties except it contains all the architectures and those subprograms not defined inside some process or some other subprogram, (i.e. the subprograms defined inside packages). It was defined since we want a formal definition for every object for which defining a cost is meaningful and important, and defining a cost for a project is extremely important, maybe the primary objective of this thesis.

3.7 Syntax objects

Definition:

- a project is a syntax object;
- an entity is a syntax object;
- an architecture is a syntax object;
- a process is a syntax object;
- a subprogram is a syntax object;

Remarks : all the objects we have defined so far are syntax objects. In the following parts of the thesis we could add additional syntax objects, like component declarations and instantiations. We will not speak about the now for sake of simplicity, and they would not add anything to the theory we are presenting, nor would they modify its properties.

3.8 Syntax classes

Definition:

- U, the set of all the projects, is a syntax class;
- E, the set of all the entities, is a syntax class;
- A, the set of all the architectures, is a syntax class;
- P, the set of all the processes, is a syntax class;
- S, the set of all the subprograms, is a syntax class;

3.9 Notation

Definition: given a syntax object v_i, we write:

- $U(v_i) \Leftrightarrow v_i \in U \Leftrightarrow v_i$ is a project;
- $E(v_i) \Leftrightarrow v_i \in E \Leftrightarrow v_i$ is an entity;
- $A(v_i) \Leftrightarrow v_i \in A \Leftrightarrow v_i$ is an architecture;
- $P(v_i) \Leftrightarrow v_i \in P \Leftrightarrow v_i$ is a process;
- $S(v_i) \Leftrightarrow v_i \in S \Leftrightarrow v_i$ is a subprogram;

Remarks: U, E, A, P, S share a dual nature of predicates and sets at the same time. Writing $U(v_i)$ implies that we are regarding U as a predicate, whereas writing $v_i \in U$ implies that U is regarded as a set.

Metaconstraints:

- $U(v_i) \Rightarrow \neg E(v_i) \wedge \neg A(v_i) \wedge \neg P(v_i) \wedge \neg S(v_i)$;
- $E(v_i) \Rightarrow \neg U(v_i) \wedge \neg A(v_i) \wedge \neg P(v_i) \wedge \neg S(v_i)$;
- $A(v_i) \Rightarrow \neg U(v_i) \wedge \neg E(v_i) \wedge \neg P(v_i) \wedge \neg S(v_i)$;
- $P(v_i) \Rightarrow \neg U(v_i) \wedge \neg E(v_i) \wedge \neg A(v_i) \wedge \neg S(v_i)$;
- $S(v_i) \Rightarrow \neg U(v_i) \wedge \neg E(v_i) \wedge \neg A(v_i) \wedge \neg P(v_i)$;

(i.e. informally, object types are mutually exclusive).

3.10 The *contains* relation

Definition: given two syntax objects v_i and v_j, we say that v_i contains v_j and we write $v_i \supset v_j$ iff one of the following cases is true:

1. $A(v_i) \wedge P(v_j)$, and process v_j is defined inside architecture v_i;
2. $P(v_i) \wedge S(v_j)$, and subprogram v_j is defined inside process v_i (in VHDL it is possible to define a subprogram inside a process);
3. $S(v_i) \wedge S(v_j)$, and subprogram v_j is defined inside subprogram v_i (VHDL allows subprogram nesting);
4. $E(v_i) \wedge A(v_j)$, and architecture v_j is belongs to entity v_i (please note that this form of containment is different from all other forms here described, since architecture declarations are not actually encapsulated in the entities to which they belong; but this choice of the people who designed the syntax of the VHDL language has no influence on the actual status of architectures which must belong to some entity. The proof is that we could easily design a new language, identical to VHDL, except for architecture declarations occur inside entity declarations, and the original semantics of VHDL is fully preserved);
5. $U(v_i) \wedge E(v_j)$ (all the entities we are considering make up a project);
6. $U(v_i) \wedge S(v_j)$, and subprogram v_j is defined inside some package which is ideally part of project v_i;

3.11 The *contains** relation

Definition: given two syntax objects v_i and v_j, we say that v_i star-contains v_j and we write $v_i \supset^* v_j$ iff one of the following cases is true:

- $v_i \supset v_j$
- $\exists v_k : v_i \supset^* v_k, v_k \supset v_j$

3.12 The *references* relation

Definition: given two syntax objects v_i and v_j, we say that v_i references v_j and we write $v_i \rightarrow v_j$ iff one of the following cases:

- v_i is an architecture and v_j is an entity, and entity v_j is referenced inside architecture v_i via an `entity ... port map ...` construct or via a `component` construct plus an external `configuration` construct;
- v_i is a process and v_j is a subprogram, and subprogram v_j is called inside the body of process v_i;

3.13 The *references** relation

Definition: given two syntax objects v_i and v_j, we say that v_i star-references v_j and we write $v_i \rightarrow^* v_j$ iff one of the following cases is true:

- $v_i \rightarrow v_j$
- $\exists v_k : v_i \rightarrow v_k, v_k \rightarrow^* v_j$

3.14 The *uses* relation

Definition: given two syntax objects v_i and v_j, we say that v_i uses v_j and we write $v_i \triangleright v_j$ iff $v_i \supset v_j$ or $v_i \rightarrow v_j$

3.15 The *uses** relation

Definition: given two syntax objects v_i and v_j, we say that v_i star-uses v_j and we write $v_i \triangleright^* v_j$ iff one of the following cases is true:

- $v_i \triangleright v_j$
- $\exists v_k : v_i \triangleright v_k, v_k \triangleright^* v_j$

3.16 The syntax object graph

Definition: a syntax object graph (SOG, for short) is a triple (V,E,u) where:

1. $V = \{v : v \text{ is a syntax object }\}$;
2. $E = \{(v, w, r) : v \in V, W \in V, r \in \{\supset, \rightarrow\}\}$;
3. $u : u \in V, U(u), \nexists q \neq u : U(u)$;

and the following metaconstraints[6] are satisfied:

1. $v_i \not\supset^* v_i$;
   (an object cannot contain itself);
2. $v_i \supset^* v_j \Rightarrow v_j \not\supset^* v_i$;
   (if an object contains another, the second one cannot contain the first);
3. $v_i \supset v_j \Rightarrow \nexists v_k : v_k \neq v_i, v_k \supset v_j$;
   (the same object cannot be directly contained in two distinct objects);
4. $A(v_i) \vee P(v_i) \vee S(v_i) \Rightarrow \exists v_j : v_j \supset v_i$;
   (every object except for projects must have a father, i.e. some other object who contains him);

Remarks: V is the set of vertices, E is the set of edges, u represents the project's universal set, which acts as the root of the SOG. Triples belonging to E are ordered, i.e. $(v, w, r) \neq (w, v, r)$. Given a SOG (V,E,u), we say that $(v, w, \supset) \in E \Leftrightarrow v \supset w$ and $(v, w, \rightarrow) \in E \Leftrightarrow v \rightarrow w$.

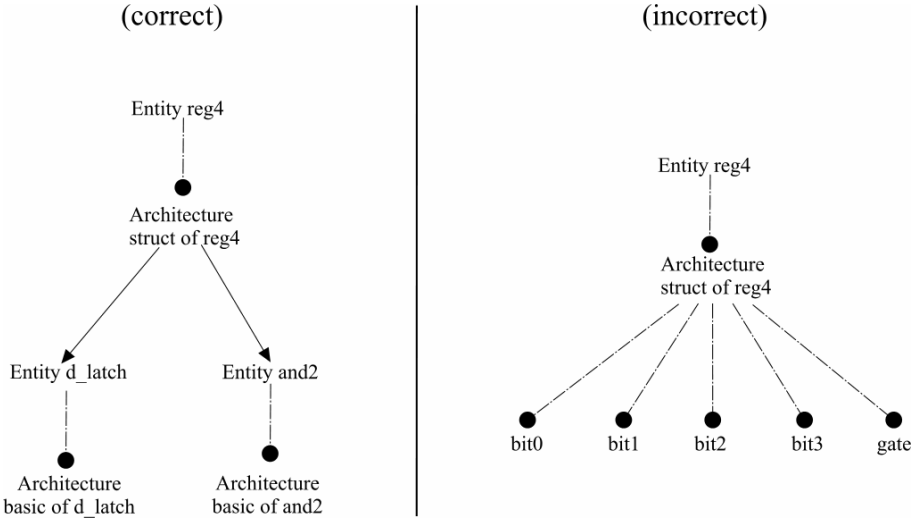
The syntax object graph is the way in which both containment and dependence are represented. To avoid confusion, please note that *references* relation connects component *types*, not *instances*. For example the code in figure 3.5 corresponds to the SOG in figure 3.1 (left), and not to the SOG on the right.

[6]These constraints are called this way because they do not derive from the formalism but from VHDL syntax, which is a *metacontext* with respect to it.

Listing 3.5: Source code for the graph example

```
entity d_latch is
  port ( d, clk : in bit; q : out bit );
end d_latch;

entity and2 is
  port ( a, b : in bit; y : out bit );
end and2;

architecture basic of d_latch is
begin
  latch_behavior : process is
  begin
    if clk = '1' then
      q <= d after 2 ns;
    end if;
    wait on clk, d;
  end process latch_behavior;
end architecture basic;

architecture basic of and2 is
begin
  and2_behavior : process is
  begin
    y <= a and b after 2 ns;
    wait on a, b;
  end process and2_behavior;
end architecture basic;

entity reg4 is
  port ( d0, d1, d2, d3, en, clk : in bit; q0, q1, q2, q3 : out bit );
end entity reg4;

architecture struct of reg4 is
  signal int_clk : bit;
begin
  bit0 : entity work.d_latch(basic)
         port map (d0, int_clk, q0);
  bit1 : entity work.d_latch(basic)
         port map (d1, int_clk, q1);
  bit2 : entity work.d_latch(basic)
         port map (d2, int_clk, q2);
  bit3 : entity work.d_latch(basic)
         port map (d3, int_clk, q3);
  gate : entity work.and2(basic)
         port map (en, clk, int_clk);
end architecture struct;
```

Figure 3.1: The correct and an incorrect syntax object graph for listing 3.5



Example: in figure 3.2 a simple SOG is represented. We have a number of entities, architectures, processes and subprograms. All entities have one architecture except for E_2, which has two associated architectures; associating more than one architectures to an entity is perfectly allowed by the VHDL language specifications, though an uncommon practice except for special purposes. Architectures $a_1, a_{2,2}, a_3$ and a_4 are very likely to be pure structural architectures, since they reference other entities and no processes; architectures a_6 and a_7 on the other side are very likely to be pure behavioral architectures, since they contain processes but no subsystem references. Finally $a_{2,1}, a_8$ and a_5 are very likely to be data-flow architectures, since they do not contain any process or subsystem reference.

Figure 3.2 is in fact a special case of a SOG, since it is a DAG (directed acyclic graph). All the actual projects we gathered in our project base, which are thoroughly described in part III exhibit the property of having an acyclic SOG, nevertheless nothing prevents cycles in a SOG, in particular:

there can be **functional referential recursion**, which is well-known traditional recursion that happen when a subprogram invokes itself (direct recursion), as in $S_1 \rightarrow S_1$ or when a subprogram invokes a second one, and this a third one, and so on, until the last one invokes the first one, as in $S_1 \rightarrow S_2, S_2 \rightarrow S_3, ..., S_{n-1} \rightarrow S_n, S_n \rightarrow S_1$ (indirect recursion);

there can be **functional structural recursion**, which is a complicated name for an extremely simple condition, usually named function nesting, consisting in the possibility to declare a subprogram while inside another subprogram;

there can be **architectural referential recursion**, which happens when an architecture includes smaller instances of itself. Though being an extremely rare practice, Peter J. Ashenden shows in paper [2] that structural recursion is perfectly legal in VHDL, and can be realized quite easily with a proper use of generic constants (which serve to specify the size of the included subsystems), of `generate` statements (in order to vary the

size of the current system according to the desired size, specified via generic constant), and of `configuration` blocks, as in the following example:

```
architecture recursive of fanout_tree is
    component buf
        port ( a : in bit ; y : out bit );
    end component;

    component fanout_tree
        generic ( h : natural ; d : positive );
        port ( input : in bit ; output : out bit_vector(0 to d**h  1) );
    end component;

    signal buffered_input : bit_vector(0 to d  1);
begin
    degenerate_tree : if h = 0 generate
        output(0) <= input;
    end generate degenerate_tree;

    compound_tree : if h > 0 generate
        subtree_array : for i in 0 to d  1 generate
            the_buffer : buf
                port map ( a => input, y => buffered_input(i ) );
            the_subtree : fanout_tree
                generic map ( h => h 1, d => d )
                port map ( input => buffered_input(i),
            output => output(i * d**(h1) to ( i +1) * d**(h1 ) 1) );
        end generate subtree_array;
    end generate compound_tree;
end recursive;

configuration recursive_fanout_tree of fanout_tree is
    for recursive
        for compound_tree
            for subtree_array
                for the_buffer : buf
                    use entity work.buf(behaviour);
                end for;
                for the_subtree : fanout_tree
                    use configuration recursive_fanout_tree;
                end for;
            end for;
        end for;
    end for;
end recursive_fanout_tree;
```

The SOG of a project containing a recursive architecture contains a loop $v_i \rightarrow^* v_i$, in particular a loop made of a single edge if $v_i \rightarrow v_i$ (direct recursion), or a loop made of multiple edges if $v_i \rightarrow v_j, v_j \rightarrow v_k, ..., v_m \rightarrow v_i$ (indirect recursion).

Figure 3.3 contains a real-life example of a SOG, representing the syntax object of a 6502

compatible core by the Free-IP project.

3.17 An example lemma

Hypothesis: $v_p \supset^* v_q \wedge S(v_p)$

Thesis: $S(v_q)$

Intuitive meaning: subprograms, through *contains* relations, can only lead to subprograms.

Proof: by induction.

Let's rename by convenience the elements in the following way: $v_p = v_1, v_2, v_3, \ldots, v_n = v_q$ so that $v_q \supset^* v_p$ can be rewritten (by definition of "$\supset^*$") in the following way: $v_1 \supset v_2$, $v_2 \supset v_3$, $\ldots$, $v_{n-1} \supset v_n$ Proving the thesis can be therefore decomposed in proving the following two:

1. $(v_1 \supset^* v_2 \wedge S(v_1)) \Rightarrow S(v_2)$;
   Since $v_1 \supset^* v_2$ and by construction $\nexists v_k : v_1 \supset v_k, v_k \supset^* v_2$, we must be in case 2 of the definition of "$\supset^*$", therefore it must be $v_1 \supset v_2$. Now, by definition of "$\supset$", the only case in which $S(v_1)$ is case 3, and in that case it is true that $S(v_2)$;

2. $(((v_1 \supset^* v_i \wedge S(v_1)) \Rightarrow S(v_i)) \wedge (v_i \supset v_{i+1})) \Rightarrow S(v_{i+1})$;
   Since $S(v_i)$ and $v_i \supset v_{i+1}$, by definition of "$\supset$", the only case in which $S(v_i)$ is case 3, and in that case it is true that $S(v_{i+1})$;

QED

3.18 The syntax class graph

Definition: a syntax class graph (SCG, for short) is a couple (V, E) where:

1. $V = \{U, E, A, P, S\}$;
2. $E = \{(v, w, r) : v \in V, W \in V, r \in \{\supset, \rightarrow\}\}$;
3. $(v, w, r) \in E \Leftrightarrow$ a SOG (W, F, p) can exist such that: $\exists m, n \in W : (m, n, r) \in F \wedge v(m) \wedge w(n)$.

Remarks: Again, V is the set of vertices (which contains the syntax classes), E is the set of edges, and triples belonging to E are ordered. Condition 3 informally means that an edge of some type (r) is present in the syntax class graph from class v to class w, if and only if it is legal to construct a SOG in which there is an edge (of type r as before) from some syntax object m of syntax class v to some other syntax object n of syntax class w.

SOGs	SCGs
There is an infinite number of possible SOGs, each representing the structure of one of the infinite possible VHDL projects	There is only one possible SCG, and it represents on a syntax class level the possible relations between syntax objects
A SOG is a practically useful objects, since it can be used to model an actual project, and cost calculations can be performed on it.	A SCG is a theoretically useful object, since it simplifies the proof of some theorems regards properties of SOGs.
An edge in a SOG going from object v_i to object v_j means that object v_i actually contains / references v_j.	An edge in a SCG going from a class w_i to a class w_j means that it is allowed for an object of class w_i to contains / reference an object of class w_j.

3.19 Lemma

Hypothesis :

1. (V, E) is a SCG;
2. (W, F, p) is a SOG;
3. $C \in V$;
4. $C \not\supset^* C$;

Thesis: $\forall w \in W : C(w) \Rightarrow w \not\supset^* w$

Informal meaning: if a given *contains*-type loop is not present in the SCG, a corresponding loop *contains*-type cannot exist in a SOG. The SCG represents all the possible loop types that can occur in a SOG, and possibly more, but it never happens that an allowed type of a SOG loop is not represented in the SCG.

Proof: by contradiction. Let us suppose the existence of a loop in the SOG:

$$\exists w \in W : C(w) \wedge w \supset^* w$$

Let's rename by convenience the elements in the following way: $v_p = v_1, v_2, v_3, \ldots, v_n = v_q$ so that $v_q \supset^* v_p$ can be rewritten (by definition of "$\supset^*$") in the following way:
$v_1 \supset v_2, v_2 \supset v_3, \ldots, v_{n-1} \supset v_n$ Proving the thesis can be therefore decomposed in proving the following two:

3.20 Theorem

Thesis: $\nexists v_i : v_i \supset^* v_i$

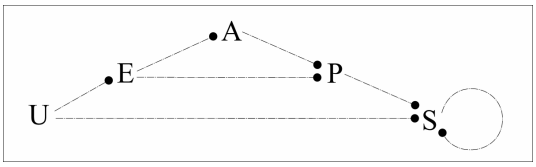
Informal meaning: a sub-SOG composed of *contains* edges only is acyclic.

Proof: by contradiction. First of all let us build a subset of a SCG in which all the nodes of the SCG definition are present but *contains*-type only edges are represented. The whole set of edges is given by the definition of the $\supset$ relation:

- $E \supset P$

- $A \supset P$
- $P \supset S$
- $S \supset S$
- $U \supset E$
- $U \supset S$

The following sub-SCG results:



By examining the sub-SCG it is possible to see that only one loop is present. The loop is composed by one edge only, $S \supset S$. Therefore, by the previous lemma, it is possible to state that no loop can exist in an SOG having *contains*-type edges only, when the looping element is not of S type (i.e. it is not a subprogram).

Now, proving the thesis reduces to proving that $v_i \supset *v_i$ loops are not possible in SOGs when $S(v_i)$. And this is trivial, since that type of loops is forbidden by metaconstraint 1 of the SOG definition.

QED

3.21 Observations

- A SOG is not, in general, a tree. To prove that it is sufficient to exhibit a counterexample in which two architectures belonging to the same project reference a third architecture.
- A SOG is not, in general, a DAG. To prove that it is sufficient to exhibit a counterexample in which there is a recursive function or structural recursion.

3.22 Theorem

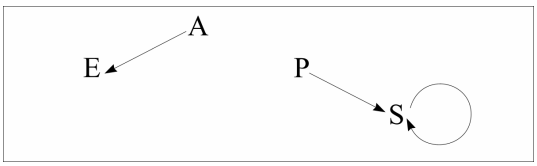
A Sub-SOG whose edges are all of *contains* type is a tree.

Informal proof: given a SOG made of *contains* edges only, the only additional requirement requested to qualify for a tree is the following: there must not exists any node having more two distinct "fathers" (i.e., nodes containing it), and this is forbidden by metacondition 3 in the SOG definition.

3.23 Theorem

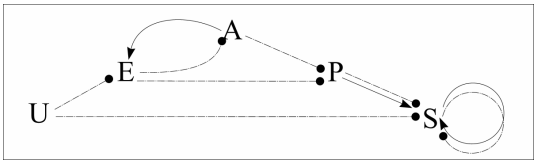
A Sub-SOG free from architectural and functional recursion is a DAG.

Informal proof: as done for the $\supset$ relation, let's represent all the cases allowed by the definition of $\rightarrow$.



It is immediate to see that there are two loops, both comprising elements belonging to the same classes: the first loop represents architectural recursion, the second one functional recursion.

In order to realize the class relation graph for $\triangleright$, we combining the two previous graph, possibly obtaining more loops than the sum of the loops contained in each of the contributing graph:



The resulting graph shows that no additional loops were born. Therefore we have three loops:

- the $E \supset^* A \rightarrow^* E$ loop (architectural referential recursion);
- the $S \supset^* S$ loop (functional structural recursion);
- the $S \rightarrow^* S$ loop (functional referential recursion, possible for the S class but forbidden for every element $\in S$ by metaconstraint 1 of the SOG definition, as said above).

Figure 3.2: An example syntax object graph

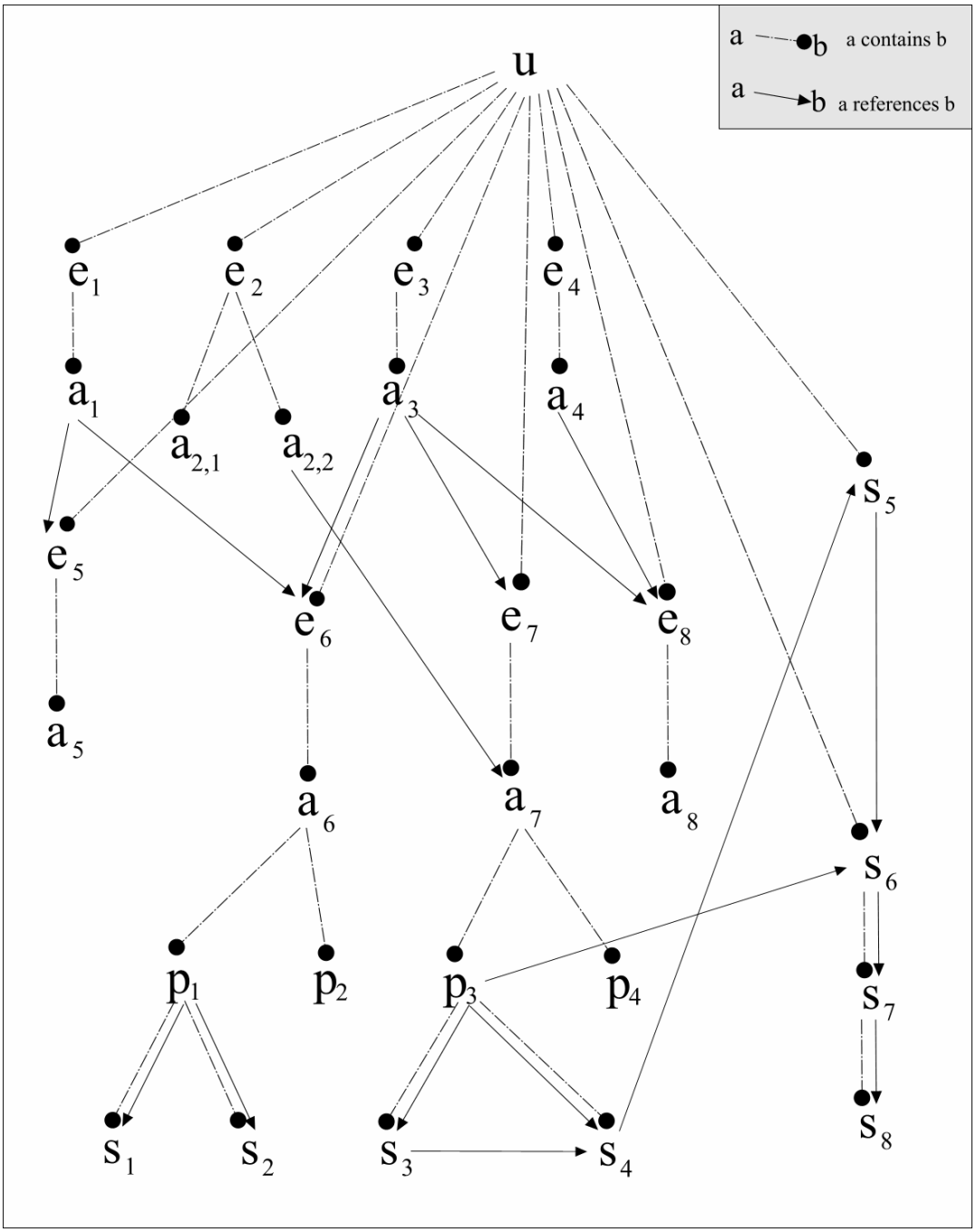
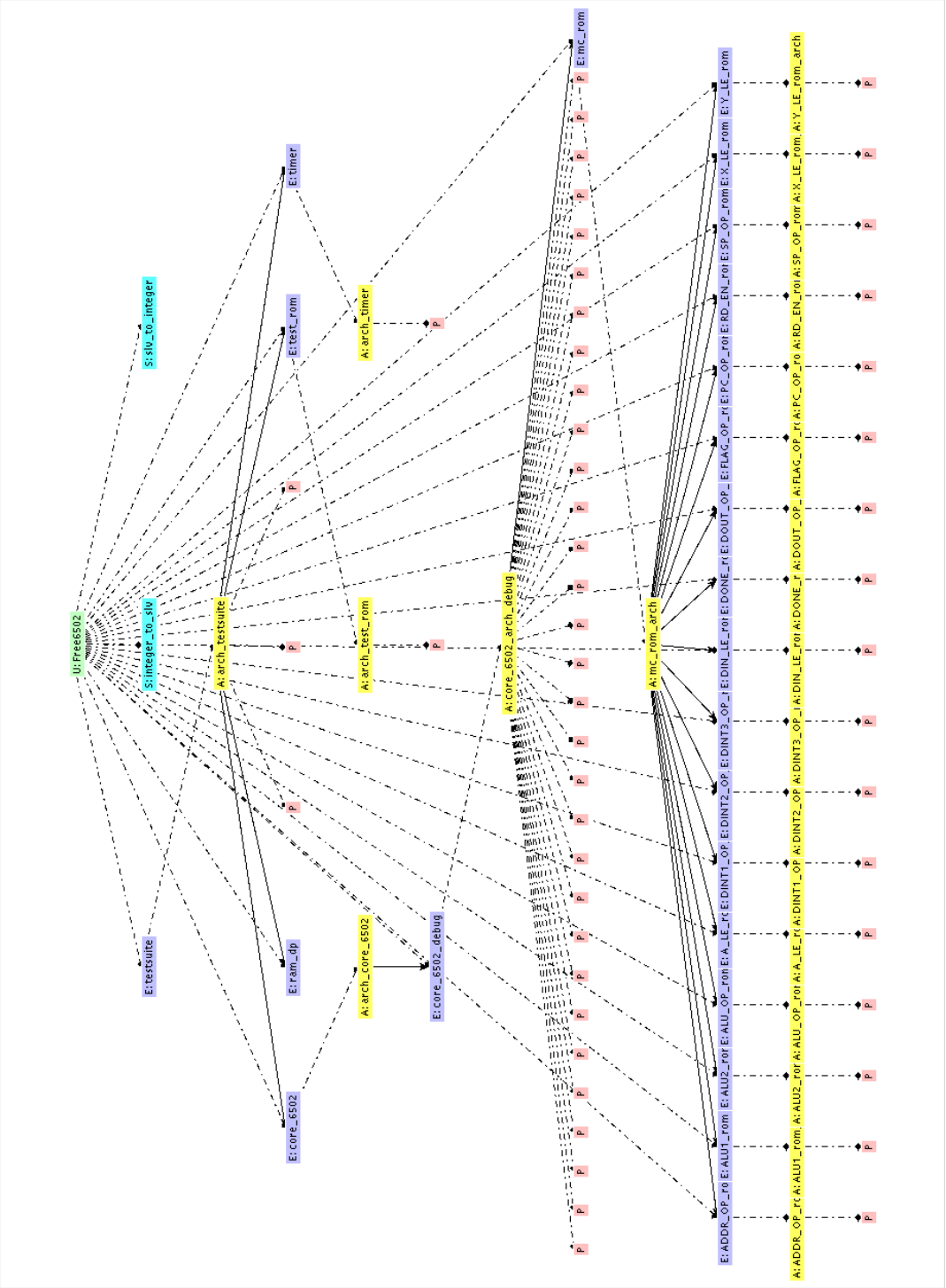


Figure 3.3: A real-life SOG, representing the Free6502 project



Chapter 4

Meta-cognitive information

In the previous sections we talked about the amount of knowledge that is available at a given time about an object. We also tried to give names to possible knowledge levels (see chapter 2, page 6). Then we introduced a way to describe syntax objects and their possible relationships, resulting in a SOG.

Now we collapse the two concepts by introducing the *known syntax object graph* (KSOG for short), which formally defines what is a valid input for our estimation algorithm. At the end of this chapter it will be easy to see that a KSOG is a formalism able to represent a VHDL embedded system design project, at any possible intermediate stage of development.

Let's consider a real, given, completed VHDL project at the time in which it has just been finished by its designers. The project is completed and fully detailed in all its parts, therefore it is rather simple to draw the SOG corresponding to all the syntax objects contained in it. Now let's consider the same project in the status it was two months before: some of the entities have been detailed but there are no associated architectures; some behavioral architectures have signal declarations but internal processes are not yet implemented; there is the conscience that an architecture currently under development could contain subsystem which could eventually contain a whole tree of smaller subsystems, whose details at that given stage of development are difficult to predict. Let's now try to draw a SOG of the project at its, incomplete, development stage. There will be syntax object which are completely known in all their details (let's mark them with a 'C' label), syntax objects known only externally (e.g. the case of the entity with no associated architectures, let's mark them with a 'E' tag), and so on. Informally speaking, we have just built a KSOG.

Let's now proceed with formal definitions in order to better understand what a KSOG is.

4.1 Known syntax objects

Definition: a known syntax object K is a couple (o,s) where:

1. o is a syntax object;
2. $s \in$ {'c', 'e', 'i', 'v'};

Remarks: the known syntax objects are the new nodes that will be used to create KSOGs. The 'c', 'e', 'i', 'v' labels respectively denote completely known, externally known, internally known and virtually completely known objects. Please note that there are no partially

known objects, and no unknown objects (despite example below, where unknown objects are shown with a question mark for exemplification purposes);

Relationships The redefinition of $\supset, \rightarrow, \supset^*, \rightarrow^*, \triangleright, \triangleright^*$ relations in order to extend them from syntax objects to known syntax objects is trivial, then left to the reader. In the following paragraphs we will immediately make use of the newly redefined relations, in a manner that the reader can be easily guessed from what we said so far.

4.2 The known syntax object graph

Definition: a known syntax object graph is a triple (W,E,p) such that :

1. $W = \{w : w$ is a known syntax object $\}$;
2. if $V = \{v : \exists w = (v, a) \in W\}$, then (V,E,p) is a SOG;

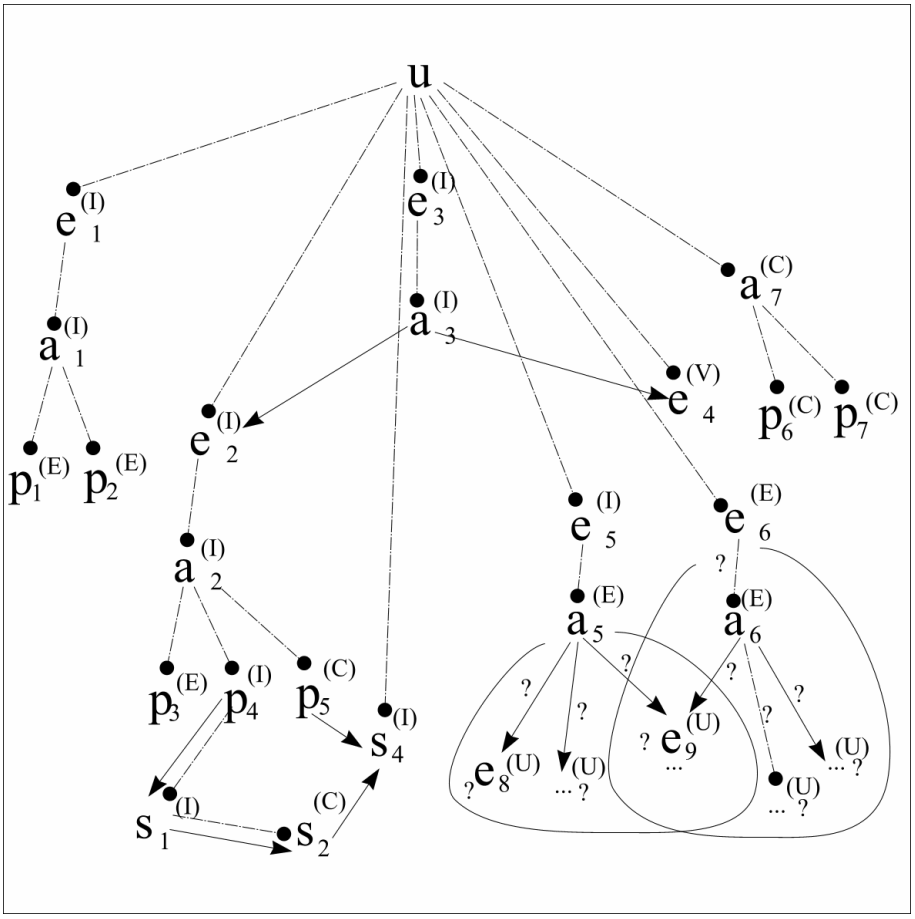
and the following metaconstraints are satisfied:

1. $(w = (o, s) : s =$'c') $\Rightarrow (\forall z : (w \supset z, w = (p, q)) \Rightarrow q =$'c') (when an object is completely known, all the objects contained in it are too);
2. $(w = (o, s) : s =$'i') $\Rightarrow (\forall z : (w \supset z, w = (p, q)) \Rightarrow q =$'e') (when an object is internally known, all contained objects must be at least externally known);
3. $(w = (o, s) : s =$'v') $\Rightarrow (\nexists z \in W : w \supset z)$ (when an object is virtually completely known, details are not specified);
4. $(w = (o, s) : s =$'e') $\Rightarrow (\nexists z \in W : w \supset z \vee w \rightarrow z)$ (by definition of externally known);

Example: figure 4.1 illustrates a non-trivial example of a KSOG. As usual, elements named u_i, e_i, a_i, p_i, s_i for some i are respectively projects, entities, architectures, processes and subprograms. Their knowledge state is represented in parentheses. Elements enclosed in a curve line and marked with "(U)" are completely unknown and are **not** part of the graph, they are only provided for clarification purposes.

Entities e_1, e_2, e_3 are internally known, therefore their details must be present, namely architectures a_1, a_2, a_3. Since are architectures a_1, a_2, a_3 are themselves internally known, their details must be declared too, like it happens for processes p_1 and p_2 contained in a_1 which are externally known (no further details are given), or like p_3, p_4, p_5, of which the first is externally known, the second is internally known (and subprogram s_1 must be at least externally known but it this case it is internally known and in particular it contains a nested subprogram s_2 which is completely known and references a package subprogram s_4), and the third is completely known and references the already-mentioned s_4 which is internally known and implements its features without containing or referring to any other element. Architecture a_5 is externally known, thus no further details are given and nothing prevents a future refinement of the specifications represented in the KSOG to include elements like e_8 or e_7, which are currently completely unknown and therefore they are **not** part of the graph. e_9 represents all the elements that could be referenced by both a_5 and a_6, thus to say that our estimation algorithm must cope in some way with the estimation of the complexity of completely unknown subgraphs which are possibly partly **overlapped**. Architecture a_7 is completely known and so must be all the objects in it contained.

Figure 4.1: An example of known syntax object graph



4.3 Branch

Definition: given a KSOG K, a branch is a known syntax object $v_i \in K$ such that it is at least internally known, and it contains at least one known syntax object, $\exists v_j \in K : v_i \supset v_j$.

Remarks: the complexity of a branch is the sum of the complexities of all the contained elements

Example: a_1, a_2, a_7 in the example KSOG are branches.

4.4 Leaf

Definition: (Part I) Given a KSOG K, any internally known, known syntax object $v_i \in K$, such that it has no contained syntax objects, i.e. $\nexists v_j \in K : v_i \supset v_j$, is a leaf.

(Part II) Given a KSOG K, any externally known, known syntax object $v_i \in K$, such that the following metainformation is available: "in all the possible future refinements of the current KSOG, v_i **will not** contain any other object", is a leaf too.

Nothing else is a leaf.

Remarks: please note that the second part of the definition includes a metainformation, which is an information not belonging to the KSOG domain but to the project domain, of which the KSOG is an incomplete representation. Moreover, providing the metainformation required by part II of the above definition requires some deep understanding of the project semantics and developer intentions that we do not want to deal with herewith. It is a sort of "oracle response."

A leaf is a terminal object in the KSOG, an object not subject to future refinements.

Example: $p_1, p_2, p_3, p_5, a_3, p_6, p_7$ in the example KSOG are leafs.

4.5 Bud

Definition: given a KSOG K, any known syntax object $v_i \in K$, such that it is neither a branch, nor a leaf, is a bud.

Remarks: given the KSOG $K(W, E, p)$, this definition, together with the above two, formally allows to induce a complete partition on W. This partition is extremely useful for estimation purposes: cost of leaf objects will be determined on the basis of particular models, branch costs will be determined by the sum of the cost yielded by the same models plus the cost yielded by suitable models applied to all the containment subtree rooted at the given branch node, bud costs will be estimated by the use of more complex heuristic models.

4.6 Bunch

Definition: given a KSOG $Q(V, E, p)$ and entity e (belonging to the SOG), we define as the **bunch** B **of the entity** e the graph $B_e(W, F)$, such that

$$W = \{w : w \in V \wedge e \supset^* w\} \cup \{e\}$$

and

$$F = \{(v_i, v_j, \supset) : (v_i, v_j, \supset) \in E \wedge v_i \in W, v_j \in W\}$$

Remarks: intuitively, a bunch is composed by an entity, all the architectures implementing that entity (note that in all except a really few cases there is exactly one entity for each architecture), and all the processes, subprograms, and other known syntax objects contained in them. Properties and purpose of bunches will be better explained in chapter 14. Bunches can also be called, more formally, **entity containment object graphs**, or ECOGs, for short.

Part II

The Tools

Chapter 5

Tool overview

In the following parts of this thesis, we will introduce a wealthy set of real VHDL projects (part III), on which our methodology is based and against which it is validated, and (in part IV) a relevant number of hypotheses and models, which collectively make up the methodology that we are presenting.

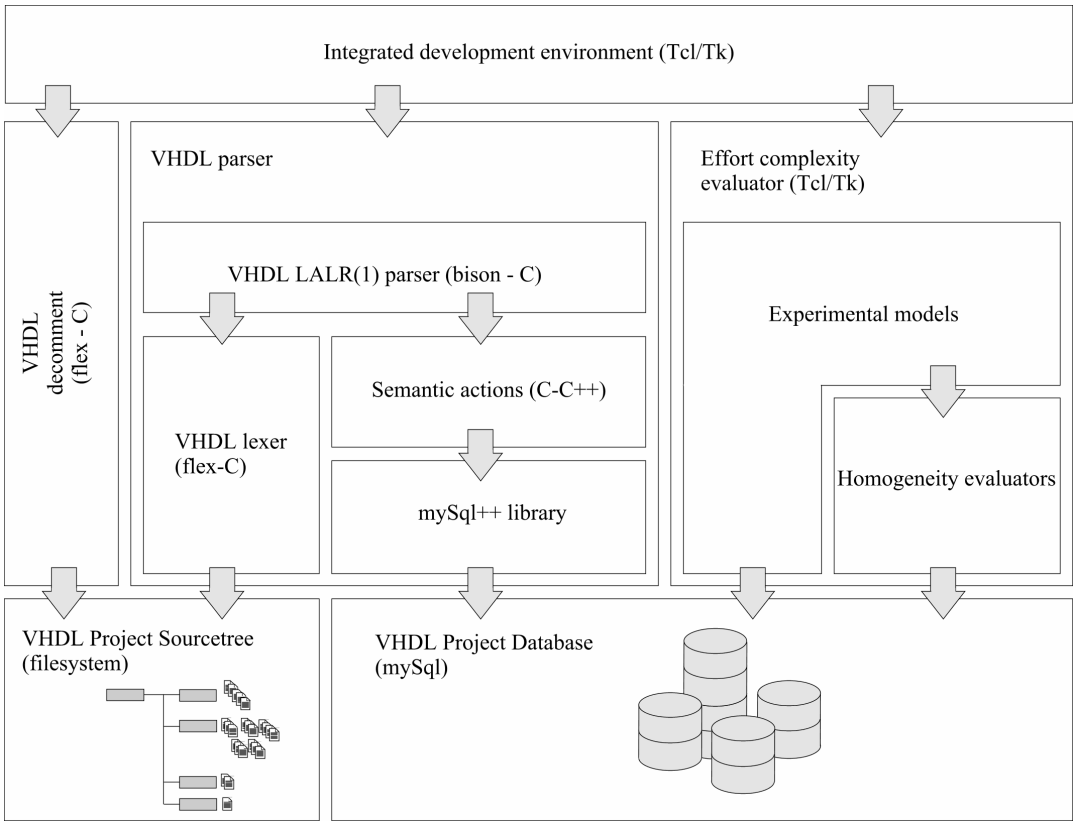
In this chapter, we focus on the tools that we designed to perform proper data extraction from VHDL projects described in part III, and to experimentally derive and apply models described in part IV. So far we have been talking about data on one hand and models on the other, but we ignored what stands in between, and constitute the most time-consuming activity of this thesis: the development of tools.

Although not being the most innovative and original part of the thesis (relying on well-known and widely spread technologies like `bison` and `SQL`), tools are a large part of the hard work. More important, without tools it would be impossible not only to check hypothesis and validate models, but also to collect and process any statistical data, which is fundamental for the task of model creation.

The tools we have developed are fundamentally four:

- a lexical filter, called `decomment`, whose purpose is to remove comments and empty lines from input source files;
- a VHDL parser, whose purpose is to read the content of a VHDL source file and represents its content in a hierarchically organized structure stored in a SQL database;
- a rich and complex set of scripts which implement all the experimental models presented in this thesis (and many more), applying models to data and collecting results;
- an integrated environment with a rich graphical user interface, that automates the operation of the above tools;

Figure 5.1: Tool diagram



Chapter 6

Dealing with comment lines

Obviously, the VHDL parser I designed can correctly deal with comments and blank lines, nevertheless, because of the way behavioral, structural and data-flow lines are defined and categorized, comment and blank lines not belonging to any process or component definition or instantiation are categorized as dataflow, thus incorrectly biasing the results and obtaining a "dataflow-ability" index which is greater than reality.

Therefore it is a better approach to remove comment and blank lines before the source files are passed to the parser. In order to do that, a simple `flex`-based lexical analyzer called `decomment` was designed.

Since this filter acts as a sort of language pre-processor, whose output is given as input to the actual parser, the parser does not know anymore of the actual position of language constructs and identifiers inside the original source code file.

Line numbers annotated into the database during the parsing operations are therefore relative to the stripped version of the source file. The same applies to line numbers reported in parse error messages. This is not a problem, since both versions (the original one and the stripped one with a `.decomment` extension added to the original name) are retained and when the user double-clicks on a source file name inside the GUI tool, both versions are opened.

The full source code of the decomment filter is reported below.

```
%option noyywrap
%s INSIDE_STRING

%{
#include <stdio.h>
#define YYLMAX 1024
%}

space                   [ \t\n\v\r\l\f]

%%
^{space}*\n
^{space}*\-\-.*\n
<INITIAL>\-\-.*\n       { puts(""); }
<INITIAL>\"             { ECHO; BEGIN(INSIDE_STRING); }
<INSIDE_STRING>\"       { ECHO; BEGIN(INITIAL); }
.                       { ECHO; }
```

```
%%

main( int argc , char** argv )
{
  ++argv, --argc; /* skip over program name */
  if ( argc > 0 )
    yyin = fopen( argv[0], " r " );
  else
    yyin = stdin;
  yylex();
}
```

The above rules are rather straightforward:

- rule in line 12 removes empty lines;
- rule in line 13 removes lines containing only a comment, possibly preceded by any number of empty spaces or lines;
- rule in line 14 removes comments which appear at the end of lines possibly containing code;
- rule in line 15 switches to INSIDE_STRING state when the beginning of a string literal is encountered;
- rule in line 16 switches back to INITIAL state when the end of a string literal is encounter;
- thanks to the above two rules, the lexical analyzer jumps in INSIDE_STRING when a string literal begins and back to INITIAL state when it ends; the INSIDE_STRING serves to prevent rules in lines 14 and 15 (comment removal) to be activated when a couple of hyphens appear inside a string literal.

By applying the above filter, we preprocessed all the VHDL source files composing our project base (tuning project set, described in the first chapter of part III) and we obtained the following statistics.

	Lines of code	Bytes
Original size:	388,790	16,478,757
Size after filtering:	319,049	14,441,893
Discarded elements:	69,741	2,036,864
Percentage of discarded elements:	17.9%	12.4%

It is possible to say that 17.9% of all the lines of VHDL in the project base and 12.4% of all the characters belong to comments or blank lines. This information should be helpful when the full number of lines of code (including comments and cosmetic empty lines) is to be estimated, instead of pure lines of code.

Average length of a line:	42.61 characters
Average length of a non-comment/blank line:	45.65 characters
Average length of a comment/blank line:	28.64 characters

The following table shows in detail the impact of comments and blank lines over each file belonging to tuning projects (which will be described in part III of this thesis). Column 'L' contains the full number of lines of which the current file is composed, column 'D' contains the number of lines after the `decomment` filter was applied. Column 'X' contains the number of lines removed during the process. Column 'X%' contains the percentage of such lines. Columns 'Lb', 'Db', 'Xb' and 'Xb%' respectively contain the same information, but they express number of bytes and not of lines.

Project/Filename	L	D	X	X%	Lb	Db	Xb	Xb%
ADC0808_int/ad_converter.vhd	199	140	59	29,6%	5200	4117	1083	20,8%
ADC0808_int/en_register.vhd	102	61	41	40,2%	3146	2201	945	30,0%
ADC0808_int/multiplex.vhd	32	19	13	40,6%	756	458	298	39,4%
ADC0808_int/nregister.vhd	43	19	24	55,8%	1196	549	647	54,1%
Am2901/alg_beh2901.vhdl	272	174	98	36,0%	7807	5513	2294	29,4%
Am2901/alu.vhdl	94	40	54	57,4%	3228	1418	1810	56,1%
Am2901/alu_inputs.vhdl	72	42	30	41,7%	1814	1023	791	43,6%
Am2901/funct_block_alg_-beh2901.vhdl	199	94	105	52,8%	5831	3201	2630	45,1%
Am2901/mem.vhdl	53	22	31	58,5%	1543	717	826	53,5%
Am2901/MVL7_functions.vhd	363	292	71	19,6%	12872	10068	2804	21,8%
Am2901/op_sh.vhdl	81	25	56	69,1%	1780	901	879	49,4%
Am2901/q_reg.vhdl	57	22	35	61,4%	1494	672	822	55,0%
Am2901/synthesis_types.vhd	18	4	14	77,8%	551	158	393	71,3%
Am2901/test_vectors.vhdl	307	173	134	43,6%	5119	3174	1945	38,0%
Am2901/test_vectors_2901.vhdl	11084	6403	4681	42,2%	135799	98611	37188	27,4%
Am2901/test_vectors_alu.vhdl	661	473	188	28,4%	11338	8784	2554	22,5%
Am2901/test_vectors_alu_-input.vhdl	223	149	74	33,2%	4612	2879	1733	37,6%
Am2901/test_vectors_mem.vhdl	441	275	166	37,6%	7654	5290	2364	30,9%
Am2901/test_vectors_op_-sh.vhdl	397	275	122	30,7%	7437	5576	1861	25,0%
Am2901/test_vectors_q_reg.vhdl	307	173	134	43,6%	5076	3173	1903	37,5%
Am2901/types.vhd	1105	472	633	57,3%	31064	14713	16351	52,6%
an-XC2S-USB/te-bl.vhd	432	298	134	31,0%	14887	9770	5117	34,4%
an-XC2S-USB/xc2sCore.vhd	90	61	29	32,2%	2125	1331	794	37,4%
an-XC2S-USB/xc2sFPGA.vhd	236	192	44	18,6%	6438	5368	1070	16,6%
an-XC2S-USB/xc2sFunc.vhd	45	18	27	60,0%	1210	410	800	66,1%
an-XC2S-XR16/Core.vhd	245	212	33	13,5%	6064	5208	856	14,1%
an-XC2S-XR16/demo.vhd	144	68	76	52,8%	5824	3468	2356	40,5%
an-XC2S-XR16/FPGA.vhd	248	207	41	16,5%	6602	5582	1020	15,4%
an-XC2S-XR16/glue.vhd	168	111	57	33,9%	7452	4409	3043	40,8%
an-XC2S-XR16/te-bl.vhd	437	305	132	30,2%	14919	9902	5017	33,6%
an-XC2S-XR16/tpl1.vhd	98	56	42	42,9%	3760	2686	1074	28,6%
an-XC2S-XR16/vgaCHAR.vhd	1167	1005	162	13,9%	32788	16044	16744	51,1%
ansRisc8/alu.vhd	261	232	29	11,1%	7955	7210	745	9,4%
ansRisc8/aluop_gen.vhd	100	64	36	36,0%	4482	2828	1654	36,9%
ansRisc8/alu_dp.vhd	131	87	44	33,6%	5821	4233	1588	27,3%
ansRisc8/ans_risc8.vhd	393	365	28	7,1%	12082	11333	749	6,2%
ansRisc8/clock_div.vhd	51	32	19	37,3%	1549	729	820	52,9%
ansRisc8/control.vhd	281	248	33	11,7%	8214	7398	816	9,9%
ansRisc8/inst_decoder.vhd	356	322	34	9,6%	10597	9774	823	7,8%
ansRisc8/ir_decode.vhd	108	77	31	28,7%	4621	3715	906	19,6%
ansRisc8/ir_reg.vhd	96	42	54	56,3%	3705	1359	2346	63,3%
ansRisc8/mux_alua.vhd	70	52	18	25,7%	2608	1961	647	24,8%
ansRisc8/mux_alub.vhd	67	48	19	28,4%	2599	1939	660	25,4%
ansRisc8/mux_cz_write.vhd	66	47	19	28,8%	2392	1725	667	27,9%
ansRisc8/mux_fin.vhd	74	50	24	32,4%	3074	2078	996	32,4%
ansRisc8/mux_fwe.vhd	65	48	17	26,2%	2529	1897	632	25,0%
ansRisc8/mux_win.vhd	76	54	22	28,9%	3108	2244	864	27,8%
ansRisc8/pic_pak.vhd	84	51	33	39,3%	4179	3282	897	21,5%

(continued on next page)

(continued from previous page)

Project/Filename	L	D	X	X%	Lb	Db	Xb	Xb%
ansRisc8/prog_count.vhd	123	80	43	35,0%	4096	2791	1305	31,9%
ansRisc8/reg_files.vhd	108	79	29	26,9%	3772	2605	1167	30,9%
ansRisc8/reg_fsr.vhd	55	25	30	54,5%	1737	652	1085	62,5%
ansRisc8/reg_ioport.vhd	59	41	18	30,5%	1780	1152	628	35,3%
ansRisc8/reg_status.vhd	93	48	45	48,4%	2745	1349	1396	50,9%
ansRisc8/reg_top.vhd	348	311	37	10,6%	10372	9548	824	7,9%
ansRisc8/reg_w.vhd	40	21	19	47,5%	1139	491	648	56,9%
ansRisc8/alu_dp_TB.vhd	232	184	48	20,7%	6824	5697	1127	16,5%
ansRisc8/alu_dp_TB_tim_-cfg.vhd	25	6	19	76,0%	1203	144	1059	88,0%
ansRisc8/Ans_RISC8_Top.vhd	114	79	35	30,7%	3399	2532	867	25,5%
ansRisc8/Ans_RISC8_Top_-TB.vhd	108	80	28	25,9%	2946	2383	563	19,1%
ansRisc8/Ans_RISC8_Top_TB_-tim_cfg.vhd	20	6	14	70,0%	791	172	619	78,3%
ansRisc8/qatools.vhd	85	62	23	27,1%	2726	1832	894	32,8%
ansRisc8/syn_rom_2048x12.vhd	334	294	40	12,0%	11634	10852	782	6,7%
ATL18/ATL18_1818_iox_-Vcomp.vhd	42	19	23	54,8%	1114	568	546	49,0%
ATL18/ATL18_1818_iox_-Vital.vhd	121	68	53	43,8%	2952	2020	932	31,6%
ATL18/ATL18_1818_iox_-Vtabs.vhd	133	88	45	33,8%	4750	3801	949	20,0%
ATL18/ATL18_1825_iox_-Vcomp.vhd	88	49	39	44,3%	2384	1663	721	30,2%
ATL18/ATL18_1825_iox_-Vital.vhd	315	186	129	41,0%	7800	5911	1889	24,2%
ATL18/ATL18_1825_iox_-Vtabs.vhd	133	88	45	33,8%	4750	3801	949	20,0%
ATL18/ATL18_1833_iox_-Vcomp.vhd	156	93	63	40,4%	4185	3207	978	23,4%
ATL18/ATL18_1833_iox_-Vital.vhd	617	372	245	39,7%	15136	11815	3321	21,9%
ATL18/ATL18_1833_iox_-Vtabs.vhd	133	88	45	33,8%	4750	3801	949	20,0%
ATL18/ATL18_18_cell_-Vcomp.vhd	4399	3208	1191	27,1%	138547	125743	12804	9,2%
ATL18/ATL18_18_cell_Vital.vhd	18793	12868	5925	31,5%	546164	472850	73314	13,4%
ATL18/ATL18_18_cell_-Vtabs.vhd	389	312	77	19,8%	12132	10820	1312	10,8%
ATL18/ATL18_18_macro_-Vcomp.vhd	81	41	40	49,4%	1957	1208	749	38,3%
ATL18/ATL18_18_macro_-Vital.vhd	284	156	128	45,1%	6385	4470	1915	30,0%
ATL18/ATL18_18_macro_-Vtabs.vhd	33	14	19	57,6%	966	475	491	50,8%
ax8/A90S1200.vhd	253	170	83	32,8%	8220	5053	3167	38,5%
ax8/A90S2313.vhd	389	289	100	25,7%	12307	8461	3846	31,3%
ax8/AX8.vhd	751	651	100	13,3%	25283	22049	3234	12,8%
ax8/AX_ALU.vhd	303	216	87	28,7%	10230	7700	2530	24,7%
ax8/AX_Pack.vhd	273	210	63	23,1%	7948	5875	2073	26,1%
ax8/AX_PCS.vhd	139	82	57	41,0%	4214	2164	2050	48,6%
ax8/AX_Port.vhd	110	53	57	51,8%	3684	1675	2009	54,5%
ax8/AX_RAM.vhd	92	34	58	63,0%	3119	996	2123	68,1%
ax8/AX_Reg.vhd	292	206	86	29,5%	10540	6948	3592	34,1%
ax8/AX_Reg2.vhd	279	214	65	23,3%	8434	6150	2284	27,1%
ax8/AX_TC16.vhd	343	280	63	18,4%	9820	7659	2161	22,0%
ax8/AX_TC8.vhd	159	102	57	35,8%	4469	2417	2052	45,9%
ax8/AX_UART.vhd	286	218	68	23,8%	7987	5697	2290	28,7%
ax8/ROM1200.vhd	71	67	4	5,6%	3030	2479	551	18,2%
ax8/ROM2313.vhd	686	682	4	0,6%	35625	28924	6701	18,8%
ax8/TestBench1200.vhd	29	19	10	34,5%	735	715	20	2,7%
ax8/TestBench2313.vhd	72	57	15	20,8%	1723	1661	62	3,6%

(continued on next page)

(continued from previous page)

Project/Filename	L	D	X	X%	Lb	Db	Xb	Xb%
dlx/alu-behaviour.vhdl	81	44	37	45,7%	2836	1605	1231	43,4%
dlx/alu.vhdl	45	13	32	71,1%	1534	325	1209	78,8%
dlx/alu_types.vhdl	39	6	33	84,6%	1440	218	1222	84,9%
dlx/bv_arithmetic-body.vhdl	982	547	435	44,3%	32574	21085	11489	35,3%
dlx/bv_arithmetic.vhdl	238	68	170	71,4%	8151	3227	4924	60,4%
dlx/bv_test-bench.vhdl	1121	769	352	31,4%	34607	25683	8924	25,8%
dlx/bv_test.vhdl	36	3	33	91,7%	1294	37	1257	97,1%
dlx/cache-behaviour.vhdl	310	238	72	23,2%	10543	8590	1953	18,5%
dlx/cache.vhdl	61	27	34	55,7%	2662	846	1816	68,2%
dlx/cache_types.vhdl	36	3	33	91,7%	1321	92	1229	93,0%
dlx/clock_gen-behaviour.vhdl	48	12	36	75,0%	1605	352	1253	78,1%
dlx/clock_gen.vhdl	42	8	34	81,0%	1594	253	1341	84,1%
dlx/clock_gen_test-bench.vhdl	50	13	37	74,0%	1599	339	1260	78,8%
dlx/clock_gen_test.vhdl	33	2	31	93,9%	1290	45	1245	96,5%
dlx/controller-behaviour.vhdl	858	701	157	18,3%	29259	25583	3676	12,6%
dlx/controller.vhdl	71	39	32	45,1%	2699	1467	1232	45,6%
dlx/dlx-behaviour.vhdl	614	529	85	13,8%	23374	20914	2460	10,5%
dlx/dlx-instrumented.vhdl	687	574	113	16,4%	26403	22696	3707	14,0%
dlx/dlx-rtl.vhdl	286	223	63	22,0%	10019	8458	1561	15,6%
dlx/dlx.vhdl	50	18	32	64,0%	2051	502	1549	75,5%
dlx/dlx_bus_monitor-behaviour.vhdl	171	116	55	32,2%	5425	3722	1703	31,4%
dlx/dlx_bus_monitor.vhdl	50	18	32	64,0%	2063	518	1545	74,9%
dlx/dlx_instr-body.vhdl	258	205	53	20,5%	10542	9245	1297	12,3%
dlx/dlx_instr.vhdl	257	184	73	28,4%	11479	9248	2231	19,4%
dlx/dlx_test-bench.vhdl	104	60	44	42,3%	3616	1771	1845	51,0%
dlx/dlx_test-bench_cache.vhdl	139	89	50	36,0%	5245	2868	2377	45,3%
dlx/dlx_test.vhdl	33	2	31	93,9%	1260	33	1227	97,4%
dlx/dlx_test_behaviour.vhdl	59	21	38	64,4%	1997	727	1270	63,6%
dlx/dlx_test_cache.vhdl	74	32	42	56,8%	2504	1213	1291	51,6%
dlx/dlx_test_instrumented.vhdl	60	21	39	65,0%	2024	698	1326	65,5%
dlx/dlx_test_rtl.vhdl	103	64	39	37,9%	3287	1940	1347	41,0%
dlx/dlx_types-body.vhdl	45	10	35	77,8%	1534	298	1236	80,6%
dlx/dlx_types.vhdl	44	9	35	79,5%	1763	455	1308	74,2%
dlx/images-body.vhdl	167	104	63	37,7%	4589	2643	1946	42,4%
dlx/images.vhdl	61	5	56	91,8%	2010	199	1811	90,1%
dlx/images_test-bench.vhdl	89	30	59	66,3%	2882	753	2129	73,9%
dlx/images_test.vhdl	35	2	33	94,3%	1282	39	1243	97,0%
dlx/ir-behaviour.vhdl	88	48	40	45,5%	2767	1445	1322	47,8%
dlx/ir.vhdl	46	14	32	69,6%	1632	407	1225	75,1%
dlx/latch-behaviour.vhdl	43	9	34	79,1%	1409	167	1242	88,1%
dlx/latch.vhdl	41	9	32	78,0%	1447	222	1225	84,7%
dlx/memory-behaviour.vhdl	226	155	71	31,4%	6825	4816	2009	29,4%
dlx/memory.vhdl	54	17	37	68,5%	2250	523	1727	76,8%
dlx/memory_test-bench.vhdl	330	259	71	21,5%	10707	9125	1582	14,8%
dlx/memory_test.vhdl	33	2	31	93,9%	1274	39	1235	96,9%
dlx/mem_types.vhdl	36	3	33	91,7%	1321	98	1223	92,6%
dlx/mux2-behaviour.vhdl	37	6	31	83,8%	1380	138	1242	90,0%
dlx/mux2.vhdl	41	9	32	78,0%	1448	220	1228	84,8%
dlx/reg_1_out-behaviour.vhdl	51	15	36	70,6%	1602	337	1265	79,0%
dlx/reg_1_out.vhdl	42	10	32	76,2%	1496	250	1246	83,3%
dlx/reg_2_1_out-behaviour.vhdl	58	21	37	63,8%	1796	500	1296	72,2%
dlx/reg_2_1_out.vhdl	44	11	33	75,0%	1566	289	1277	81,5%
dlx/reg_2_out-behaviour.vhdl	56	20	36	64,3%	1728	462	1266	73,3%
dlx/reg_2_out.vhdl	42	10	32	76,2%	1512	265	1247	82,5%
dlx/reg_3_out-behaviour.vhdl	61	25	36	59,0%	1851	583	1268	68,5%
dlx/reg_3_out.vhdl	42	10	32	76,2%	1527	278	1249	81,8%
dlx/reg_file-behaviour.vhdl	76	29	47	61,8%	2259	897	1362	60,3%
dlx/reg_file.vhdl	46	14	32	69,6%	1563	339	1224	78,3%
ERC32/board.vhd	506	400	106	20,9%	14741	10072	4669	31,7%
ERC32/factlib.vhd	492	311	181	36,8%	17991	9987	8004	44,5%

(continued on next page)

(continued from previous page)

Project/Filename	L	D	X	X%	Lb	Db	Xb	Xb%
ERC32/fpurt_lib.vhd	3369	2530	839	24,9%	125841	90731	35110	27,9%
ERC32/iurt_lib.vhd	3300	2498	802	24,3%	136606	96610	39996	29,3%
ERC32/meclibrary.vhd	22140	15284	6856	31,0%	807154	603926	203228	25,2%
ERC32/memory.vhd	304	244	60	19,7%	9074	8727	347	3,8%
ERC32/mms.vhd	3787	2595	1192	31,5%	130977	93917	37060	28,3%
ERC32/sparc_lib.vhd	3995	2940	1055	26,4%	166612	123754	42858	25,7%
Free6502/free6502.vhd	974	772	202	20,7%	31394	25645	5749	18,3%
Free6502/microcode.vhd	2623	2268	355	13,5%	115962	106289	9673	8,3%
Free6502/ramlib_sim.vhd	191	128	63	33,0%	6020	3935	2085	34,6%
Free6502/testrom.vhd	4176	4128	48	1,1%	264148	213819	50329	19,1%
Free6502/testsuite.vhd	251	170	81	32,3%	7973	5666	2307	28,9%
GL85/i8085.vhd	1790	1673	117	6,5%	115442	106197	9245	8,0%
GL85/acc_ctrl.vhd	68	41	27	39,7%	2865	2197	668	23,3%
GL85/alulogic.vhd	88	70	18	20,5%	3330	2943	387	11,6%
GL85/alu_8bit.vhd	65	43	22	33,8%	2119	1530	589	27,8%
GL85/alu_ctrl.vhd	81	61	20	24,7%	3784	3201	583	15,4%
GL85/bc_pc_sp.vhd	54	38	16	29,6%	1956	1674	282	14,4%
GL85/buf8.vhd	21	15	6	28,6%	782	700	82	10,5%
GL85/ctl_lgc1.vhd	48	28	20	41,7%	1649	1160	489	29,7%
GL85/ctl_lgc2.vhd	62	45	17	27,4%	1965	1749	216	11,0%
GL85/dataaddr.vhd	69	54	15	21,7%	2300	2093	207	9,0%
GL85/decod2_4.vhd	21	14	7	33,3%	597	509	88	14,7%
GL85/decod3_8.vhd	31	24	7	22,6%	1174	1086	88	7,5%
GL85/flagunit.vhd	116	97	19	16,4%	5584	5038	546	9,8%
GL85/g16bctr.vhd	55	36	19	34,5%	1982	1572	410	20,7%
GL85/g4bctr.vhd	78	69	9	11,5%	3625	3511	114	3,1%
GL85/gl85.vhd	109	91	18	16,5%	4788	4545	243	5,1%
GL85/hldlogic.vhd	21	14	7	33,3%	572	484	88	15,4%
GL85/hltlogic.vhd	23	16	7	30,4%	658	570	88	13,4%
GL85/hl_de_wz.vhd	75	53	22	29,3%	2743	2406	337	12,3%
GL85/inst_reg.vhd	58	34	24	41,4%	1885	1376	509	27,0%
GL85/interupt.vhd	76	57	19	25,0%	2782	2465	317	11,4%
GL85/intrupt1.vhd	72	59	13	18,1%	2992	2705	287	9,6%
GL85/intrupt2.vhd	67	54	13	19,4%	2879	2553	326	11,3%
GL85/intrupt3.vhd	59	42	17	28,8%	2172	1792	380	17,5%
GL85/inv8.vhd	21	15	6	28,6%	782	700	82	10,5%
GL85/m5.vhd	30	22	8	26,7%	904	820	84	9,3%
GL85/mcdecode.vhd	120	91	29	24,2%	6140	5082	1058	17,2%
GL85/mux2to1.vhd	26	15	11	42,3%	686	543	143	20,8%
GL85/mux_4bit.vhd	31	24	7	22,6%	1214	1126	88	7,2%
GL85/ocnand.vhd	26	18	8	30,8%	892	674	218	24,4%
GL85/oprlogic.vhd	69	54	15	21,7%	2618	2460	158	6,0%
GL85/parity1.vhd	22	15	7	31,8%	635	548	87	13,7%
GL85/pc_cntrl.vhd	114	92	22	19,3%	5713	4974	739	12,9%
GL85/prioenco.vhd	32	25	7	21,9%	1103	1015	88	8,0%
GL85/rdwrgen.vhd	92	76	16	17,4%	4191	3822	369	8,8%
GL85/reg8bits.vhd	43	30	13	30,2%	1601	1274	327	20,4%
GL85/regctrl0.vhd	117	73	44	37,6%	5661	3897	1764	31,2%
GL85/regctrl1.vhd	199	160	39	19,6%	10033	8970	1063	10,6%
GL85/regctrl2.vhd	64	55	9	14,1%	2623	2479	144	5,5%
GL85/regpad.vhd	89	75	14	15,7%	3702	3602	100	2,7%
GL85/regpair.vhd	28	20	8	28,6%	742	653	89	12,0%
GL85/regpairs.vhd	28	20	8	28,6%	745	655	90	12,1%
GL85/reg_8bit.vhd	35	24	11	31,4%	1043	862	181	17,4%
GL85/reg_ctrl.vhd	190	153	37	19,5%	8905	8184	721	8,1%
GL85/reg_ram.vhd	28	20	8	28,6%	742	653	89	12,0%
GL85/shflogic.vhd	58	44	14	24,1%	1954	1802	152	7,8%
GL85/sn54181.vhd	94	86	8	8,5%	4600	4511	89	1,9%
GL85/sn85150.vhd	41	31	10	24,4%	1509	1336	173	11,5%
GL85/sp_cntrl.vhd	66	56	10	15,2%	3190	2902	288	9,0%
GL85/stater.vhd	77	61	16	20,8%	3269	2961	308	9,4%

(continued on next page)

(continued from previous page)

Project/Filename	L	D	X	X%	Lb	Db	Xb	Xb%
GL85/tempctrl.vhd	47	40	7	14,9%	2291	2088	203	8,9%
GL85/vectrgen.vhd	42	26	16	38,1%	1280	922	358	28,0%
hc11core/clkgen.vhd	53	26	27	50,9%	1564	699	865	55,3%
hc11core/dev.vhd	76	52	24	31,6%	2138	1362	776	36,3%
hc11core/hc11rtl.vhd	2194	2076	118	5,4%	71602	69917	1685	2,4%
hc11core/ram.vhd	72	45	27	37,5%	2122	1246	876	41,3%
hc11core/srec_rom.vhd	175	116	59	33,7%	5822	3650	2172	37,3%
hc11core/syncore.vhd	47	30	17	36,2%	1793	1296	497	27,7%
hc11core/testcore.vhd	65	41	24	36,9%	1934	1352	582	30,1%
i80386/i80386.vhdl	1221	972	249	20,4%	59417	45318	14099	23,7%
i8051_source_2.8/i8051_all.vhd	254	200	54	21,3%	10302	8905	1397	13,6%
i8051_source_2.8/i8051_alu.vhd	429	322	107	24,9%	14228	12020	2208	15,5%
i8051_source_2.8/i8051_ctr.vhd	5206	3514	1692	32,5%	211436	181072	30364	14,4%
i8051_source_2.8/i8051_dbg.vhd	279	249	30	10,8%	10645	9983	662	6,2%
i8051_source_2.8/i8051_dec.vhd	152	125	27	17,8%	9310	8416	894	9,6%
i8051_source_2.8/i8051_lib.vhd	306	284	22	7,2%	17737	17248	489	2,8%
i8051_source_2.8/i8051_ram.vhd	272	199	73	26,8%	10196	8077	2119	20,8%
i8051_source_2.8/i8051_rom.vhd	581	569	12	2,1%	12141	8415	3726	30,7%
i8051_source_2.8/i8051_tsb.vhd	93	61	32	34,4%	3221	2540	681	21,1%
i8051_source_2.8/i8051_xrm.vhd	77	34	43	55,8%	2227	1182	1045	46,9%
Jane/Header.vhdl	339	258	81	23,9%	9885	9256	629	6,4%
Jane/Processor.vhdl	542	293	249	45,9%	12856	10152	2704	21,0%
Jane/Test.vhdl	115	65	50	43,5%	2305	2108	197	8,5%
Leon/acache.vhd	250	182	68	27,2%	7854	6400	1454	18,5%
Leon/ahbarb.vhd	277	193	84	30,3%	9507	7432	2075	21,8%
Leon/ahbstat.vhd	127	74	53	41,7%	3302	2303	999	30,3%
Leon/ahbtest.vhd	136	95	41	30,1%	4148	3201	947	22,8%
Leon/amba.vhd	291	91	200	68,7%	15191	4770	10421	68,6%
Leon/ambacomp.vhd	279	205	74	26,5%	7757	5859	1898	24,5%
Leon/apbmst.vhd	135	85	50	37,0%	4158	2951	1207	29,0%
Leon/bprom.vhd	280	258	22	7,9%	16539	16008	531	3,2%
Leon/cache.vhd	124	84	40	32,3%	3570	2463	1107	31,0%
Leon/cachemem.vhd	146	82	64	43,8%	4645	3713	932	20,1%
Leon/clkgen.vhd	71	34	37	52,1%	2516	1197	1319	52,4%
Leon/config.vhd	149	97	52	34,9%	7076	5459	1617	22,9%
Leon/dcache.vhd	507	385	122	24,1%	17861	14741	3120	17,5%
Leon/debug.vhd	787	717	70	8,9%	29340	28075	1265	4,3%
Leon/device.vhd	52	6	46	88,5%	2196	137	2059	93,8%
Leon/div.vhd	175	123	52	29,7%	5457	4101	1356	24,8%
Leon/fp.vhd	912	769	143	15,7%	33038	28420	4618	14,0%
Leon/fp1eu.vhd	688	573	115	16,7%	24458	21055	3403	13,9%
Leon/fpulib.vhd	56	27	29	51,8%	1889	905	984	52,1%
Leon/ftcell.vhd	25	8	17	68,0%	695	159	536	77,1%
Leon/ftlib.vhd	26	4	22	84,6%	898	69	829	92,3%
Leon/icache.vhd	342	238	104	30,4%	11651	9421	2230	19,1%
Leon/iface.vhd	600	456	144	24,0%	19825	15419	4406	22,2%
Leon/ioport.vhd	174	113	61	35,1%	5835	4314	1521	26,1%
Leon/irqctrl.vhd	143	89	54	37,8%	4407	3151	1256	28,5%
Leon/irqctrl2.vhd	170	118	52	30,6%	5371	4188	1183	22,0%
Leon/iu.vhd	2674	2186	488	18,2%	91790	81619	10171	11,1%
Leon/lconf.vhd	70	40	30	42,9%	2347	1469	878	37,4%
Leon/leon.vhd	155	92	63	40,6%	4849	3464	1385	28,6%
Leon/leon_pci.vhd	321	222	99	30,8%	12060	9846	2214	18,4%
Leon/macro.vhd	111	55	56	50,5%	2786	1486	1300	46,7%
Leon/mcore.vhd	326	195	131	40,2%	10050	7443	2607	25,9%
Leon/mctrl.vhd	720	566	154	21,4%	24053	21709	2344	9,7%
Leon/meiko.vhd	58	35	23	39,7%	1881	1101	780	41,5%
Leon/mul.vhd	335	272	63	18,8%	11042	9708	1334	12,1%
Leon/multlib.vhd	86183	86124	59	0,1%	5736019	5735044	975	0,0%
Leon/pci_arb.vhd	413	171	242	58,6%	18790	6407	12383	65,9%
Leon/pci_esa.vhd	64	35	29	45,3%	2452	1231	1221	49,8%

(continued on next page)

(continued from previous page)

Project/Filename	L	D	X	X%	Lb	Db	Xb	Xb%
Leon/pci_is.vhd	86	47	39	45,3%	3328	1759	1569	47,1%
Leon/proc.vhd	233	171	62	26,6%	6800	4974	1826	26,9%
Leon/rstgen.vhd	61	24	37	60,7%	1726	638	1088	63,0%
Leon/sparcv8.vhd	273	203	70	25,6%	13984	12564	1420	10,2%
Leon/target.vhd	755	522	233	30,9%	34691	25028	9663	27,9%
Leon/tech_atc25.vhd	1115	919	196	17,6%	36455	32517	3938	10,8%
Leon/tech_atc35.vhd	916	738	178	19,4%	27198	24353	2845	10,5%
Leon/tech_fs90.vhd	789	663	126	16,0%	27311	24546	2765	10,1%
Leon/tech_generic.vhd	621	462	159	25,6%	18494	15126	3368	18,2%
Leon/tech_map.vhd	827	669	158	19,1%	23806	21810	1996	8,4%
Leon/tech_umc18.vhd	1173	961	212	18,1%	34597	31013	3584	10,4%
Leon/tech_virtex.vhd	547	463	84	15,4%	17079	15655	1424	8,3%
Leon/timers.vhd	248	170	78	31,5%	8007	5922	2085	26,0%
Leon/uart.vhd	394	278	116	29,4%	11916	8652	3264	27,4%
Leon/wprot.vhd	142	99	43	30,3%	4269	3393	876	20,5%
Leon/bprom.vhd	64	37	27	42,2%	2071	1079	992	47,9%
Leon/iram.vhd	271	208	63	23,2%	8919	6973	1946	21,8%
Leon/leonlib.vhd	110	74	36	32,7%	3602	2648	954	26,5%
Leon/mspram.vhd	350	268	82	23,4%	10937	8911	2026	18,5%
Leon/tbdef.vhd	39	10	29	74,4%	1385	244	1141	82,4%
Leon/tbgen.vhd	383	267	116	30,3%	11953	9712	2241	18,7%
Leon/tbleon.vhd	312	236	76	24,4%	7684	5957	1727	22,5%
Leon/tblib.vhd	31	7	24	77,4%	1141	131	1010	88,5%
Leon/tb_msp.vhd	110	70	40	36,4%	3226	2100	1126	34,9%
Leon/testmod.vhd	152	107	45	29,6%	5379	4086	1293	24,0%
pic16C5X/bench.vhd	65	31	34	52,3%	1990	949	1041	52,3%
pic16C5X/fadr_mux.vhd	44	14	30	68,2%	1682	399	1283	76,3%
pic16C5X/pic_alu.vhd	120	81	39	32,5%	4875	3831	1044	21,4%
pic16C5X/pic_core.vhd	514	421	93	18,1%	15650	13477	2173	13,9%
pic16C5X/pic_ctrl.vhd	449	396	53	11,8%	22487	16942	5545	24,7%
pic16C5X/pic_rom.vhd	119	61	58	48,7%	4535	3078	1457	32,1%
pic16C5X/reg_8rst.vhd	53	26	27	50,9%	1733	733	1000	57,7%
pic16C5X/reg_8t.vhd	48	24	24	50,0%	1526	677	849	55,6%
pic16C5X/reg_cons.vhd	39	14	25	64,1%	1332	401	931	69,9%
pic16C5X/reg_file.vhd	123	37	86	69,9%	5841	1392	4449	76,2%
pic16C5X/reg_fsr.vhd	52	26	26	50,0%	1745	771	974	55,8%
pic16C5X/reg_inst.vhd	62	31	31	50,0%	2121	946	1175	55,4%
pic16C5X/reg_io.vhd	68	36	32	47,1%	2696	1471	1225	45,4%
pic16C5X/reg_pc.vhd	86	47	39	45,3%	2975	1516	1459	49,0%
pic16C5X/reg_s.vhd	65	40	25	38,5%	2160	1239	921	42,6%
pic16C5X/reg_w.vhd	52	27	25	48,1%	1741	843	898	51,6%
ppx16/P16C55.vhd	223	153	70	31,4%	6620	4488	2132	32,2%
ppx16/P16F84.vhd	261	190	71	27,2%	7977	5790	2187	27,4%
ppx16/PPX16.vhd	325	239	86	26,5%	9452	6766	2686	28,4%
ppx16/PPX_ALU.vhd	318	219	99	31,1%	9945	7419	2526	25,4%
ppx16/PPX_Ctrl.vhd	116	59	57	49,1%	4737	2397	2340	49,4%
ppx16/PPX_Pack.vhd	196	138	58	29,6%	5980	3810	2170	36,3%
ppx16/PPX_PCS.vhd	161	102	59	36,6%	5144	2972	2172	42,2%
ppx16/PPX_Port.vhd	111	52	59	53,2%	3839	1646	2193	57,1%
ppx16/PPX_RAM.vhd	102	42	60	58,8%	3361	1157	2204	65,6%
ppx16/PPX_TMR.vhd	157	95	62	39,5%	4493	2320	2173	48,4%
ppx16/ROM55.vhd	534	530	4	0,7%	25511	20330	5181	20,3%
ppx16/ROM84.vhd	362	358	4	1,1%	17765	14304	3461	19,5%
ppx16/TestBench55.vhd	30	21	9	30,0%	813	795	18	2,2%
ppx16/TestBench84.vhd	23	16	7	30,4%	531	517	14	2,6%
rd1007/Sd_cnfg.vhd	210	122	88	41,9%	9584	6219	3365	35,1%
rd1007/Sd_rfrsh.vhd	106	44	62	58,5%	4008	1566	2442	60,9%
rd1007/Sd_sig.vhd	311	201	110	35,4%	11181	7912	3269	29,2%
rd1007/Sd_state.vhd	128	64	64	50,0%	5041	2772	2269	45,0%
rd1007/Sd_top.vhd	217	128	89	41,0%	9832	5567	4265	43,4%
SuperscalarDlx/Dlx.vhd	3879	2388	1491	38,4%	179277	124412	54865	30,6%

(continued on next page)

(continued from previous page)

Project/Filename	L	D	X	X%	Lb	Db	Xb	Xb%
SuperscalarDlx/DlxPackage.vhd	900	743	157	17,4%	34954	32133	2821	8,1%
SuperscalarDlx/Environment.vhd	403	313	90	22,3%	12541	10421	2120	16,9%
SuperscalarDlx/TestBench.vhd	106	72	34	32,1%	4270	2496	1774	41,5%
synth_fft/and_gates.vhd	43	37	6	14,0%	1003	900	103	10,3%
synth_fft/baseindex.vhd	61	50	11	18,0%	1998	1739	259	13,0%
synth_fft/but.vhd	48	25	23	47,9%	761	693	68	8,9%
synth_fft/butter_lib.vhd	58	47	11	19,0%	3271	3125	146	4,5%
synth_fft/control2.vhd	151	113	38	25,2%	2965	2829	136	4,6%
synth_fft/controller.vhd	140	107	33	23,6%	3389	2654	735	21,7%
synth_fft/counter.vhd	29	25	4	13,8%	814	645	169	20,8%
synth_fft/cycles_but.vhd	83	48	35	42,2%	1388	1152	236	17,0%
synth_fft/dff.vhd	22	19	3	13,6%	627	548	79	12,6%
synth_fft/divide.vhd	33	27	6	18,2%	1222	878	344	28,2%
synth_fft/ioadd.vhd	34	30	4	11,8%	1012	944	68	6,7%
synth_fft/iod_staged.vhd	42	35	7	16,7%	1201	948	253	21,1%
synth_fft/lblock.vhd	22	19	3	13,6%	523	479	44	8,4%
synth_fft/mult.vhd	30	27	3	10,0%	832	738	94	11,3%
synth_fft/multiply.vhd	87	59	28	32,2%	2192	1969	223	10,2%
synth_fft/mux_add.vhd	24	21	3	12,5%	571	520	51	8,9%
synth_fft/mux_but.vhd	24	21	3	12,5%	585	531	54	9,2%
synth_fft/negate.vhd	35	31	4	11,4%	823	800	23	2,8%
synth_fft/normalize.vhd	91	74	17	18,7%	2139	2094	45	2,1%
synth_fft/out_result.vhd	34	24	10	29,4%	629	577	52	8,3%
synth_fft/print.vhd	55	34	21	38,2%	1233	840	393	31,9%
synth_fft/ram.vhd	48	39	9	18,8%	1236	1024	212	17,2%
synth_fft/ram_shift.vhd	33	20	13	39,4%	798	527	271	34,0%
synth_fft/rblock.vhd	22	19	3	13,6%	541	498	43	7,9%
synth_fft/rom.vhd	47	41	6	12,8%	1162	1112	50	4,3%
synth_fft/romadd_gen.vhd	105	72	33	31,4%	1867	1770	97	5,2%
synth_fft/shift2.vhd	61	53	8	13,1%	1397	1341	56	4,0%
synth_fft/stage.vhd	27	24	3	11,1%	663	631	32	4,8%
synth_fft/subtractor.vhd	102	85	17	16,7%	2189	2090	99	4,5%
synth_fft/summer.vhd	48	38	10	20,8%	1259	1109	150	11,9%
synth_fft/swap.vhd	66	58	8	12,1%	1874	1711	163	8,7%
synth_fft/synth_main.vhd	315	272	43	13,7%	12068	11915	153	1,3%
synth_fft/synth_test.vhd	400	328	72	18,0%	13414	12986	428	3,2%
T51/I8052.vhd	232	149	83	35,8%	6914	4302	2612	37,8%
T51/ROM52.vhd	159	155	4	2,5%	6584	5153	1431	21,7%
T51/T51.vhd	989	852	137	13,9%	39287	28316	10971	27,9%
T51/T51_ALU.vhd	634	408	226	35,6%	21031	14004	7027	33,4%
T51/T51_Pack.vhd	178	119	59	33,1%	5566	3536	2030	36,5%
T51/T51_Port.vhd	102	45	57	55,9%	3387	1383	2004	59,2%
T51/T51_RAM.vhd	125	58	67	53,6%	3974	1647	2327	58,6%
T51/TestBench52.vhd	46	32	14	30,4%	997	741	256	25,7%
T80/MonZ80.vhd	889	885	4	0,4%	39436	30705	8731	22,1%
T80/NoICE.vhd	175	154	21	12,0%	3935	3732	203	5,2%
T80/NoICE_TB.vhd	71	56	15	21,1%	1681	1635	46	2,7%
T80/ROM80.vhd	2070	2066	4	0,2%	74308	53767	20541	27,6%
T80/T80.vhd	1014	869	145	14,3%	27006	23393	3613	13,4%
T80/T80a.vhd	222	160	62	27,9%	5882	3704	2178	37,0%
T80/T80s.vhd	157	101	56	35,7%	4522	2328	2194	48,5%
T80/T80_ALU.vhd	335	268	67	20,0%	10902	8410	2492	22,9%
T80/T80_MCode.vhd	1582	1337	245	15,5%	43827	38004	5823	13,3%
T80/T80_Pack.vhd	162	108	54	33,3%	5712	3356	2356	41,2%
T80/TestBench.vhd	125	105	20	16,0%	2876	2806	70	2,4%
TE51/TE51eval.vhd	3107	1964	1143	36,8%	120406	63383	57023	47,4%
xapp146/multi_dvm.vhd	1167	594	573	49,1%	37094	22164	14930	40,2%
xapp146/multi_dvm_tb.vhd	177	89	88	49,7%	3765	2586	1179	31,3%
xapp146/shift16.vhd	76	32	44	57,9%	1758	988	770	43,8%
xapp146/shift8.vhd	75	34	41	54,7%	1854	1067	787	42,4%
xapp146/top_level.vhd	332	144	188	56,6%	8605	4775	3830	44,5%

(continued on next page)

(continued from previous page)

Project/Filename	L	D	X	X%	Lb	Db	Xb	Xb%
xapp146/top_level_tb.vhd	213	149	64	30,0%	4731	4070	661	14,0%
xapp146/upcnt5.vhd	65	31	34	52,3%	1533	986	547	35,7%
xapp328_proj_nav/mp3_-post.vhd	3680	3406	274	7,4%	209157	189423	19734	9,4%
xapp328/cnt25.vhd	108	50	58	53,7%	3300	1593	1707	51,7%
xapp328/cnt3.vhd	63	27	36	57,1%	1549	809	740	47,8%
xapp328/cnt5.vhd	78	32	46	59,0%	2113	898	1215	57,5%
xapp328/command_state_-machine.vhd	268	111	157	58,6%	8713	3199	5514	63,3%
xapp328/dnld_interface.vhd	235	99	136	57,9%	7774	3274	4500	57,9%
xapp328/flash_cntr.vhd	761	381	380	49,9%	25037	12507	12530	50,0%
xapp328/i2c_master.vhd	556	348	208	37,4%	16593	10344	6249	37,7%
xapp328/lcd_control.vhd	215	132	83	38,6%	6485	4232	2253	34,7%
xapp328/main_ctrl_state_-machine.vhd	330	166	164	49,7%	11056	4687	6369	57,6%
xapp328/mp3_cpld.vhd	703	570	133	18,9%	23518	19868	3650	15,5%
xapp328/mp3_post.vhd	3680	3406	274	7,4%	209157	189423	19734	9,4%
xapp328/mpeg_chip_ctrl.vhd	284	143	141	49,6%	8485	3991	4494	53,0%
xapp328/on_off_logic.vhd	76	31	45	59,2%	2379	798	1581	66,5%
xapp328/parallel_port.vhd	162	65	97	59,9%	5244	2104	3140	59,9%
xapp328/play_logic_state_-machine.vhd	196	84	112	57,1%	6178	2174	4004	64,8%
xapp328/play_modes.vhd	238	128	110	46,2%	6442	3414	3028	47,0%
xapp328/power_ctrl.vhd	161	64	97	60,2%	4584	1537	3047	66,5%
xapp328/shift.vhd	78	34	44	56,4%	1920	1049	871	45,4%
xapp328/sound_control.vhd	254	128	126	49,6%	7164	3491	3673	51,3%
xapp328/upcnt2.vhd	69	31	38	55,1%	1806	998	808	44,7%
xapp328/upcnt3.vhd	69	31	38	55,1%	1806	998	808	44,7%
xapp328/upcnt4.vhd	70	31	39	55,7%	1793	1003	790	44,1%
xapp328/updwncnt4.vhd	85	35	50	58,8%	2024	967	1057	52,2%
xapp328/osc_14mhz.vhd	53	25	28	52,8%	1113	482	631	56,7%
xapp328/osc_2mhz.vhd	53	25	28	52,8%	1108	480	628	56,7%
xapp328/overall.vhd	376	328	48	12,8%	12460	11963	497	4,0%
xapp328/overall_post.vhd	426	377	49	11,5%	15826	15321	505	3,2%
xapp328/pkg_convert.vhd	63	54	9	14,3%	1587	1362	225	14,2%
xapp328/pulldown.vhd	18	10	8	44,4%	257	229	28	10,9%
xapp328/pullup.vhd	16	10	6	37,5%	243	219	24	9,9%
xapp328/tst_dac3550a.vhd	469	291	178	38,0%	13367	8532	4835	36,2%
xapp328/tst_mas3507d.vhd	189	97	92	48,7%	4955	2213	2742	55,3%
xapp328/tst_pc.vhd	215	97	118	54,9%	6650	3638	3012	45,3%
xapp328/tst_strata_flash.vhd	238	137	101	42,4%	7329	4278	3051	41,6%
xapp328/tst_user_interface.vhd	272	153	119	43,8%	7130	4365	2765	38,8%
xapp333/i2c.vhd	284	168	116	40,8%	9449	4734	4715	49,9%
xapp333/i2c_control.vhd	937	617	320	34,2%	30272	17473	12799	42,3%
xapp333/i2c_timesim.vhd	13710	13691	19	0,1%	440661	439999	662	0,2%
xapp333/micro_master_tb.vhd	464	300	164	35,3%	15051	8895	6156	40,9%
xapp333/micro_slave_tb.vhd	320	203	117	36,6%	9141	5797	3344	36,6%
xapp333/micro_tb.vhd	522	360	162	31,0%	16696	10471	6225	37,3%
xapp333/micro_test.vhd	124	100	24	19,4%	3815	3251	564	14,8%
xapp333/micro_test_post.vhd	187	135	52	27,8%	6750	4664	2086	30,9%
xapp333/pullup.vhd	16	10	6	37,5%	243	219	24	9,9%
xapp333/shift.vhd	76	34	42	55,3%	1739	1033	706	40,6%
xapp333/uc_interface.vhd	463	245	218	47,1%	11987	7089	4898	40,9%
xapp333/upcnt4.vhd	66	31	35	53,0%	1465	984	481	32,8%
xapp333/upcnt4_tb.vhd	193	140	53	27,5%	5305	3493	1812	34,2%
xapp333/upcnt4_tb_post.vhd	204	142	62	30,4%	5811	3611	2200	37,9%
xapp336/decoder.vhd	169	83	86	50,9%	4051	2538	1513	37,3%
xapp336/dec_func.vhd	363	194	169	46,6%	10575	7803	2772	26,2%
xapp336/dis_gen_low.vhd	215	89	126	58,6%	6452	3167	3285	50,9%
xapp336/dis_gen_up.vhd	212	86	126	59,4%	6405	3080	3325	51,9%
xapp336/encoder.vhd	162	77	85	52,5%	4313	2470	1843	42,7%

(continued on next page)

(continued from previous page)

Project/Filename	L	D	X	X%	Lb	Db	Xb	Xb%
xapp336/enc_func.vhd	281	129	152	54,1%	7837	4616	3221	41,1%
xapp336/err_check.vhd	191	80	111	58,1%	4991	2735	2256	45,2%
xapp336/err_det.vhd	168	63	105	62,5%	4250	2052	2198	51,7%
xapp336/main_dec.vhd	338	178	160	47,3%	8691	4640	4051	46,6%
xapp336/main_enc_low.vhd	379	199	180	47,5%	10782	5379	5403	50,1%
xapp336/main_enc_up.vhd	380	200	180	47,4%	10837	5419	5418	50,0%
xapp336/main_tb.vhd	152	91	61	40,1%	4831	2975	1856	38,4%
xapp336/main_tb_post.vhd	227	166	61	26,9%	7905	6505	1400	17,7%
xapp336/pkg_convert.vhd	82	55	27	32,9%	2161	1453	708	32,8%
xapp336/pkg_spc_char.vhd	71	37	34	47,9%	2032	1547	485	23,9%
xapp336/s_gen.vhd	162	60	102	63,0%	4329	1767	2562	59,2%
xapp336/timesim.vhd	8716	8697	19	0,2%	333561	332875	686	0,2%
xapp336/tst_bench.vhd	356	176	180	50,6%	10317	5912	4405	42,7%
xapp336/tst_bench_post.vhd	356	173	183	51,4%	10329	5803	4526	43,8%
xapp336_8/dec_err_check.vhd	191	80	111	58,1%	4991	2735	2256	45,2%
xapp336_8/dec_func.vhd	363	194	169	46,6%	10575	7803	2772	26,2%
xapp336_8/dis_gen.vhd	205	82	123	60,0%	6134	2966	3168	51,6%
xapp336_8/enc_func.vhd	293	138	155	52,9%	8327	4947	3380	40,6%
xapp336_8/err_check.vhd	191	80	111	58,1%	4991	2735	2256	45,2%
xapp336_8/main_dec.vhd	333	176	157	47,1%	8478	4569	3909	46,1%
xapp336_8/main_enc.vhd	386	205	181	46,9%	11012	5552	5460	49,6%
xapp336_8/main_tb.vhd	169	95	74	43,8%	5112	2988	2124	41,5%
xapp336_8/pkg_convert.vhd	82	55	27	32,9%	2156	1450	706	32,7%
xapp336_8/pkg_spc_char.vhd	71	37	34	47,9%	1827	1342	485	26,5%
xapp336_8/s_gen.vhd	162	60	102	63,0%	4330	1767	2563	59,2%
xapp336_8/tst_bench.vhd	355	169	186	52,4%	10034	5591	4443	44,3%
xapp345/irda_uart.vhd	107	77	30	28,0%	2875	2279	596	20,7%
xapp345/irda_uart_tb.vhd	167	107	60	35,9%	4692	3330	1362	29,0%
xapp345/jk_ff.vhd	67	41	26	38,8%	1554	1080	474	30,5%
xapp345/pkg_util.vhd	95	68	27	28,4%	2086	1459	627	30,1%
xapp345/rxcver.vhd	219	131	88	40,2%	6896	3795	3101	45,0%
xapp345/sirendec.vhd	203	141	62	30,5%	4769	4104	665	13,9%
xapp345/time_sim.vhd	1721	1600	121	7,0%	96724	88903	7821	8,1%
xapp345/txmit.vhd	182	111	71	39,0%	5713	3099	2614	45,8%
xapp345/uart.vhd	103	69	34	33,0%	2717	1977	740	27,2%
xapp345/uart_tb.vhd	173	107	66	38,2%	3906	2793	1113	28,5%
xapp348/sck_logic.vhd	235	141	94	40,0%	8359	4448	3911	46,8%
xapp348/spi_control_sm.vhd	425	244	181	42,6%	17248	8351	8897	51,6%
xapp348/spi_interface_tb.vhd	298	185	113	37,9%	9730	6159	3571	36,7%
xapp348/spi_master_tb.vhd	870	540	330	37,9%	35772	20981	14791	41,3%
xapp348/spi_rcv_shift_reg.vhd	164	79	85	51,8%	5631	2767	2864	50,9%
xapp348/spi_xmit_shift_reg.vhd	103	44	59	57,3%	2912	1452	1460	50,1%
xapp348/time_sim.vhd	2352	2192	160	6,8%	132254	121344	10910	8,2%
xapp348/uc_interface.vhd	527	296	231	43,8%	19574	10844	8730	44,6%
xapp348/upcnt4.vhd	58	27	31	53,4%	1251	843	408	32,6%
xapp348/upcnt5.vhd	56	27	29	51,8%	1131	802	329	29,1%
xapp349/time_sim.vhd	5923	5904	19	0,3%	175559	174863	696	0,4%
xapp349/uc_interface.vhd	514	259	255	49,6%	20156	10271	9885	49,0%
xapp349/uc_interface_tb.vhd	807	484	323	40,0%	32861	18579	14282	43,5%
xapp354/amd_flash_denali.vhd	99	90	9	9,1%	2634	2437	197	7,5%
xapp354/amd_flash_tb.vhd	537	328	209	38,9%	14572	8649	5923	40,6%
xapp354/nand_interface.vhd	270	141	129	47,8%	7840	4697	3143	40,1%
xapp354/pkg_convert.vhd	77	54	23	29,9%	2021	1362	659	32,6%
xapp354/samsung_flash_-denali.vhd	30	24	6	20,0%	883	707	176	19,9%
xapp354/samsung_flash_tb.vhd	517	309	208	40,2%	13857	8234	5623	40,6%
xapp354/time_sim.vhd	222	184	38	17,1%	9656	7964	1692	17,5%
xapp355/adc_interface.vhd	1167	594	573	49,1%	37136	22179	14957	40,3%
xapp355/adc_interface_tb.vhd	176	89	87	49,4%	3743	2598	1145	30,6%
xapp355/shift16.vhd	76	32	44	57,9%	1758	988	770	43,8%
xapp355/shift8.vhd	75	34	41	54,7%	1852	1062	790	42,7%

(continued on next page)

(continued from previous page)

Project/Filename	L	D	X	X%	Lb	Db	Xb	Xb%
xapp355/time_sim.vhd	3603	3362	241	6,7%	207048	189903	17145	8,3%
xapp355/top_level.vhd	323	142	181	56,0%	8247	4730	3517	42,6%
xapp355/top_level_tb.vhd	211	146	65	30,8%	4772	3972	800	16,8%
xapp355/upcnt5.vhd	66	31	35	53,0%	1541	984	557	36,1%
xapp356/adc_interface.vhd	705	305	400	56,7%	21343	10054	11289	52,9%
xapp356/pkg_constants.vhd	248	75	173	69,8%	9775	4442	5333	54,6%
xapp356/shift16.vhd	75	32	43	57,3%	1742	974	768	44,1%
xapp356/shift8.vhd	74	34	40	54,1%	1836	1048	788	42,9%
xapp356/temp_interface.vhd	257	105	152	59,1%	6843	3183	3660	53,5%
xapp356/time_sim.vhd	4184	3913	271	6,5%	242616	223245	19371	8,0%
xapp356/top_level.vhd	557	269	288	51,7%	17033	8856	8177	48,0%
xapp356/top_level_tb.vhd	514	307	207	40,3%	12338	7650	4688	38,0%
xapp356/upcnt11.vhd	68	31	37	54,4%	1558	976	582	37,4%
xapp356/upcnt15.vhd	68	31	37	54,4%	1558	976	582	37,4%
xapp356/upcnt5.vhd	65	31	34	52,3%	1525	970	555	36,4%
xapp356/xpath.vhd	1124	434	690	61,4%	31478	13444	18034	57,3%
xapp357/clk_divider.vhd	52	19	33	63,5%	1139	495	644	56,5%
xapp357/LED_TEST.VHD	256	119	137	53,5%	7643	3747	3896	51,0%
xapp358/rx-RECEIVE.vhd	230	167	63	27,4%	9521	5990	3531	37,1%
xapp358/rx-timesim.vhd	18202	18183	19	0,1%	530173	529502	671	0,1%
xapp358/tb-DISPLAYCOUNT.vhd	66	39	27	40,9%	2574	1154	1420	55,2%
xapp358/tb-RECEIVE.vhd	232	167	65	28,0%	9642	5986	3656	37,9%
xapp358/tb-SHIFTENABLE.vhd	97	69	28	28,9%	4568	2252	2316	50,7%
xapp358/tb-SHIFTOUT.vhd	72	42	30	41,7%	3169	1542	1627	51,3%
xapp358/tb-TRANSMIT.vhd	212	162	50	23,6%	8436	5649	2787	33,0%
xapp358/tb-TX_RX_ENTITY.vhd	90	64	26	28,9%	3212	1783	1429	44,5%
xapp358/tb-TX_RX_-TestBench.vhd	95	55	40	42,1%	2633	1563	1070	40,6%
xapp358/tx-DISPLAYCOUNT.vhd	66	39	27	40,9%	2574	1154	1420	55,2%
xapp358/tx-SHIFTENABLE.vhd	97	69	28	28,9%	4568	2252	2316	50,7%
xapp358/tx-SHIFTOUT.vhd	72	42	30	41,7%	3169	1542	1627	51,3%
xapp358/tx-timesim.vhd	8723	8704	19	0,2%	250265	249584	681	0,3%
xapp358/tx-TRANSMIT.vhd	212	162	50	23,6%	8436	5649	2787	33,0%
xapp358/tx-TXtestbench.vhd	87	51	36	41,4%	2387	1355	1032	43,2%
xapp363/clk_gen.vhd	45	20	25	55,6%	978	462	516	52,8%
xapp363/clk_top.vhd	76	44	32	42,1%	2229	1430	799	35,8%
xapp363/gpio_top.vhd	52	25	27	51,9%	1765	960	805	45,6%
xapp363/sam_top.vhd	574	421	153	26,7%	27076	21055	6021	22,2%
xapp363/smedia_state.vhd	392	301	91	23,2%	13092	11250	1842	14,1%
xapp363/smedia_testbench.vhd	112	78	34	30,4%	4088	3534	554	13,6%
xapp363/smedia_top.vhd	167	116	51	30,5%	7236	5192	2044	28,2%
xapp363/spi.vhd	135	86	49	36,3%	5145	3761	1384	26,9%
xapp363/spi_switch.vhd	95	58	37	38,9%	3066	2112	954	31,1%
xapp363/ssp_icc.vhd	702	634	68	9,7%	36493	34664	1829	5,0%
xapp363/ssp_icc_smedia_-testbench.vhd	277	178	99	35,7%	12205	9511	2694	22,1%
xapp363/ssp_icc_switch.vhd	171	118	53	31,0%	6861	5292	1569	22,9%
xapp365/iso9141.vhd	356	216	140	39,3%	13563	8060	5503	40,6%
xapp365/iso_clk_divider.vhd	98	59	39	39,8%	3113	2073	1040	33,4%
xapp365/rcvr.vhd	147	110	37	25,2%	3546	2892	654	18,4%
xapp365/txmit.vhd	120	83	37	30,8%	2602	2057	545	20,9%
xapp367/AudioController.vhd	40	24	16	40,0%	892	665	227	25,4%
xapp367/Chatterbox.vhd	387	274	113	29,2%	9279	7803	1476	15,9%
xapp367/DTMFController.vhd	50	34	16	32,0%	1269	1052	217	17,1%
xapp367/FlipFlop.vhd	25	19	6	24,0%	458	412	46	10,0%
xapp367/FlipFlopR.vhd	27	21	6	22,2%	569	479	90	15,8%
xapp367/IrqController.vhd	117	86	31	26,5%	2493	2114	379	15,2%
xapp367/MemoryManager.vhd	225	101	124	55,1%	8636	5219	3417	39,6%
xapp367/PowerSupplyController.vhd	39	23	16	41,0%	811	583	228	28,1%
xapp367/RFtransceiverController.vhd	52	35	17	32,7%	1326	1093	233	17,6%
xapp367/time_sim.vhd	1291	1151	140	10,8%	72837	63388	9449	13,0%

(continued on next page)

(continued from previous page)

Project/Filename	L	D	X	X%	Lb	Db	Xb	Xb%
xapp369/decode_man.vhd	608	287	321	52,8%	17963	9620	8343	46,4%
xapp369/time_sim.vhd	4106	3850	256	6,2%	235676	217460	18216	7,7%
xapp369/top_level.vhd	303	143	160	52,8%	8770	4933	3837	43,8%
xapp370/clk_divider.vhd	39	22	17	43,6%	1011	549	462	45,7%
xapp370/cooltrak.vhd	350	248	102	29,1%	11695	8692	3003	25,7%
xapp370/multi_dvm.vhd	1202	610	592	49,3%	38286	22437	15849	41,4%
xapp370/shift16.vhd	77	32	45	58,4%	1788	988	800	44,7%
xapp370/shift8.vhd	65	29	36	55,4%	1748	920	828	47,4%
xapp370/speed.vhd	69	48	21	30,4%	1867	1472	395	21,2%
xapp370/temp.vhd	1169	595	574	49,1%	36822	21869	14953	40,6%
xapp370/time_sim.vhd	4492	4241	251	5,6%	256641	238727	17914	7,0%
xapp370/upcnt5.vhd	66	31	35	53,0%	1561	984	577	37,0%
total	388790	319049	69741	17,9%	16478757	14441893	2036864	12,4%

Chapter 7

Grammatical aspects

This chapter describes the full set of productions composing the formal grammar of the VHDL language, in its 1993 standard, as described in [4]. The productions here listed do not match 100% the productions I used to implement the VHDL parser, mainly because the first ones are written in EBNF (Extended Backus-Naur Form, not supported by `bison`) notation, which comprises the following regular language operators:

- optionality – a symbol or a sequence of symbols can be repeated 0 or 1 times, here represented by enclosing those symbols in square brackets;
- repeatability – a symbol or a sequence of symbols can be repeated 0 or more times, here represented by enclosing those symbols in braces;

Example:

⟨*non_terminal_symbol_name*⟩ ::= `terminal` ⟨*another_symbol_name*⟩
| {, ⟨*repeatable_symbols*⟩ } [⟨*optional_symbol*⟩]

Please note that terminal symbols are written in `monospaced` font, and this is extremely important to distinguish operators from meta-operators, since EBNF and VHDL operators overlap. For example braces, square brackets and pipes appear both as terminal symbols and as EBNF symbols.

⟨*abstract_literal*⟩ ::= ⟨*decimal_literal*⟩ | ⟨*based_literal*⟩

⟨*access_type_definition*⟩ ::= `access` ⟨*subtype_indication*⟩

⟨*actual_designator*⟩ ::= ⟨*expression*⟩ | ⟨*signal_name*⟩ | ⟨*variable_name*⟩ | ⟨*file_name*⟩ | `open`

⟨*actual_parameter_part*⟩ ::= ⟨*parameter_association_list*⟩

⟨*actual_part*⟩ ::= ⟨*actual_designator*⟩ | ⟨*function_name*⟩ `(` ⟨*actual_designator*⟩ `)` | ⟨*type_mark*⟩ `(` ⟨*actual_designator*⟩ `)`

⟨*adding_operator*⟩ ::= `+` | `-` | `&`

⟨*aggregate*⟩ ::= `(` ⟨*element_association*⟩ {, ⟨*element_association*⟩} `)`

⟨*alias_declaration*⟩ ::= `alias` ⟨*alias_designator*⟩ [: ⟨*subtype_indication*⟩] `is` ⟨*name*⟩ [⟨*signature*⟩] `;`

⟨*alias_designator*⟩ ::= ⟨*identifier*⟩ | ⟨*character_literal*⟩ | ⟨*operator_symbol*⟩

⟨*allocator*⟩ ::= `new` ⟨*subtype_indication*⟩ | `new` ⟨*qualified_expression*⟩

⟨*architecture_body*⟩ ::= `architecture` ⟨*identifier*⟩ `of` ⟨*entity_name*⟩ `is` ⟨*architecture_declarative_part*⟩ `begin` ⟨*architecture_statement_part*⟩ `end` [`architecture`] [⟨*architecture_simple_name*⟩] `;`

⟨*architecture_declarative_part*⟩ ::= {⟨*block_declarative_item*⟩}

⟨*architecture_statement_part*⟩ ::= {⟨*concurrent_statement*⟩}

⟨*array_type_definition*⟩ ::= ⟨*unconstrained_array_definition*⟩ | ⟨*constrained_array_definition*⟩

⟨*assertion*⟩ ::= `assert` ⟨*condition*⟩ [`report` ⟨*expression*⟩] [`severity` ⟨*expression*⟩]

⟨*assertion_statement*⟩ ::= [⟨*label*⟩ :] ⟨*assertion*⟩ `;`

⟨*association_element*⟩ ::= [⟨*formal_part*⟩ `=>`] ⟨*actual_part*⟩

⟨*association_list*⟩ ::= ⟨*association_element*⟩ {, ⟨*association_element*⟩}

⟨*attribute_declaration*⟩ ::= `attribute` ⟨*identifier*⟩ : ⟨*type_mark*⟩ `;`

⟨*attribute_designator*⟩ ::= ⟨*attribute_simple_name*⟩

⟨*attribute_name*⟩ ::= ⟨*prefix*⟩ [⟨*signature*⟩] `'` ⟨*attribute_designator*⟩ [`(` ⟨*expression*⟩ `)`]

⟨*attribute_specification*⟩ ::= `attribute` ⟨*attribute_designator*⟩ `of` ⟨*entity_specification*⟩ `is` ⟨*expression*⟩ `;`

⟨*base*⟩ ::= ⟨*integer*⟩

⟨*base_specifier*⟩ ::= `B` | `O` | `X`

⟨*base_unit_declaration*⟩ ::= ⟨*identifier*⟩ `;`

⟨*based_integer*⟩ ::= ⟨*extended_digit*⟩ {[⟨*underline*⟩] ⟨*extended_digit*⟩}

⟨*based_literal*⟩ ::= ⟨*base*⟩ # ⟨*based_integer*⟩ [. ⟨*based_integer*⟩] # [⟨*exponent*⟩]

⟨*basic_character*⟩ ::= ⟨*basic_graphic_character*⟩ | ⟨*format_effector*⟩

⟨*basic_graphic_character*⟩ ::= ⟨*upper_case_letter*⟩ | ⟨*digit*⟩ | ⟨*special_character*⟩ | ⟨*space_character*⟩

⟨*basic_identifier*⟩ ::= ⟨*letter*⟩ {[⟨*underline*⟩] ⟨*letter_or_digit*⟩}

⟨*binding_indication*⟩ ::= [use ⟨*entity_aspect*⟩] [⟨*generic_map_aspect*⟩] [⟨*port_map_aspect*⟩]

⟨*bit_string_literal*⟩ ::= ⟨*base_specifier*⟩ " [⟨*bit_value*⟩] "

⟨*bit_value*⟩ ::= ⟨*extended_digit*⟩ {[⟨*underline*⟩] ⟨*extended_digit*⟩}

⟨*block_configuration*⟩ ::= `for` ⟨*block_specification*⟩ {⟨*use_clause*⟩} {⟨*configuration_item*⟩} `end for ;`

⟨*block_declarative_item*⟩ ::= ⟨*subprogram_declaration*⟩ | ⟨*subprogram_body*⟩ | ⟨*type_declaration*⟩ | ⟨*subtype_declaration*⟩ | ⟨*constant_declaration*⟩ | ⟨*signal_declaration*⟩ | ⟨*shared_variable_declaration*⟩ | ⟨*file_declaration*⟩ | ⟨*alias_declaration*⟩ | ⟨*component_declaration*⟩ | ⟨*attribute_declaration*⟩ | ⟨*attribute_specification*⟩ | ⟨*configuration_specification*⟩ | ⟨*disconnection_specification*⟩ | ⟨*use_clause*⟩ | ⟨*group_template_declaration*⟩ | ⟨*group_declaration*⟩

⟨*block_declarative_part*⟩ ::= {⟨*block_declarative_item*⟩}

⟨*block_header*⟩ ::= [⟨*generic_clause*⟩ [⟨*generic_map_aspect*⟩ `;`]] [⟨*port_clause*⟩ [⟨*port_map_aspect*⟩ `;`]]

⟨*block_specification*⟩ ::= ⟨*architecture_name*⟩ | ⟨*block_statement_label*⟩ | ⟨*generate_statement_label*⟩ [`(` ⟨*index_specification*⟩ `)`]

⟨*block_statement*⟩ ::= ⟨*block_label*⟩ : block [`(` ⟨*guard_expression*⟩ `)`] [`is`] ⟨*block_header*⟩ ⟨*block_declarative_part*⟩ `begin` ⟨*block_statement_part*⟩ `end` block [⟨*block_label*⟩] `;`

⟨*block_statement_part*⟩ ::= {⟨*concurrent_statement*⟩}

⟨*case_statement*⟩ ::= [⟨*case_label*⟩ :] `case` ⟨*expression*⟩ `is` ⟨*case_statement_alternative*⟩ {⟨*case_statement_alternative*⟩}
`end case` [⟨*case_label*⟩] `;`

⟨*case_statement_alternative*⟩ ::= `when` ⟨*choices*⟩ `=>` ⟨*sequence_of_statements*⟩

⟨*character_literal*⟩ ::= `'`⟨*graphic_character*⟩`'`

⟨*choice*⟩ ::= ⟨*simple_expression*⟩ | ⟨*discrete_range*⟩ | ⟨*element_simple_name*⟩ | `others`

⟨*choices*⟩ ::= ⟨*choice*⟩ {| ⟨*choice*⟩}

⟨*component_configuration*⟩ ::= for ⟨*component_specification*⟩ [⟨*binding_indication*⟩ `;`] [⟨*block_configuration*⟩]
`end for ;`

⟨*component_declaration*⟩ ::= component ⟨*identifier*⟩ [`is`] [⟨*local_generic_clause*⟩] [⟨*local_port_clause*⟩] `end` component [⟨*component_simple_name*⟩] `;`

⟨*component_instantiation_statement*⟩ ::= ⟨*instantiation_label*⟩ : ⟨*instantiated_unit*⟩ [⟨*generic_map_aspect*⟩]
[⟨*port_map_aspect*⟩] `;`

⟨*component_specification*⟩ ::= ⟨*instantiation_list*⟩ : ⟨*component_name*⟩

⟨*composite_type_definition*⟩ ::= ⟨*array_type_definition*⟩ | ⟨*record_type_definition*⟩

⟨*concurrent_assertion_statement*⟩ ::= [⟨*label*⟩ :] [`postponed`] ⟨*assertion*⟩ `;`

⟨*concurrent_procedure_call_statement*⟩ ::= [⟨*label*⟩ :] [`postponed`] ⟨*procedure_call*⟩ `;`

⟨*concurrent_signal_assignment_statement*⟩ ::= [⟨*label*⟩ :] [`postponed`] ⟨*conditional_signal_assignment*⟩
| [⟨*label*⟩ :] [`postponed`] ⟨*selected_signal_assignment*⟩

⟨*concurrent_statement*⟩ ::= ⟨*block_statement*⟩ | ⟨*process_statement*⟩ | ⟨*concurrent_procedure_call_statement*⟩
| ⟨*concurrent_assertion_statement*⟩ | ⟨*concurrent_signal_assignment_statement*⟩ |
⟨*component_instantiation_statement*⟩ | ⟨*generate_statement*⟩

⟨*condition*⟩ ::= ⟨*boolean_expression*⟩

⟨*condition_clause*⟩ ::= `until` ⟨*condition*⟩

⟨*conditional_signal_assignment*⟩ ::= ⟨*target*⟩ `<=` ⟨*options*⟩ ⟨*conditional_waveforms*⟩ `;`

⟨*conditional_waveforms*⟩ ::= {⟨*waveform*⟩ `when` ⟨*condition*⟩ `else`} ⟨*waveform*⟩ [`when` ⟨*condition*⟩]

⟨*configuration_declaration*⟩ ::= `configuration` ⟨*identifier*⟩ `of` ⟨*entity_name*⟩ `is` ⟨*configuration_declarative_part*⟩
⟨*block_configuration*⟩ `end` [`configuration`] [⟨*configuration_simple_name*⟩] `;`

⟨*configuration_declarative_item*⟩ ::= ⟨*use_clause*⟩ | ⟨*attribute_specification*⟩ | ⟨*group_declaration*⟩

⟨*configuration_declarative_part*⟩ ::= {⟨*configuration_declarative_item*⟩}

⟨*configuration_item*⟩ ::= ⟨*block_configuration*⟩ | ⟨*component_configuration*⟩

⟨*configuration_specification*⟩ ::= `for` ⟨*component_specification*⟩ ⟨*binding_indication*⟩ `;`

⟨*constant_declaration*⟩ ::= `constant` ⟨*identifier_list*⟩ : ⟨*subtype_indication*⟩ [:= ⟨*expression*⟩] `;`

⟨*constrained_array_definition*⟩ ::= `array` ⟨*index_constraint*⟩ `of` ⟨*element_subtype_indication*⟩

⟨*constraint*⟩ ::= ⟨*range_constraint*⟩ | ⟨*index_constraint*⟩

⟨*context_clause*⟩ ::= {⟨*context_item*⟩}

⟨*context_item*⟩ ::= ⟨*library_clause*⟩ | ⟨*use_clause*⟩

⟨*decimal_literal*⟩ ::= ⟨*integer*⟩ [. ⟨*integer*⟩] [⟨*exponent*⟩]

⟨*declaration*⟩ ::= ⟨*type_declaration*⟩ | ⟨*subtype_declaration*⟩ | ⟨*object_declaration*⟩ | ⟨*interface_declaration*⟩ | ⟨*alias_declaration*⟩ | ⟨*attribute_declaration*⟩ | ⟨*component_declaration*⟩ | ⟨*group_template_declaration*⟩ | ⟨*group_declaration*⟩ | ⟨*entity_declaration*⟩ | ⟨*configuration_declaration*⟩ | ⟨*subprogram_declaration*⟩ | ⟨*package_declaration*⟩

⟨*delay_mechanism*⟩ ::= `transport` | [`reject` ⟨*time_expression*⟩] `inertial`

⟨*design_file*⟩ ::= ⟨*design_unit*⟩ {⟨*design_unit*⟩}

⟨*design_unit*⟩ ::= ⟨*context_clause*⟩ ⟨*library_unit*⟩

⟨*designator*⟩ ::= ⟨*identifier*⟩ | ⟨*operator_symbol*⟩

⟨*direction*⟩ ::= `to` | `downto`

⟨*disconnection_specification*⟩ ::= `disconnect` ⟨*guarded_signal_specification*⟩ `after` ⟨*time_expression*⟩ `;`

⟨*discrete_range*⟩ ::= ⟨*discrete_subtype_indication*⟩ | ⟨*range*⟩

⟨*element_association*⟩ ::= [⟨*choices*⟩ `=>`] ⟨*expression*⟩

⟨*element_declaration*⟩ ::= ⟨*identifier_list*⟩ : ⟨*element_subtype_definition*⟩ `;`

⟨*element_subtype_definition*⟩ ::= ⟨*subtype_indication*⟩

⟨*entity_aspect*⟩ ::= `entity` ⟨*entity_name*⟩ [`(` ⟨*architecture_identifier*⟩ `)`] | `configuration` ⟨*configuration_name*⟩ | `open`

⟨*entity_class*⟩ ::= `entity` | `architecture` | `configuration` | `procedure` | `function` | `package` | `type` | `subtype` | `constant` | `signal` | `variable` | `component` | `label` | `literal` | `units` | `group` | `file`

⟨*entity_class_entry*⟩ ::= ⟨*entity_class*⟩ [`<>`]

⟨*entity_class_entry_list*⟩ ::= ⟨*entity_class_entry*⟩ {, ⟨*entity_class_entry*⟩}

⟨*entity_declaration*⟩ ::= `entity` ⟨*identifier*⟩ `is` ⟨*entity_header*⟩ ⟨*entity_declarative_part*⟩ [`begin` ⟨*entity_statement_part*⟩] `end` [`entity`] [⟨*entity_simple_name*⟩] `;`

⟨*entity_declarative_item*⟩ ::= ⟨*subprogram_declaration*⟩ | ⟨*subprogram_body*⟩ | ⟨*type_declaration*⟩ | ⟨*subtype_declaration*⟩ | ⟨*constant_declaration*⟩ | ⟨*signal_declaration*⟩ | ⟨*shared_variable_declaration*⟩ | ⟨*file_declaration*⟩ | ⟨*alias_declaration*⟩ | ⟨*attribute_declaration*⟩ | ⟨*attribute_specification*⟩ | ⟨*disconnection_specification*⟩ | ⟨*use_clause*⟩ | ⟨*group_template_declaration*⟩ | ⟨*group_declaration*⟩

⟨*entity_declarative_part*⟩ ::= {⟨*entity_declarative_item*⟩}

⟨*entity_designator*⟩ ::= ⟨*entity_tag*⟩ [⟨*signature*⟩]

⟨*entity_header*⟩ ::= [⟨*formal_generic_clause*⟩] [⟨*formal_port_clause*⟩]

⟨*entity_name_list*⟩ ::= ⟨*entity_designator*⟩ {, ⟨*entity_designator*⟩} | `others` | `all`

⟨*entity_specification*⟩ ::= ⟨*entity_name_list*⟩ : ⟨*entity_class*⟩

⟨*entity_statement*⟩ ::= ⟨*concurrent_assertion_statement*⟩ | ⟨*passive_concurrent_procedure_call_statement*⟩ | ⟨*passive_process_statement*⟩

⟨*entity_statement_part*⟩ ::= {⟨*entity_statement*⟩}

⟨*entity_tag*⟩ ::= ⟨*simple_name*⟩ | ⟨*character_literal*⟩ | ⟨*operator_symbol*⟩

⟨*enumeration_literal*⟩ ::= ⟨*identifier*⟩ | ⟨*character_literal*⟩

⟨*enumeration_type_definition*⟩ ::= `(` ⟨*enumeration_literal*⟩ {, ⟨*enumeration_literal*⟩} `)`

⟨*exit_statement*⟩ ::= [⟨*label*⟩ :] `exit` [⟨*loop_label*⟩] [`when` ⟨*condition*⟩] `;`

⟨*exponent*⟩ ::= `E` [`+`] ⟨*integer*⟩ | `E -` ⟨*integer*⟩

⟨*expression*⟩ ::= ⟨*relation*⟩ {`and` ⟨*relation*⟩} | ⟨*relation*⟩ {`or` ⟨*relation*⟩} | ⟨*relation*⟩ {`xor` ⟨*relation*⟩} | ⟨*relation*⟩ [`nand` ⟨*relation*⟩] | ⟨*relation*⟩ [`nor` ⟨*relation*⟩] | ⟨*relation*⟩ {`xnor` ⟨*relation*⟩}

⟨*extended_digit*⟩ ::= ⟨*digit*⟩ | ⟨*letter*⟩

⟨*extended_identifier*⟩ ::= ⟨*graphic_character*⟩ {⟨*graphic_character*⟩}

⟨*factor*⟩ ::= ⟨*primary*⟩ [`**` ⟨*primary*⟩] | `abs` ⟨*primary*⟩ | `not` ⟨*primary*⟩

⟨*file_declaration*⟩ ::= `file` ⟨*identifier_list*⟩ : ⟨*subtype_indication*⟩ [⟨*file_open_information*⟩] `;`

⟨*file_logical_name*⟩ ::= ⟨*string_expression*⟩

⟨*file_open_information*⟩ ::= [`open` ⟨*file_open_kind_expression*⟩] `is` ⟨*file_logical_name*⟩

⟨*file_type_definition*⟩ ::= `file of` ⟨*type_mark*⟩

⟨*floating_type_definition*⟩ ::= ⟨*range_constraint*⟩

⟨*formal_designator*⟩ ::= ⟨*generic_name*⟩ | ⟨*port_name*⟩ | ⟨*parameter_name*⟩

⟨*formal_parameter_list*⟩ ::= ⟨*parameter_interface_list*⟩

⟨*formal_part*⟩ ::= ⟨*formal_designator*⟩ | ⟨*function_name*⟩ `(` ⟨*formal_designator*⟩ `)` | ⟨*type_mark*⟩ `(` ⟨*formal_designator*⟩ `)`

⟨*full_type_declaration*⟩ ::= `type` ⟨*identifier*⟩ `is` ⟨*type_definition*⟩ `;`

⟨*function_call*⟩ ::= ⟨*function_name*⟩ [`(` ⟨*actual_parameter_part*⟩ `)`]

⟨*generate_statement*⟩ ::= ⟨*generate_label*⟩ : ⟨*generation_scheme*⟩ `generate` [{⟨*block_declarative_item*⟩} `begin`] {⟨*concurrent_statement*⟩ } `end generate` [⟨*generate_label*⟩] `;`

⟨*generation_scheme*⟩ ::= `for` ⟨*generate_parameter_specification*⟩ | `if` ⟨*condition*⟩

⟨*generic_clause*⟩ ::= `generic (` ⟨*generic_list*⟩ `) ;`

⟨*generic_list*⟩ ::= ⟨*generic_interface_list*⟩

⟨*generic_map_aspect*⟩ ::= `generic map (` ⟨*generic_association_list*⟩ `)`

⟨*graphic_character*⟩ ::= ⟨*basic_graphic_character*⟩ | ⟨*lower_case_letter*⟩ | ⟨*other_special_character*⟩

⟨*group_constituent*⟩ ::= ⟨*name*⟩ | ⟨*character_literal*⟩

⟨*group_constituent_list*⟩ ::= ⟨*group_constituent*⟩ {, ⟨*group_constituent*⟩}

⟨*group_declaration*⟩ ::= `group` ⟨*identifier*⟩ : ⟨*group_template_name*⟩ `(` ⟨*group_constituent_list*⟩ `) ;`

⟨*group_template_declaration*⟩ ::= `group` ⟨*identifier*⟩ `is (` ⟨*entity_class_entry_list*⟩ `) ;`

⟨*guarded_signal_specification*⟩ ::= ⟨*guarded_signal_list*⟩ : ⟨*type_mark*⟩

⟨*identifier*⟩ ::= ⟨*basic_identifier*⟩ | ⟨*extended_identifier*⟩

⟨*identifier_list*⟩ ::= ⟨*identifier*⟩ {, ⟨*identifier*⟩}

⟨*if_statement*⟩ ::= [⟨*if_label*⟩ :] `if` ⟨*condition*⟩ `then` ⟨*sequence_of_statements*⟩ {`elsif` ⟨*condition*⟩ `then` ⟨*sequence_of_statements*⟩} [`else` ⟨*sequence_of_statements*⟩] `end if` [⟨*if_label*⟩] `;`

⟨*incomplete_type_declaration*⟩ ::= `type` ⟨*identifier*⟩ `;`

⟨*index_constraint*⟩ ::= `(`⟨*discrete_range*⟩ {, ⟨*discrete_range*⟩}`)`

⟨*index_specification*⟩ ::= ⟨*discrete_range*⟩ | ⟨*static_expression*⟩

⟨*index_subtype_definition*⟩ ::= ⟨*type_mark*⟩ | range ⟨⟩

⟨*indexed_name*⟩ ::= ⟨*prefix*⟩ `(` ⟨*expression*⟩ {, ⟨*expression*⟩} `)`

⟨*instantiated_unit*⟩ ::= [`component`] ⟨*component_name*⟩
| entity ⟨*entity_name*⟩ [`(` ⟨*architecture_identifier*⟩ `)`]
| configuration ⟨*configuration_name*⟩

⟨*instantiation_list*⟩ ::= ⟨*instantiation_label*⟩ {, ⟨*instantiation_label*⟩} | others | `all`

⟨*integer*⟩ ::= ⟨*digit*⟩ {[⟨*underline*⟩] ⟨*digit*⟩}

⟨*integer_type_definition*⟩ ::= ⟨*range_constraint*⟩

⟨*interface_constant_declaration*⟩ ::= [`constant`] ⟨*identifier_list*⟩ : [in] ⟨*subtype_indication*⟩ [:= ⟨*static_expression*⟩]

⟨*interface_declaration*⟩ ::= ⟨*interface_constant_declaration*⟩ | ⟨*interface_signal_declaration*⟩ | ⟨*interface_variable_declaration*⟩ | ⟨*interface_file_declaration*⟩

⟨*interface_element*⟩ ::= ⟨*interface_declaration*⟩

⟨*interface_file_declaration*⟩ ::= `file` ⟨*identifier_list*⟩ : ⟨*subtype_indication*⟩

⟨*interface_list*⟩ ::= ⟨*interface_element*⟩ {`;` ⟨*interface_element*⟩}

⟨*interface_signal_declaration*⟩ ::= [`signal`] ⟨*identifier_list*⟩ : [⟨*mode*⟩] ⟨*subtype_indication*⟩ [`bus`] [:= ⟨*static_expression*⟩]

⟨*interface_variable_declaration*⟩ ::= [`variable`] ⟨*identifier_list*⟩ : [⟨*mode*⟩] ⟨*subtype_indication*⟩ [:= ⟨*static_expression*⟩]

⟨*iteration_scheme*⟩ ::= `while` ⟨*condition*⟩ | `for` ⟨*loop_parameter_specification*⟩

⟨*label*⟩ ::= ⟨*identifier*⟩

⟨*letter*⟩ ::= ⟨*upper_case_letter*⟩ | ⟨*lower_case_letter*⟩

⟨*letter_or_digit*⟩ ::= ⟨*letter*⟩ | ⟨*digit*⟩

⟨*library_clause*⟩ ::= `library` ⟨*logical_name_list*⟩ `;`

⟨*library_unit*⟩ ::= ⟨*primary_unit*⟩ | ⟨*secondary_unit*⟩

⟨*literal*⟩ ::= ⟨*numeric_literal*⟩ | ⟨*enumeration_literal*⟩ | ⟨*string_literal*⟩ | ⟨*bit_string_literal*⟩ | `null`

⟨*logical_name*⟩ ::= ⟨*identifier*⟩

⟨*logical_name_list*⟩ ::= ⟨*logical_name*⟩ {, ⟨*logical_name*⟩}

⟨*logical_operator*⟩ ::= `and` | `or` | `nand` | `nor` | `xor` | `xnor`

⟨*loop_statement*⟩ ::= [⟨*loop_label*⟩ :] [⟨*iteration_scheme*⟩] `loop` ⟨*sequence_of_statements*⟩ `end loop` [⟨*loop_label*⟩] `;`

⟨*miscellaneous_operator*⟩ ::= `**` | `abs` | `not`

⟨*mode*⟩ ::= `in` | `out` | `inout` | `buffer` | `linkage`

⟨*multiplying_operator*⟩ ::= `*` | `/` | `mod` | `rem`

⟨*name*⟩ ::= ⟨*simple_name*⟩ | ⟨*operator_symbol*⟩ | ⟨*selected_name*⟩ | ⟨*indexed_name*⟩ | ⟨*slice_name*⟩ | ⟨*attribute_name*⟩

⟨*next_statement*⟩ ::= [⟨*label*⟩ :] `next` [⟨*loop_label*⟩] [`when` ⟨*condition*⟩] `;`

⟨*null_statement*⟩ ::= [⟨*label*⟩ :] `null ;`

⟨*numeric_literal*⟩ ::= ⟨*abstract_literal*⟩ | ⟨*physical_literal*⟩

⟨*object_declaration*⟩ ::= ⟨*constant_declaration*⟩ | ⟨*signal_declaration*⟩ | ⟨*variable_declaration*⟩ | ⟨*file_declaration*⟩

⟨*operator_symbol*⟩ ::= ⟨*string_literal*⟩

⟨*options*⟩ ::= [`guarded`] [⟨*delay_mechanism*⟩]

⟨*package_body*⟩ ::= `package body` ⟨*package_simple_name*⟩ `is` ⟨*package_body_declarative_part*⟩ `end` [`package body`] [⟨*package_simple_name*⟩] `;`

⟨*package_body_declarative_item*⟩ ::= ⟨*subprogram_declaration*⟩ | ⟨*subprogram_body*⟩ | ⟨*type_declaration*⟩ | ⟨*subtype_declaration*⟩ | ⟨*constant_declaration*⟩ | ⟨*shared_variable_declaration*⟩ | ⟨*file_declaration*⟩ | ⟨*alias_declaration*⟩ | ⟨*use_clause*⟩ | ⟨*group_template_declaration*⟩ | ⟨*group_declaration*⟩

⟨*package_body_declarative_part*⟩ ::= {⟨*package_body_declarative_item*⟩}

⟨*package_declaration*⟩ ::= `package` ⟨*identifier*⟩ `is` ⟨*package_declarative_part*⟩ `end` [`package`] [⟨*package_simple_name*⟩] `;`

⟨*package_declarative_item*⟩ ::= ⟨*subprogram_declaration*⟩ | ⟨*type_declaration*⟩ | ⟨*subtype_declaration*⟩ | ⟨*constant_declaration*⟩ | ⟨*signal_declaration*⟩ | ⟨*shared_variable_declaration*⟩ | ⟨*file_declaration*⟩ | ⟨*alias_declaration*⟩ | ⟨*component_declaration*⟩ | ⟨*attribute_declaration*⟩ | ⟨*attribute_specification*⟩ | ⟨*disconnection_specification*⟩ | ⟨*use_clause*⟩ | ⟨*group_template_declaration*⟩ | ⟨*group_declaration*⟩

⟨*package_declarative_part*⟩ ::= {⟨*package_declarative_item*⟩}

⟨*parameter_specification*⟩ ::= ⟨*identifier*⟩ `in` ⟨*discrete_range*⟩

⟨*physical_literal*⟩ ::= [⟨*abstract_literal*⟩] ⟨*unit_name*⟩

⟨*physical_type_definition*⟩ ::= ⟨*range_constraint*⟩ `units` ⟨*primary_unit_declaration*⟩ {⟨*secondary_unit_declaration*⟩} `end units` [⟨*physical_type_simple_name*⟩]

⟨*port_clause*⟩ ::= `port (` ⟨*port_list*⟩ `) ;`

⟨*port_list*⟩ ::= ⟨*port_interface_list*⟩

⟨*port_map_aspect*⟩ ::= `port map (` ⟨*port_association_list*⟩ `)`

⟨*prefix*⟩ ::= ⟨*name*⟩ | ⟨*function_call*⟩

⟨*primary*⟩ ::= ⟨*name*⟩ | ⟨*literal*⟩ | ⟨*aggregate*⟩ | ⟨*function_call*⟩ | ⟨*qualified_expression*⟩ | ⟨*type_conversion*⟩ | ⟨*allocator*⟩ | `(` ⟨*expression*⟩ `)`

⟨*primary_unit*⟩ ::= ⟨*entity_declaration*⟩ | ⟨*configuration_declaration*⟩ | ⟨*package_declaration*⟩

⟨*procedure_call*⟩ ::= ⟨*procedure_name*⟩ [`(` ⟨*actual_parameter_part*⟩ `)`]

⟨*procedure_call_statement*⟩ ::= [⟨*label*⟩ :] ⟨*procedure_call*⟩ `;`

⟨*process_declarative_item*⟩ ::= ⟨*subprogram_declaration*⟩ | ⟨*subprogram_body*⟩ | ⟨*type_declaration*⟩ | ⟨*subtype_declaration*⟩ | ⟨*constant_declaration*⟩ | ⟨*variable_declaration*⟩ | ⟨*file_declaration*⟩ | ⟨*alias_declaration*⟩ | ⟨*attribute_declaration*⟩ | ⟨*attribute_specification*⟩ | ⟨*use_clause*⟩ | ⟨*group_template_declaration*⟩ | ⟨*group_declaration*⟩

⟨*process_declarative_part*⟩ ::= {⟨*process_declarative_item*⟩}

⟨*process_statement*⟩ ::= [⟨*process_label*⟩ :] [postponed] process [(⟨*sensitivity_list*⟩)] [is] ⟨*process_declarative_part*⟩ begin ⟨*process_statement_part*⟩ end [postponed] process [⟨*process_label*⟩] ;

⟨*process_statement_part*⟩ ::= {⟨*sequential_statement*⟩}

⟨*qualified_expression*⟩ ::= ⟨*type_mark*⟩ ' (⟨*expression*⟩) | ⟨*type_mark*⟩ ' ⟨*aggregate*⟩

⟨*range*⟩ ::= ⟨*range_attribute_name*⟩
| ⟨*simple_expression*⟩ ⟨*direction*⟩ ⟨*simple_expression*⟩

⟨*range_constraint*⟩ ::= range ⟨*range*⟩

⟨*record_type_definition*⟩ ::= record ⟨*element_declaration*⟩ {⟨*element_declaration*⟩} end record [⟨*record_type_simple_name*⟩]

⟨*relation*⟩ ::= ⟨*shift_expression*⟩ [⟨*relational_operator*⟩ ⟨*shift_expression*⟩]

⟨*relational_operator*⟩ ::= = | /= | < | <= | > | >=

⟨*report_statement*⟩ ::= [⟨*label*⟩ :] report ⟨*expression*⟩ [severity ⟨*expression*⟩] ;

⟨*return_statement*⟩ ::= [⟨*label*⟩ :] return [⟨*expression*⟩] ;

⟨*scalar_type_definition*⟩ ::= ⟨*enumeration_type_definition*⟩ | ⟨*integer_type_definition*⟩ | ⟨*floating_type_definition*⟩ | ⟨*physical_type_definition*⟩

⟨*secondary_unit*⟩ ::= ⟨*architecture_body*⟩ | ⟨*package_body*⟩

⟨*secondary_unit_declaration*⟩ ::= ⟨*identifier*⟩ = ⟨*physical_literal*⟩ ;

⟨*selected_name*⟩ ::= ⟨*prefix*⟩ . ⟨*suffix*⟩

⟨*selected_signal_assignment*⟩ ::= with ⟨*expression*⟩ select ⟨*target*⟩ ⟨= ¡options⟩ ⟨*selected_waveforms*⟩ ;

⟨*selected_waveforms*⟩ ::= {⟨*waveform*⟩ when ⟨*choices*⟩ ,} ⟨*waveform*⟩ when ⟨*choices*⟩

⟨*sensitivity_clause*⟩ ::= on ⟨*sensitivity_list*⟩

⟨*sensitivity_list*⟩ ::= ⟨*signal_name*⟩ {, ⟨*signal_name*⟩}

⟨*sequence_of_statements*⟩ ::= {⟨*sequential_statement*⟩}

⟨*sequential_statement*⟩ ::= ⟨*wait_statement*⟩ | ⟨*assertion_statement*⟩ | ⟨*report_statement*⟩ | ⟨*signal_assignment_statement*⟩ | ⟨*variable_assignment_statement*⟩ | ⟨*procedure_call_statement*⟩ | ⟨*if_statement*⟩ | ⟨*case_statement*⟩ | ⟨*loop_statement*⟩ | ⟨*next_statement*⟩ | ⟨*exit_statement*⟩ | ⟨*return_statement*⟩ | ⟨*null_statement*⟩

⟨*shift_expression*⟩ ::= ⟨*simple_expression*⟩ [⟨*shift_operator*⟩ ⟨*simple_expression*⟩]

⟨*shift_operator*⟩ ::= sll | srl | sla | sra | rol | ror

⟨*sign*⟩ ::= + | -

⟨*signal_assignment_statement*⟩ ::= [⟨*label*⟩ :] ⟨*target*⟩ <= [⟨*delay_mechanism*⟩] ⟨*waveform*⟩ ;

⟨*signal_declaration*⟩ ::= signal ⟨*identifier_list*⟩ : ⟨*subtype_indication*⟩ [⟨*signal_kind*⟩] [:= ⟨*expression*⟩] ;

⟨*signal_kind*⟩ ::= register | bus

⟨*signal_list*⟩ ::= ⟨*signal_name*⟩ {, ⟨*signal_name*⟩} | others | all

⟨*signature*⟩ ::= [[⟨*type_mark*⟩ {, ⟨*type_mark*⟩}] [return ⟨*type_mark*⟩]]

⟨*simple_expression*⟩ ::= [⟨*sign*⟩] ⟨*term*⟩ {⟨*adding_operator*⟩ ⟨*term*⟩}

⟨*simple_name*⟩ ::= ⟨*identifier*⟩

⟨*slice_name*⟩ ::= ⟨*prefix*⟩ `(` ⟨*discrete_range*⟩ `)`

⟨*string_literal*⟩ ::= `"` {⟨*graphic_character*⟩} `"`

⟨*subprogram_body*⟩ ::= ⟨*subprogram_specification*⟩ `is` ⟨*subprogram_declarative_part*⟩ `begin` ⟨*subprogram_statement_part*⟩ `end` [⟨*subprogram_kind*⟩] [⟨*designator*⟩] `;`

⟨*subprogram_declaration*⟩ ::= ⟨*subprogram_specification*⟩ `;`

⟨*subprogram_declarative_item*⟩ ::= ⟨*subprogram_declaration*⟩ | ⟨*subprogram_body*⟩ | ⟨*type_declaration*⟩ | ⟨*subtype_declaration*⟩ | ⟨*constant_declaration*⟩ | ⟨*variable_declaration*⟩ | ⟨*file_declaration*⟩ | ⟨*alias_declaration*⟩ | ⟨*attribute_declaration*⟩ | ⟨*attribute_specification*⟩ | ⟨*use_clause*⟩ | ⟨*group_template_declaration*⟩ | ⟨*group_declaration*⟩

⟨*subprogram_declarative_part*⟩ ::= {⟨*subprogram_declarative_item*⟩}

⟨*subprogram_kind*⟩ ::= `procedure` | `function`

⟨*subprogram_specification*⟩ ::= `procedure` ⟨*designator*⟩ [`(` ⟨*formal_parameter_list*⟩ `)`]
| [`pure` | `impure`] `function` ⟨*designator*⟩ [`(` ⟨*formal_parameter_list*⟩ `)`] `return` ⟨*type_mark*⟩

⟨*subprogram_statement_part*⟩ ::= {⟨*sequential_statement*⟩}

⟨*subtype_declaration*⟩ ::= `subtype` ⟨*identifier*⟩ `is` ⟨*subtype_indication*⟩ `;`

⟨*subtype_indication*⟩ ::= [⟨*resolution_function_name*⟩] ⟨*type_mark*⟩ [⟨*constraint*⟩]

⟨*suffix*⟩ ::= ⟨*simple_name*⟩ | ⟨*character_literal*⟩ | ⟨*operator_symbol*⟩ | `all`

⟨*target*⟩ ::= ⟨*name*⟩ | ⟨*aggregate*⟩

⟨*term*⟩ ::= ⟨*factor*⟩ {⟨*multiplying_operator*⟩ ⟨*factor*⟩}

⟨*timeout_clause*⟩ ::= `for` ⟨*time_expression*⟩

⟨*type_conversion*⟩ ::= ⟨*type_mark*⟩ `(` ⟨*expression*⟩ `)`

⟨*type_declaration*⟩ ::= ⟨*full_type_declaration*⟩ | ⟨*incomplete_type_declaration*⟩

⟨*type_definition*⟩ ::= ⟨*scalar_type_definition*⟩ | ⟨*composite_type_definition*⟩ | ⟨*access_type_definition*⟩ | ⟨*file_type_definition*⟩

⟨*type_mark*⟩ ::= ⟨*type_name*⟩ | ⟨*subtype_name*⟩

⟨*unconstrained_array_definition*⟩ ::= `array` `(` ⟨*index_subtype_definition*⟩ {`,` ⟨*index_subtype_definition*⟩} `)` `of` ⟨*element_subtype_indication*⟩

⟨*use_clause*⟩ ::= `use` ⟨*selected_name*⟩ {`,` ⟨*selected_name*⟩} `;`

⟨*variable_assignment_statement*⟩ ::= [⟨*label*⟩ `:`] ⟨*target*⟩ `:=` ⟨*expression*⟩ `;`

⟨*variable_declaration*⟩ ::= [`shared`] `variable` ⟨*identifier_list*⟩ `:` ⟨*subtype_indication*⟩ [`:=` ⟨*expression*⟩] `;`

⟨*wait_statement*⟩ ::= [⟨*label*⟩ `:`] `wait` [⟨*sensitivity_clause*⟩] [⟨*condition_clause*⟩] [⟨*timeout_clause*⟩] `;`

⟨*waveform*⟩ ::= ⟨*waveform_element*⟩ {`,` ⟨*waveform_element*⟩} `unaffected`

⟨*waveform_element*⟩ ::= ⟨*value_expression*⟩ [`after` ⟨*time_expression*⟩] | `null` [`after` ⟨*time_expression*⟩]

Chapter 8

Implemented data base

8.1 Motivation

The actual implementation of model tuning, validation and use, plus practically any other implementative aspect of the methodology we propose in this thesis, access the syntactic information of the VHDL projects not directly (that is, opening VHDL source files and reading their contents as required), but indirectly, by accessing a SQL database where all syntactic information of all projects were stored by the VHDL parser.

In other words, every time a VHDL source file is analyzed by the parser, a number of records are annotated in a database, containing information about the objects encountered in the parsing and their properties. Every time a model is to be tuned, validated or used, those records are accessed in order to extract –by means of appropriate queries– the input data which the models need.

In this chapter we will briefly describe the details of this database: how it is structured and which relationships exist between the parsed objects and the records written into the database.

The reason why we preferred to rely upon a SQL database with respect to other (maybe simpler) solutions are the following:

- practically any category of syntax objects (e.g. architectures, entities, processes, ...) is populated by thousands of elements; executing a query involving two categories (a *join*, in SQL jargon) without SQL and without an optimized database engine, would require performing a cartesian product possibly containing millions of items. A *naive* linear search in that cartesian space would perform very badly. To improve performance we should write better, optimized code, specifically for that task. Doing it would practically mean writing a database engine. But we do not want to loose time to reinvent the wheel; why not simply using a ready-made, mature, reliable, easy-to-use database engine?

- most models require as input data resulting from rather complex queries; implementing those queries in SQL is very simple. Instead, implementing all the queries used or experimented during the elaboration of this thesis with specialized source code, for example working on bread-and-butter text files, would have required such a great amount of work that would have taken several years to complete.

A little disclaimer: by browsing both the Tcl/Tk and the C/C++ portion of the tools we developed, it is not uncommon to encounter SQL queries that seem to be written in

an inefficient way; you may get the feeling that we have not fully exploited the power of the SQL language. That is perfectly true, since the particular database server implementation we chose, mySql, supports a limited subset of all the SQL constructs. In fact, nested queries (`SELECT ... WHERE ... IN (SELECT ...);`) are not supported, neither are `INTERSECT`, `UNION` and `EXCEPT` operators.

8.2 Requirements

This section is an attempt to transpose the VHDL language constraints into something that approximates as close as possible the language of database management systems specifications.

Note: *entity* is an *overloaded* term, in the sense that in this context it can assume at least two meanings: it is the name of a syntax object of the VHDL language (this is the *default* meaning) and a name of a type of node in an entity-relationship graph. We are sure that the reader will not be confused by this semantic ambiguity.

The database must be able to store information on all the syntax objects present in a set of VHDL projects, each composed by an arbitrary number of VHDL language source files, located at arbitrary paths, with arbitrary names. Files are uniquely identified by their pathname; we will suppose that all the files come from the local filesystem.

The database must keep proper status information for every file, namely whether the file has been successfully parsed or is new/modified[1].

Projects, which are uniquely identified by their name, maintain the double nature of a collection of VHDL files (like real-life software engineering projects, managed via makefiles or with the use of integrated development environments) and of a collection of VHDL entities, like they were theoretically defined in the previous chapters.

For each project there is a certain number of entities, which have names that uniquely identify them inside a project namespace (that is, two entities can have the same name only provided that they belong to different projects).

Entity declarations appear in a certain files, at a given line number, and they can include entity port declarations and entity generic declarations. Entity ports are uniquely identified by their name in the context of a given entity (there cannot exist two ports in the same entity with the same name), have a mode (`IN, OUT, INOUT, LINKAGE, BUFFER`), are of a given type and appear at a given line (obviously in the same file as the entity declaration).

There are zero or more architectures implementing a given entity. An architecture declaration appears at a certain line of a file (which can be different from the file where the corresponding entity was declared). Each architecture is uniquely identified by its name in its entity namespace (that is, no two architectures with the same name can implement a given entity).

An architecture can declare an arbitrary number of components, component instances, signals and processes, which all appear at a given line in the same file where the architecture declaration appears (obvious, since they are included in that declaration), and all are uniquely identified by their names inside their architectures.

Components can contain port and generic declarations, which behave exactly as entity port and entity generic objects. For each component there can be an arbitrary number of component instances.

Signals are typed, and can be connected to component ports.

[1] In the GUI tool we will implement a specific user command to *touch* a file, causing the parser to parse that file again on the next project parse cycle.

There can be an arbitrary number of configurations, which can appear in any file belonging to the project; a configuration is composed by a number of configuration entries, each associating a given component declared inside an architecture with a different[2] architecture of an entity.

Processes declare variables, which are uniquely identified inside their process by their name, have a type and appear at a given line (of the same file as the process). Every time a process references a signal or a variable (by *defining* them, that is associating a new value to them, or by *using* them, that is, accessing their values, according to software engineering terminology), that reference is taken into account and logged into the database.

A process can declare an arbitrary number of subprograms, each one having a name, a number of arguments and possibly one return type. Subprograms can appear inside other subprograms. Subprogram arguments have a name (unique inside a subprogram declaration), a type and a mode (telling if the parameter is to be used for input and/or output). Subprograms can declare local variables which share the same properties as process variables.

Types can be subdivided between project-local types, which are defined with a `TYPE` construct inside the given project files, whereas global types are language built-in types or types defined in external libraries.

8.3 Entity-relationship model

An entity-relationship diagram, compatible with the above requirements, is presented in figure 8.1.

8.4 Shortcuts and work-arounds

The specifications presented above are in some points simplified with respect to actual ones (the actual number of attributes is higher, there are some minor details which have not been taken into account), and a common case: elements like processes can appear not only in architectures but also in entities (the so-called *passive* processes described in the theory chapter), subprograms can appear not only in processes but also in architectures, entities and alone. Components can be declared also out of any architecture. All this particular cases have one simple solution that was implemented all over the database: null-named syntax objects.

Every time a process is declared in an entity but out of any architecture, the corresponding architecture name for the record associated to that process will be set to `'(null)'`. Every time a subprogram is declared in an architecture but not inside a process, the corresponding process name for the record associated to that subprogram will be `'(null)'` again. The same happens for both process name and architecture name in records representing subprograms declared inside entities. If a subprogram declaration appears in a package (package are not considered, in our database model), its record will have null process name, architecture name and entity name.

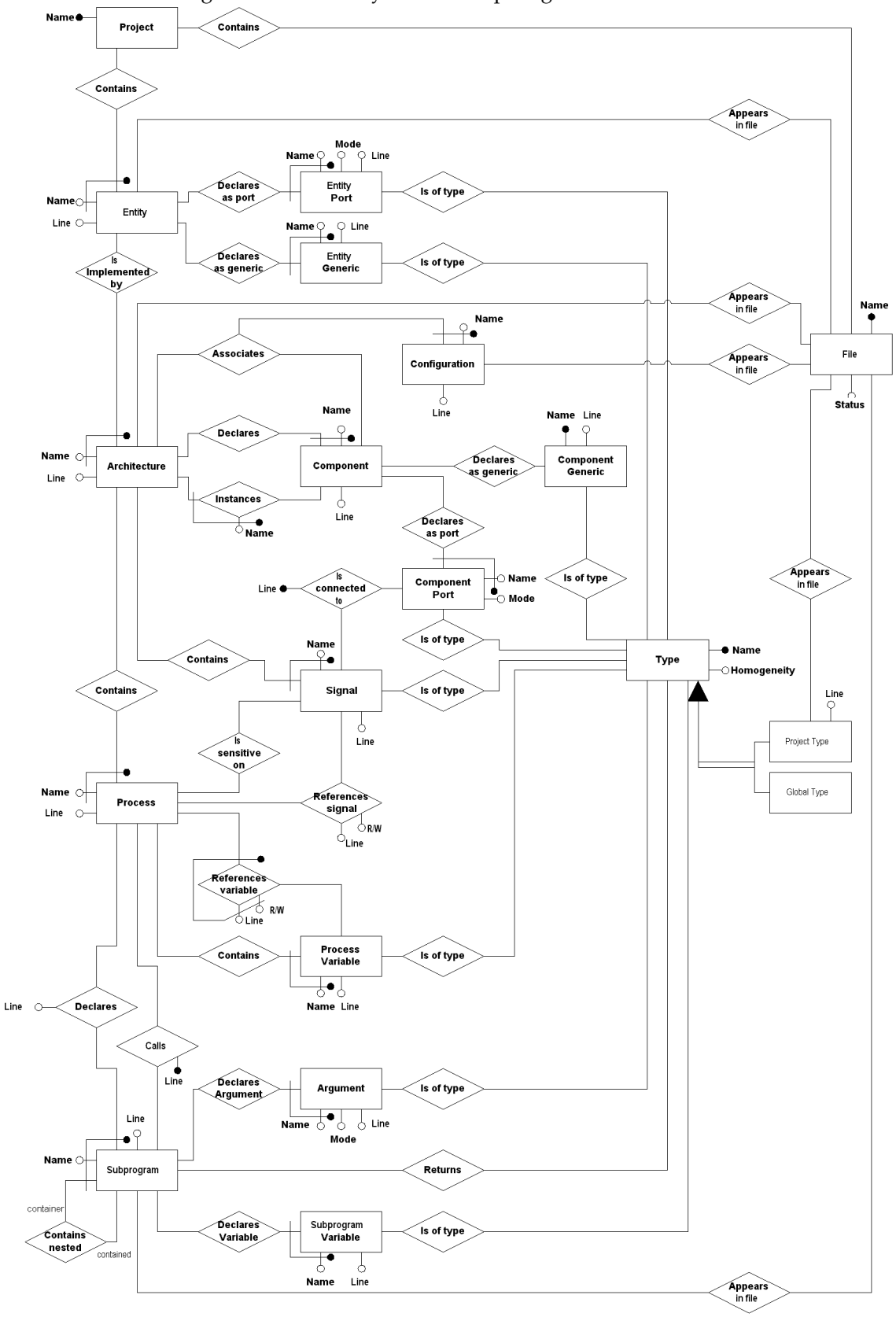
For reasons of convenience, the database is not in normal form: in carefully selected cases, the same information is replicated in more tables. For example process declarations appear in the same file as the architecture declarations in which they appear (because process declarations are actually *inside* architectures), therefore for each entry in the process table, it will always happen that the file name is the same as the corresponding entry in the

[2] unless architecture structural recursion is desired.

architecture table. Nevertheless, for performance and debugging purposes, it is incredibly convenient to immediately know in which file a process declaration appear by simply listing a table content. The same applies, for example, to process variables: if the table were normalized, and file name present only in architecture names, it would be necessary to do a three table join query to know where a variable is declared.

Such information redundancy occurs almost everywhere when homogeneity data is considered: though homogeneity is conceptually simple to calculate, it is highly inefficient to recursively perform a set of homogeneity resolution queries every time a signal or a variable is encountered: it is much better to save homogeneity data in a sort of *cache* which is simply implemented as an additional column for selected syntax objects, which can be null. The value of that column is initially null at row creation and receives a value when the syntax object homogeneity is resolve. From now on, every other algorithm requiring that homogeneity value will not resolve it again, instead the cached value will be used.

Figure 8.1: The Entity-Relationship diagram of our database.



8.5 Database tables

In the following subsections we will describe the structure of all the tables used in the VHDL parser. There are several more tables which are internally used by the model engine, mainly to store partial results in an efficient way, but there is no point in describing them here.

For each table the structure is given, together with a human-readable explanation of the meaning of each field and an example of the records that would be added to the database by the VHDL parser when processing a given input source code. For a few cases, an additional scheme is given where the relationship between the table content and the syntactic structure of the source code (from which data is extracted) is illustrated.

Source code examples come from either [3] or from real examples used for model tuning or validation. In the first case the project name is "example" and does not serve to group associated files, but just to provide a value for the "PROJECT_NAME" field (or, to be more formal, *relation* attribute);

Tables are listed in alphabetical order.

8.5.1 Table ARCHITECTURES

Structure for table `ARCHITECTURES`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	FILE_NAME	text			
5	START_LINE	integer			0
6	START_COLUMN	integer			0
7	END_LINE	integer			0
8	IDENTIFIER_LENGTH	tinyint(4)			0
9	HOMOGENEITY	integer	Yes		NULL
10	LINE_COUNT	integer	Yes		NULL

The VHDL parser adds an entry to this table for every architecture definition it encounters. An entry in this table has the following meaning: in project (1), an architecture (3) of entity (2) is declared in file (4) from line (5) to line (7), for a total number of lines equal to (10). The name of the architecture begins at column (6) of line (5) and is (8) characters long. If the homogeneity has been already calculated for this architecture, it is equal to (9), otherwise (9) is NULL.

From a formal point of view, the contents of (3) and (2) are taken respectively from the symbols *identifier* and *entity_name* of the following production:

⟨*architecture_body*⟩ ::= `architecture` ⟨*identifier*⟩ `of` ⟨*entity_name*⟩ `is`
⟨*architecture_declarative_part*⟩ `begin` ⟨*architecture_statement_part*⟩
`end` [`architecture`] [⟨*architecture_simple_name*⟩] `;`

Example: the following section of code, coming from file `fg_01_11.vhd`, and belonging to project "example":

```
architecture struct of reg4 is
  signal int_clk : bit;
begin
  bit0 : entity work.d_latch(basic)
```

```
5       port map (d0, int_clk, q0);
    bit1 : entity work.d_latch(basic)
       port map (d1, int_clk, q1);
    bit2 : entity work.d_latch(basic)
       port map (d2, int_clk, q2);
10  bit3 : entity work.d_latch(basic)
       port map (d3, int_clk, q3);
   gate : entity work.and2(basic)
      port map (en, clk, int_clk);
end architecture struct;
```

causes the following tuple to be added to the table:

PROJECT_NAME	Example
ENTITY_NAME	reg4
ARCHITECTURE_NAME	struct
FILE_NAME	/root/sources/fg_01_11.vhd
START_LINE	1
START_COLUMN	14
END_LINE	19
IDENTIFIER_LENGTH	6
HOMOGENEITY	NULL
LINE_COUNT	19

8.5.2 Table COMPONENT_CONNECTIONS

Structure for table `COMPONENT_CONNECTIONS`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	COMPONENT_NAME	tinytext		Primary key	
5	INSTANCE_NAME	tinytext		Primary key	
6	PARAMETER_ORDER	integer		Primary key	0
7	CONNECTED_SIGNAL	tinytext		Primary key	
8	FILE_NAME	text			
9	LINE	integer			0

Every time a component port is connected (to a signal or to a port of the architecture that surrounds the component, it does not matter), that connection is annotated in this table.

An entry in this table has the following meaning: in project (1), in architecture (3) of entity (2), the port number (6) of (5) (which is an instance of component (4)) is connected to a signal named (7). The connection appears at line (9) of file (8).

The following table represents the tuples added by the VHDL parser when it processes file `fg_01_11.vhd` (the same as in the previous example).

PROJECT _NAME	ENTITY _NAME	ARCH. _NAME	COMP. _NAME	INSTANCE _NAME	P. O.	CONNECTED _SIGNAL	FILE_NAME	LINE
Example	reg4	struct	and2	gate	1	en	/root/sources/ fg_01_11.vhd	17
Example	reg4	struct	and2	gate	2	clk	/root/sources/ fg_01_11.vhd	17
Example	reg4	struct	and2	gate	3	int_clk	/root/sources/ fg_01_11.vhd	17
Example	reg4	struct	d_latch	bit0	1	d0	/root/sources/ fg_01_11.vhd	8
Example	reg4	struct	d_latch	bit0	2	int_clk	/root/sources/ fg_01_11.vhd	8
Example	reg4	struct	d_latch	bit0	3	q0	/root/sources/ fg_01_11.vhd	8
Example	reg4	struct	d_latch	bit1	1	d1	/root/sources/ fg_01_11.vhd	10
Example	reg4	struct	d_latch	bit1	2	int_clk	/root/sources/ fg_01_11.vhd	10
Example	reg4	struct	d_latch	bit1	3	q1	/root/sources/ fg_01_11.vhd	10
Example	reg4	struct	d_latch	bit2	1	d2	/root/sources/ fg_01_11.vhd	12
Example	reg4	struct	d_latch	bit2	2	int_clk	/root/sources/ fg_01_11.vhd	12
Example	reg4	struct	d_latch	bit2	3	q2	/root/sources/ fg_01_11.vhd	12
Example	reg4	struct	d_latch	bit3	1	d3	/root/sources/ fg_01_11.vhd	14
Example	reg4	struct	d_latch	bit3	2	int_clk	/root/sources/ fg_01_11.vhd	14
Example	reg4	struct	d_latch	bit3	3	q3	/root/sources/ fg_01_11.vhd	14

8.5.3 Table COMPONENT_DECLARATIONS

Structure for table `COMPONENT_DECLARATIONS`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	COMPONENT_NAME	tinytext		Primary key	
5	FILE_NAME	text			
6	LINE_NUMBER	integer			0

The VHDL parser adds an entry to this table every time a component declaration is found. An entry in this table has the following meaning: in project (1), architecture (3) of entity (2) declares a component called (4). That declaration appears at line (6) of file (5).

Example: the following section of code, coming from file `fg_13_05.vhd`, and belonging to project "example":

```
library star_lib;
use star_lib.edge_triggered_Dff;

configuration reg4_gate_level of reg4 is

  for struct  -- architecture of reg4

    for bit0 : flipflop
      use entity edge_triggered_Dff(hi_fanout);
    end for;
```

```
    for others : flipflop
      use entity edge_triggered_Dff(basic);
    end for;
  end for; -- end of architecture struct
end configuration reg4_gate_level;

entity fg_13_05 is
end entity fg_13_05;

architecture test of fg_13_05 is
  component reg4 is
    port ( clk, clr : in bit; d : in bit_vector(0 to 3);
           q : out bit_vector(0 to 3) );
  end component reg4;

  signal clk, clr : bit;
  signal d, q : bit_vector(0 to 3);
begin
  flag_reg : component reg4
    port map ( clk => clk, clr => clr, d => d, q => q );
end architecture test;


configuration fg_13_05_test of fg_13_05 is
  for test
    for flag_reg : reg4
      use configuration work.reg4_gate_level;
    end for;
  end for;
end configuration fg_13_05_test;
```

causes the following tuple to be added to the table:

PROJECT_NAME	Example
ENTITY_NAME	fg_13_05
ARCHITECTURE_NAME	test
COMPONENT_NAME	reg4
FILE_NAME	/root/sources/fg_13_05.vhd
LINE_NUMBER	34

8.5.4 Table COMPONENT_DECLARATION_GENERICS

Structure for table `COMPONENT_DECLARATION_GENERICS`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext			
4	COMPONENT_NAME	tinytext			
5	GENERIC_NAME	tinytext		Primary key	
6	GENERIC_TYPE	tinytext			
7	FILE_NAME	text			
8	LINE	int(11)			0
9	HOMOGENEITY	int(11)	Yes		NULL

The VHDL parser adds an entry to this table every time a generic declaration is found within a component declaration. An entry in this table has the following meaning: in project (1), architecture (3) of entity (2) contains the declaration of a component called (4), which declares a generic named (5), of type (6). That declaration appears at line (8) of file (7). If the homogeneity has been already calculated for this generic, it is equal to (9), otherwise (9) is NULL.

Example: the following section of code, coming from file `bprom.vhd`, and belonging to project "Leon":

```
architecture behav of virtex_prom256 is
    component iram
      generic ( index    : integer   := 0;          -- Byte lane (0 - 3)
                Abits    : Positive  := 10;         -- Default 10 address bits (1 Kbyte)
                echk     : integer   := 0;          -- Generate EDAC checksum
                tacc     : integer   := 10;         -- access time (ns)
                fname    : string    := "ram.dat" ); -- File to read from
      port (
        A    : in std_logic_vector;
        D    : inout std_logic_vector(7 downto 0);
        CE1  : in std_logic;
        WE   : in std_logic;
        OE   : in std_logic
        );
    end component;

signal gnd          :  std_logic  := '0';
signal vcc          :  std_logic  := '1';
signal address      :  std_logic_vector(7 downto 0);
signal data         :  std_logic_vector(31 downto 0);
begin
  x : process(clk)
  begin
    if rising_edge(clk) then
      address <= addr;
    end if;
  end process;

    romarr : for i in 0 to 3 generate
      rom0 : iram
    generic map (index => i, abits => 8, echk => 0, tacc => 10,
                 fname => "tsource/bprom.dat")
        port map (A => address(7 downto 0),
          D => data((31 - i*8) downto (24-i*8)), CE1 => gnd,
          WE => VCC, OE => gnd);
    end generate;

    do <= data;
end;
```

causes the following tuples to be added to the table:

PROJECT _NAME	ENTITY _NAME	ARCHITECTURE _NAME	COMPONENT _NAME	GENERIC _NAME	GENERIC _TYPE	FILE_NAME	LINE	HOMOGENEITY
Leon	virtex_prom256	behav	iram	index	integer	/root/sources/leon1-2.4.0 testbench/bprom.vhd	5	NULL
Leon	virtex_prom256	behav	iram	Abits	Positive	/root/sources/leon1-2.4.0 testbench/bprom.vhd	6	NULL
Leon	virtex_prom256	behav	iram	echk	integer	/root/sources/leon1-2.4.0 testbench/bprom.vhd	7	NULL
Leon	virtex_prom256	behav	iram	tacc	integer	/root/sources/leon1-2.4.0 testbench/bprom.vhd	8	NULL
Leon	virtex_prom256	behav	iram	fname	string	/root/sources/leon1-2.4.0 testbench/bprom.vhd	9	NULL

8.5.5 Table COMPONENT_INSTANTIATIONS

Structure for table `COMPONENT_INSTANTIATIONS`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	COMPONENT_NAME	tinytext		Primary key	
5	INSTANCE_NAME	tinytext		Primary key	
6	SYNTAX	tinytext	Yes		NULL
7	FILE_NAME	text			
8	LINE_NUMBER	integer			0

The VHDL parser adds an entry to this table every time it encounters the declaration of a component instance, or the instantiation of an entity. An entry in this table has the following meaning: in project (1), an architecture (3) of an entity (2) contains the use of an instance of component (4), named (5) in file (7) at line (8). If (6) is `ENTITY`, then (4) is the name of an entity (usually different from (2), unless you are building a recursive structural architecture) and it was instanced with a construct like the following:

```
architecture struct of reg4 is
begin
  bit0 : entity work.d_latch(basic)
         port map (d0, int_clk, q0);
end
```

Otherwise, if (6) is `COMPONENT`, then (4) is the name of a component and it was declared and instanced with constructs like the following:

```
architecture test of fg_13_05 is
  component reg4 is
    port ( clk , clr       : in bit ;
           d               : in bit_vector(0 to 3);
           q               : out bit_vector(0 to 3) );
  end component reg4;
 [...]
begin
  flag_reg : component reg4
    port map ( clk => clk, clr => clr , d => d, q => q );
end architecture test;
```

Example: the same code from file `fg_13_05.vhd`, belonging to project "example", causes the following entries to be added to table:

PROJECT _NAME	ENTITY _NAME	ARCHITECTURE _NAME	COMPONENT _NAME	INSTANCE _NAME	SYNTAX	FILE_NAME	LINE _NUMBER
Example	fg_13_05	test	reg4	flag_reg	COMPONENT	/root/sources/ fg_13_05.vhd	42
Example	reg4	struct	and2	gate	ENTITY	/root/sources/ fg_01_11.vhd	17
Example	reg4	struct	d_latch	bit0	ENTITY	/root/sources/ fg_01_11.vhd	8
Example	reg4	struct	d_latch	bit1	ENTITY	/root/sources/ fg_01_11.vhd	10
Example	reg4	struct	d_latch	bit2	ENTITY	/root/sources/ fg_01_11.vhd	12
Example	reg4	struct	d_latch	bit3	ENTITY	/root/sources/ fg_01_11.vhd	14

8.5.6 Table COMPONENT_PORTS

Structure for table `COMPONENT_PORTS`					
	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	COMPONENT_NAME	tinytext		Primary key	
5	PORT_NAME	tinytext		Primary key	
6	PORT_MODE	tinytext			
7	PORT_TYPE	tinytext			
8	FILE_NAME	text			
9	LINE	int(11)			0

The VHDL parser adds an entry to this table for every port declared inside a component declaration. An entry in this table has the following meaning: in project (1), inside architecture (3) of entity (2), there exists a declaration of component (4); this component has a port named (3) of type (5) and mode (4). That declaration appears in file (8) at line (9).

Example: the following table shows the rows that would be added to this table by running the VHDL parser on the same section of code (coming from file `bprom.vhd`, and belonging to project "Leon") as the example for table `COMPONENT_DECLARATION_GENERICS`.

PROJECT_NAME	ENTITY_NAME	ARCHITECTURE_NAME	COMPONENT_NAME	PORT_NAME	PORT_MODE	PORT_TYPE	FILE_NAME	LINE
Leon	virtex_prom256	behav	iram	A	IN	std_logic_vector	/root/sources/leon1-2.4.0 testbench/bprom.vhd	11
Leon	virtex_prom256	behav	iram	D	INOUT	std_logic_vector	/root/sources/leon1-2.4.0 testbench/bprom.vhd	12
Leon	virtex_prom256	behav	iram	CE1	IN	std_logic	/root/sources/leon1-2.4.0 testbench/bprom.vhd	13
Leon	virtex_prom256	behav	iram	WE	IN	std_logic	/root/sources/leon1-2.4.0 testbench/bprom.vhd	14
Leon	virtex_prom256	behav	iram	OE	IN	std_logic	/root/sources/leon1-2.4.0 testbench/bprom.vhd	16

8.5.7 Table CONFIGURATIONS

Structure for table `CONFIGURATIONS`					
	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	CONFIGURATION_NAME	tinytext		Primary key	
3	CONFIGURATION_ENTITY	tinytext		Primary key	
4	ENTITY_NAME	tinytext		Primary key	
5	ARCHITECTURE_NAME	tinytext		Primary key	
6	COMPONENT_NAME	tinytext		Primary key	
7	INSTANCE_NAME	tinytext		Primary key	
8	USE_ENTITY	tinytext			
9	USE_ARCHITECTURE	tinytext			
10	SYNTAX	tinytext			
11	FILE_NAME	text			
12	LINE	integer			0

We think that the best way to understand how information coming from `CONFIGURATION` constructs in source file affect rows added to this table is to thoroughly examine the following example, which comes from file `meclibrary.vhd`, line 21781 and following, and is by far the most complex example of configuration available in our project base:

```
library MECLibrary;
use MECLibrary.all;

configuration MECConfiguration of MEC is
  for Schematic


    for all : MECGen
      use entity MECLibrary.MECGen(Schematic);
      for Schematic

--------------------------------
--MEC Functionallity
--------------------------------
        for all : MECFunc
          use entity MECLibrary.MECFunc(Mini_Spec);
          for Mini_Spec

--------------------------------
--UARTS
--------------------------------
            for all : uarts
              use entity MECLibrary.uarts(mini_spec);
              for Mini_Spec
                for all : uartcontrol
                  use entity MECLibrary.uartcontrol(mini_spec);
                end for;
                for all : uart
                  use entity MECLibrary.uart(VHDL_RTL);
                end for;
              end for;
            end for;

--------------------------------
--Timers
--------------------------------
            for all : timers
              use entity MECLibrary.timers(mini_spec);
              for Mini_Spec
                for all : timercontrol
                  use entity MECLibrary.timercontrol(mini_spec);
                end for;
                for all : rtctimer
                  use entity MECLibrary.rtctimer(mini_spec);
                end for;
                for all : genpurptimer
                  use entity MECLibrary.genpurptimer(mini_spec);
                end for;
              end for;
            end for;

--------------------------------
--Test and Debug
--------------------------------
            for all : tap
              use entity MECLibrary.tap(mini_spec);
```

```
            end for;

-------------------------------
--System Bus Interface
-------------------------------
            for all : systembusinterface
              use entity MECLibrary.systembusinterface(mini_spec);
              for Mini_Spec
                for all : addressdecoder
                  use entity MECLibrary.addressdecoder(mini_spec);
                    for Mini_Spec
                      for all : LBuff
                        use entity MECLibrary.LBuff(mini_spec);
                      end for;
                    end for;
                end for;
                for all : mec_latch_e
                  use entity MECLibrary.mec_latch_e(mini_spec);
                end for;
                for all : busarbiter
                  use entity MECLibrary.busarbiter(mini_spec);
                end for;
                for all : edac
                  use entity MECLibrary.edac(mini_spec);
                end for;
                for all : accesscontrol
                  use entity MECLibrary.accesscontrol(mini_spec);
                    for Mini_Spec
                      for all : LBuff
                        use entity MECLibrary.LBuff(mini_spec);
                      end for;
                      for all : AccessControl_fsm
                        use entity MECLibrary.AccessControl_fsm(mini_spec);
                      end for;
                    end for;
                end for;
                for all : mem_io_config_e
                  use entity MECLibrary.mem_io_config_e(mini_spec);
                end for;
              end for;
            end for;


-------------------------------
--MEC Control and Support Functions
-------------------------------
            for all : meccontrolandsupportfunctions
            use entity MECLibrary.meccontrolandsupportfunctions(mini_spec);
              for Mini_Spec
                for all : systemclocks
                  use entity MECLibrary.systemclocks(mini_spec);
                end for;
                for all : startupctlandres
                  use entity MECLibrary.startupctlandres(mini_spec);
                end for;
                for all : powerdownmodectl
                  use entity MECLibrary.powerdownmodectl(mini_spec);
                end for;
              end for;
            end for;

-------------------------------
```

```
--Interrupt and Errorhandling
21900 ----------------------------------
            for all : interruptanderrorhandling
            use entity MECLibrary.interruptanderrorhandling(mini_spec);
              for Mini_Spec
                for all : fault_handler
21905             use entity MECLibrary.fault_handler(mini_spec);
                end for;

                for all : watchdog
                  use entity MECLibrary.watchdog(mini_spec);
21910           end for;

                for all : int_handler
                  use entity MECLibrary.int_handler(mini_spec);
                end for;
21915

                for all : error_handler
                  use entity MECLibrary.error_handler(mini_spec);
                end for;
              end for;
21920       end for;

----------------------------------
--Data Mux
----------------------------------
21925       for all : datamux
              use entity MECLibrary.datamux(mini_spec);
            end for;
          end for;      --End MECFunc(Mini_Spec)
        end for;        --End MECFunc
21930


----------------------------------
--Buffers
----------------------------------
21935   for all : clk_buffer
          use entity MECLibrary.clk_buffer(clk_buffer_body);
        end for;

        for all : test_buffer
21940     use entity MECLibrary.test_buffer(test_buffer_body);
        end for;

        for all : iu_in_buffer
          use entity MECLibrary.iu_in_buffer(iu_in_buffer_body);
21945   end for;

        for all : setupandcheck_buffer
          use entity MECLibrary.setupandcheck_buffer(setupandcheck_buffer_body);
        end for;
21950

        for all : dma_buffer
          use entity MECLibrary.dma_buffer(dma_buffer_body);
        end for;

21955   for all : checkbits_buffer
          use entity MECLibrary.checkbits_buffer(checkbits_buffer_body);
        end for;

        for all : systemerror_buffer
21960     use entity MECLibrary.systemerror_buffer(systemerror_buffer_body);
```

```
        end for;

        for all : serial_buffer
          use entity MECLibrary.serial_buffer(serial_buffer_body);
21965   end for;

        for all : sysctrl_buffer
          use entity MECLibrary.sysctrl_buffer(sysctrl_buffer_body);
        end for;
21970
        for all : iu_out_buffer
          use entity MECLibrary.iu_out_buffer(iu_out_buffer_body);
        end for;

21975   for all : data_buffer
          use entity MECLibrary.data_buffer(data_buffer_body);
        end for;

        for all : mem_buffer
21980     use entity MECLibrary.mem_buffer(mem_buffer_body);
        end for;

        for all : io_buffer
          use entity MECLibrary.io_buffer(io_buffer_body);
21985   end for;

        for all : externalinterrupt_buffer
          use entity MECLibrary.externalinterrupt_buffer(externalinterrupt_buffer_body);
        end for;
21990
      end for;          --End MECGen(Schematic)
    end for;            --End MECGen
  end for;              --End MEC(Schematic)
end MECConfiguration; --End MEC configuration
```

The above section of code it causes the following lines to be annotated in the database:

PROJECT _NAME	CONFIGURATION _NAME	C. E.	ENTITY _NAME	ARCHIT. _NAME	COMPONENT _NAME	I. N.	USE_ENTITY	USE_ ARCH.	SYNTAX	FILE_NAME	LINE
ERC32	MECConfiguration	MEC	MECGen	Schematic	io_buffer	ALL	io_buffer	io_buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21984
ERC32	MECConfiguration	MEC	MECGen	Schematic	externalinterrupt _buffer	ALL	externalinterrupt _buffer	externalinterrupt _buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21988
ERC32	MECConfiguration	MEC	MECGen	Schematic	mem_buffer	ALL	mem_buffer	mem _buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21980
ERC32	MECConfiguration	MEC	MECGen	Schematic	iu_out_buffer	ALL	iu_out_buffer	iu_out _buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21972
ERC32	MECConfiguration	MEC	MECGen	Schematic	data_buffer	ALL	data_buffer	data _buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21976
ERC32	MECConfiguration	MEC	MECGen	Schematic	sysctrl_buffer	ALL	sysctrl_buffer	sysctrl _buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21968
ERC32	MECConfiguration	MEC	MECGen	Schematic	systemerror _buffer	ALL	systemerror _buffer	systemerror _buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21960
ERC32	MECConfiguration	MEC	MECGen	Schematic	serial_buffer	ALL	serial_buffer	serial _buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21964
ERC32	MECConfiguration	MEC	MECGen	Schematic	dma_buffer	ALL	dma_buffer	dma _buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21952
ERC32	MECConfiguration	MEC	MECGen	Schematic	checkbits _buffer	ALL	checkbits _buffer	checkbits _buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21956
ERC32	MECConfiguration	MEC	MECGen	Schematic	setupandcheck _buffer	ALL	setupandcheck _buffer	setupandcheck _buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21948
ERC32	MECConfiguration	MEC	MECGen	Schematic	test_buffer	ALL	test_buffer	test _buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21940
ERC32	MECConfiguration	MEC	MECGen	Schematic	iu_in_buffer	ALL	iu_in_buffer	iu_in _buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21944
ERC32	MECConfiguration	MEC	MECGen	Schematic	clk_buffer	ALL	clk_buffer	clk _buffer_body	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21936
ERC32	MECConfiguration	MEC	MECGen	Schematic	MECFunc	ALL	MECFunc	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21796
ERC32	MECConfiguration	MEC	MECFunc	Mini_Spec	datamux	ALL	datamux	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21926
ERC32	MECConfiguration	MEC	interruptand errorhandling	Mini_Spec	int_handler	ALL	int_handler	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21913
ERC32	MECConfiguration	MEC	interruptand errorhandling	Mini_Spec	error_handler	ALL	error_handler	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21917
ERC32	MECConfiguration	MEC	MECFunc	Mini_Spec	interruptand errorhandling	ALL	interruptand errorhandling	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21902
ERC32	MECConfiguration	MEC	interruptand errorhandling	Mini_Spec	watchdog	ALL	watchdog	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21909
ERC32	MECConfiguration	MEC	interruptand errorhandling	Mini_Spec	fault_handler	ALL	fault_handler	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21905
ERC32	MECConfiguration	MEC	MECFunc	Mini_Spec	meccontroland supportfunctions	ALL	meccontroland supportfunctions	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21884
ERC32	MECConfiguration	MEC	meccontroland supportfunctions	Mini_Spec	startupctlandres	ALL	startupctlandres	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21890
ERC32	MECConfiguration	MEC	meccontroland supportfunctions	Mini_Spec	powerdownmodectl	ALL	powerdownmodectl	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21893
ERC32	MECConfiguration	MEC	meccontroland supportfunctions	Mini_Spec	systemclocks	ALL	systemclocks	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21887
ERC32	MECConfiguration	MEC	systembusinterface	Mini_Spec	mem_io_config_e	ALL	mem_io_config_e	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21874
ERC32	MECConfiguration	MEC	MECFunc	Mini_Spec	systembusinterface	ALL	systembusinterface	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21843
ERC32	MECConfiguration	MEC	accesscontrol	Mini_Spec	LBuff	ALL	LBuff	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21866
ERC32	MECConfiguration	MEC	accesscontrol	Mini_Spec	AccessControl_fsm	ALL	AccessControl _fsm	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21869
ERC32	MECConfiguration	MEC	systembusinterface	Mini_Spec	accesscontrol	ALL	accesscontrol	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21863
ERC32	MECConfiguration	MEC	systembusinterface	Mini_Spec	edac	ALL	edac	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21860
ERC32	MECConfiguration	MEC	systembusinterface	Mini_Spec	busarbiter	ALL	busarbiter	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21857
ERC32	MECConfiguration	MEC	systembusinterface	Mini_Spec	mec_latch_e	ALL	mec_latch_e	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21854
ERC32	MECConfiguration	MEC	systembusinterface	Mini_Spec	addressdecoder	ALL	addressdecoder	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21846
ERC32	MECConfiguration	MEC	timers	Mini_Spec	genpurptimer	ALL	genpurptimer	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21827
ERC32	MECConfiguration	MEC	MECFunc	Mini_Spec	timers	ALL	timers	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21818
ERC32	MECConfiguration	MEC	addressdecoder	Mini_Spec	LBuff	ALL	LBuff	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21849
ERC32	MECConfiguration	MEC	MECFunc	Mini_Spec	tap	ALL	tap	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21836
ERC32	MECConfiguration	MEC	timers	Mini_Spec	rtctimer	ALL	rtctimer	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21824
ERC32	MECConfiguration	MEC	uarts	Mini_Spec	uartcontrol	ALL	uartcontrol	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21806
ERC32	MECConfiguration	MEC	MECFunc	Mini_Spec	uarts	ALL	uarts	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21803
ERC32	MECConfiguration	MEC	timers	Mini_Spec	timercontrol	ALL	timercontrol	Mini_Spec	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21821
ERC32	MECConfiguration	MEC	uarts	Mini_Spec	uart	ALL	uart	VHDL_RTL	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21809
ERC32	MECConfiguration	MEC	MEC	Schematic	MECGen	ALL	MECGen	Schematic	ENTITY	/root/sources/ erc32vhdl-1.0/meclibrary.vhd	21789

8.5.8 Table ENTITIES

Structure for table `ENTITIES`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	FILE_NAME	text			
4	START_LINE	integer			0
5	START_COLUMN	integer			0
6	END_LINE	integer			0
7	IDENTIFIER_LENGTH	tinyint(4)			0
8	HOMOGENEITY	integer	Yes		NULL
9	LINE_COUNT	integer	Yes		NULL

The VHDL parser adds an entry to this table for every entity definition it encounters. An entry in this table has the following meaning: in project (1), an entity (2) is declared in file (3) from line (4) to line (6), for a total number of lines equal to (9). The name of the entity begins at column (5) of line (4) and is (7) characters long. If the homogeneity has been already calculated for this entity, it is equal to (8), otherwise (8) is NULL.

From a formal point of view, the contents of (2) is taken from the symbol *identifier* of the following production:

⟨*entity_declaration*⟩ ::= `entity` ⟨*identifier*⟩ `is` ⟨*entity_header*⟩ ⟨*entity_declarative_part*⟩ [`begin` ⟨*entity_statement_part*⟩] `end` [`entity`] [⟨*entity_simple_name*⟩] `;`

Example: the following section of code, coming from file `tb_01_03.vhd`, belonging to project "example":

```
entity shift_adder is
  port ( addend : in integer;
         augend : in integer;
         sum : out integer;
         add_control : in bit );
end entity shift_adder;

-----------

architecture behavior of shift_adder is
begin
end architecture behavior;

-----------

entity reg is
  port ( d : in integer;
         q : out integer;
         en : in bit;
         reset : in bit );
end entity reg;

-----------

architecture behavior of reg is
begin
end architecture behavior;
```

```
-----------

entity shift_reg is
  port ( d : in integer;
         q : out bit;
         load : in bit;
         clk : in bit );
end entity shift_reg;

-----------

architecture behavior of shift_reg is
begin
end architecture behavior;
```

causes the following tuples to be added to the table:

PROJECT _NAME	ENTITY _NAME	FILE _NAME	START _LINE	START_ COLUMN	END_ LINE	IDENTIFIER _LENGTH	HOMOGENEITY	LINE _COUNT
Example	reg	/root/sources/ tb_01_03.vhd	15	8	18	3	NULL	4
Example	shift_adder	/root/sources/ tb_01_03.vhd	1	8	5	11	NULL	5
Example	shift_reg	/root/sources/ tb_01_03.vhd	28	8	31	9	NULL	4

8.5.9 Table GENERICS

Structure for table `GENERICS`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	GENERIC_NAME	tinytext		Primary key	
4	GENERIC_TYPE	tinytext			
5	FILE_NAME	text			
6	LINE	int(11)			0
7	HOMOGENEITY	int(11)	Yes		NULL

The VHDL parser adds an entry to this table every a generic declaration is found inside an entity. An entry in this table has the following meaning: in project (1) there is an entity called (2) which declares a generic named (3) of type (4). The declaration appears at line (6) of file (5). If the homogeneity has been already calculated for this generic, it is equal to (7), otherwise (7) is NULL.

Example: the following section of code, coming from file `ramlib_sim.vhd`, and belonging to project "Free6502":

```
entity ram_dp is
    generic (addr_bits          :  integer;
             data_bits          :  integer;
             register_out_flag  :  integer := 0;
             block_type         :  integer := 0);
    port (reset      : in  std_logic;
          wr_clk     : in  std_logic;
          wr_en      : in  std_logic;
          wr_addr    : in  std_logic_vector (addr_bits-1 downto 0);
          wr_data    : in  std_logic_vector(data_bits-1 downto 0);
```

```
        rd_clk      : in  std_logic;
        rd_addr     : in  std_logic_vector (addr_bits-1 downto 0);
        rd_data     : out std_logic_vector(data_bits-1 downto 0)
      );

  subtype word is std_logic_vector (data_bits-1 downto 0);
  constant nwords : integer := 2 ** addr_bits;
  type ram_type is array (0 to nwords-1) of word;
end ram_dp;
```

causes the following rows to be added to the table:

PROJECT_NAME	ENTITY_NAME	GENERIC_NAME	GENERIC_TYPE	FILE_NAME FILE_NAME	LINE	HOMO-GENEITY
Free6502	ram_dp	addr_bits	integer	/root/sources/Free6502/ramlib_sim.vhd	80	NULL
Free6502	ram_dp	block_type	integer	/root/sources/Free6502/ramlib_sim.vhd	83	NULL
Free6502	ram_dp	data_bits	integer	/root/sources/Free6502/ramlib_sim.vhd	81	NULL
Free6502	ram_dp	register_out_flag	integer	/root/sources/Free6502/ramlib_sim.vhd	82	NULL

8.5.10 Table GLOBAL_TYPES

Structure for table `GLOBAL_TYPES`

	Field name	Type	Can be null	Is key	Default
1	TYPE_NAME	tinytext			
2	HOMOGENEITY	integer			0

The meaning use of this table is explained in the next subsection.

8.5.11 Table LOCAL_TYPES

Structure for table `LOCAL_TYPES`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	TYPE_NAME	tinytext		Primary key	
3	MULTIPLIER	integer			0
4	REFERENCE_TYPE	tinytext		Primary key	
5	FILE_NAME	text			
6	LINE	integer		Primary key	0

This table contains the homogeneity components for each type declared in project files, according to the following conventions: when (4) is `'(null)'`, in (3) is contained the absolute homogeneity of the type named (2). When (4) is not `'(null)'`, the homogeneity of type (2) is given by (3) times the homogeneity of (4). This last case happens for example when a `record` is declared.

This table is mainly used by the homogeneity resolution routines of our model engine. In order to resolve homogeneity of a record type with a given name, all records of this table having the desired (2) are looked for and, for each records, homogeneity of type (4) is recursively resolved via the same algorithm, then multiplied by (3) and accumulated into a summation variable. The final summation value is the desired homogeneity.

The algorithm above, without modifications, fails whenever type (4) is not present in another row of the table under column (2). This is quite common for language predefined types and types defined in external libraries not provided among project types. A simple solution to this problem is manually including the missing type in table `GLOBAL_TYPES`, and modifying the above algorithm in order to look for desired type in that table when it could not be found in `LOCAL_TYPES`. `GLOBAL_TYPES` is modified by the user only, whereas `LOCAL_TYPES` by the parser only, this allowing for a good design clearness.

The following example illustrates how a record composed by five fields of different types is transcribed into the database. The declaration comes from file `iface.vhd`, project "Leon".

```
type io_out_type is record
   piol          : std_logic_vector(15 downto 0);      -- I/O port outputs
   piodir        : std_logic_vector(15 downto 0);      -- I/O port direction
   errorn        : std_logic;                          -- CPU in error mode
   wdog          : std_logic;                          -- watchdog output
   pci_arb_gnt_n : std_logic_vector(0 to 3);
end record;
```

The corresponding rows added to the table follow.

PROJECT_NAME	TYPE_NAME	MULTIPLIER	REFERENCE_TYPE	FILE_NAME	LINE
Leon	io_out_type	1	std_logic_vector	/root/sources/leon1-2.4.0 processor/iface.vhd	16
Leon	io_out_type	1	std_logic_vector	/root/sources/leon1-2.4.0 processor/iface.vhd	13
Leon	io_out_type	1	std_logic	/root/sources/leon1-2.4.0 processor/iface.vhd	15
Leon	io_out_type	1	std_logic	/root/sources/leon1-2.4.0 processor/iface.vhd	14
Leon	io_out_type	1	std_logic_vector	/root/sources/leon1-2.4.0 processor/iface.vhd	12

8.5.12 Table OBJECT_REFERENCES

Structure for table `OBJECT_REFERENCES`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	PROCESS_NAME	tinytext		Primary key	
5	FILE_NAME	text			
6	LINE	integer			0
7	REFERENCED_OBJECT	tinytext		Primary key	
8	REFERENCE_TYPE	tinytext			

The VHDL parser adds an entry to this table every time the value of a variable or of a signal is accessed. An entry in this table has the following meaning: in project (1), a signal or a variable of name (7) is accessed inside process (4) in architecture (3) of entity (2), in file (5) at line (6). If the value of the variable or of the signal is read, (8) contains `USE`; if the value was changed, (8) contains `DEFINITION`.

If the assignment is outside a process, (4) is left to an empty string; if the assignment is outside an architecture, (3) is left to an empty string.

The same section of code as before causes the following tuples to be added:

PROJECT _NAME	ENTITY _NAME	ARCH. _NAME	PROCESS _NAME	FILE _NAME	LINE	REFERENCED _OBJECT	REFERENCE _TYPE
Example	ch_04_04	test	process_04_1_i	/root/sources/ ch_04_04.vhd	43	count	USE
Example	ch_04_04	test	process_04_1_i	/root/sources/ ch_04_04.vhd	43	count	DEFINITION
Example	ch_04_04	test	process_04_1_i	/root/sources/ ch_04_04.vhd	44	index	USE
Example	ch_04_04	test	process_04_1_i	/root/sources/ ch_04_04.vhd	44	free_map	USE
Example	ch_04_04	test	process_04_1_i	/root/sources/ ch_04_04.vhd	49	A	USE
Example	ch_04_04	test	process_04_1_i	/root/sources/ ch_04_04.vhd	49	true	USE
Example	ch_04_04	test	process_04_1_i	/root/sources/ ch_04_04.vhd	49	false	USE

8.5.13 Table PORTS

Structure for table `PORTS`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	PORT_NAME	tinytext		Primary key	
4	PORT_MODE	varchar(10)			
5	PORT_TYPE	varchar(64)			
6	FILE_NAME	text			
7	LINE	integer			0
8	HOMOGENEITY	integer	Yes		NULL

The VHDL parser adds an entry to this table for every port declared inside an entity definition. An entry in this table has the following meaning: in project (1), entity (2) has a port named (3) of type (5) and mode (4), declared in file (6) at line (7). If the homogeneity of the port has been already calculated, it is equal to (8), otherwise (8) is NULL.

The following productions are all the productions required to reach the port declaration statements from the *entity_declaration* symbol. From a formal point of view, the contents of (2), (3), (4) and (5) are taken respectively from the symbol *identifier* of the first production, and from the symbols *identifier_list*, *mode* and *subtype_indication* of the last production.

⟨*entity_declaration*⟩ ::= `entity` ⟨*identifier*⟩ `is` ⟨*entity_header*⟩ ⟨*entity_declarative_part*⟩
[`begin` ⟨*entity_statement_part*⟩] `end` [`entity`] [⟨*entity_simple_name*⟩] `;`

⟨*entity_header*⟩ ::= [⟨*formal_generic_clause*⟩] [⟨*formal_port_clause*⟩]

⟨*port_clause*⟩ ::= `port (` ⟨*port_list*⟩ `) ;`

⟨*port_list*⟩ ::= ⟨*port_interface_list*⟩

⟨*interface_list*⟩ ::= ⟨*interface_element*⟩ {`;` ⟨*interface_element*⟩}

⟨*interface_element*⟩ ::= ⟨*interface_declaration*⟩

⟨*interface_declaration*⟩ ::= ⟨*interface_constant_declaration*⟩ | ⟨*interface_signal_declaration*⟩ |
⟨*interface_variable_declaration*⟩ |
⟨*interface_file_declaration*⟩

⟨*interface_signal_declaration*⟩ ::= [`signal`] ⟨*identifier_list*⟩ : [⟨*mode*⟩]
⟨*subtype_indication*⟩ [`bus`] [:= ⟨*static_expression*⟩]

Example: running the parser on the same example file used for the table ENTITIES, causes the following entries to be added to table PORTS.

PROJECT _NAME	ENTITY _NAME	PORT _NAME	MODE _MODE	PORT _TYPE	FILE_NAME	LINE	HOMO- GENEITY
Example	reg	d	IN	integer	/root/sources/ tb_01_03.vhd	16	NULL
Example	reg	en	IN	bit	/root/sources/ tb_01_03.vhd	17	NULL
Example	reg	q	OUT	integer	/root/sources/ tb_01_03.vhd	16	NULL
Example	reg	reset	IN	bit	/root/sources/ tb_01_03.vhd	17	NULL
Example	shift_adder	addend	IN	integer	/root/sources/ tb_01_03.vhd	2	NULL
Example	shift_adder	add _control	IN	bit	/root/sources/ tb_01_03.vhd	4	NULL
Example	shift_adder	augend	IN	integer	/root/sources/ tb_01_03.vhd	2	NULL
Example	shift_adder	sum	OUT	integer	/root/sources/ tb_01_03.vhd	3	NULL
Example	shift_reg	clk	IN	bit	/root/sources/ tb_01_03.vhd	30	NULL
Example	shift_reg	d	IN	integer	/root/sources/ tb_01_03.vhd	29	NULL
Example	shift_reg	load	IN	bit	/root/sources/ tb_01_03.vhd	30	NULL
Example	shift_reg	q	OUT	bit	/root/sources/ tb_01_03.vhd	29	NULL

8.5.14 Table PROCESSES

Structure for table `PROCESSES`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	PROCESS_NAME	tinytext		Primary key	
5	FILE_NAME	text			
6	START_LINE	integer			0
7	START_COLUMN	integer			0
8	END_LINE	integer			0
9	IDENTIFIER_LENGTH	tinyint(4)			0
10	HOMOGENEITY	integer	Yes		NULL
11	LINE_COUNT	integer	Yes		NULL

The VHDL parser adds an entry to this table for every process definition it encounters. An entry in this table has the following meaning: in project (1), a process of name (4) is declared inside architecture (3) of entity (2), in file (5) from line (6) to line (8), for a total number of lines equal to (11). The name of the process begins at column (7) of line (6) and is (9) characters long. If the homogeneity has been already calculated for this entity, it is equal to (10), otherwise (10) is NULL.

Note: if a label was associated with the process (i.e., the optional process label symbol is not omitted, see formal grammar production here quoted), that label will appear in (4), otherwise a unique process name will be generated like the following: `Unnamed-lineNNN`, where NNN is the same line number as (6).

⟨*process_statement*⟩ ::= [⟨*process_label*⟩ :] [`postponed`] `process` [`(` ⟨*sensitivity_list*⟩ `)`] [`is`] ⟨*process_declarative_part*⟩ `begin` ⟨*process_statement_part*⟩ `end` [`postponed`] process [⟨*process_label*⟩] `;`

Example: the following section of code, coming from file `ch_04_04.vhd`, belonging to project "example":

```
entity ch_04_04 is

end entity ch_04_04;

-------------------------------

architecture test of ch_04_04 is
begin

  process_04_1_i : process is
    -- code from book:
    type A is array (1 to 4, 31 downto 0) of boolean;
    -- end of code from book

```

```
    variable free_map : bit_vector(1 to 10) := "0011010110";
    variable count : natural;

  begin

    -- code from book (just the conditions):

    assert A'left (1) = 1;        assert A'low(1) = 1;
    assert A'right (2) = 0 ;      assert A'high(2) = 31;

    assert A'length (1) = 4;      assert A'length(2) = 32;

    assert A'ascending(1) = true;     assert A'ascending(2) = false;

    assert A'low = 1;     assert A'length = 4;

    --

    count := 0;
    for index in free_map'range loop
      if free_map(index) = '1' then
        count := count + 1;
      end if;
    end loop;

    -- end of code from book

    wait;
  end process process_04_1_i;

end architecture test;
```

causes the following tuple to be added to the table:

PROJECT_NAME	Example
ENTITY_NAME	ch_04_04
ARCHITECTURE_NAME	test
PROCESS_NAME	process_04_1_i
FILE_NAME	/root/sources/ch_04_04.vhd
START_LINE	13
START_COLUMN	3
END_LINE	49
IDENTIFIER_LENGTH	14
HOMOGENEITY	NULL
LINE_COUNT	37

8.5.15 Table PROCESS_VARIABLES

Structure for table PROCESS_VARIABLES					
	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	PROCESS_NAME	tinytext	Yes		NULL
5	VARIABLE_NAME	tinytext		Primary key	
6	VARIABLE_TYPE	varchar(64)			
7	FILE_NAME	text			
8	LINE	integer			0
9	HOMOGENEITY	integer	Yes		NULL

The VHDL parser adds an entry to this table every time the definition of a variable is encountered. An entry in this table has the following meaning: in project (1), a variable of name (5) and type (6) is defined inside process (4) in architecture (3) of entity (2), in file (7) at line (8). If the homogeneity has been already calculated for this variable, it is equal to (9), otherwise (9) is NULL.

The same section of code as before causes the following tuples to be added:

PROJECT_NAME	Example	Example
ENTITY_NAME	ch_04_04	ch_04_04
ARCHITECTURE_NAME	test	test
PROCESS_NAME	process_04_1_i	process_04_1_i
VARIABLE_NAME	count	free_map
VARIABLE_TYPE	natural	bit_vector
FILE_NAME	/root/sources/ch_04_04.vhd	/root/sources/ch_04_04.vhd
LINE	22	21
HOMOGENEITY	NULL	NULL

8.5.16 Table PROJECT_FILES

Structure for table PROJECT_FILES					
	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	FILE_NAME	text		Primary key	
3	FILE_STATUS	tinytext	Yes		NULL

This table is not used nor maintained by the parser. Instead it is used by the GUI environment for bookkeeping purposes. It indicates which projects were created and whether they have been parsed or not. An entry in this table has the following meaning: file (2) belongs to project (1); if it has already been parsed, (3) contains the string `compiled`, otherwise it is left to NULL. For example, at a given time, it could contain the following entries:

PROJECT_NAME	FILE_NAME	FILE_STATUS
Free6502	/root/sources/Free6502/testrom.vhd	compiled
Free6502	/root/sources/Free6502/testsuite.vhd	compiled
Free6502	/root/sources/Free6502/microcode.vhd	compiled
Free6502	/root/sources/Free6502/ramlib_sim.vhd	compiled
Free6502	/root/sources/Free6502/free6502.vhd	compiled
gl85	/root/sources/gl85-structural/alulogic.vhd	compiled
gl85	/root/sources/gl85-structural/alu_ctrl.vhd	compiled
gl85	/root/sources/gl85-behavioral/i8085.vhd	compiled
gl85	/root/sources/gl85-structural/acc_ctrl.vhd	compiled
gl85	/root/sources/gl85-structural/alu_8bit.vhd	compiled
Example	/root/sources/fg_01_11.vhd	NULL
Example	/root/sources/tb_01_03.vhd	compiled
Example	/root/sources/ch_04_04.vhd	compiled

8.5.17 Table SENSITIVITY_ELEMENTS

Structure for table `SENSITIVITY_ELEMENTS`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	PROCESS_NAME	tinytext		Primary key	
5	SENSITIVITY_SIGNAL_NAME	tinytext		Primary key	
6	FILE_NAME	text			
7	LINE_NUMBER	integer			0

The VHDL parser adds a row to this table for each signal appearing in a process sensitivity list. An entry in this table has the following meaning: in project (1), signal (5) appears in the sensitivity list of a process named (4), declared inside architecture (3) of entity (2). The declaration appears at line (7) of file (6).

Example: the code fragment below:

```
architecture test of ch_03_01 is

  signal en : bit := '0';
  signal data_in : integer := 0;

begin

  process_3_1_a : process (en, data_in) is

    variable stored_value : integer := 0;

  begin

    -- code from book:

    if en = '1' then
      stored_value := data_in;
    end if;
```

```
    -- end of code from book

30  end process process_3_1_a;

    stimulus : process is
    begin
      en <= '1' after 10 ns , '0' after 20 ns;
35    data_in <= 1 after 5 ns, 2 after 15 ns, 3 after 25 ns;
      wait;
    end process stimulus;

end architecture test;
```

causes the following tuples to be added to the table.

PROJECT_NAME	Example	Example
ENTITY_NAME	ch_03_01	ch_03_01
ARCHITECTURE_NAME	test	test
PROCESS_NAME	process_3_1_a	process_3_1_a
SENSITIVITY_SIGNAL_NAME	data_in	en
FILE_NAME	/root/sources/ch_03_01.vhd	/root/sources/ch_03_01.vhd
LINE_NUMBER	16	16

8.5.18 Table SIGNALS

Structure for table `SIGNALS`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	PROCESS_NAME	tinytext	Yes		NULL
5	SIGNAL_NAME	tinytext		Primary key	
6	SIGNAL_TYPE	varchar(64)			
7	FILE_NAME	text			
8	LINE	integer			0
9	HOMOGENEITY	integer	Yes		NULL

The VHDL parser adds an entry to this table every time the a signal definition is encountered. An entry in this table has the following meaning: in project (1), a signal named (5) declared with type (6) is defined in architecture (3) of entity (2) (if defined inside a process, the name of the process is (4)), in file (7) at line (8). If the homogeneity has been already calculated for this signal, it is equal to (9), otherwise (9) is NULL.

The sections of code already listed, coming from files `fg_13_05.vhd` and `fg_01_11.vhd` cause the following records to be added to the table:

PROJECT _NAME	ENTITY _NAME	ARCH. _NAME	PROCESS _NAME	SIGNAL _NAME	SIGNAL _TYPE	FILE _NAME	LINE	HOMO-GENEITY
Example	fg_13_05	test	NULL	clk	bit	/root/fg_13_05.vhd	36	NULL
Example	fg_13_05	test	NULL	clr	bit	/root/fg_13_05.vhd	36	NULL
Example	fg_13_05	test	NULL	d	bit_vector	/root/fg_13_05.vhd	37	NULL
Example	fg_13_05	test	NULL	q	bit_vector	/root/fg_13_05.vhd	37	NULL
Example	reg4	struct	NULL	int_clk	bit	/root/fg_01_11.vhd	3	NULL

8.5.19 Table SUBPROGRAM_ARGUMENTS

Structure for table SUBPROGRAM_ARGUMENTS

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	PROCESS_NAME	tinytext		Primary key	
5	SUBPROGRAM_NAME	tinytext		Primary key	
6	SUBPROGRAM_START_LINE	integer		Primary key	0
7	ARGUMENT_NAME	tinytext		Primary key	
8	ARGUMENT_MODE	varchar(10)			
9	ARGUMENT_TYPE	varchar(64)			
10	FILE_NAME	text			
11	LINE	integer			0
12	HOMOGENEITY	integer	Yes		NULL

The VHDL parser adds an entry to this table every time the a variable is declared inside a subprogram argument list. An entry in this table has the following meaning: in project (1), in a process named (4) defined in architecture (3) of entity (2) there exists a subprogram called (5), starting at line (6) of file (11). In the argument list of that subprogram there is an argument named (7), of type (9) and mode (8). That argument appears in the same file at line (11).

If the homogeneity has been already calculated for this argument, it is equal to (12), otherwise (12) is NULL.

```
entity ch_07_03 is
end entity ch_07_03;

----------------

library bv_utilities ;

architecture test of ch_07_03 is
  use bv_utilities.bv_arithmetic.all;
  constant T_delay_adder : delay_length := 10 ns;

  -- code from book:
  function bv_add ( bv1, bv2 : in bit_vector ) return bit_vector is
  begin
    -- ...
    -- not in book
    return bv1 + bv2;
    -- end not in book
  end function bv_add;

  signal source1, source2, sum : bit_vector(0 to 31);
  -- end of code from book
begin
  -- code from book:
  adder : sum <= bv_add(source1, source2) after T_delay_adder;
  -- end of code from book
  stimulus : process is
  begin
    wait for 50 ns;
    source1 <= X"00000002"; source2 <= X"00000003"; wait for 50 ns;
```

```
                                source2 <= X"FFFFFFF0"; wait for 50 ns;
    source1 <= X"00000010";                             wait for 50 ns;

    wait;
  end process stimulus;
end architecture test;
```

The above section of code causes the following two records to be added to the table.

PROJECT_NAME	Example	Example
ENTITY_NAME	ch_07_03	ch_07_03
ARCHITECTURE_NAME	test	test
PROCESS_NAME	(null)	(null)
SUBPROGRAM_NAME	bv_add	bv_add
SUBPROGRAM_START_LINE	19	19
ARGUMENT_NAME	bv1	bv2
ARGUMENT_MODE	IN	IN
ARGUMENT_TYPE	bit_vector	bit_vector
FILE_NAME	/root/ch_07_03.vhd	/root/ch_07_03.vhd
LINE	19	19
HOMOGENEITY	NULL	NULL

8.5.20 Table SUBPROGRAM_CALLS

Structure for table SUBPROGRAM_CALLS

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	PROCESS_NAME	tinytext			
5	INSIDE_SUBPROGRAM	tinytext			
6	INSIDE_SUBPROGRAM_LINE	integer			0
7	SUBPROGRAM_NAME	tinytext		Primary key	
8	FILE_NAME	text			
9	LINE	integer			0

A row is added to this table every time a subprogram is called. An entry in this table has the following meaning: in project (1), in a process named (4) defined in architecture (3) of entity (2), possibly inside the body of another subprogram named (5) and beginning at line (6) of file (8), there exists a subprogram call for the subprogram (7). That call appears in the same file at line (9).

The same code fragment as above causes the parser to add the following tuple to the table.

PROJECT_NAME	Example
ENTITY_NAME	ch_07_03
ARCHITECTURE_NAME	test
PROCESS_NAME	(null)
INSIDE_SUBPROGRAM	(null)
INSIDE_SUBPROGRAM_STARTLINE	0
SUBPROGRAM_NAME	bv_add
FILE_NAME	/root/sources/ch_07_03.vhd
LINE	7

8.5.21 Table SUBPROGRAM_DECLARATIONS

Structure for table `SUBPROGRAM_DECLARATIONS`

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	PROCESS_NAME	tinytext		Primary key	
5	INSIDE_SUBPROGRAM	tinytext			
6	INSIDE_SUBPROGRAM _STARTLINE	integer			0
7	SUBPROGRAM_NAME	tinytext		Primary key	
8	SUBPROGRAM_SYNTAX	tinytext			
9	RETURN_TYPE	tinytext	Yes		NULL
10	FILE_NAME	text			
11	START_LINE	integer		Primary key	0
12	START_COLUMN	integer			0
13	END_LINE	integer			0
14	IDENTIFIER_LENGTH	integer			0
15	HOMOGENEITY	integer	Yes		NULL
16	LINE_COUNT	integer	Yes		NULL

The VHDL parser adds an entry to this table every time a subprogram declaration is found. An entry in this table has the following meaning: in process (4) appearing to architecture (3) of entity (2) in project (1) (possibly nested in another subprogram called (5) and beginning at line(6)), occurs the declaration of a subprogram named (7), returning (if any) a value of type (9). (8) assumes the value `'PROCEDURE'` or `'FUNCTION'` depending on how the subprogram was declared. The declaration starts at line (13) of file (10), and ends at line (13), therefore lasting (16) lines.

If the homogeneity has been already calculated for this subprogram, it is equal to (15), otherwise (15) is NULL.

Example: the following section of code, coming from file `iu.vhd`, and belonging to project "Leon":

```
  function regdec(cwp, regin : std_logic_vector; fp  :  std_logic)
295     return std_logic_vector is
  variable reg   : std_logic_vector(4 downto 0);
  variable ra    : std_logic_vector(RABITS -1 downto 0);
  begin
    reg := regin; ra(4 downto 0) := reg;
300     if  ((FPTYPE = meiko) and FPEN) and (fp = '1') then
```

```
      ra(RABITS −1 downto 5) := F0ADDR(RABITS−5 downto 1);
    elsif reg(4 downto 3) = "00" then ra(RABITS −1 downto 4) := R0ADDR;
    else
−− pragma translate_off
      if not (is_x(cwp & ra(4))) then
−− pragma translate_on
        ra(NWINLOG2+3 downto 4) := (cwp + ra(4));
    if CWPOPT then ra(RABITS−1) := '0';
    elsif ra(RABITS−1 downto 4) = R0ADDR then
      ra(RABITS−1 downto 4) := (others => '0');
    end if;
−− pragma translate_off
      end if;
−− pragma translate_on
    end if;
    return(ra);
  end;
```

causes the following tuple to be added to the table:
(table layout has been reversed for convenience)

PROJECT_NAME	Leon
ENTITY_NAME	iu
ARCHITECTURE_NAME	rtl
PROCESS_NAME	decode_stage
INSIDE_SUBPROGRAM	(null)
INSIDE_SUBPROGRAM_STARTLINE	0
SUBPROGRAM_NAME	regdec
SUBPROGRAM_SYNTAX	FUNCTION
RETURN_TYPE	std_logic_vector
FILE_NAME	/root/sources/leon1-2.4.0 processor/iu.vhd
START_LINE	294
START_COLUMN	12
END_LINE	317
IDENTIFIER_LENGTH	6
HOMOGENEITY	NULL
LINE_COUNT	24

8.5.22 Table SUBPROGRAM_VARIABLES

Structure for table SUBPROGRAM_VARIABLES

	Field name	Type	Can be null	Is key	Default
1	PROJECT_NAME	text		Primary key	
2	ENTITY_NAME	tinytext		Primary key	
3	ARCHITECTURE_NAME	tinytext		Primary key	
4	PROCESS_NAME	tinytext	Yes		NULL
5	SUBPROGRAM_NAME	tinytext			
6	SUBPROGRAM_START_LINE	int(11)			0
7	VARIABLE_NAME	tinytext		Primary key	
8	VARIABLE_TYPE	varchar(64)			
9	FILE_NAME	text			
10	LINE	int(11)			0
11	HOMOGENEITY	int(11)	Yes		NULL

The VHDL parser adds an entry to this table for every subprogram variable declaration it encounters. An entry in this table has the following meaning: in project (1), there is a process named (4) declared inside an architecture (3) of entity (2). Inside this process there exists a subprogram named (5), beginning at line (6) of file (9), which declares a variable named (7), with type (8) at line (10). If the homogeneity of this variable has been already calculated, it is equal to (9), otherwise (9) is NULL.

Example: the same section of code used for the previous example above, causes the following tuples to be added to the table:

PROJECT_NAME	Leon	Leon
ENTITY_NAME	iu	iu
ARCHITECTURE_NAME	rtl	rtl
PROCESS_NAME	decode_stage	decode_stage
SUBPROGRAM_NAME	regdec	regdec
SUBPROGRAM_START_LINE	293	293
VARIABLE_NAME	ra	reg
VARIABLE_TYPE	std_logic_vector	std_logic_vector
FILE_NAME	/root/sources/leon1-2.4.0 processor/iu.vhd	/root/sources/leon1-2.4.0 processor/iu.vhd
LINE	296	295
HOMOGENEITY	NULL	NULL

Chapter 9

Installation instructions

The following installation instructions describe how to set up an environment in which the tools developed for this thesis can run smoothly. The described steps are designed to be executed on a i386 Linux box running Red Hat Linux version 7.2; they should work as well on any distribution of Linux which supports RPM installation packages and a System V-style initialization mechanism. If you are in different conditions (you are running a different Linux distribution, you have no RPM package support, you are not running Linux at all, you are not running a *n*x flavor at all, etc...), please find the same packages that are here mentioned in a format that is suitable to be used on your operating system and install them according to the instructions that came with your OS, then find out yourself how to build, configure and install them.

The core package of the tools developed for this thesis is available on the world wide web at the following address: `http://www.scarpaz.com/vhdlthesis` Simply connect to the above address to find out which is the latest release of the tools and to download them. Then follow the steps below:

- Step 1: mySql is a simple SQL database engine used by the tools to implement the database described in chapter 8. Find mySql distribution files, either on CD2 of Red Hat CDs or searching the Internet; then install them by issuing the following commands:

  ```
  rpm -i mysql-3.23.41-1.i386.rpm
  rpm -i mysql-devel-3.23.41-1.i386.rpm
  rpm -i mysql-server-3.23.41-1.i386.rpm
  ```

- Step 2: start mySql server for the current session by issuing the following command:

  ```
  /etc/rc.d/init.d/mysqld start
  ```

  Please note that the above causes mySql server to keep running until the next reboot. In order to make mySql persistent, read on the following step.

- Step 3: in order to have mySql server automatically started at boot time in the sessions following the current one, enable launch of the `mysqld` service in runlevel 5. This can be done in Red Hat Linux by using the `serviceconf` graphic utility, or by manually creating an appropriate symbolic link to file `/etc/init.d/mysqld` in directory `/etc/rc5.d`;

- Step 4: mySql++ is a set of C++ API classes, that come in very handy when interfacing with mySql server in programs written in C++. Our VHDL parser uses mySql++ in order to store project information in the SQL database described in chapter 8. You have to find mySql++ distribution files by searching the Internet; At the time I'm writing, mySql++ is not available as an RPM packages and it must be downloaded in tar-gzipped format, then compiled and installed. This can be done by copying your `.tar.gz` file in an appropriate, empty directory and then issuing the following commands:

```
tar xvzf mysql++-1.7.9.tar.gz
cd mysql++-1.7.9
configure
sh build.sh
make install
```

- Step 5: build results are now available at the following path: `/usr/local/lib`. Since this path is not searched by the dynamic link library loader, move the libraries from here to either `/usr/lib` or `/usr/lib/mysql`. On other *n*x flavors either add this path to your library path by modifying the `LIBPATH` environment variable or file `/etc/ld.so.conf`.

- Step 6: in order to perform model identification tasks, the Vhdl GUI Tool requires the presence of a MATLAB-compatible interpreter, such as `octave`. In order to install `octave`, find its distribution files, either on CD2 of Red Hat CDs or searching the Internet. Note that `octave` usually requires gnuplot, liblapack and libblas installed on your computer. Just find the RPM packages listed below and issue the following commands:

```
rpm -i blas-3.0-12.i386.rpm
rpm -i lapack-3.0-12.i386.rpm
rpm -i gnuplot-3.7.1-13.i386.rpm
rpm -i octave-2.1.34-3.i386.rpm
```

- Step 7: now you need to unpack the thesis tools. This can be done by simply locating the file you downloaded from our website and unpack it by issuing:

```
tar xvzf vhdl-tools-2002xxxx.tar.gz
```

- Step 8: this step consists in building the VHDL parser (the other tools are written in Tcl/Tk and do not need to be compiled). In order to start build, simply type:

```
make
```

- Step 9: if the previous phase completed successfully, you can safely start the GUI tool, by simply typing:

```
./vhdl_gui
```

  You do not need to perform any database initialization; the first time the tool is run, all table creation operations are automatically performed by the VHDL parser.

Part III

The Data

Chapter 10

Tuning project base

This thesis is deeply rooted in the analysis of existing, real-life VHDL projects. A large number of VHDL source files has been gathered from world-wide public domain resources. We tried to build a project base as heterogeneous as possible, with projects of all the sizes, thus to avoid biasing of the resulting estimation.

Project summary	
Number of projects:	41
Number of VHDL files:	573
Number of VHDL lines:	388,790
Cumulative size:	16.5M
Number of projects:	41
Number of entities:	945
Number of architectures:	967
Number of component declarations:	952
Number of component instantiations:	46,653
Number of subprogram declarations:	587
- of which: functions:	386
- of which: procedures:	201
Number of ports:	9,276
Number of signals:	58,836
Number of variables in processes:	1,747
Number of variables in subprograms:	387

For each project we indicate the author, a brief description (usually an excerpt from the original documentation provided by the author), the project size (expressed in number of files and in kilobytes) and the archive name under which the project is redistributed (for reference and verification purposes only) at the website of this thesis.

When the original documentation includes a block diagram or an equivalent representation, capable of giving a rough idea of the project complexity, that diagram is here given.

File count and size in kilobytes are related with the only VHDL source code files, possibly including test-benches and similar files, but excluding any non-VHDL resources, such as documentation, test vectors, waveforms, waveform generators and so on.

The following table illustrates the projects distribution on the basis of their size. Starting from the size of the smallest project (8k), 10 categories have been prepared in geometric

progression and projects were classified on the basis of their size. The following table gives the number of projects for each category and the cumulative size of all the projects belonging to that category.

Project size	Count	Cumulative size
8–16k	2	18k
16–32k	3	71k
32–64k	4	208k
64–128k	9	828k
128–256k	9	1824k
256–512k	7	2264k
512–1024k	5	3334k
1024–2048k	1	1409k
2048–4096k	0	0k
4096–8192k	1	6493k

The same data are represented in figure 10.1 and 10.2. As you can see, the highest number of projects is around 128k in size, but most of the VHDL code is contained in one single large project (namely, the LEON processor, approximately 6.5M).

Figure 10.1: Project distribution by size

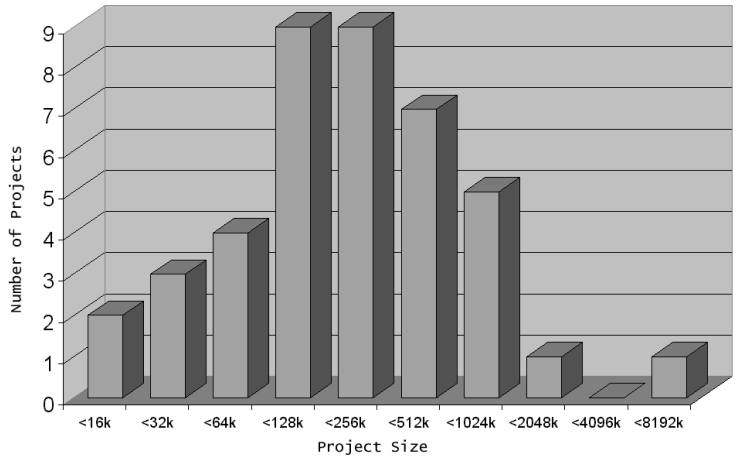
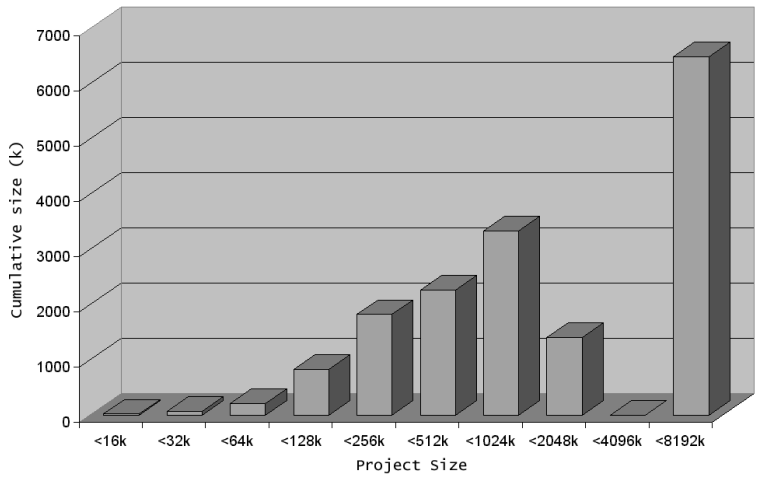


Figure 10.2: Cumulative code amount distribution per project size

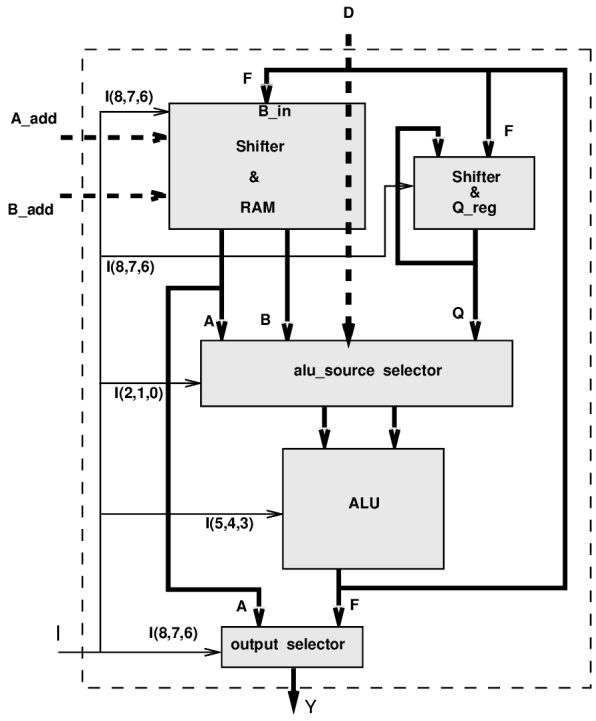


10.1 AMD Am2901

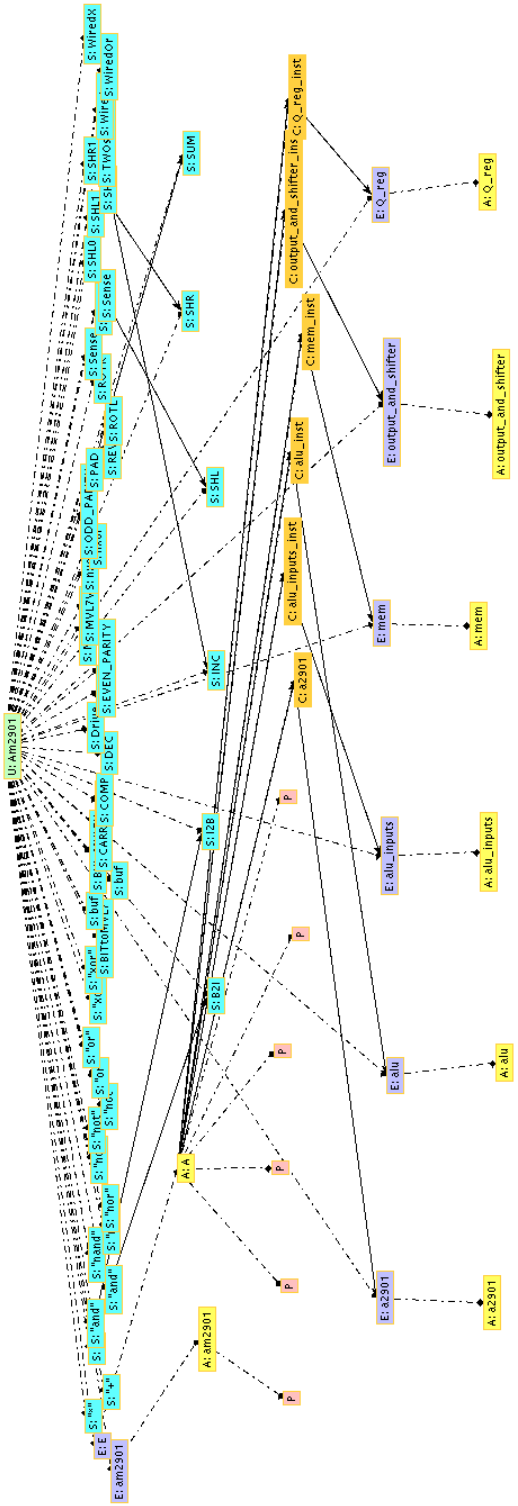
Project information	
Project name	AMD Am2901
Author	Indraneel Ghosh, Champaka Ramachandran, University of California, Irvine
Project Size	17 files, 245k
Archive	2901.tar.gz

The Am 2901 four-bit microprocessor slice (from Advanced Micro Devices Inc.), a high-speed cascadable ALU intended for use in CPUs, peripheral controllers and programmable microprocessors.

Block diagram from original documentation:



Syntax object graph:

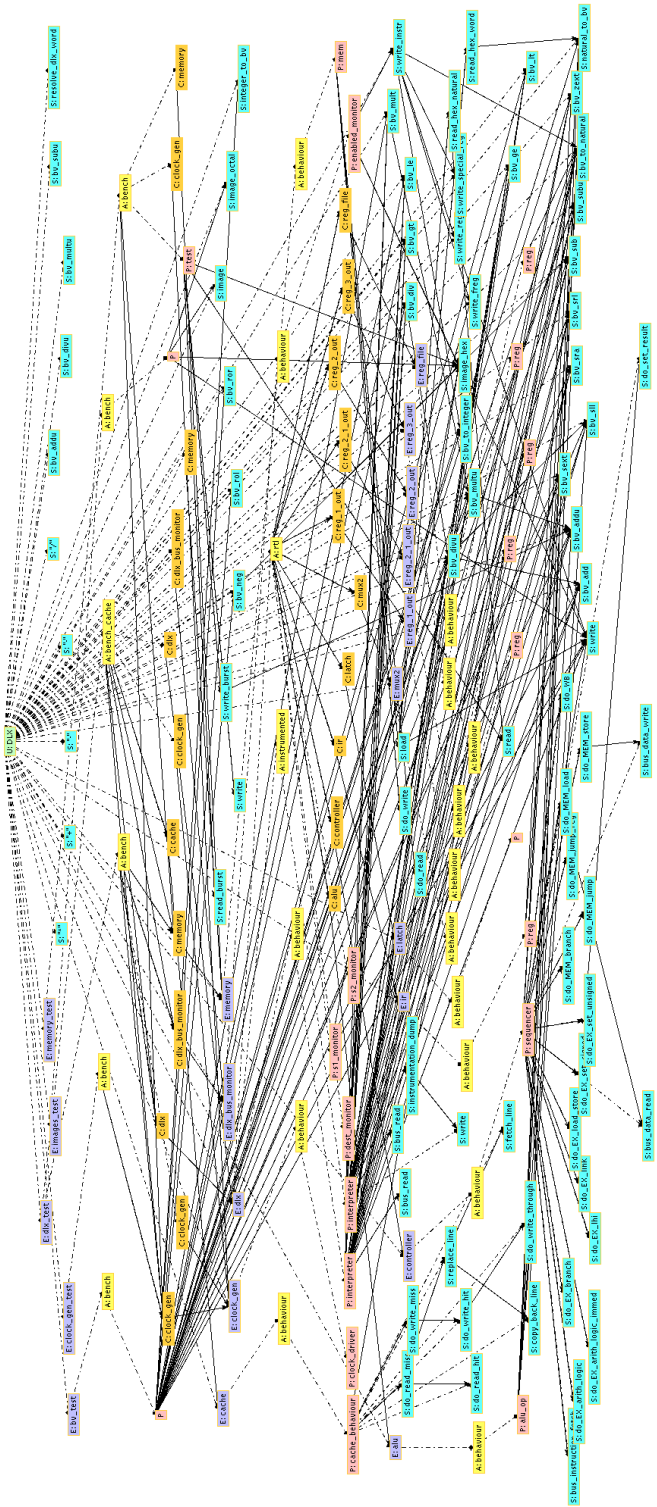

U: Am2901
S: WiredX
S: WiredOr
S: SHR1
S: SHL1
S: TWOS
S: SUM
S: SHR
S: Sense
S: SHL0
S: ROTL
S: PAD
S: SHL
S: MVLZ
S: ODD_PAR
S: EVEN_PARITY
S: INC
S: DEC
S: Drive
S: COMP
S: CARR
S: T2B
S: B2I
S: buf
S: BITTo
S: "xnor"
S: "xor"
S: "or"
S: "nor"
S: "nand"
S: "and"
S: "0"
S: "1"
S: "a"
S: E
E: am2901
A: am2901
A: A
P
C: output_and_shifter_ins
C: Q_reg_inst
C: mem_inst
C: alu_inst
C: alu_inputs_inst
C: a2901
E: Q_reg
E: output_and_shifter
E: mem
E: alu_inputs
E: alu
E: a2901
A: Q_reg
A: output_and_shifter
A: mem
A: alu_inputs
A: alu
A: a2901

10.2 DLX Microprocessor

Project information	
Project name	DLX Microprocessor
Author	Peter J. Ashenden, University of Adelaide, Australia
Project Size	59 files, 310k
Archive	dlx.tar.gz

A behavioral description of the well-known DLX example microprocessor;

Syntax object graph:



10.3 Superscalar DLX Microprocessor

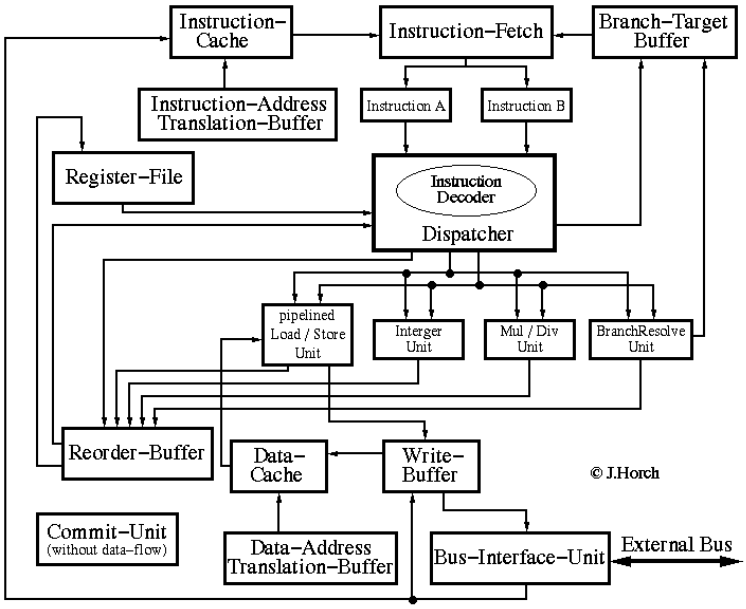
Project information	
Project name	Superscalar DLX Microprocessor
Author	Joachim Horch, Technical University of Darmstadt, Germany
Project Size	4 files, 231k
Archive	SuperscalarDlx.zip

From original documentation: VHDL design of the DLX processor described in "Hennessy, Patterson; Computer Architecture: A Quantitative Approach".

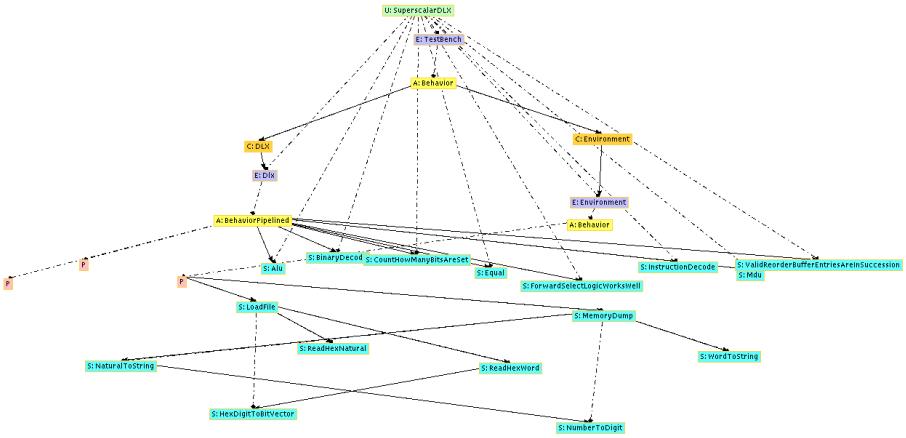
Superscalar DLX Features:

- Pipelined, superscalar
  - Two instructions per clock,
  - Branch-Target buffer
  - Reorder-Buffer to commit instructions in program order
  - Precise exception processing
- Four execution units with Reservation-Station
  - Branch-Resolve unit
  - Arithmetic-Logic unit
  - Multiply-Divide unit
  - Load-Store unit
- Write-Buffer
- 64 byte Instruction-Cache
- 64 byte Data-Cache
- 4 entry Instruction-Address-Translation-Buffer, page size: 128 byte
- 4 entry Data-Address-Translation-Buffer, page size: 128 byte

Block diagram:



Syntax object graph:



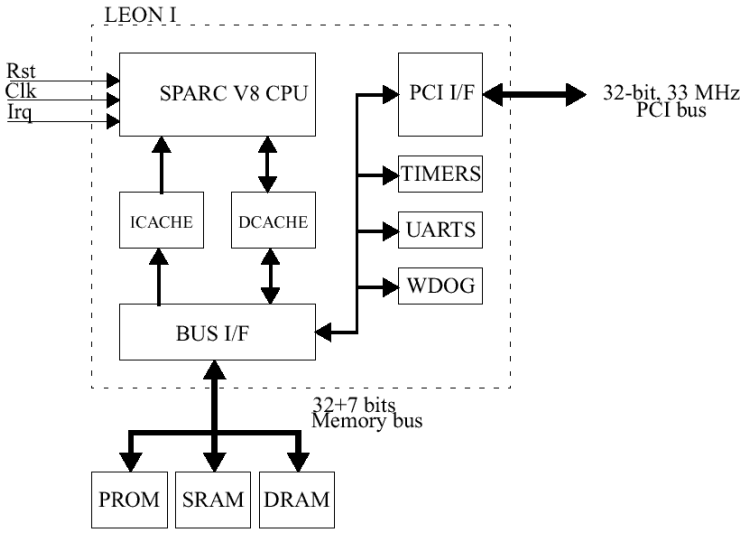
10.4 LEON 1 Microprocessor

Project information	
Project name	LEON 1 Microprocessor
Author	Jiri Gaisler, ESA/ETEC
Project Size	64 files, 6493k
Archive	leon1-2.4.0.tar.gz

From original documentation: 32-bit processor with SPARC V8 architecture. On-chip features: separate instruction and data caches, hardware multiplier and divider, interrupt

controller, two 24-bit timers, two UARTs, power-down, watchdog, 16-bit I/O port and memory controller.

Block diagram:



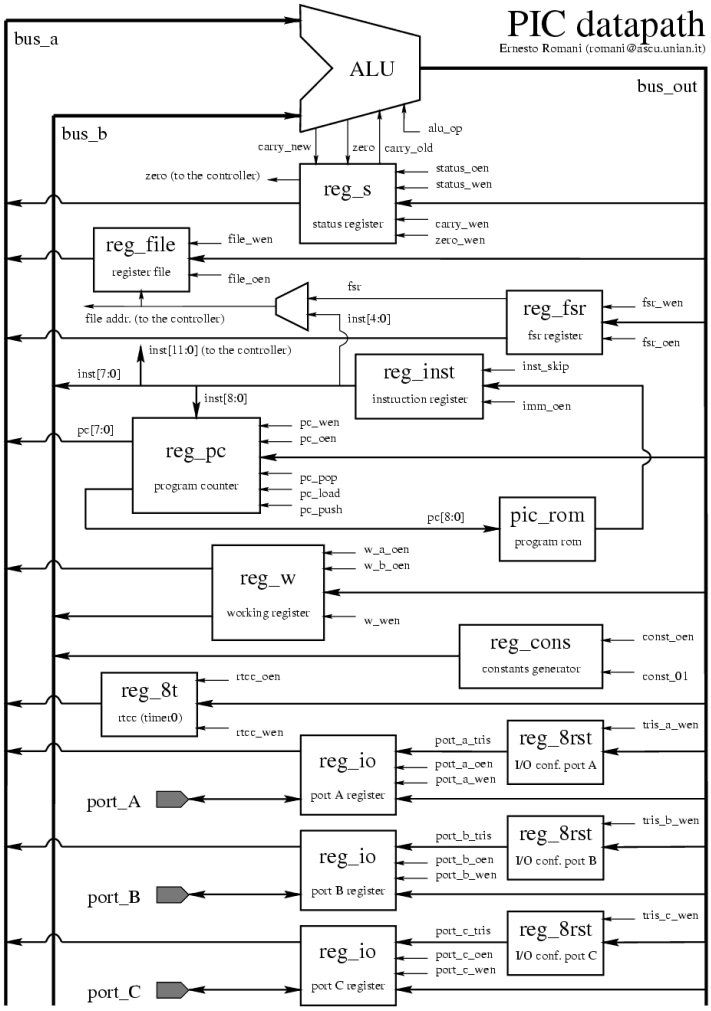
The syntax object for this project was not reported since too extended to fit on a single page.

10.5 PIC-16C5X microcontroller

Project information	
Project name	PIC-16C5X microcontroller
Author	Ernesto Romani
Project Size	17 files, 65k
Archive	pic16C5x.tar.gz

From original documentation: This package contains a synthesizable VHDL description of the Microchip's PIC-16C5X microcontroller. The PIC consists of two main components: the control unit (controller) and the data processing unit (datapath). The control unit is responsible for the interpretation of instructions and for the generation of the control signals required to execute a certain instruction. The data processing part performs the actions specified by the control unit. It consists of units which elaborate and/or store information, and paths (busses) which trasport information between units.

Block diagram (datapath):

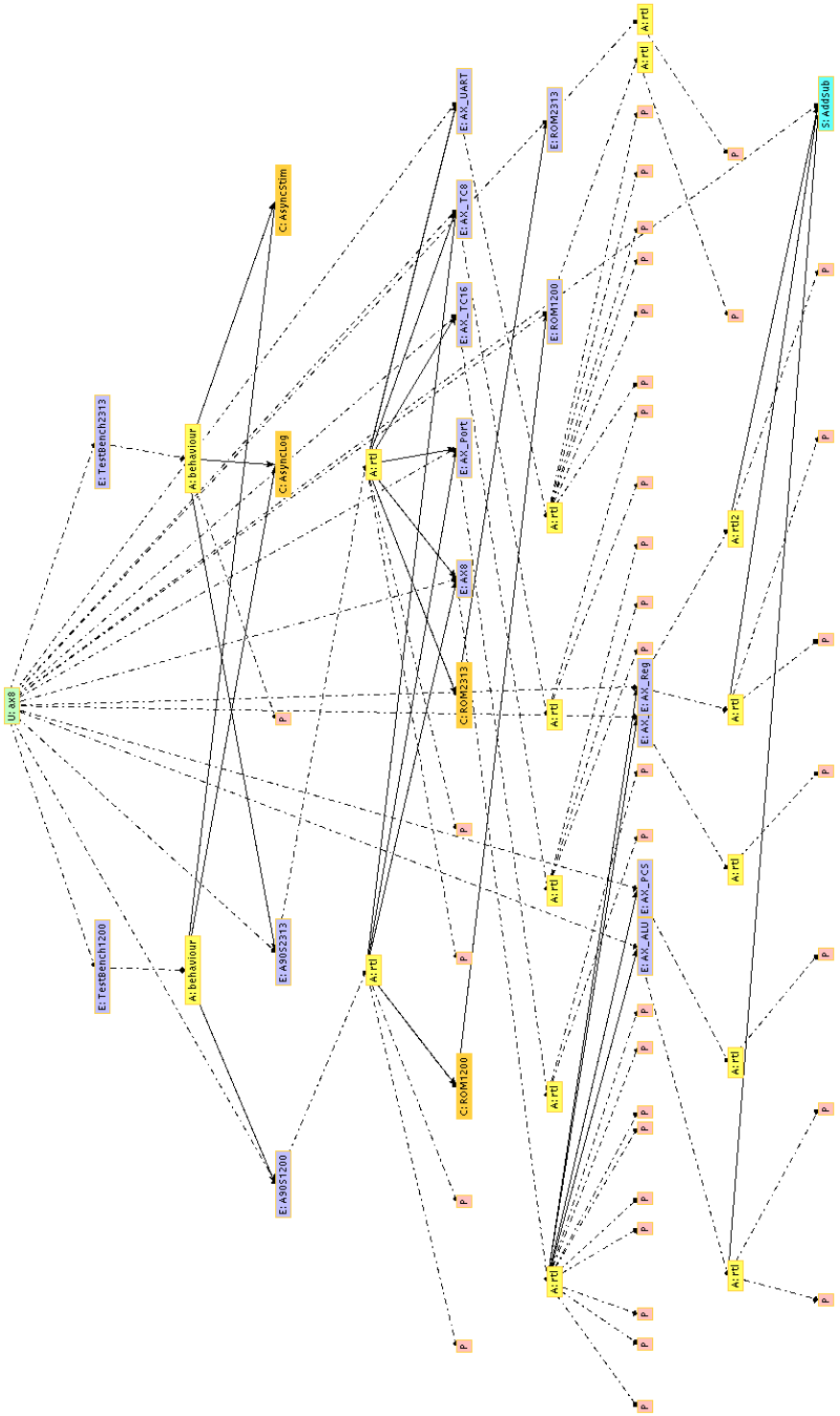


10.6 AX8

Project information	
Project name	AX8
Author	Daniel Wallner
Project Size	17 files, 157 k
Archive	ax8_0146.zip

AX8 is an Atmel 90S1200/90S2313 compatible microcontroller.

Syntax object graph:


U: AX8
E: TestBench2313
A: behaviour
C: AsyncLog
C: AsyncStim
A: rtl
E: AX_UART
E: AX_TC8
E: AX_TC16
E: AX_Port
E: AX8
C: ROM2313
E: ROM2313
E: ROM1200
E: TestBench1200
E: A90S2313
E: A90S1200
C: ROM1200
E: AX_E: AX_Reg
E: AX_PCS
E: AX_ALU
A: rtl2
S: AddSub

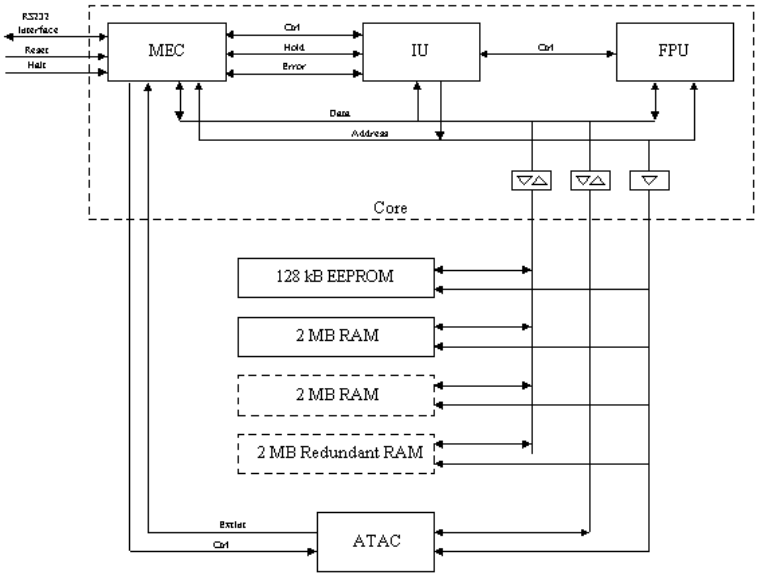
10.7 ERC32

Project information	
Project name	ERC32
Author	ESA/ESTEC
Project Size	8 files, 1409k
Archive	erc32vhdl-1.0.tar.gz

A Radiation-tolerant SPARC V7 processor developed for space applications. Fully functional and timing accurate models of integer unit, floating-point unit and memory controller. Manufactured by Temic/MHS on a 0.8 um CMOS/EPI radiation-tolerant technology.

The syntax object for this project was not reported since too extended to fit on a single page.

Block diagram :



10.8 Free6502

Project information	
Project name	Free6502
Author	The Free-IP project, www.free-ip.com
Project Size	5 files, 425k
Archive	Free6502_v07.zip

From original documentation: The Free-6502 core is a 6502 compatible CPU core. Basic features are:

- 6502 binary code compatible.
- Written in 100% VHDL.

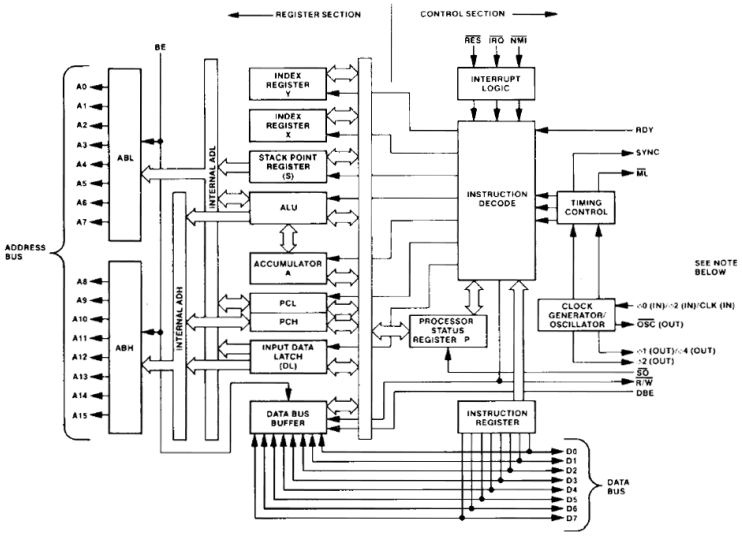
- Simple synchronous 8-bit bus interface.

- Entirely synchronous design.

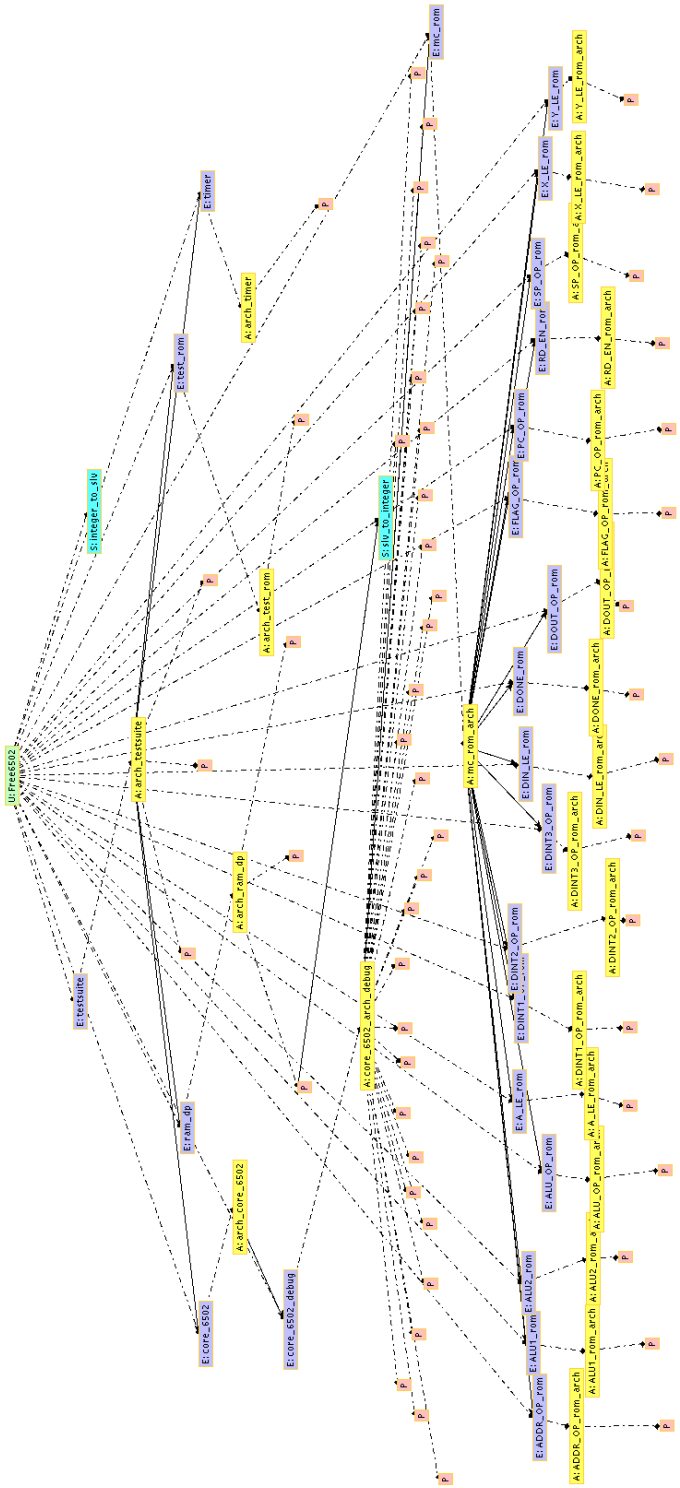
- Registered I/O for simple porting and integration.

- Supports all standard 6502 instructions.

Block diagram :



Syntax object graph:

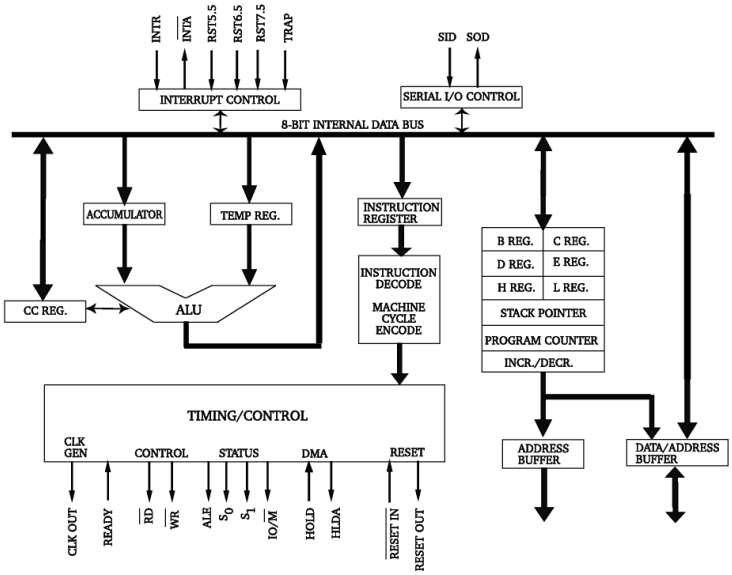

U:FreeS502
E:testsuite
A:arch_testsuite
E:timer
A:arch_timer
E:test_rom
A:arch_test_rom
E:ram_dp
A:arch_ram_dp
E:core_6502
A:arch_core_6502
E:core_6502_debug
S:integer_to_slv
S:slv_to_integer
A:core_6502_arch_debug
E:imc_rom
A:imc_rom_arch
E:Y_LE_rom
A:Y_LE_rom_arch
E:X_LE_rom
A:X_LE_rom_arch
E:SP_OP_rom
A:SP_OP_rom_a
E:RD_EN_rom
A:RD_EN_rom_arch
E:PC_OP_rom
A:PC_OP_rom_arch
E:FLAG_OP_rom
A:FLAG_OP_rom_arch
E:DOUT_OP_rom
A:DOUT_OP_rom_arch
E:DONE_rom
A:DONE_rom_arch
E:DIN_LE_rom
A:DIN_LE_rom_ar
E:DINT3_OP_rom
A:DINT3_OP_rom_arch
E:DINT2_OP_rom
A:DINT2_OP_rom_arch
E:DINT1_OP_rom
A:DINT1_OP_rom_arch
E:A_LE_rom
A:A_LE_rom_arch
E:ALU_OP_rom
A:ALU_OP_rom_ar...
E:ALU2_rom
A:ALU2_rom_a
E:ALU1_rom
A:ALU1_rom_arch
E:ADDR_OP_rom
A:ADDR_OP_rom_arch

10.9 GL85

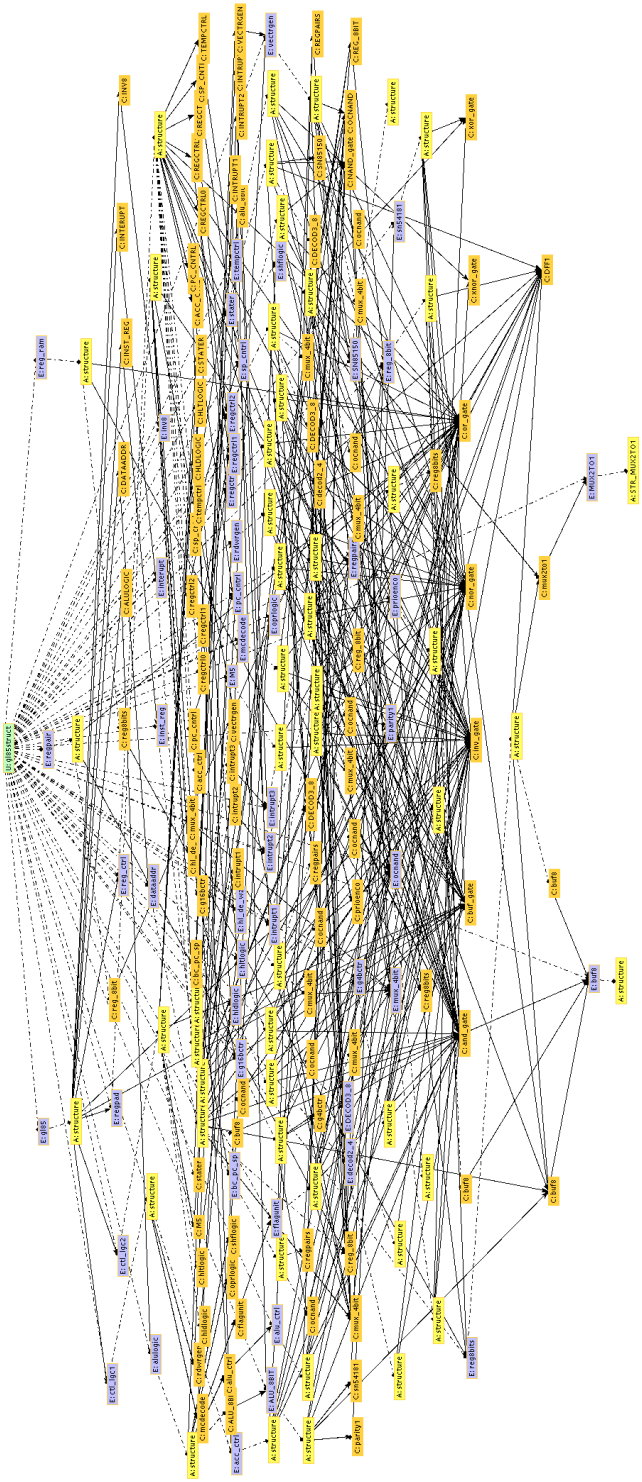
Project information	
Project name	GL85
Author	Attest Software / F. Gail Gray and Jim Armstrong, Virginia Tech
Project Size	53 files, 248k
Archive	gl85.tar.gz

From original documentation: Structural and behavioral level models of a circuit called GL85. This circuit is an op-code clone of the 8085 microprocessor (however, it is not pin compatible), a quite popular processor in the late 70's and early 80's, which also served as a subset of the Z80 microprocessor that is still in use. The behavior has about 1800 source lines, and the structure has 3000 logic gates and flip-flops. It has been extensively simulated and all of the op-codes checked out with a design verification set of 2277 vectors.

Block diagram :



Syntax object graph:

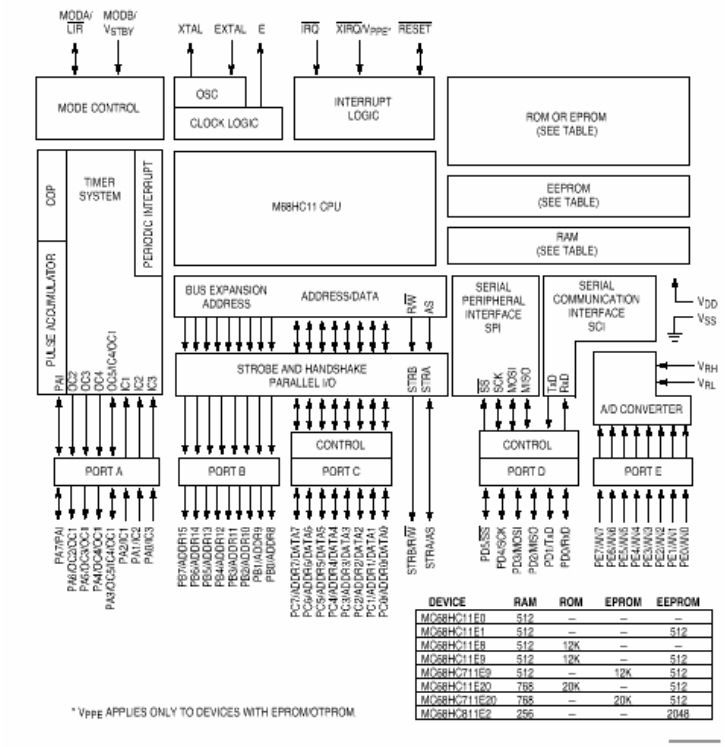


10.10 HC11

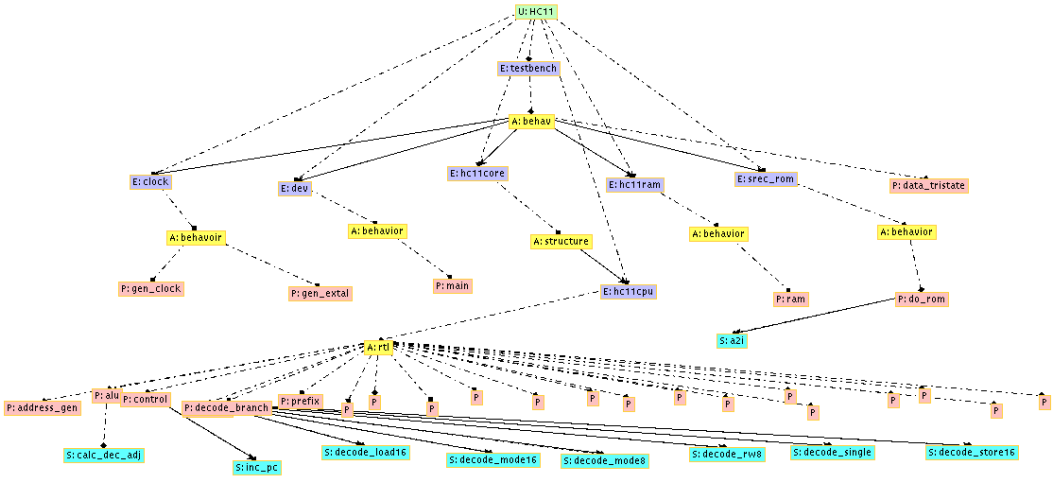
Project information	
Project name	HC11
Author	Scott Thibault, Green Mountain Computing Systems
Project Size	7 files, 87 k
Archive	hc11core.zip

From original documentation: A fully-synthesizable VHDL model of the HC11 CPU, several testbench files, and a simple HC11 debugger. The CPU implements all instructions except the two divide instructions (note: although the DAA instruction is implemented, it is commented out due to its effect on performance).

Block diagram (represents original 68HC11, not this HC11 implementation):



Syntax object graph:



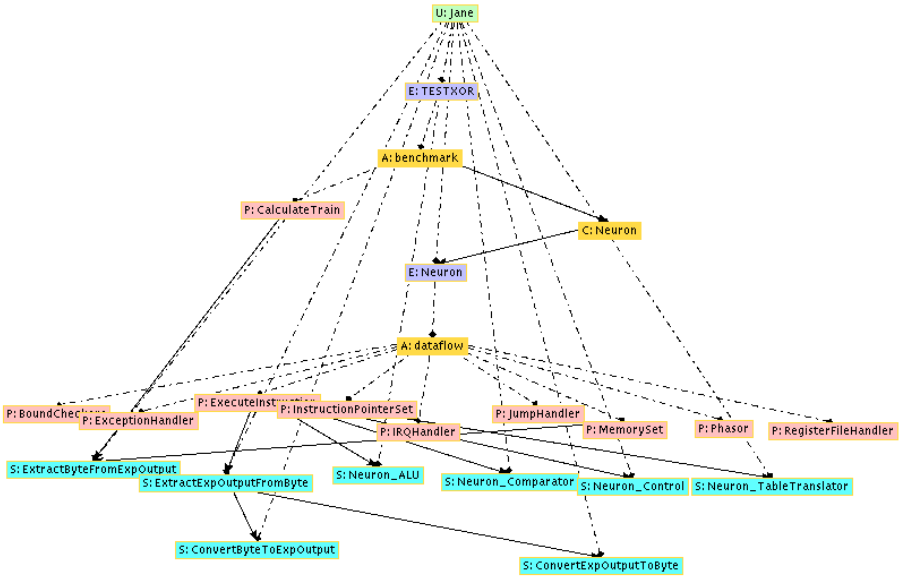
10.11 JANE

Project information	
Project name	JANE
Author	Suresh Kumar Devanathan
Project Size	3 files, 25k
Archive	Jane.zip

From original documentation: JANE is a 4-bit programmable neural network microprocessor, which could be customized to emulate any type of neuron.

- MIMD design
- 8-bit opcode / microcode
- parallel scalability
- 4-bit signed fixed point binary number support
- support for any type of neural net algorithm
- SISC design
- 5 IRQ devices/exceptions

Syntax object graph:

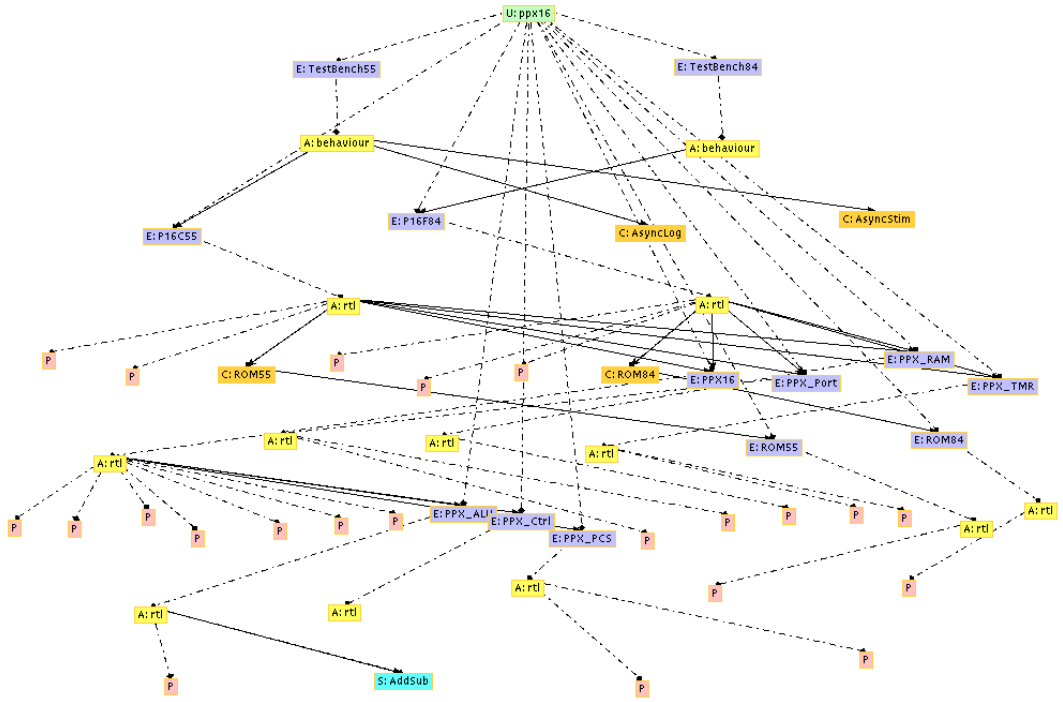


10.12 PIC16xx

Project information	
Project name	PIC16xx
Author	Daniel Wallner
Project Size	14 files, 106 k
Archive	ppx16_0146.zip

From original documentation: PIC16xx compatible microcontroller core.

Syntax object graph:

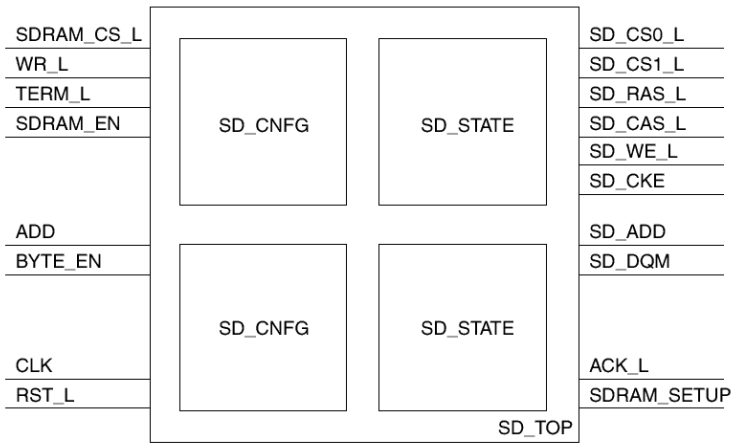


10.13 SDRAM controller

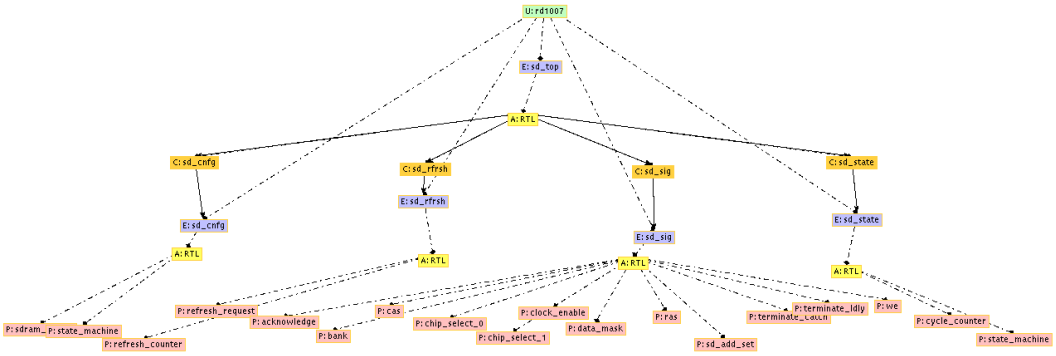
Project information	
Project name	SDRAM controller
Author	Lattice Semiconductor
Project Size	5 files, 39k
Archive	rd_1007_vhdl.zip

From original documentation: This SDRAM Controller is designed to interface to standard microprocessors. The controller is independent of processor type. This design, as implemented, supports two 16MB memory regions configured as 4 M x 32 bits. Each region consists of two Micron MT48LC4M16A2 devices.

Block diagram :



Syntax object graph:

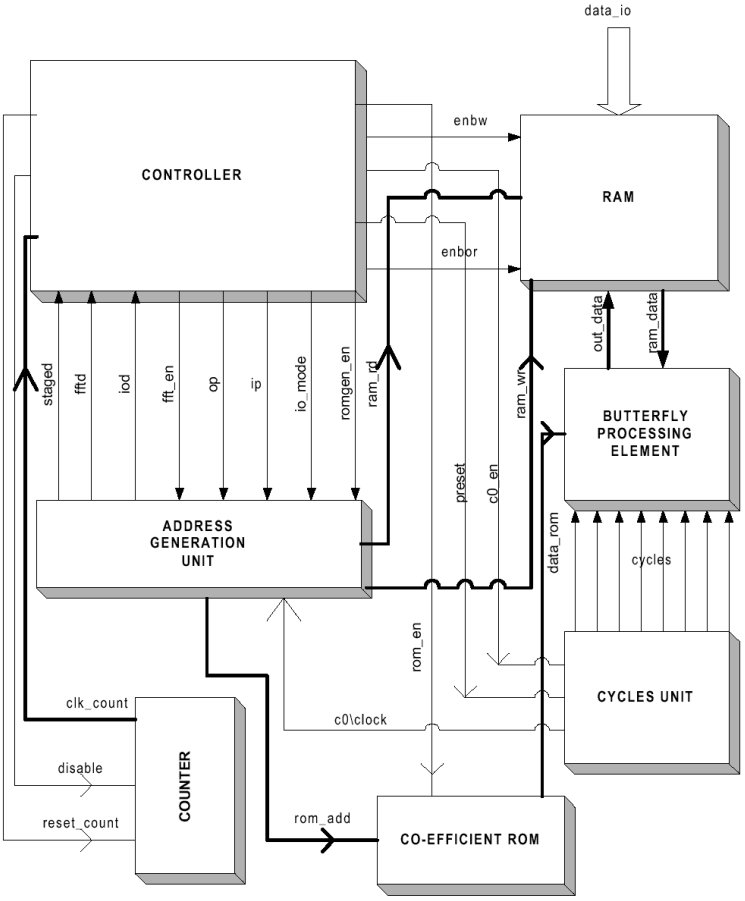


10.14 FFT processor

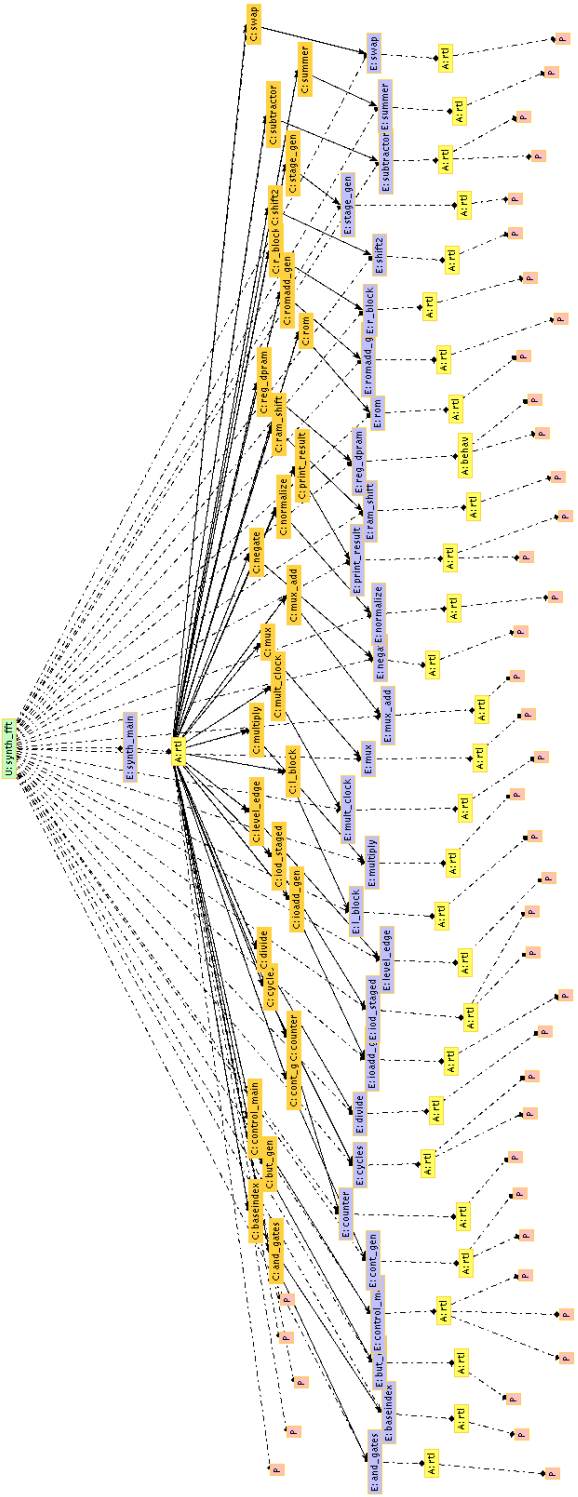
Project information	
Project name	FFT processor
Author	Ray Ranjan Varghese, Chanjal G.Tharayl, University of Kottayam, India
Project Size	34 files, 68k
Archive	synth_fft.zip

From original documentation: 8-point FFT processor based on IEEE workshop report.

Block diagram :



Syntax object graph:

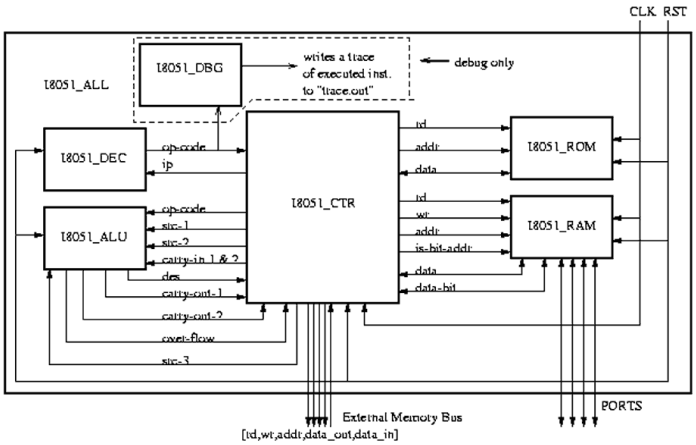


10.15 i8051

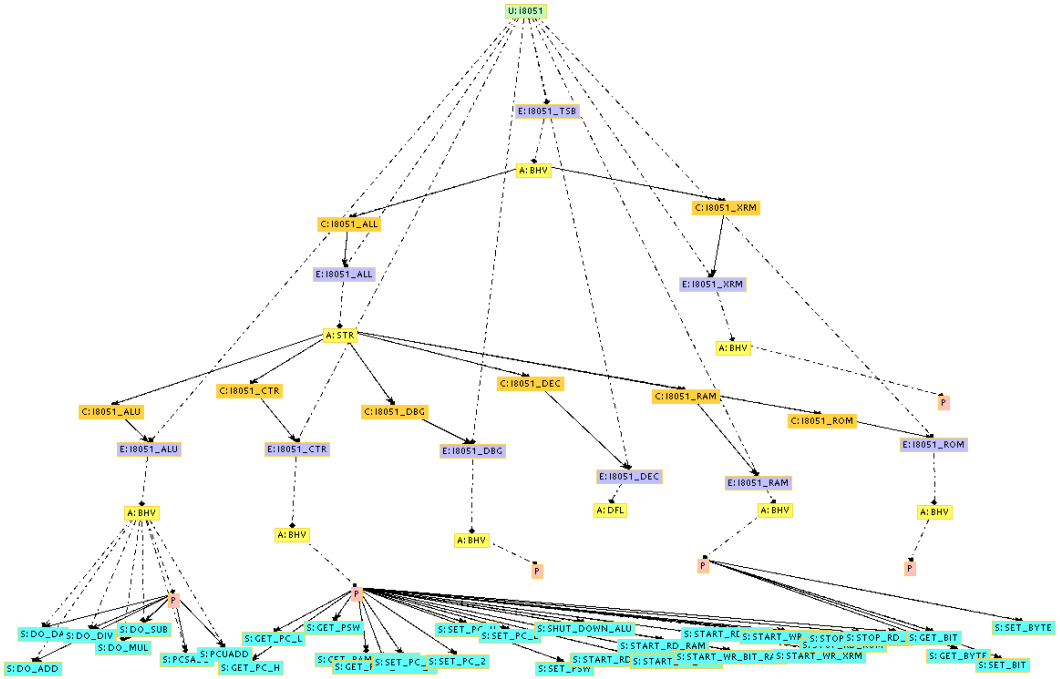
Project information	
Project name	i8051
Author	Tony Givargis, University of California at Riverside
Project Size	10 files, 301k
Archive	i8051_source_2.8.tar.gz

From original documentation: Synthesizable VHDL Model of Intel 8051, a 8-bit microcontroller with 64K of program addressing space, 64K of data memory. The implementation is 100% instruction compatible.

Block diagram :



Syntax object graph:

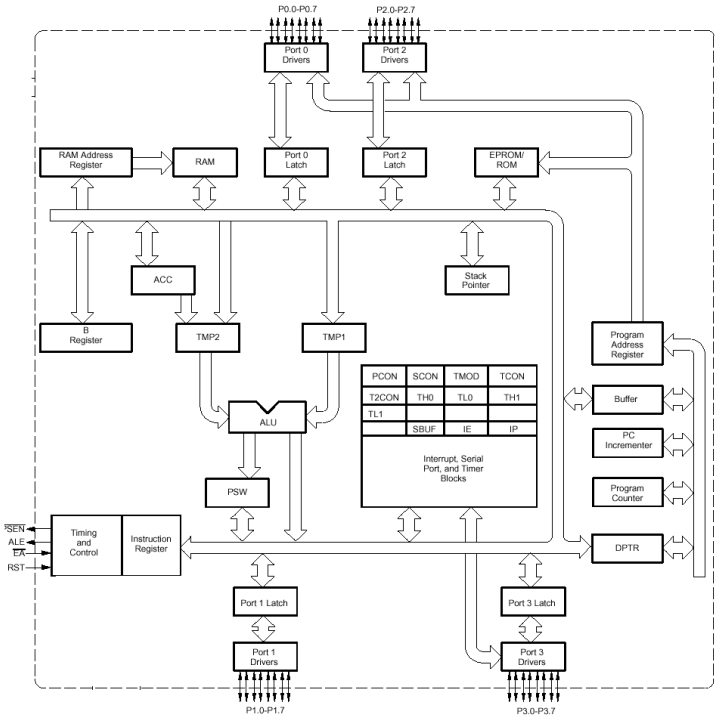


10.16 TE51

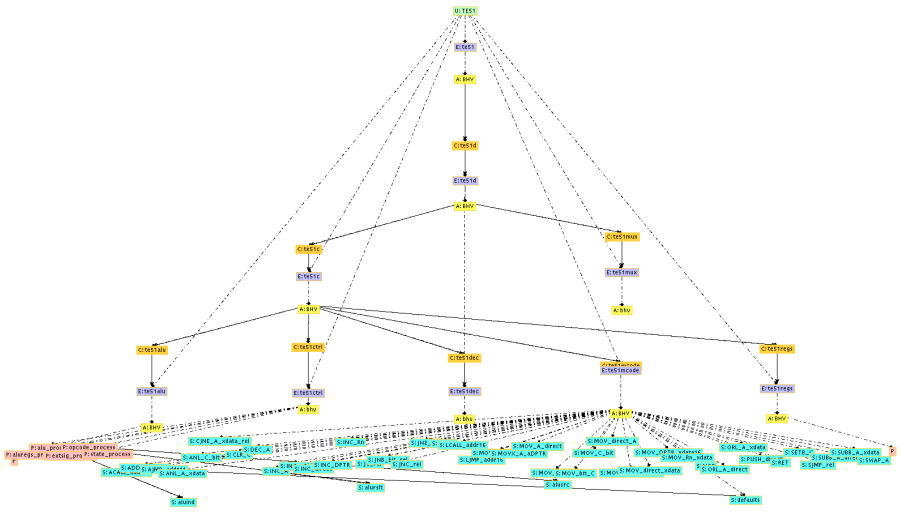
Project information	
Project name	TE51
Author	Trenz electronic
Project Size	1 file, 120k
Archive	ps-te51.zip

From original documentation: The TE51 microcontroller core is cycle compatible to MCS 51, on-chip 128 byte data RAM, minimum gate count, optimized for FPGA implementation.

Block diagram :



Syntax object graph:



10.17 Spartan-II

Project information	
Project name	Spartan-II
Author	Trenz electronic
Project Size	8 files, 77k
Archive	an-XC2S-XR16.zip

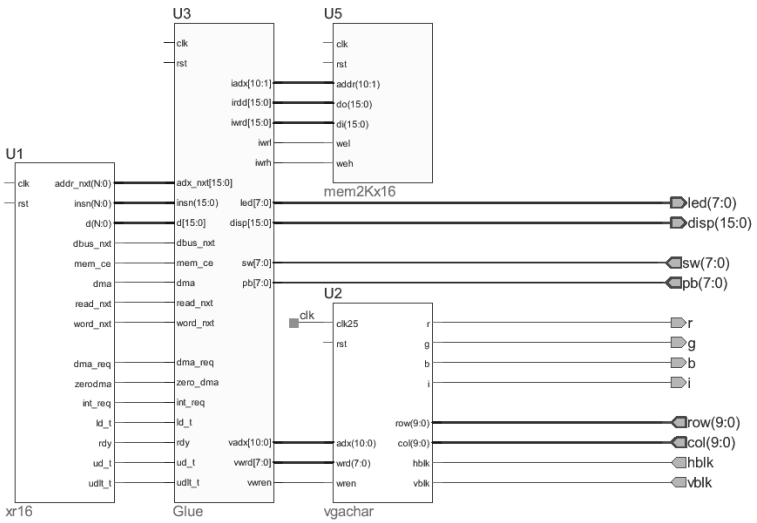
From original documentation: The Spartan-II Development System is a System-on-chip designed to provide a simple yet powerful platform for FPGA development, based on a XR16 CPU. The XR16s core features:

- pipelined RISC with sixteen 16bit registers and 16bit instructions;
- 3 stage pipeline (fetch, decode, execute);
- approximately 1.4 cycles per instruction;
- 64kB address range;
- integrated DMA engine;
- interrupt handling;
- very efficient FPGA implementation;
- C-compiler available;

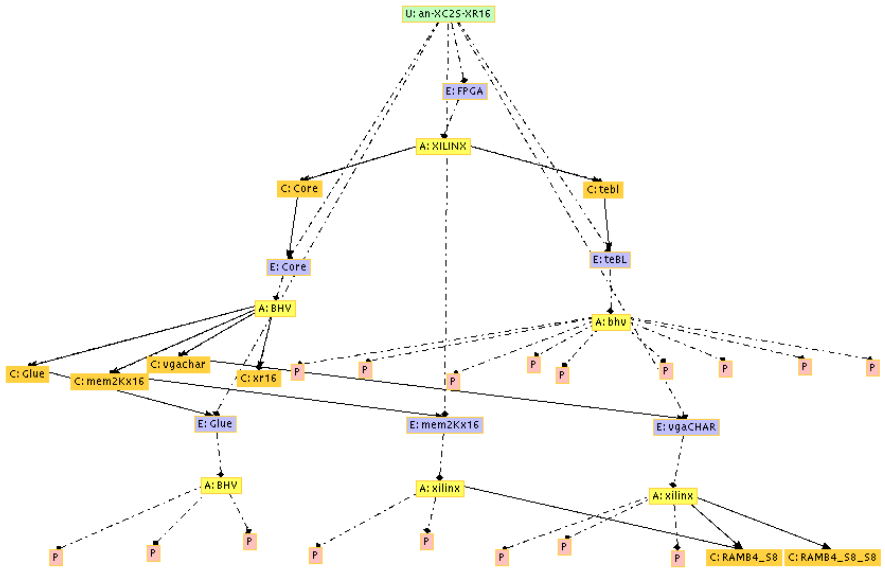
Based on the XR16 CPU, this application note implements a System-on-Chip with the following features:

- 16bit RISC CPU;
- 4kB of firmware ROM/ general purpose RAM;
- VGA text display with 64x32 characters;
- ASCII character generator;

Block diagram:



Syntax object graph:



10.18 System-on-Chip with USB support

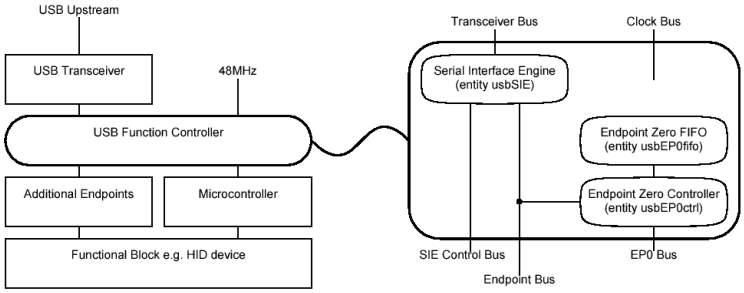
Project information	
Project name	System-on-Chip with USB support
Author	Trenz electronic
Project Size	4 files, 24k
Archive	an-XC2S-USB.zip

A System-on-chip with a full-speed USB function controller.

- Fully compliant to USB 1.1 specification:
  - Full-Speed (12Mbps) operation;
  - Support for 4 Endpoints, including up to 3 user-configurable endpoints;
  - Supports Bulk, Interrupt or Isochronous data transfers;
- Hardwired USB protocol layer
  - No firmware intervention required
  - Up to 10Mbps bandwidth
- Very compact design
  - On-chip digital PLL
  - On-chip Endpoint FIFOs
  - Minimum gate count
  - Optimized for FPGA implementation

- Lowest possible design risk
  - Free behavioral model
  - Comprehensive reference application
  - USB Packet-oriented testbench
  - Synthesizable VHDL model
- Very low cost.

Block diagram:

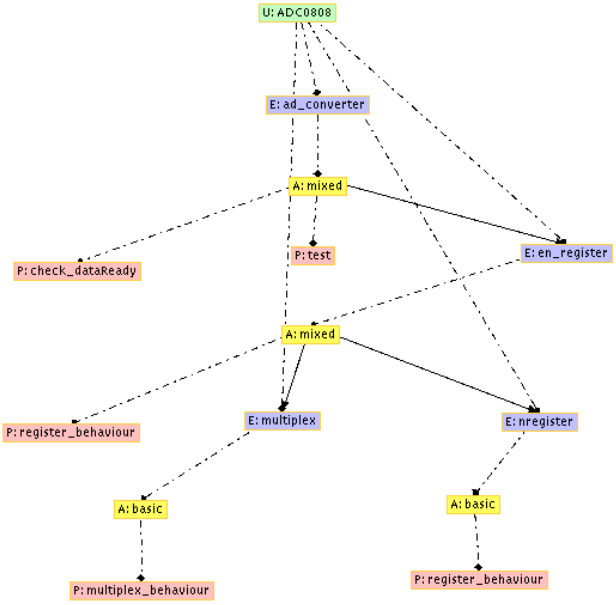


10.19 ADC0808 interface

Project information	
Project name	ADC0808 interface
Author	Shauna Rae, University of Alberta, Canada
Project Size	4 files, 10k
Archive	ADC0808_interface.zip

From original documentation: Interface circuitry for ADC0808, to be implemented on an Altera 10k20RC240-4 FPGA. The ADC0808/ADC0809 is an 8-bit ADC that also contains an 8 channel multiplexer.

Syntax object graph;



10.20 PIC16c5x

Project information	
Project name	PIC16c5x
Author	AnsLab
Project Size	30 files, 131k
Archive	Ans_risc8.zip

From original documentation: VHDL design of MicroChips PIC16c5x, an 8-Bit, ROM Based micro controller with RISC Architecture. Features:

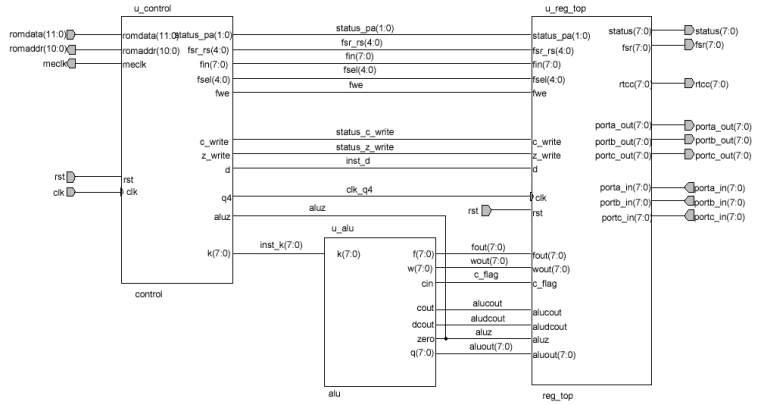
- Functionally compatible with PIC16C5x family;
- 31 available single-word instructions;
- 12-bits wide instructions;
- Up to 2K words internal program memory (Synchronous Output Data ROM);
- Up to 72Bytes(4 Bank of 16 Register + 8 GPR on Bank 0) internal data memory;
- 8-bit wide data;
- Three pair of 8-bit IO Port;
- 6 Special Function Registers;
- Two level deep stack;
- Direct, Indirect and relative addressing modes for data and instructions;

- WDT, RTC not implemented;

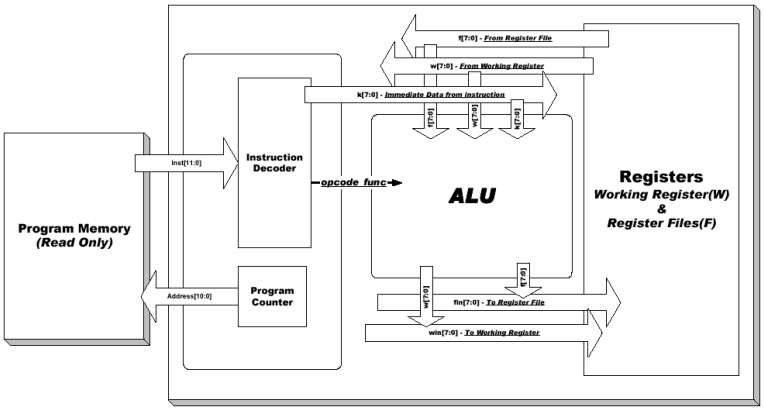
- Dedicated RC8ASM Assembler that generate hex file(romfile.hex) for 12-Bit program memory;

- Verified with PIC core test program (Assembly);

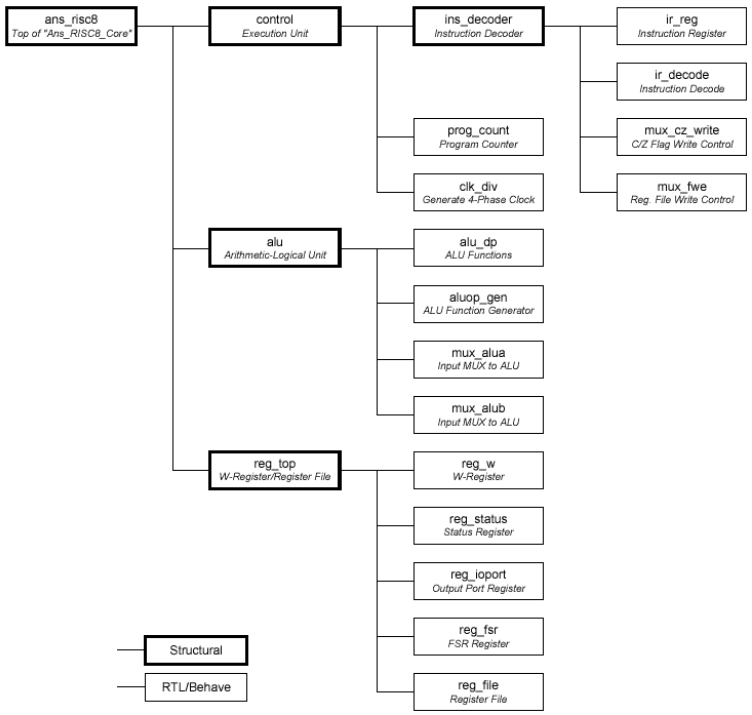
Top-level block diagram:



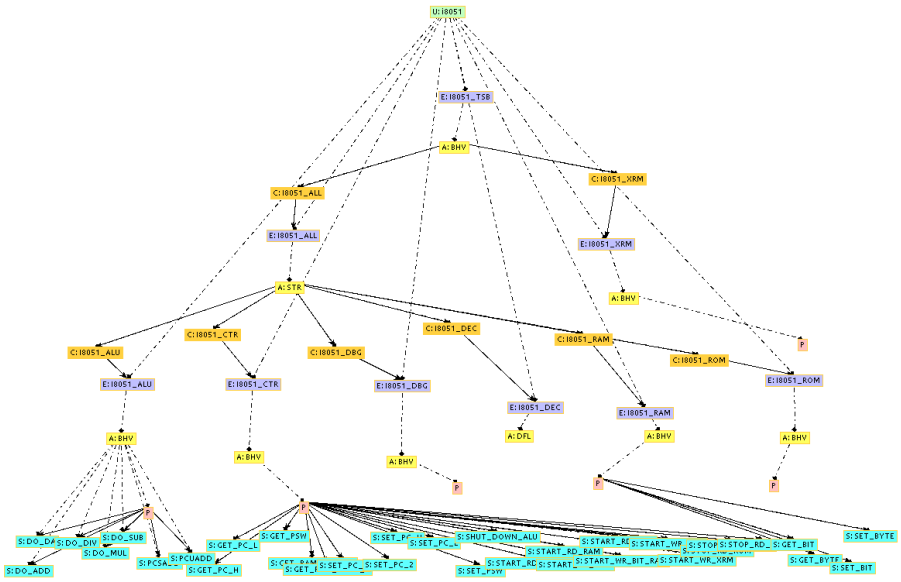
Internal data-path diagram :



Entity hierarchy :



Syntax object graph;

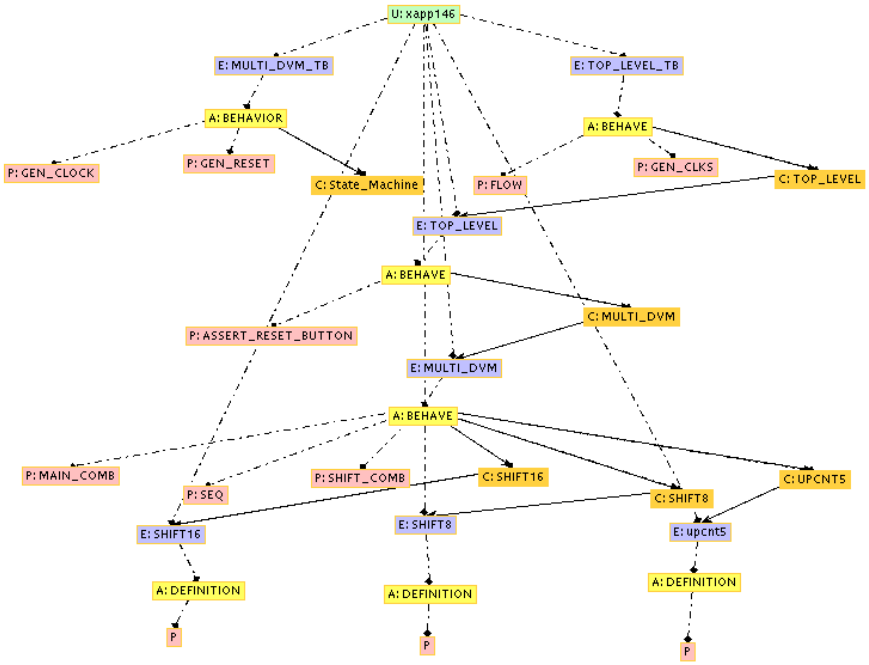


10.21 XAPP146

Project information	
Project name	XAPP146
Author	Xilinx
Project Size	7 files, 59k
Archive	xapp146.zip

From original documentation: ADS7870 8-channel voltmeter module; 8-channel voltmeter Springboard module for Handspring palmtops. Implemented on CoolRunner XPLA3 CPLD located on the Insight Springboard Development Board.

Syntax object graph;

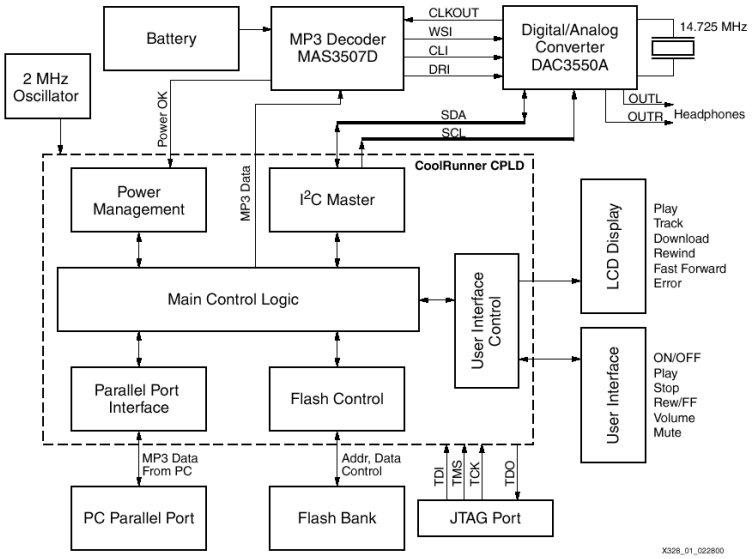


10.22 XAPP328

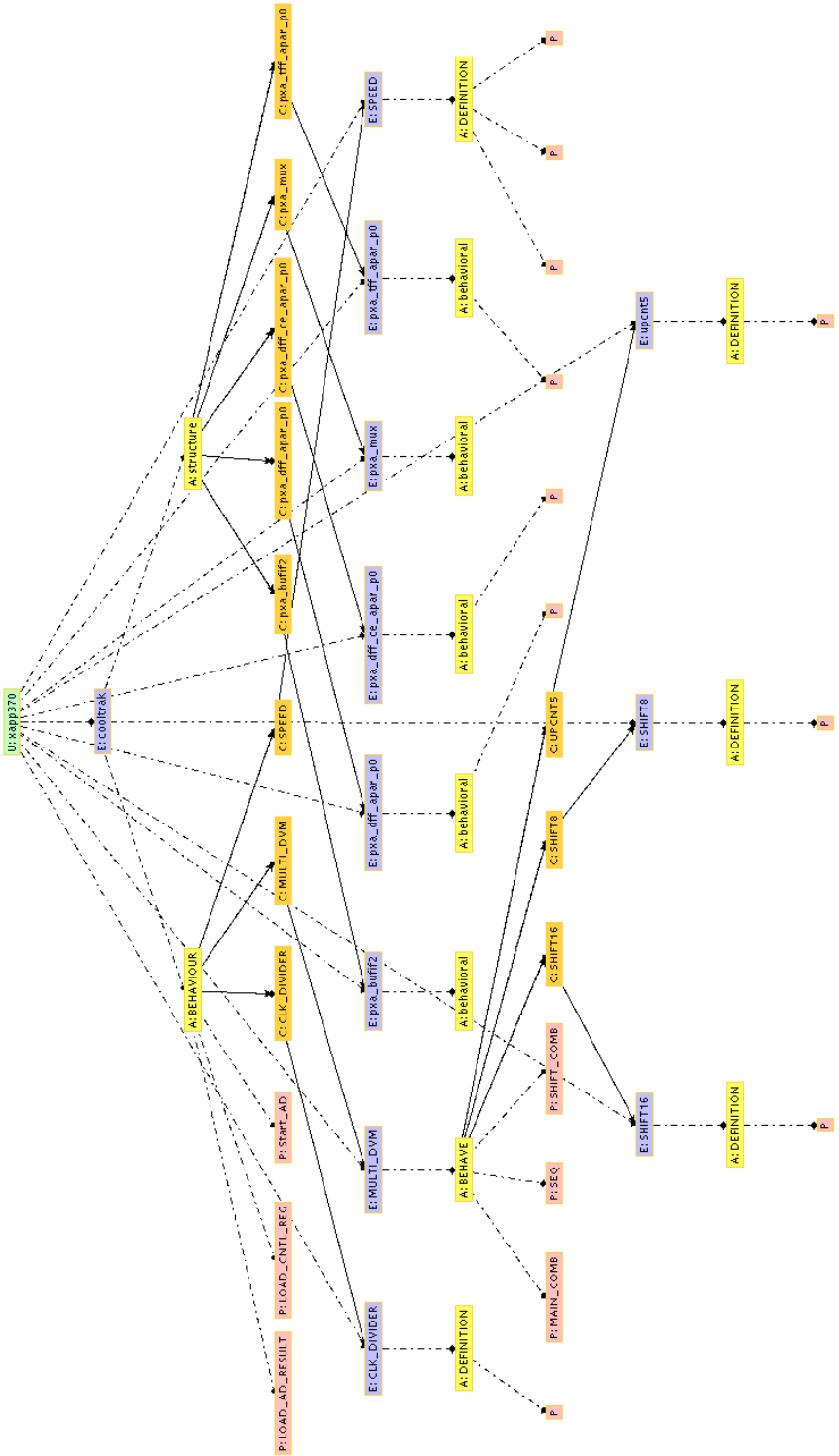
Project information	
Project name	XAPP328
Author	Xilinx
Project Size	36 files, 646k
Archive	xapp328.zip

From original documentation: MP3 Portable Player; MP3 portable player using a 3.3V 256 macrocell, CoolRunner CPLD (XCR3256XL) as the main controller.

Block diagram :



Syntax object graph:

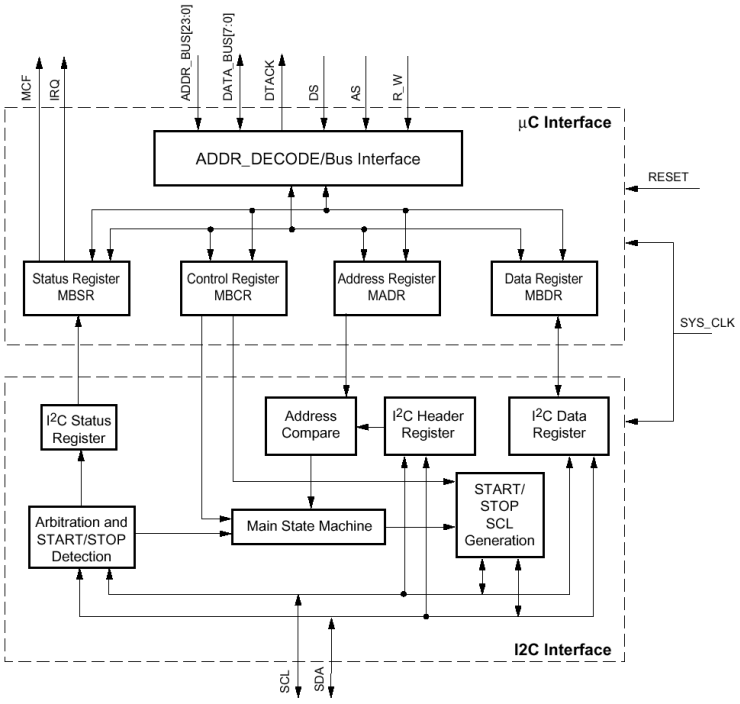

U:xapp370
E:cooltrak
A:BEHAVIOUR
A:structure
C:pxa_dff_apar_p0
C:pxa_mux
C:pxa_dff_ce_apar_p0
C:pxa_dff_apar_p0
C:pxa_buff2
C:SPEED
C:MULTI_DVM
C:CLK_DIVIDER
P:Start_AD
P:LOAD_CNTL_REG
P:LOAD_AD_RESULT
E:SPEED
E:pxa_dff_apar_p0
E:pxa_mux
E:pxa_dff_ce_apar_p0
E:pxa_dff_apar_p0
E:pxa_buff2
E:MULTI_DVM
E:CLK_DIVIDER
A:behavioral
A:BEHAVE
A:DEFINITION
P
C:UPCNT5
C:SHIFT8
C:SHIFT16
P:SHIFT_COMB
P:SEQ
P:MAIN_COMB
E:upcnt5
E:SHIFT8
E:SHIFT16

10.23 XAPP333

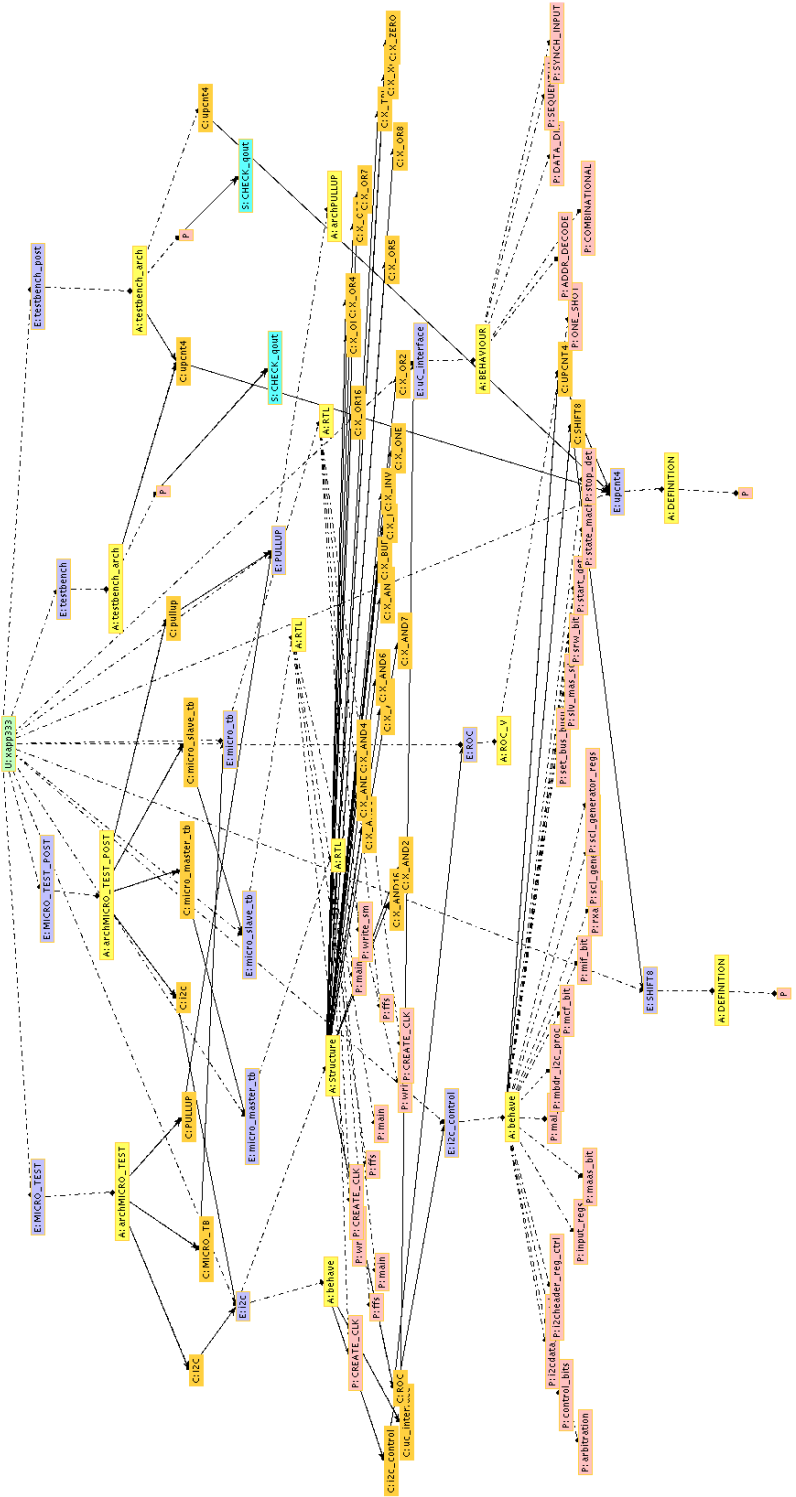
Project information	
Project name	XAPP333
Author	Xilinx
Project Size	14 files, 558k
Archive	xapp333.zip

From original documentation: VHDL implementation of an I 2 C controller in a Xilinx CoolRunner XPLA3 256 macrocell CPLD.

Block diagram:



Syntax object graph:

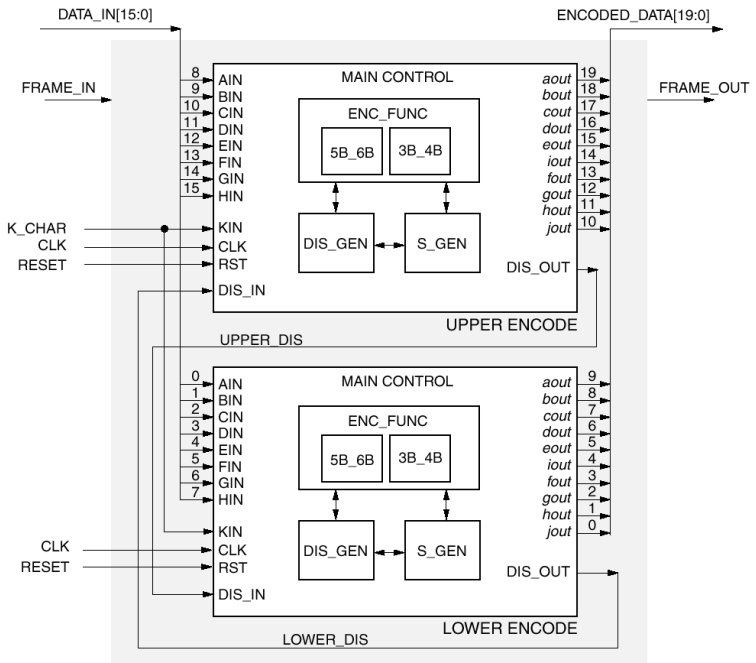

U:xapp333
E:testbench_post
A:testbench_arch
C:upcnt4
S:CHECK_qout
P
E:testbench
C:pullup
E:PULLUP
A:RTL
C:micro_slave_tb
E:micro_tb
E:MICRO_TEST_POST
A:archMICRO_TEST_POST
C:micro_master_tb
E:micro_slave_tb
C:I2C
E:micro_master_tb
E:MICRO_TEST
A:archMICRO_TEST
C:PULLUP
C:MICRO_TB
E:I2C
C:ROC
C:I2C_Control
C:uic_interface
A:archPULLUP
C:X_ZERO
C:X_ORB
C:X_ORS
C:X_OR4
C:X_OR16
C:X_OR2
C:X_ONE
C:X_INV
C:X_AND8
C:X_AND6
C:X_AND7
C:X_AND4
C:X_AND16
C:X_AND2
A:Structure
P:main
P:ffs
P:wr
P:CREATE_CLK
A:behave
E:I2C_Control
E:ROC
A:ROC_V
A:BEHAVIOUR
E:uic_interface
P:DATA_DI
P:SEQUENCE
P:SYNCH_INPUT
P:COMBINATIONAL
P:ADDR_DECODE
P:ONE_SHOT
C:UPCNT4
C:SHIFT8
E:upcnt4
A:DEFINITION
E:SHIFT8
P:state_mscf
P:start_det
P:set_bus_line
P:siw_bit
P:siw_mas_sl
P:mcf_bit
P:rxa
P:scl_gen
P:scl_generator_regs
P:mbdr_i2c_proc
P:mal
P:mass_bit
P:input_regs
P:i2cheader_reg_ctrl
P:i2cdata
P:control_bits
P:arbitration

10.24 XAPP336

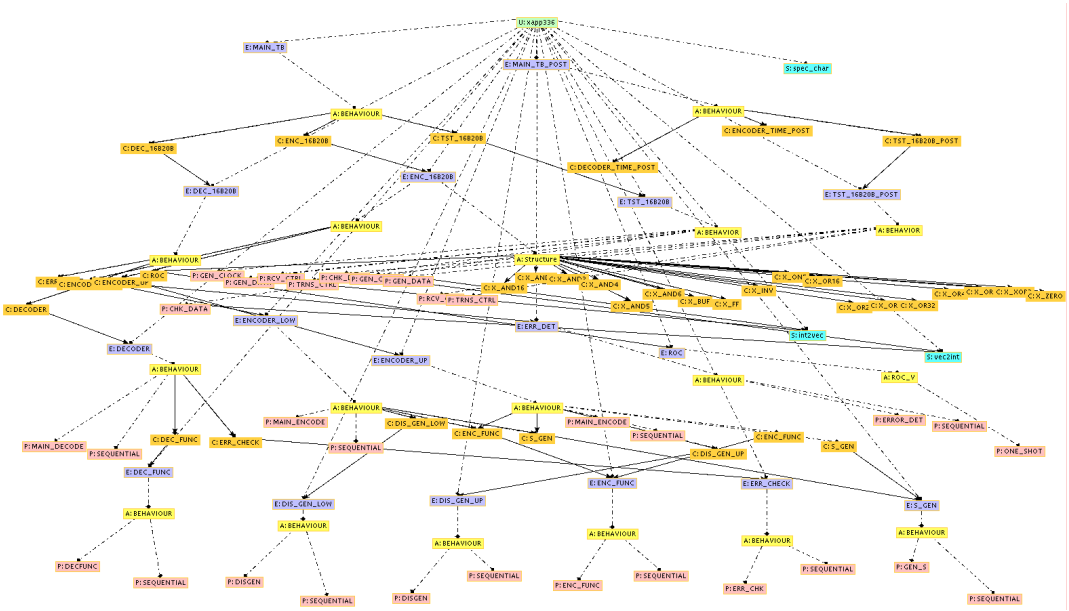
Project information	
Project name	XAPP336
Author	Xilinx
Project Size	31 files, 532k
Archive	xapp336.zip, xapp336_8b10b.zip

From original documentation: 16 bit/20 bit and 8 bit/10 bit and Encoder/Decoder; VHDL implementation of a fibre channel byte-oriented transmission encoder and decoder in a Xilinx CoolRunner CPLD.

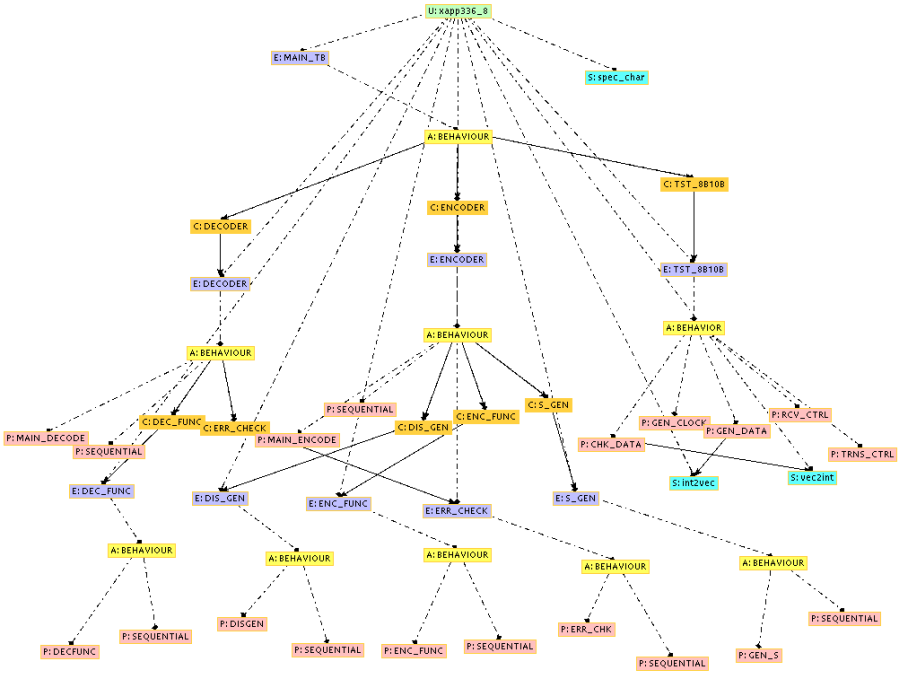
Block diagram :



Syntax object graph (for the 16-20 bit version):



Syntax object graph (for the 8-10 bit version):

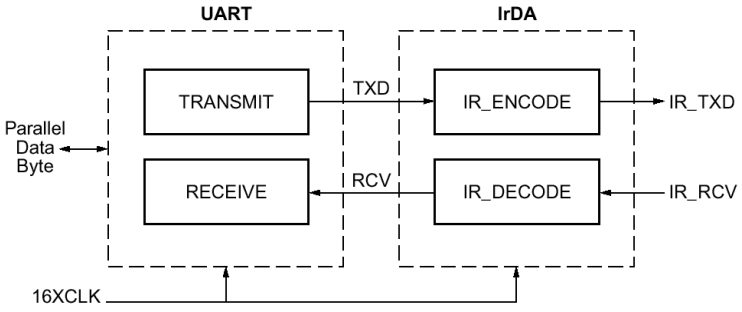


10.25 XAPP345

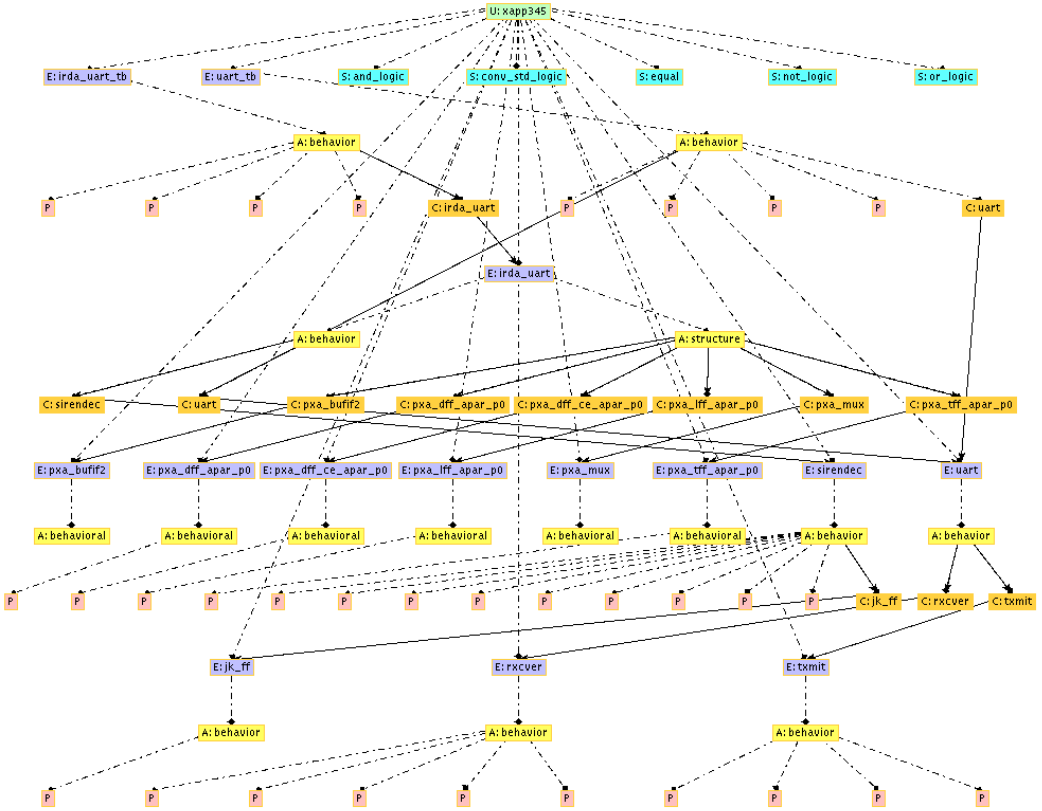
Project information	
Project name	XAPP345
Author	Xilinx
Project Size	10 files, 131k
Archive	xapp345_vhdl.zip

From original documentation: The fundamental building blocks required to create a half-duplex IrDA and full-duplex UART system using a CoolRunner XPLA3 CPLD.

Block diagram :



Syntax object graph:

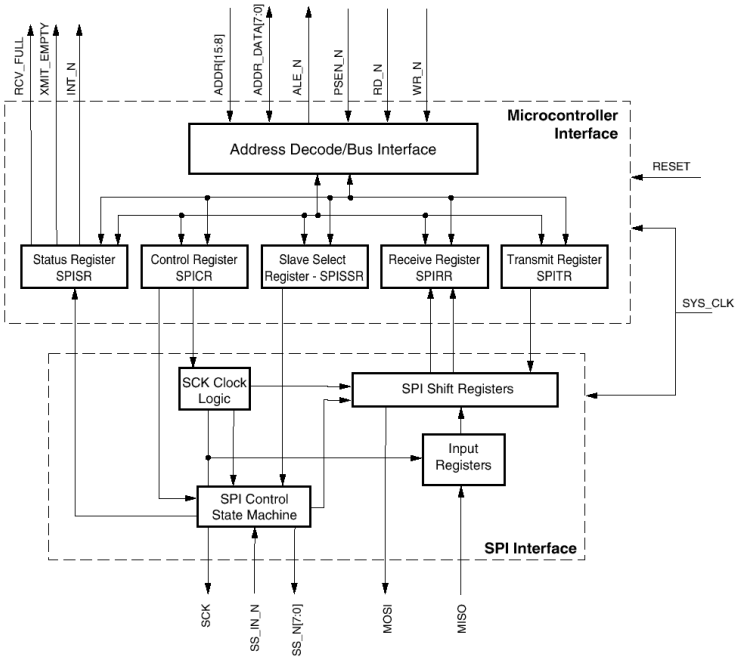


10.26 XAPP348

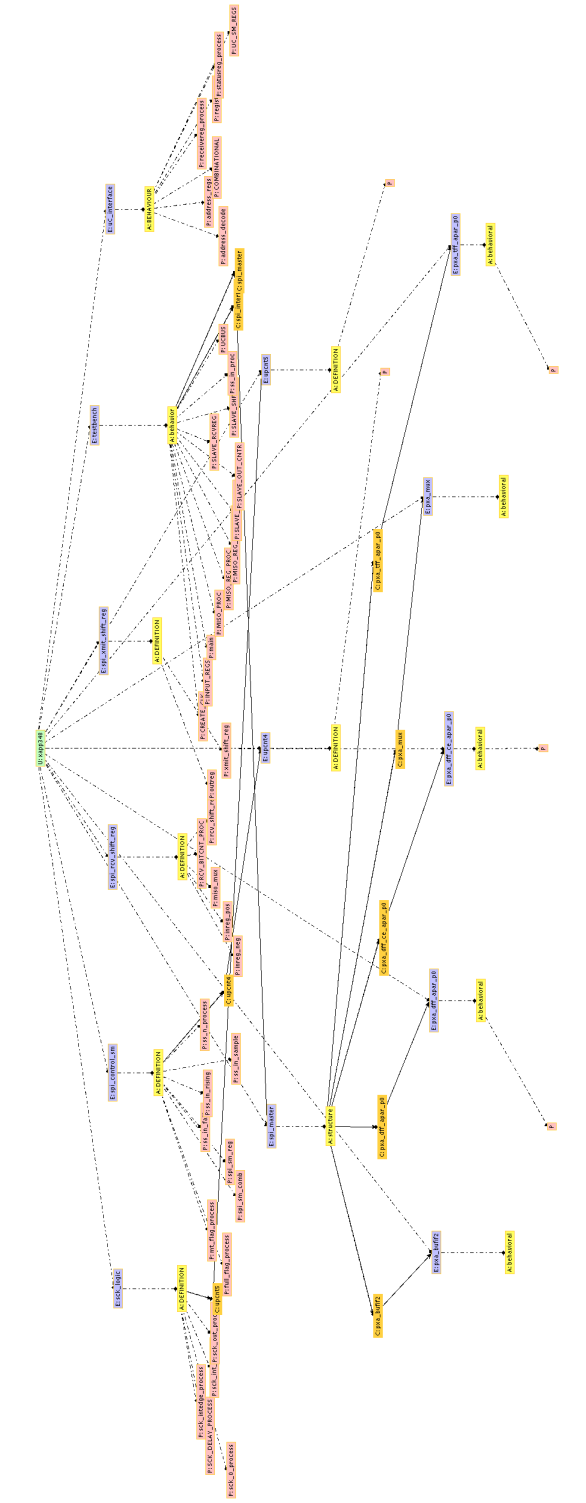
Project information	
Project name	XAPP348
Author	Xilinx
Project Size	10 files, 233k
Archive	xapp348.zip

From original documentation: Serial Peripheral Interface Master;VHDL implementation of a full-duplex, synchronous, SPI (Serial Peripheral Interface) master in a Xilinx CoolRunner XPLA3 CPLD.

Block diagram :



Syntax object graph:

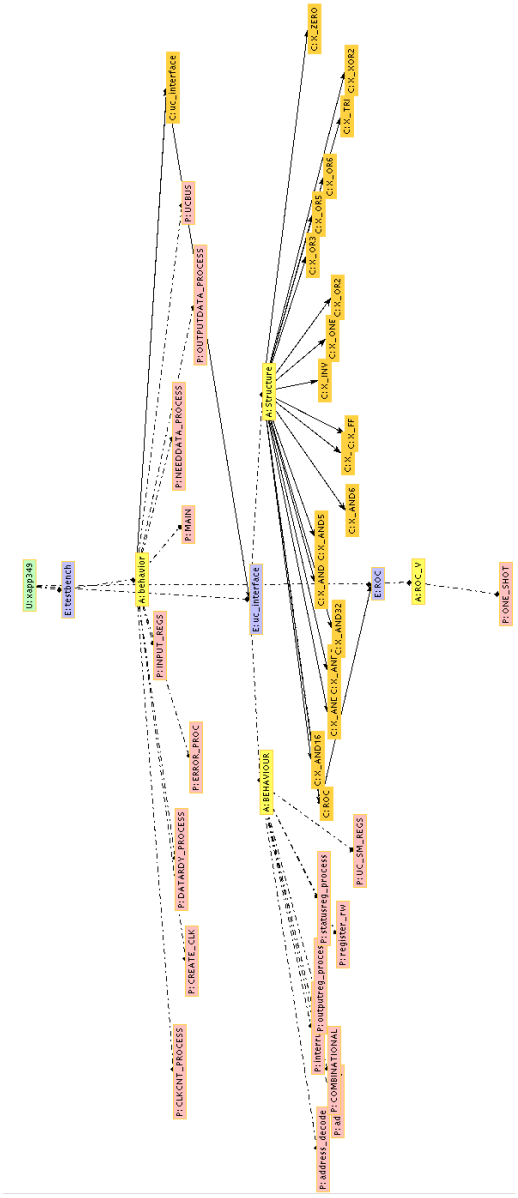


10.27 XAPP349

Project information	
Project name	XAPP349
Author	Xilinx
Project Size	3 files, 228k
Archive	xapp349.zip

From original documentation: The CoolRunner 8051 Microcontroller Interface, designed from using the Intel data sheet for the 8051 Microcontroller.

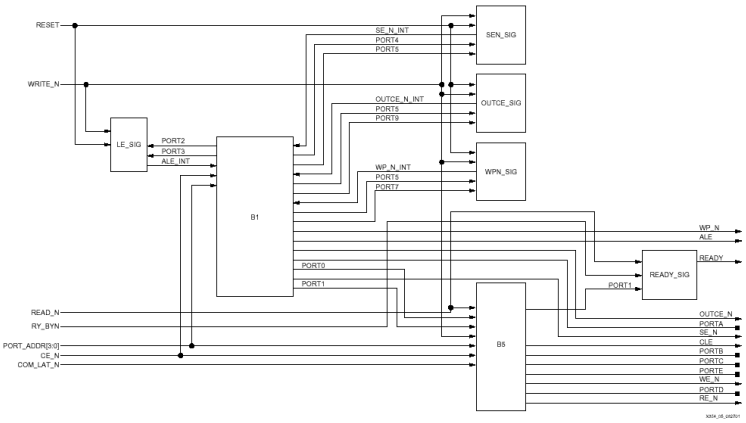
Syntax object graph:



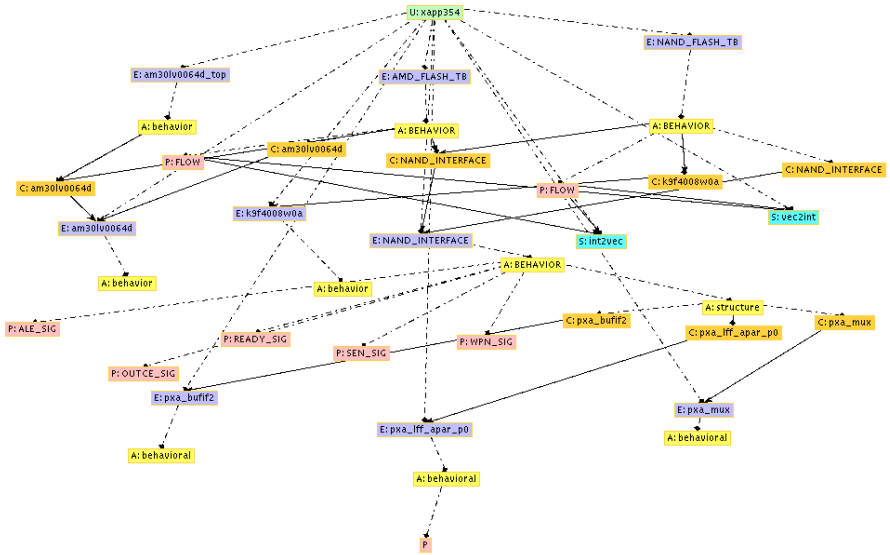
10.28 XAPP354

Project information	
Project name	XAPP354
Author	Xilinx
Project Size	7 files, 51k
Archive	xapp354_vhdl.zip

From original documentation: NAND Flash memory interface.
Block diagram :



Syntax object graph:

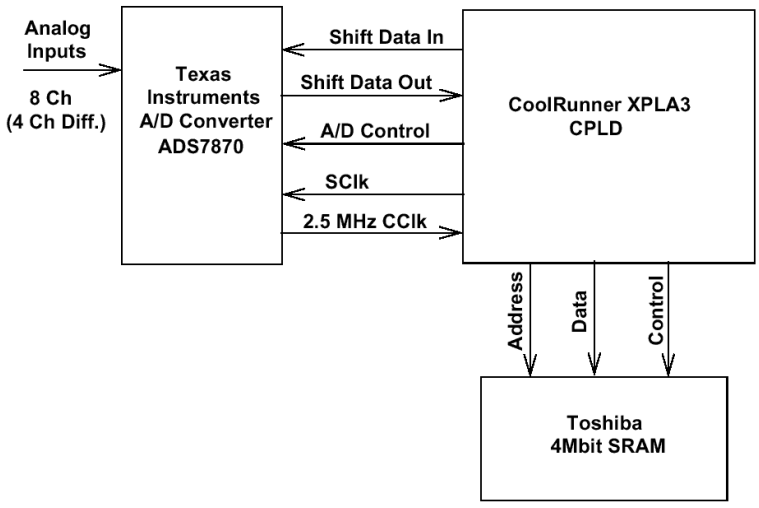


10.29 XAPP355

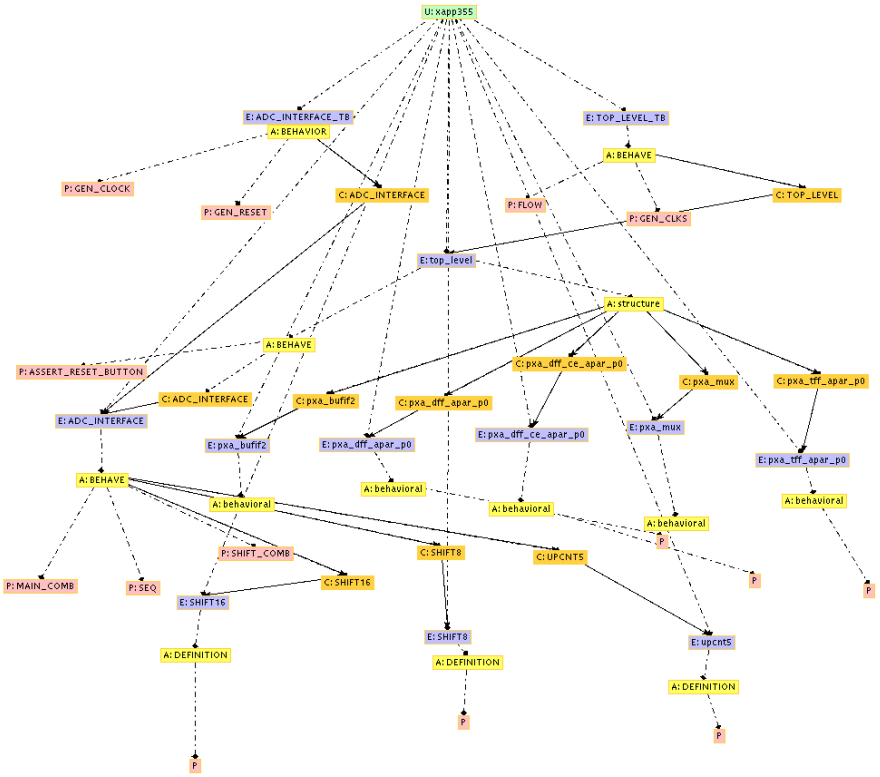
Project information	
Project name	XAPP355
Author	Xilinx
Project Size	8 files, 266k
Archive	xapp355.zip

From original documentation: Serial ADC Interface;VHDL implementation for controlling a Texas Instruments ADS7870 Analog to Digital Converter (ADC) in a Xilinx CoolRunner XPLA3 CPLD.

Block diagram :



Syntax object graph:

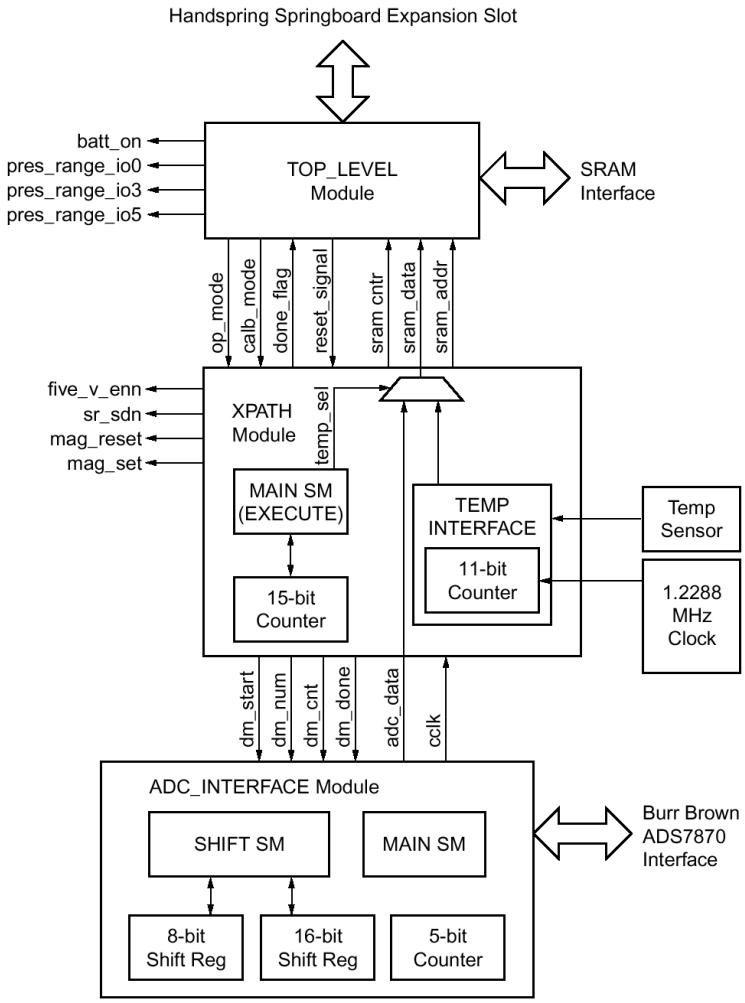


10.30 XAPP356

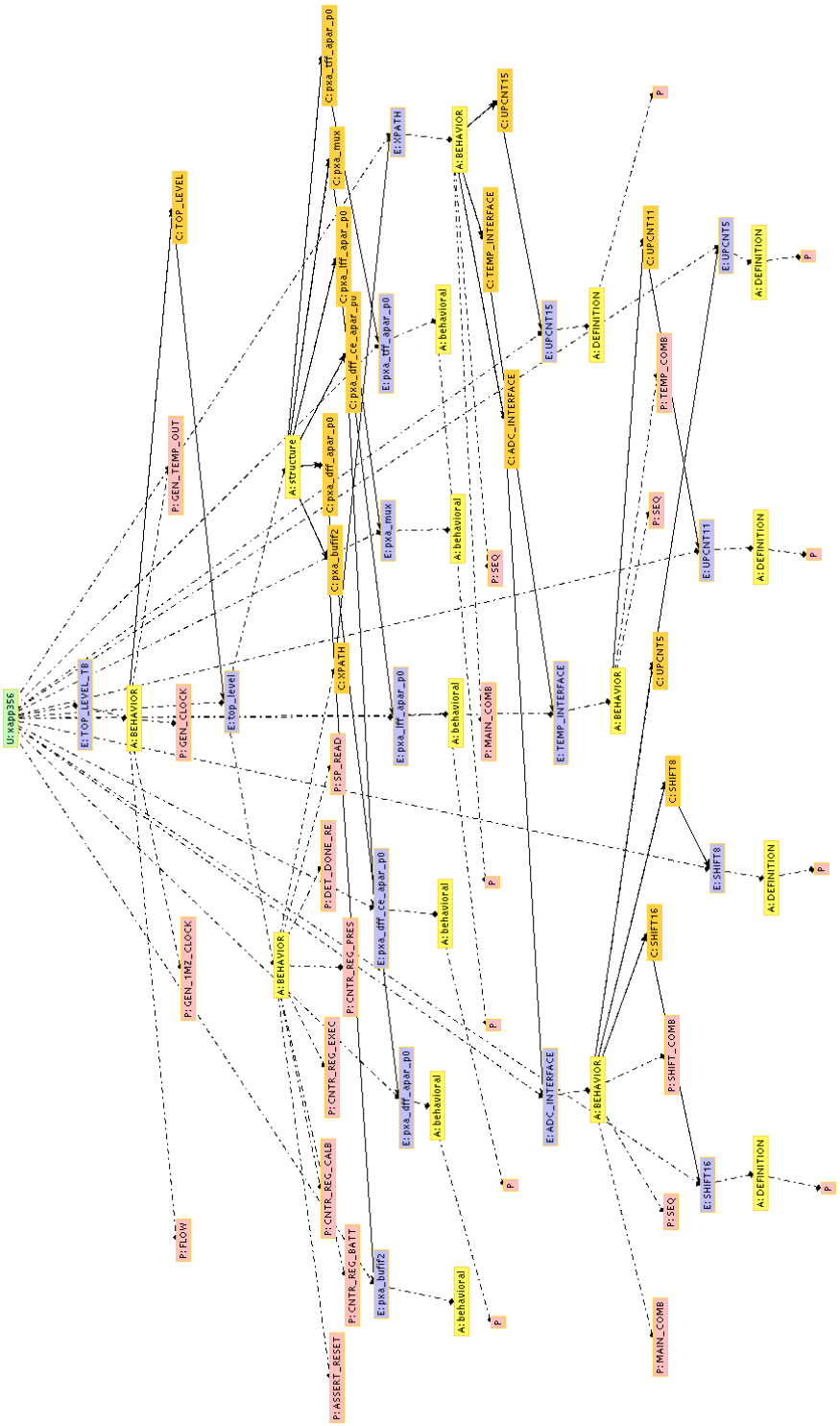
Project information	
Project name	XAPP356
Author	Xilinx
Project Size	12 files, 349k
Archive	xapp356.zip

From original documentation: XPATH Springboard Module Design;Implementation of a XPATH (Xilinx Pressure Altimeter Temperature Heading) Handspring Springboard module using Xilinx CoolRunner XPLA3 CPLD and Handspring Visor.

Block diagram :



Syntax object graph:

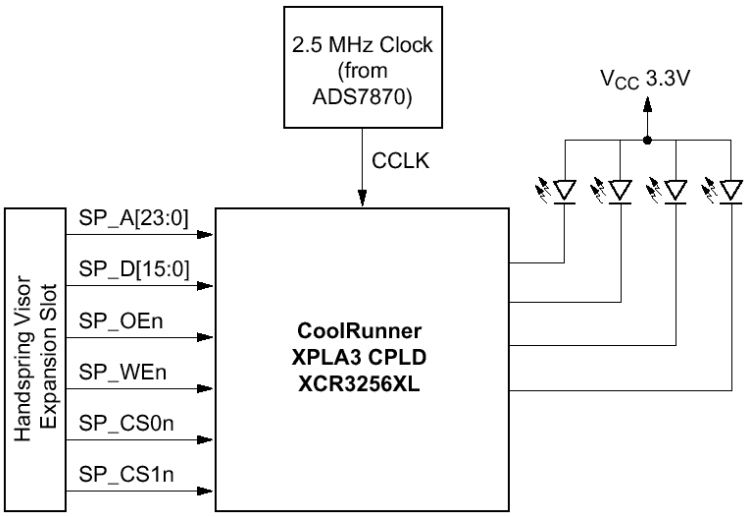

U:xapp356
E:TOP_LEVEL_T8
A:BEHAVIOR
P:GEN_CLOCK
P:GEN_TEMP_OUT
C:TOP_LEVEL
E:top_level
P:GEN_1MZ_CLOCK
P:FLOW
P:ASSERT_RESET
A:BEHAVIOR
P:DET_DONE_RE
P:SP_READ
P:CNTR_REG_PRES
P:CNTR_REG_EXEC
P:CNTR_REG_CALB
P:CNTR_REG_BATT
C:pxa_dff_apar_p0
C:pxa_mux
C:pxa_lff_apar_p0
C:pxa_dff_ce_apar_p0
C:pxa_dff_apar_p0
C:pxa_buff2
C:XPATH
E:pxa_dff_apar_p0
E:pxa_mux
E:pxa_dff_ce_apar_p0
E:pxa_dff_apar_p0
E:pxa_buff2
A:structure
E:XPATH
A:behavioral
A:behavioral
A:behavioral
A:behavioral
A:behavioral
P:
P:SEQ
P:MAIN_COMB
C:UPCNT15
C:TEMP_INTERFACE
C:ADC_INTERFACE
E:UPCNT15
E:TEMP_INTERFACE
E:ADC_INTERFACE
A:BEHAVIOR
A:DEFINITION
A:BEHAVIOR
C:UPCNT11
C:UPCNT5
C:SHIFT8
C:SHIFT16
P:TEMP_COMB
P:SEQ
P:SHIFT_COMB
P:SEQ
P:MAIN_COMB
E:UPCNT5
E:UPCNT11
E:SHIFT8
E:SHIFT16
A:DEFINITION
A:DEFINITION
A:DEFINITION
A:DEFINITION
P
P
P
P
P

10.31 XAPP357

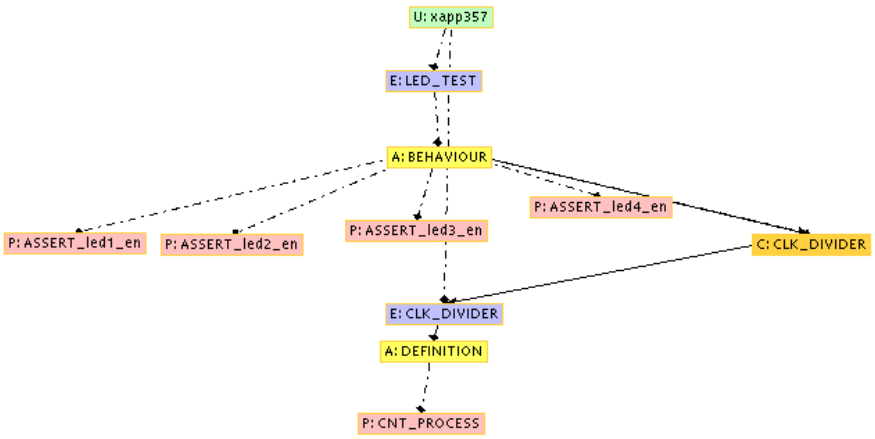
Project information	
Project name	XAPP357
Author	Xilinx
Project Size	2 files, 8k
Archive	xapp357.zip

From original documentation: LED Springboard Module Design. Implementation of a test Handspring Springboard module with 4 LEDs using Xilinx CoolRunner XPLA3 CPLD and Handspring Visor.

Block diagram :



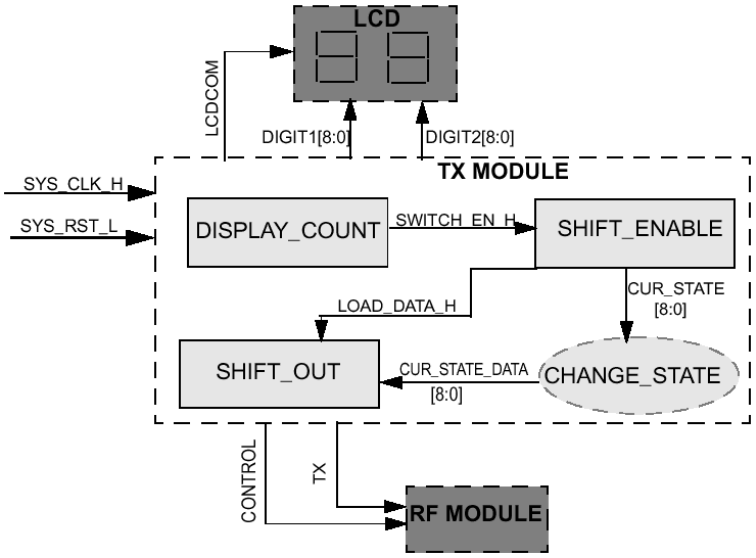
Syntax object graph:



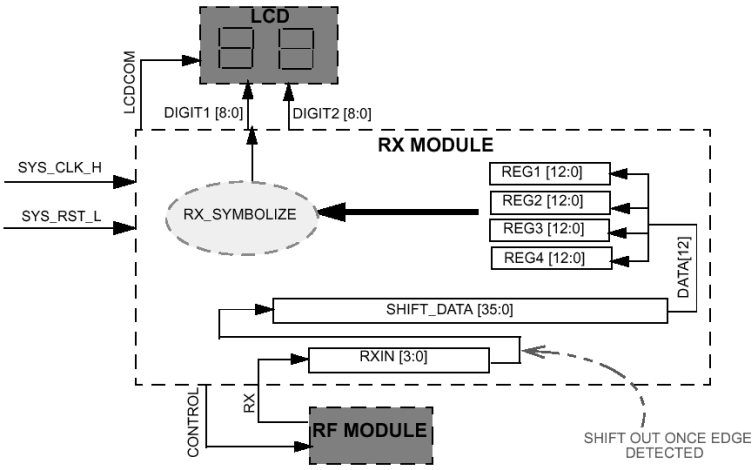
10.32 XAPP358

Project information	
Project name	XAPP358
Author	Xilinx
Project Size	15 files, 845k
Archive	xapp358.zip

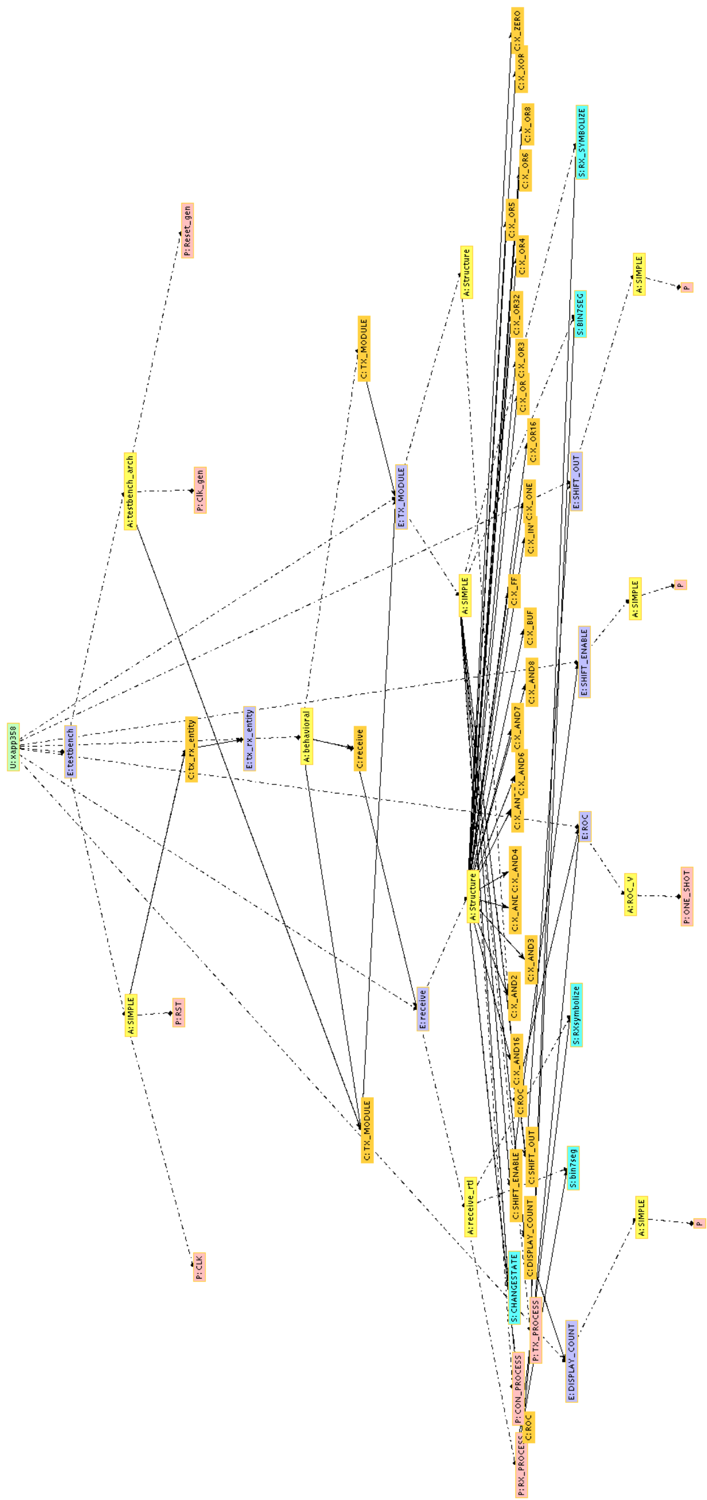
From original documentation: Implementation of a generic Wireless Transceiver.
Transmit module block diagram :



Receive module block diagram :



Syntax object graph:

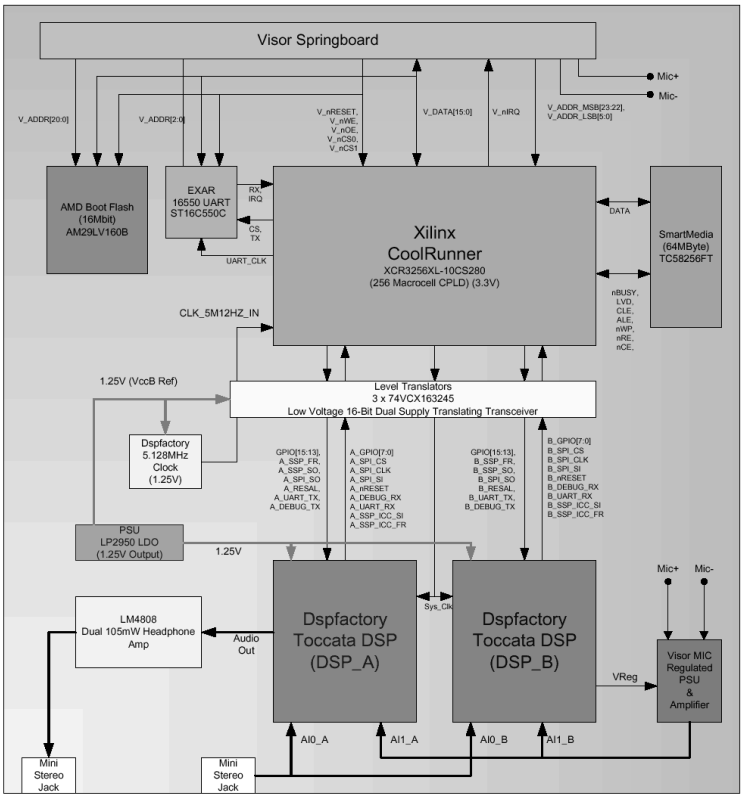

U: xapp358
E: testbench
A: testbench_arch
P: Reset_gen
P: Clk_gen
A: SIMPLE
P: RST
P: CLK
C: tx_rx_entity
E: tx_rx_entity
A: behavioral
C: receive
E: receive
C: TX_MODULE
E: TX_MODULE
C: TX_MODULE
A: Structure
A: SIMPLE
A: Structure
A: receive_rtl
C: X_ZERO
C: X_XOR
C: X_OR8
C: X_OR6
C: X_OR5
C: X_OR4
C: X_OR32
C: X_OR3
C: X_OR
C: X_OR16
C: X_ONE
C: X_FF
C: X_BUF
C: X_AND8
C: X_AND7
C: X_AND6
C: X_AND4
C: X_AND3
C: X_AND2
C: X_AND16
C: ROC
C: SHIFT_ENABLE
C: SHIFT_OUT
C: DISPLAY_COUNT
S: CHANGESTATE
P: TX_PROCESS
P: CON_PROCESS
C: ROC
P: RX_PROCESS
S: RX_SYMBOLIZE
S: BIN7SEG
E: SHIFT_OUT
E: SHIFT_ENABLE
E: ROC
S: RXsymbolize
S: bin7seg
E: DISPLAY_COUNT
A: SIMPLE
P
A: SIMPLE
P
A: ROC_V
P: ONE_SHOT
A: SIMPLE
P

10.33 XAPP363

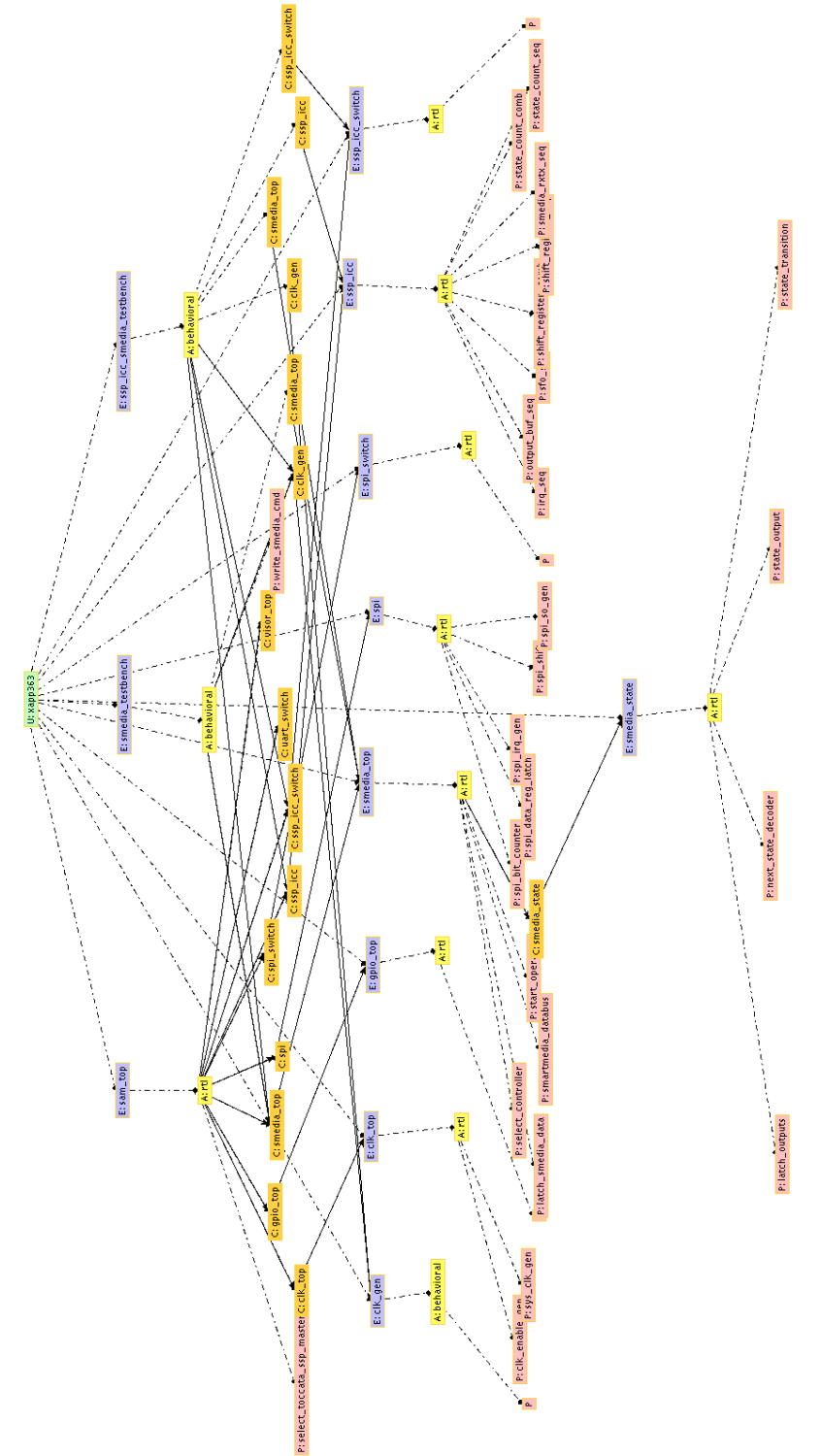
Project information	
Project name	XAPP363
Author	Xilinx
Project Size	12 files, 120k
Archive	xapp363.zip

From original documentation: Sonic Access Springboard Module; Portable audio processing module that can be used in applications ranging from real-time spectrograms to speech analysis.

Block diagram :



Syntax object graph:

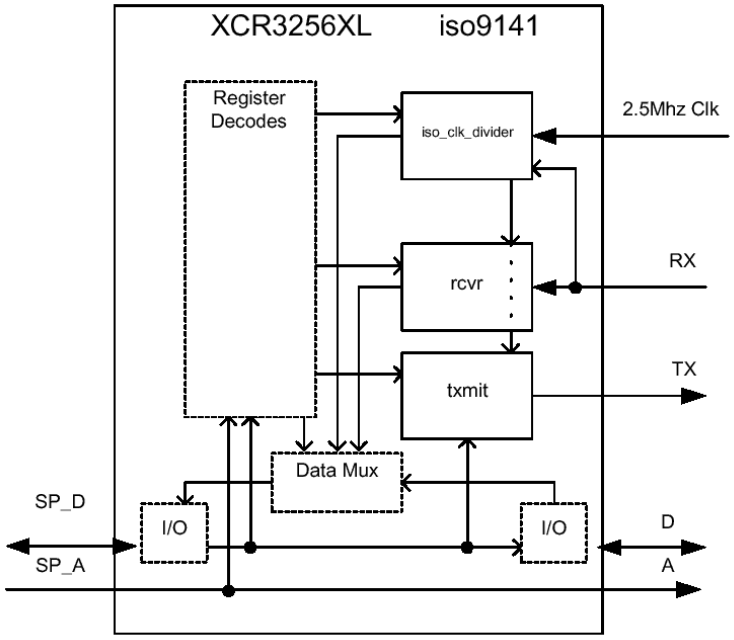

U:xapp383
E:smedia_testbench
A:behavioral
E:ssp_icc_smedia_testbench
A:behavioral
E:sam_top
A:rtl
E:smedia_testbench
A:behavioral
C:ssp_icc_switch
C:ssp_icc
C:smedia_top
C:clk_gen
C:smedia_top
C:clk_gen
C:vsscr_top
P:vwrite_smedia_cmd
C:uart_switch
C:ssp_icc_switch
C:ssp_icc
C:spi_switch
C:spi
C:smedia_top
C:gpio_top
C:clk_top
E:clk_gen
C:select_toccata_ssp_master
C:clk_top
E:ssp_icc_switch
E:ssp_icc
E:spi_switch
E:spi
E:smedia_top
E:gpio_top
E:clk_top
E:clk_gen
A:rtl
A:rtl
A:rtl
A:rtl
A:rtl
A:rtl
A:rtl
A:behavioral
P
P:state_count_seq
P:state_count_comb
P:smedia_rxtx_seq
P:shift_regi
P:shift_register
P:sfo_
P:output_buf_seq
P:irq_seq
P
P:spi_slvr
P:spi_so_gen
P:spi_irq_gen
P:spi_data_reg_latch
P:spi_bit_counter
P:start_oper
P:smartmedia_databus
P:select_controller
P:latch_smedia_data
P:clk_enable_gen
P:sys_clk_gen
P
E:smedia_state
C:smedia_state
A:rtl
P:state_transition
P:state_output
P:next_state_decoder
P:latch_outputs

10.34 XAPP365

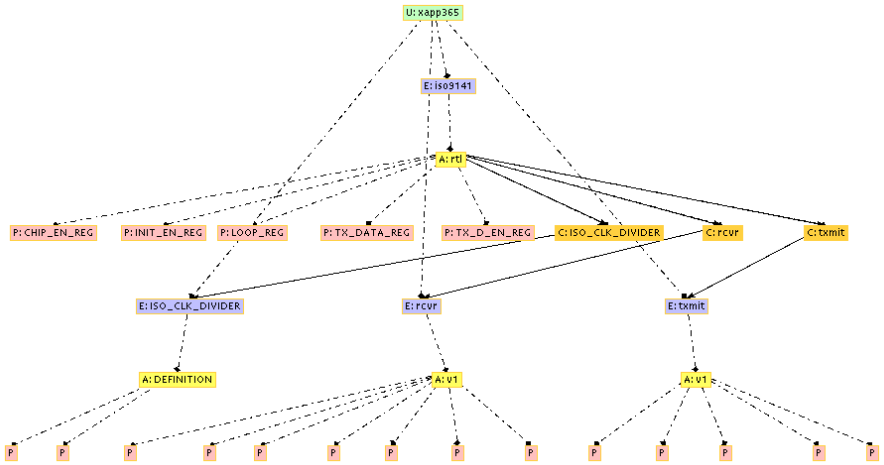
Project information	
Project name	XAPP365
Author	Xilinx
Project Size	4 files, 22k
Archive	xapp365.zip

From original documentation: Automotive Scan Tool Springboard Module. Portable test device interfacing with On-Board Diagnostics (OBD) systems found in cars to display data and trouble codes.

Block diagram :



Syntax object graph:

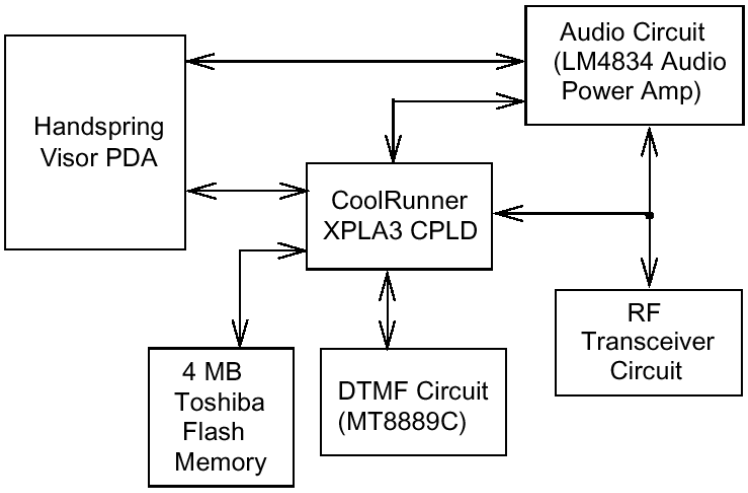


10.35 XAPP367

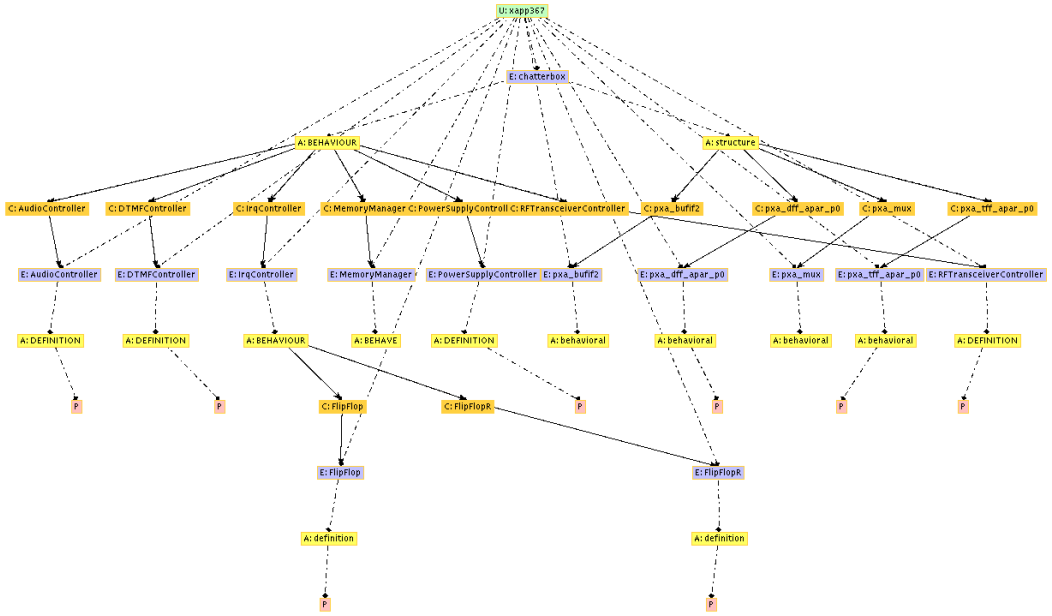
Project information	
Project name	XAPP367
Author	Xilinx
Project Size	10 files, 98k
Archive	xapp367.zip

From original documentation: Chatterbox Springboard Module. Short distance, wireless, two-way radio communications device that combines walkie-talkie type voice communications with security features.

Block diagram :



Syntax object graph:

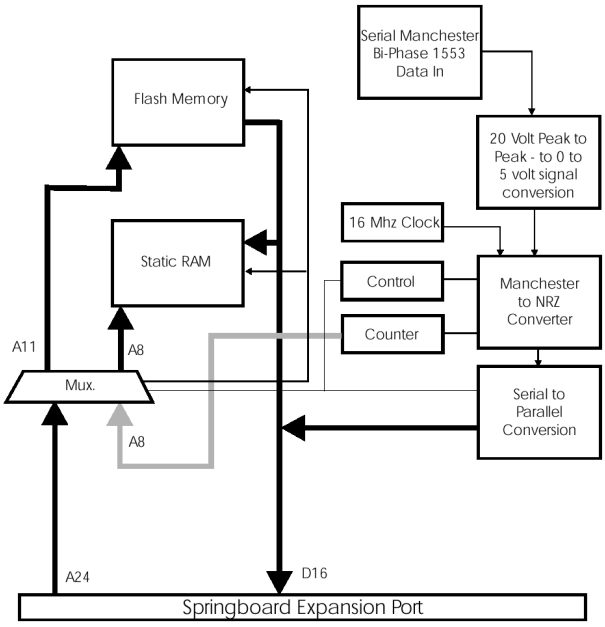


10.36 XAPP369

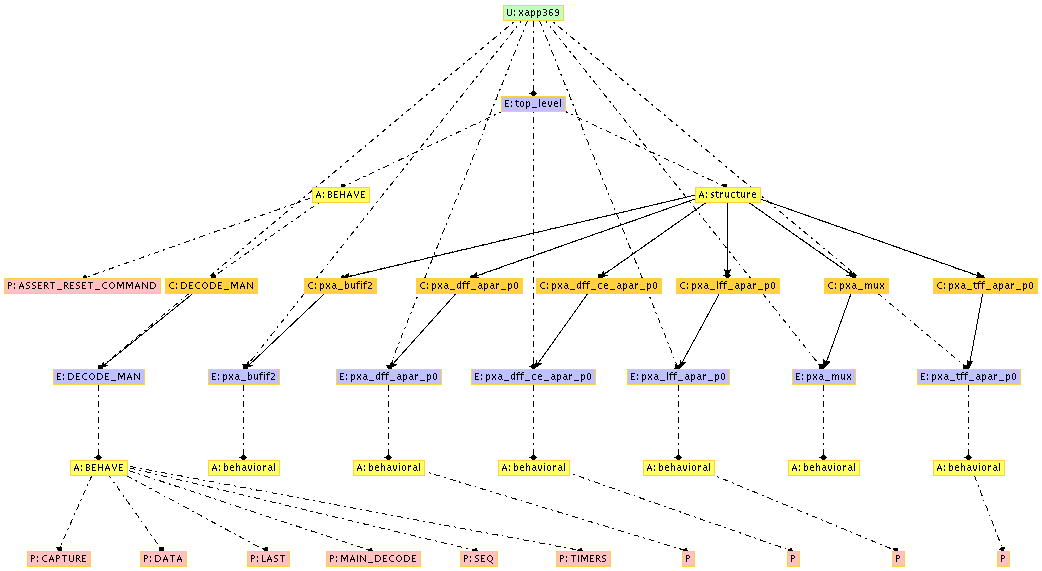
Project information	
Project name	XAPP369
Author	Xilinx
Project Size	3 files, 262 k
Archive	xapp369.zip

From original documentation: 1553 Bus Data Analyzer Springboard Module;The bus data analyzer is a Springboard module that tests the operating functionality of the standard military 1553 bus.

Block diagram :



Syntax object graph:

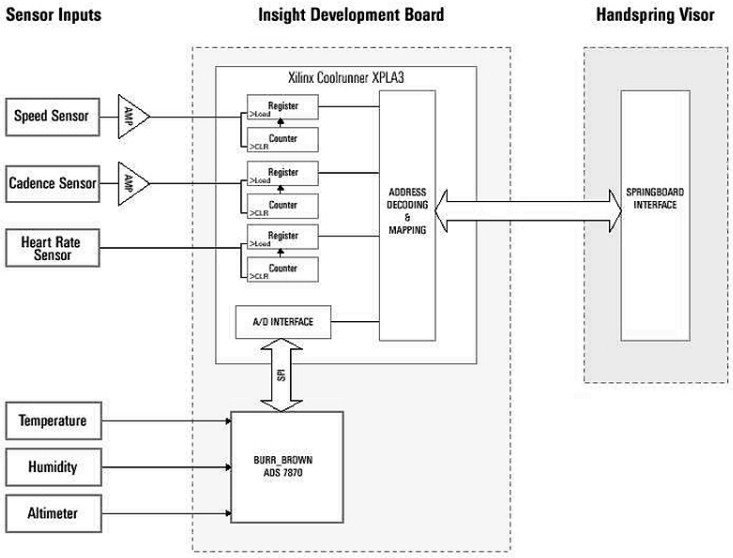


10.37 XAPP370

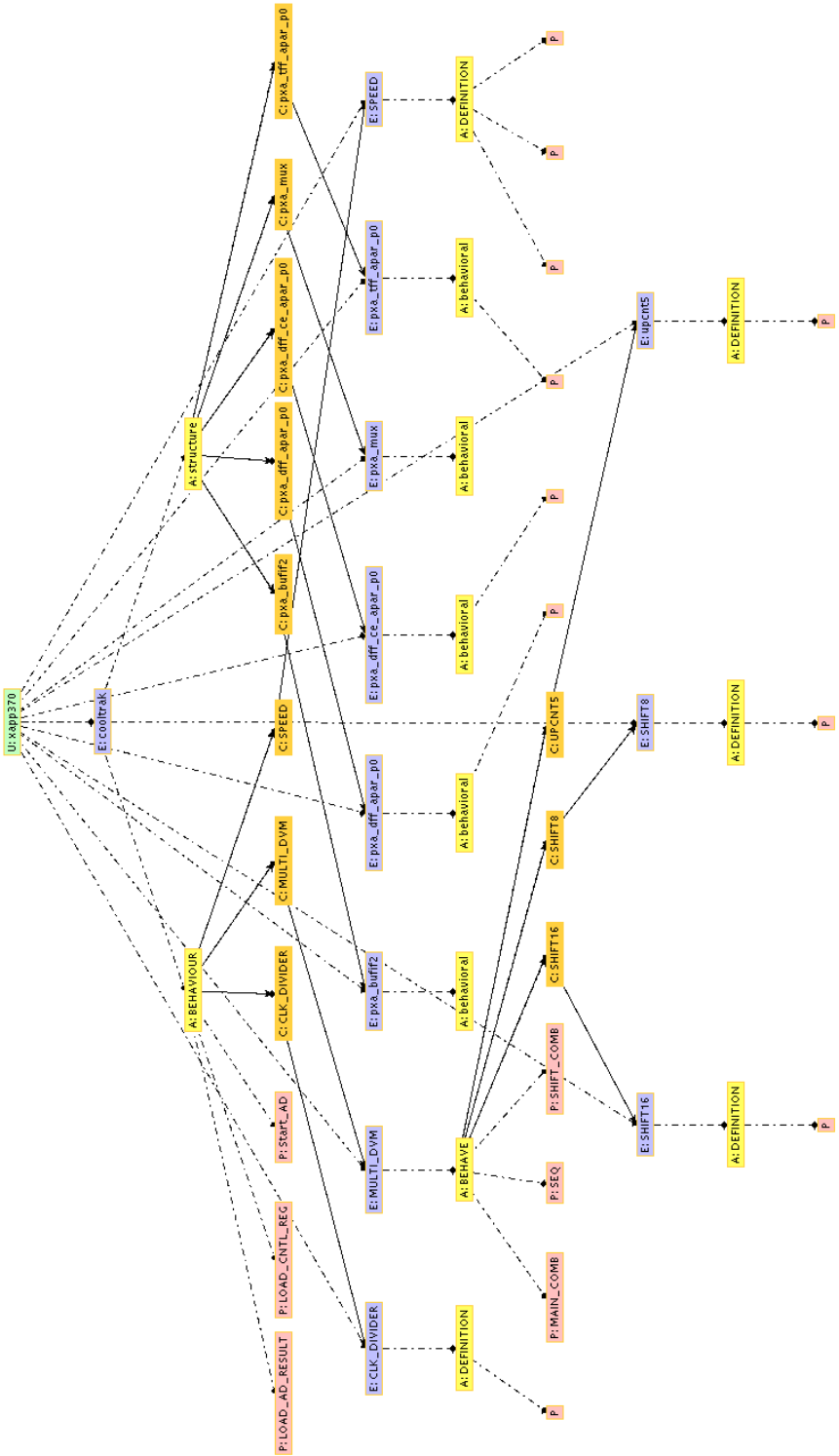
Project information	
Project name	XAPP370
Author	Xilinx
Project Size	9 files, 351k
Archive	xapp370.zip

From original documentation: Bicycle Computer Springboard Module. Advanced cycling computer, providing time, speed, distance, pedal cadence, air temperature, humidity and heart rate statistics.

Block diagram :



Syntax object graph:


U:xapp370
E:control
A:structure
A:BEHAVIOUR
C:pxa_dff_apar_p0
C:pxa_mux
C:pxa_dff_ce_apar_p0
C:pxa_dff_apar_p0
C:pxa_buff2
C:SPEED
C:MULTI_DVM
C:CLK_DIVIDER
P:Start_AD
P:LOAD_CNTL_REG
P:LOAD_AD_RESULT
E:SPEED
E:pxa_dff_apar_p0
E:pxa_mux
E:pxa_dff_ce_apar_p0
E:pxa_dff_apar_p0
E:pxa_buff2
E:MULTI_DVM
E:CLK_DIVIDER
A:DEFINITION
A:behavioral
A:behavioral
A:behavioral
A:behavioral
A:behavioral
A:BEHAVE
A:DEFINITION
P
P
P
P
P
P
P
P
P
P
P
C:UPCNT5
C:SHIFT8
C:SHIFT16
P:SHIFT_COMB
P:SEQ
P:MAIN_COMB
E:upcnt5
E:SHIFT8
E:SHIFT16
A:DEFINITION
A:DEFINITION
A:DEFINITION
P
P
P

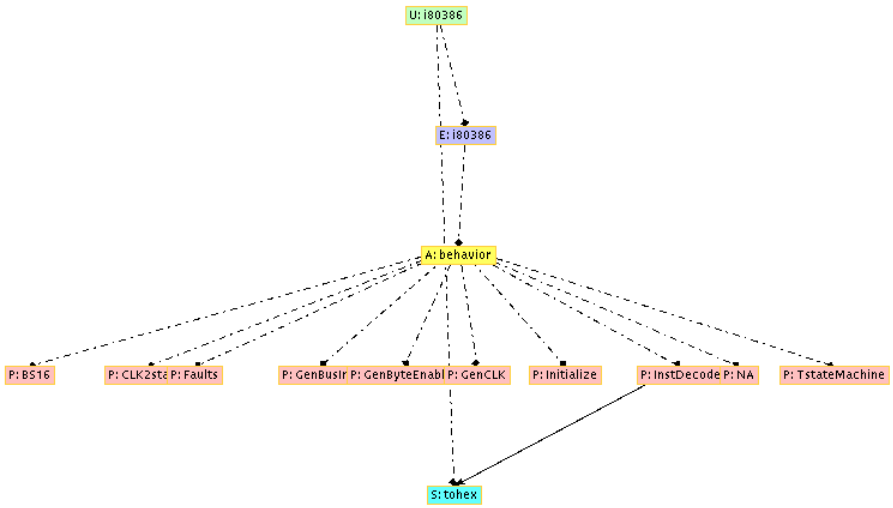
10.38 i80386

Project information	
Project name	i80386
Author	Mark Dakur, Convergent, Inc.
Project Size	1 files, 59k
Archive	i80386.vhdl

From original documentation: This VHDL model emulates the Intel 80386 32-bit CPU to the instruction and bus timing level. The i80386 consists of 6 functional units defined as follows:

- Bus Interface Unit BIunit. Accepts internal requests for code fetches from the CPunit and data transfers from the Eunit and prioritizes the requests. It is the interface to the external pins (ports) of the package.
- Code Prefetch Unit CPunit Performs the program look ahead function. When the BIunit is not performing bus cycles to execute an instruction, it uses the BIunit to to fetch sequentially along the instruction byte stream. These prefetched instructions are stored in the 16-byte Code Queue to await processing by the IDunit.
- Instruction Decode Unit IDunit. Instructions Supported:
  - nop
  - mov eax,"immediate 32 bit data"
  - mov ebx,"immediate 32 bit data"
  - mov eax,[ebx]
  - mov [ebx],eax
  - in al,"byte address"
  - out "byte address",al
  - inc eax
  - inc ebx
  - jmp "label" (relative nears and shorts)
- Execution Unit Eunit
  - Control Unit Cunit
  - Data Unit Dunit
  - Protection Test Unit PTunit
- Segmentation Unit Sunit
- Paging Unit Punit: Page Translator Unit PTunit: Translation Lookaside Buffer TLB:
  - Page Directory
  - Page Table

Syntax object graph;

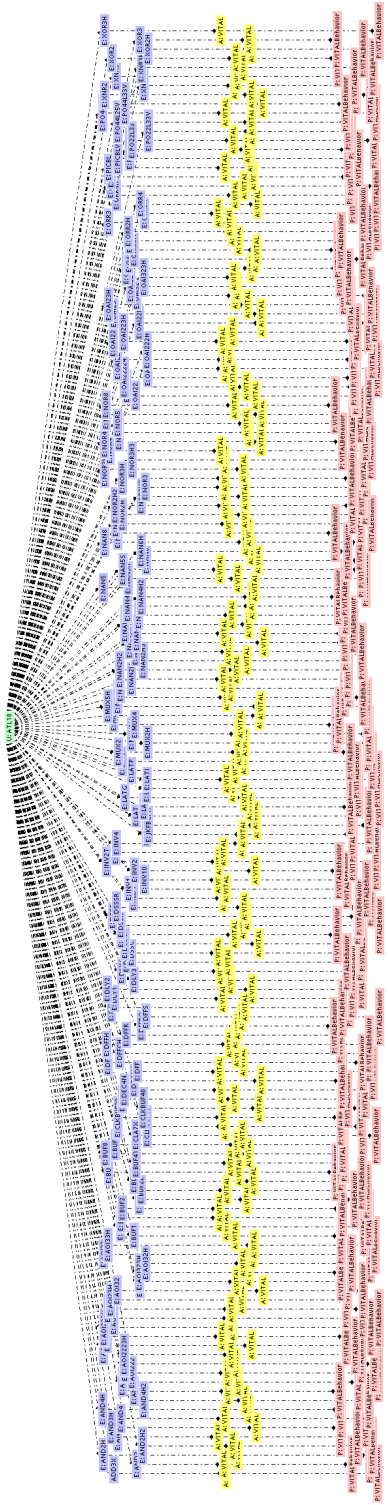


10.39 ATL18

Project information	
Project name	ATL18
Author	Atmel Corp.
Project Size	15 files, 753k
Archive	atl_18_18.tar.gz

From original documentation: Atmel ATL18 Vital VHDL library.

Syntax object graph;

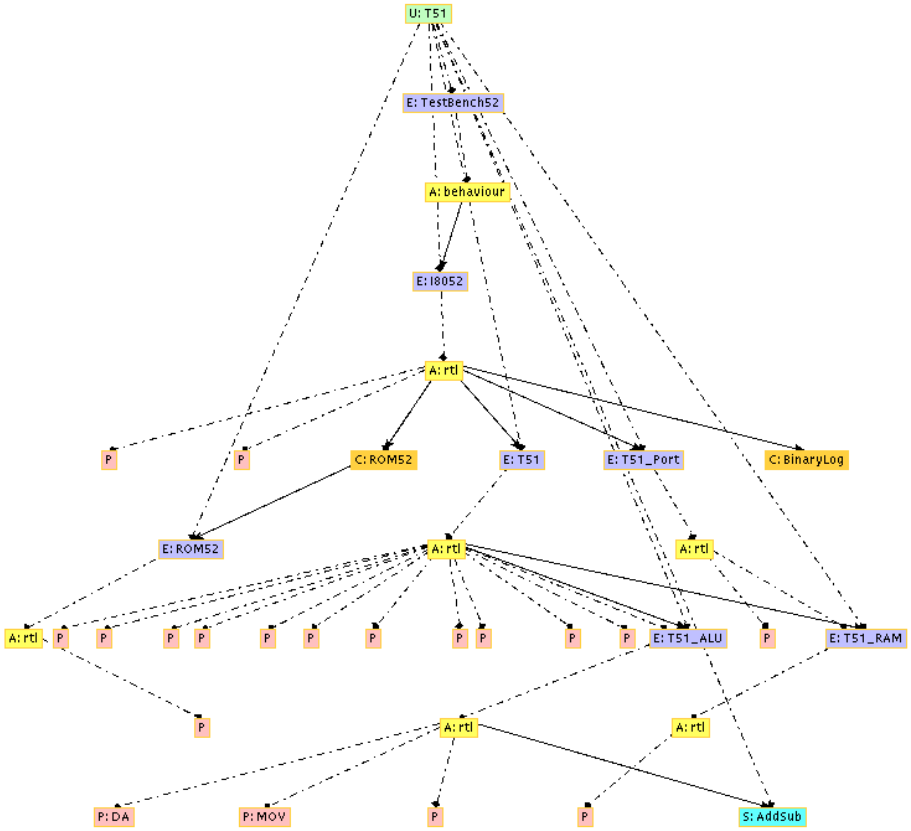


10.40 T51

Project information	
Project name	T51
Author	Daniel Wallner
Project Size	8 files, 87k
Archive	T51_0148.zip

From original documentation: T51 is a 8051 compatible microcontroller core: No MUL, no DIV, no external RAM memory, only one interrupt.

Syntax object graph:



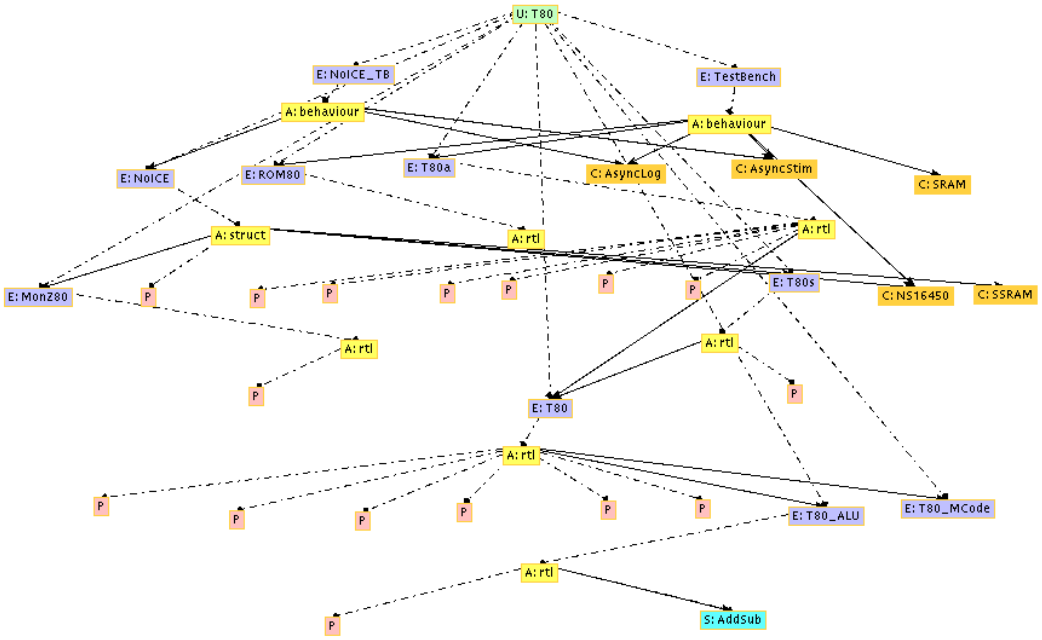
10.41 T80

Project information	
Project name	T80
Author	Daniel Wallner
Project Size	11 files, 220k
Archive	T80_0208.zip

From original documentation: T80 is a Z80 compatible microcontroller core; two architectures are provided:

- synchronous top level; with different timing than the original Z80 and inputs needs to be synchronous, and outputs may glitch;
- asynchronous top level.

Syntax object graph;



Chapter 11

Validation project base

This chapter describes projects belonging to the validation set exactly as previous chapter describe the ones belonging to the tuning set.

Project summary	
Number of projects:	19
Number of VHDL files:	469
Number of VHDL lines:	222,188
Cumulative size:	12.0M
Number of projects:	19
Number of entities:	571
Number of architectures:	570
Number of component declarations:	634
Number of component instantiations:	35,478
Number of subprogram declarations:	298
- of which: functions:	152
- of which: procedures:	146
Number of ports:	4,984
Number of signals:	39,017
Number of variables in processes:	2,449
Number of variables in subprograms:	299

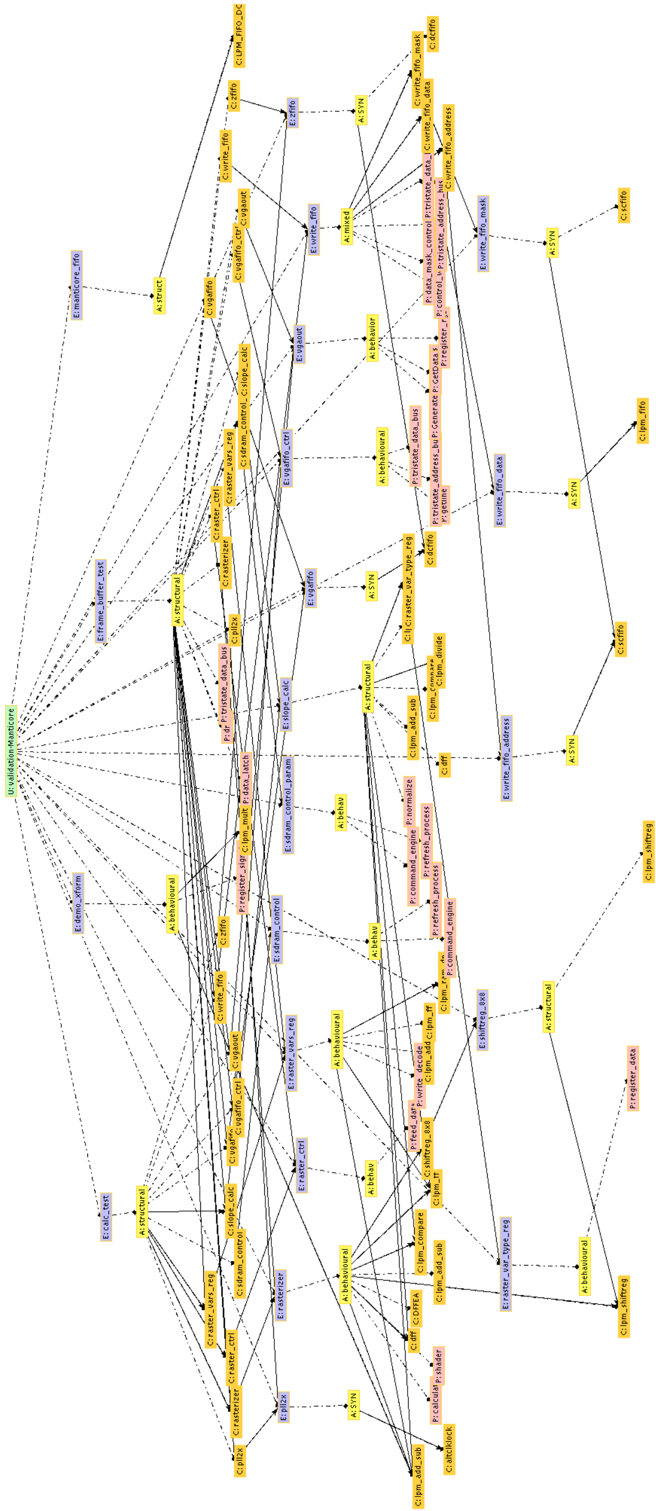
Again, for each project a brief summary is given, exactly structured as in previous chapter.

11.1 Manticore

Project information	
Project name	Manticore
Author	Benj Carson and Jeff Mrochuk, http://icculus.org/manticore/
Project Size	23 files, 336k
Archive	manticore.tar.gz

From original documentation: Manticore is an open source hardware design for a 3D graphics accelerator. It is written entirely in VHDL. It is currently capable of rendering triangles on a VGA display. The design includes a VGA output module, an open source (written entirely by the authors) SDRAM controller and a triangle rasterizer. Eventually it will incorporate standard 2D graphics primitives, multiple resolutions and colour depths, hardware lighting support and a PCI or perhaps AGP interface. The design was originally developed on an Altera APEX20K200E FPGA and Nios development board. The design was able to operate at 50MHz with this hardware. Ultimately, an open board design will be developed, creating an entirely open source PC graphics accelerator.

Syntax object graph:

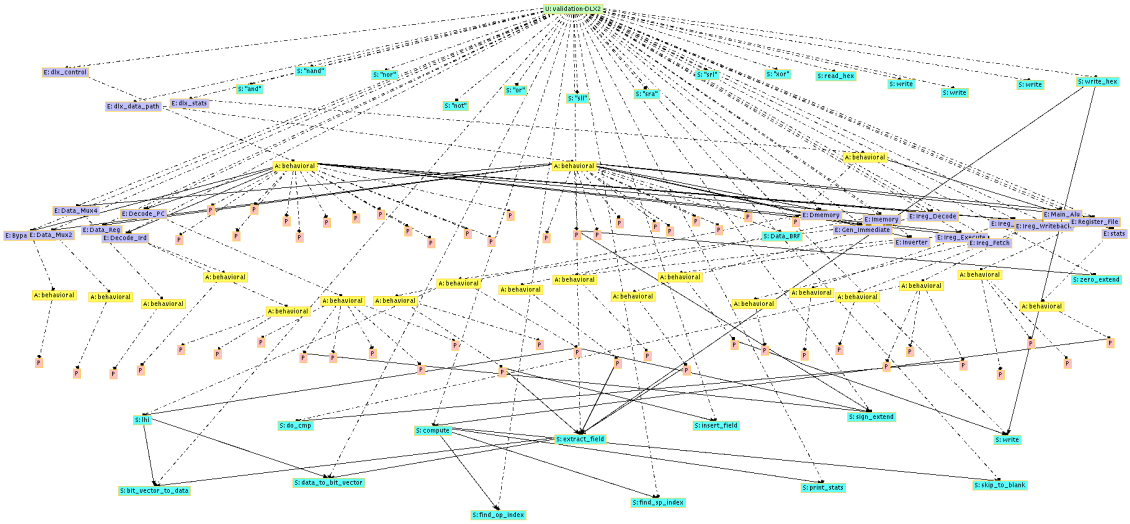


11.2 DLX2

Project information	
Project name	DLX2
Author	Avaneendra Gupta, Paul R. Stephan
Project Size	20 files, 80k
Archive	dlx2.tar.gz

From original documentation: a behavioral level model of the DLX instruction set architecture designed using VHDL. The model implements all the DLX instructions described in the text "Computer Architecture: A Quantitative Approach" by Patterson and Hennesey, except the floating point instructions and the multiply/divide instructions. The model implements a 5-stage pipeline, and complete 2-level data-bypassing.

Syntax object graph:

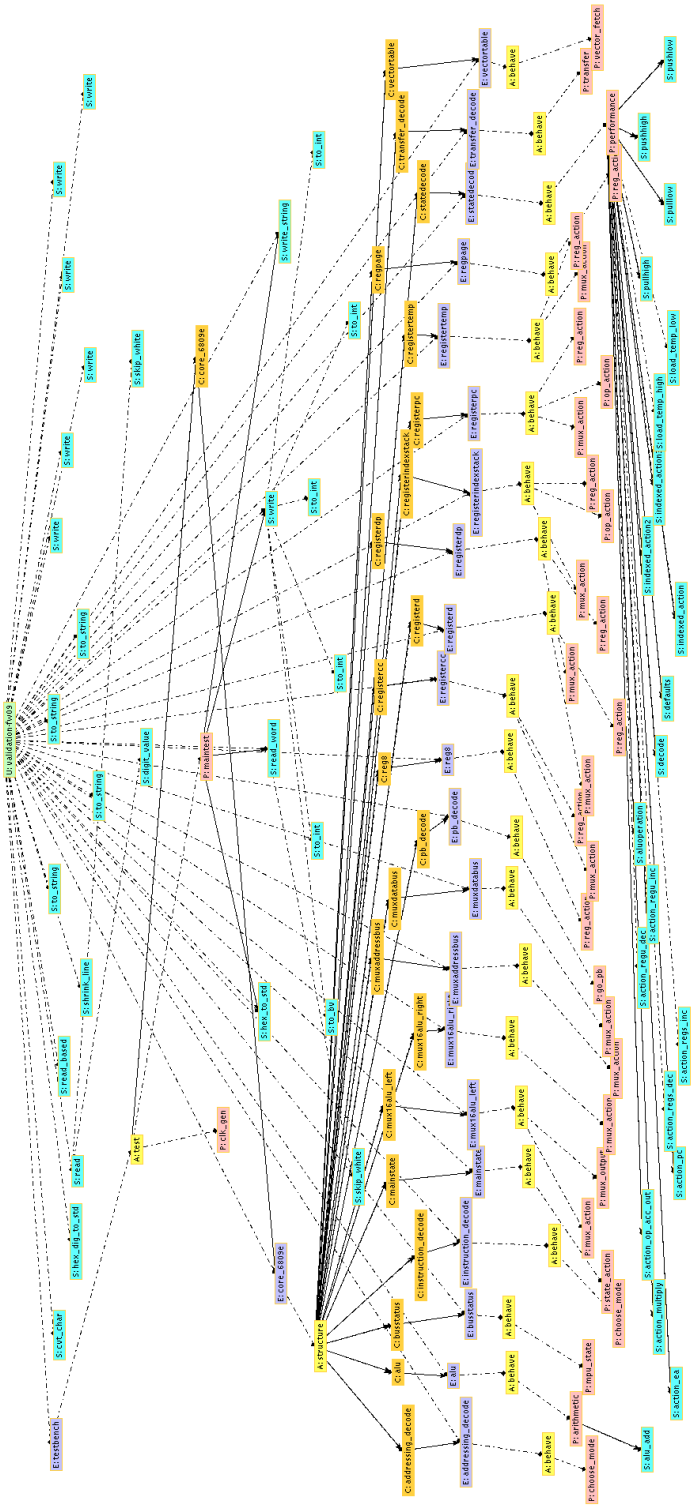


11.3 fw09

Project information	
Project name	fw09
Author	Flint Weller
Project Size	26 files, 239k
Archive	fw09vhdl.zip

A structural model of the mc6809e processor.

Syntax object graph:



(note: in the above SOG representation, $S \triangleright S$ links were not represented since it would have rendered the graph exageratedly complicated)

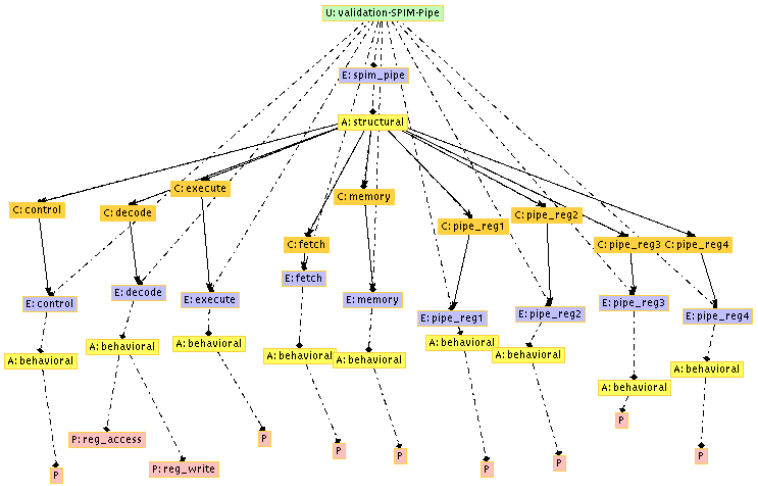
11.4 SpimPipe

Project information	
Project name	SpimPipe
Author	Sudhakar Yalamanchili, Georgia Institute of Technology, Atlanta GA
Project Size	10 files, 27k
Archive	pipelined_spim.zip

From original documentation: a model for an elementary pipelined SPIM, i.e., no forwarding, hazard detection etc. Pipeline registers are explicitly provided rather than have a synthesis compiler infer them or rather than placing them inside the individual modules. This model is structured in this manner so that students explicitly see the modules and can work with them in assignments.

This model was originally synthesized under the Altera 7.21 student Edition. More recently the model provided here was compiled under Aldec version 3.6 for simulation.

Syntax object graph:

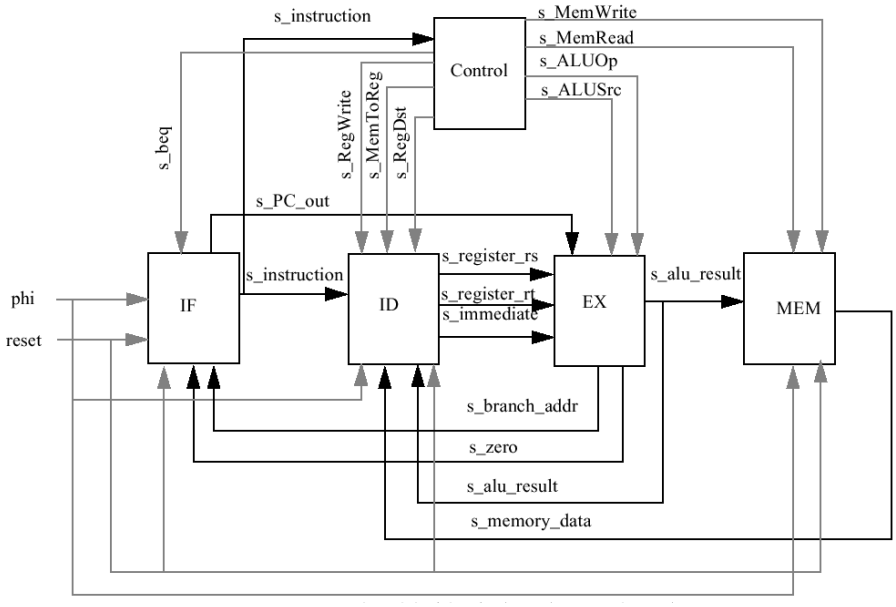


11.5 Spim

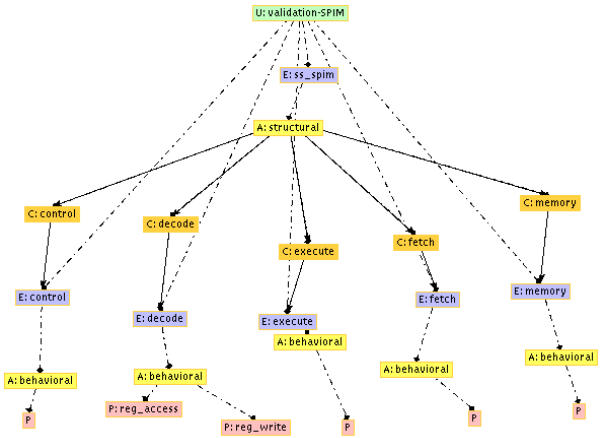
Project information	
Project name	Spim
Author	Sudhakar Yalamanchili, Georgia Institute of Technology, Atlanta GA
Project Size	6 files, 18k
Archive	single_cycle_spim.zip

From original documentation: a model for an elementary single cycle SPIM datapath. This model was originally synthesized under the Altera 7.21 student Edition. More recently the model provided here was compiled under Aldec version 3.6 for simulation.

Block diagram:



Syntax object graph:



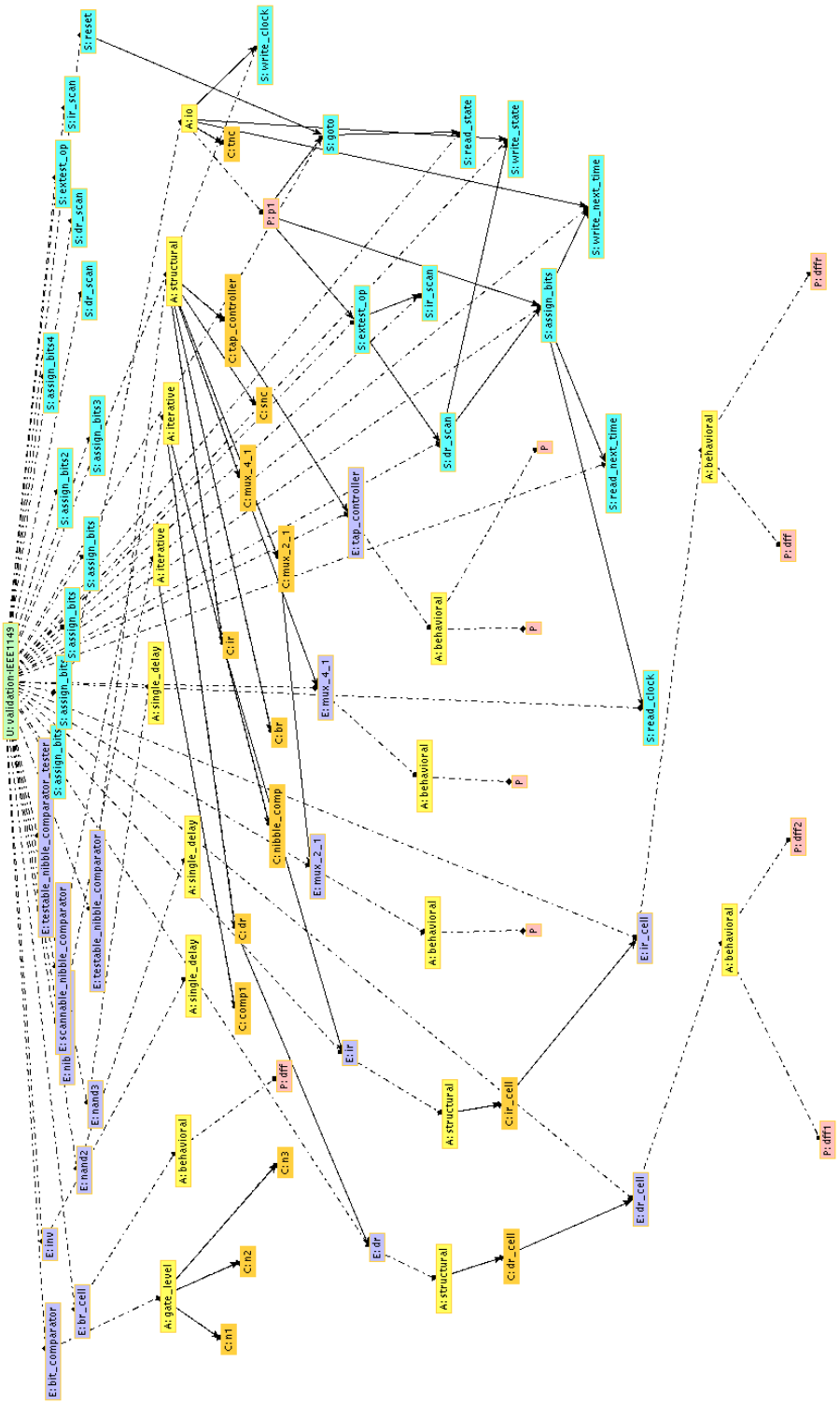
11.6 IEEE1149

Project information	
Project name	IEEE1149
Author	Peter M. Campbell, Zainalabedin Navabi, Northeastern University, Boston MA
Project Size	19 files, 81k
Archive	IEEE_1149.tar.Z

From original documentation: the implementation of IEEE Std 1149.1- 1990, IEEE Standard Test Access Port and Boundary-Scan Architecture, using behavioral VHDL. The IEEE 1149.1 standard provides a structured method for implementing testability in circuit designs and may be used to provide many different levels of testability. By implementing IEEE Std

1149.1-1990 in VHDL, designs which use the standard may be constructed and simulated to determine the operation of the design and the effectiveness of the included testability. This paper describes the basic components of IEEE 1149.1 as well as the test bench used to stimulate the finished logic. The test bench includes low-level and high-level functions which ease the test application process, provide high- level control, and are portable between different implementations of the test logic. An example which employs the test logic and uses the test bench functions for test application is included.

Syntax object graph:


U: validation:IEEE1149
E: bit_comparator
E: br_cell
E: inv
E: nand2
E: nand3
E: nib
E: scannable_nibble_comparator
E: testable_nibble_comparator
E: testable_nibble_comparator_tester
S: assign_bit
S: assign_bits
S: assign_bits2
S: assign_bits3
S: assign_bits4
S: dr_scan
S: extest_op
S: ir_scan
S: reset
S: write_clock
A: gate_level
A: behavioral
C: n1
C: n2
C: n3
P: dff
A: single_delay
C: comp1
C: dr
E: dr
A: structural
C: dr_cell
E: dr_cell
C: ir
E: ir
C: ir_cell
E: ir_cell
C: br
E: mux_4_1
C: nibble_comp
E: mux_2_1
A: iterative
C: mux_4_1
C: mux_2_1
C: tap_controller
E: tap_controller
C: tmc
A: io
S: goto
P: p1
S: read_state
S: write_state
S: write_next_time
S: assign_bits
S: read_next_time
S: read_clock
P
P: dffr
P: dffc
P: dffi

11.7 LFSR

Project information	
Project name	LFSR
Author	Karen M. Serafino and Michael A. Dukes, U.S. Government
Project Size	21 files, 53k
Archive	lfsr.tar.gz

From original documentation: VHDL Code for a Linear-Feedback Shift Register accompanying WL Tech Report WL-TR-91-5037.

Block diagram:

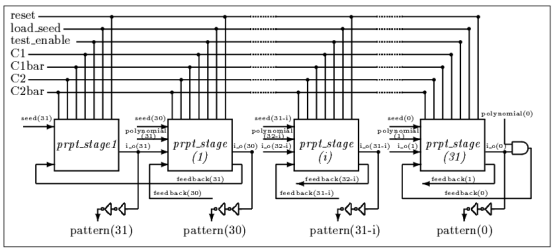


Figure 3. Schematic of **prpt**.

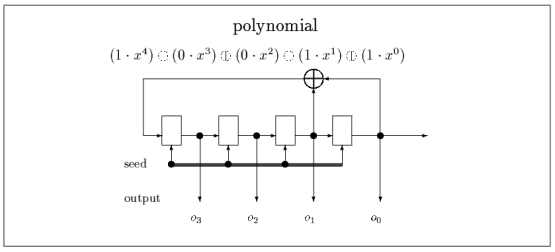
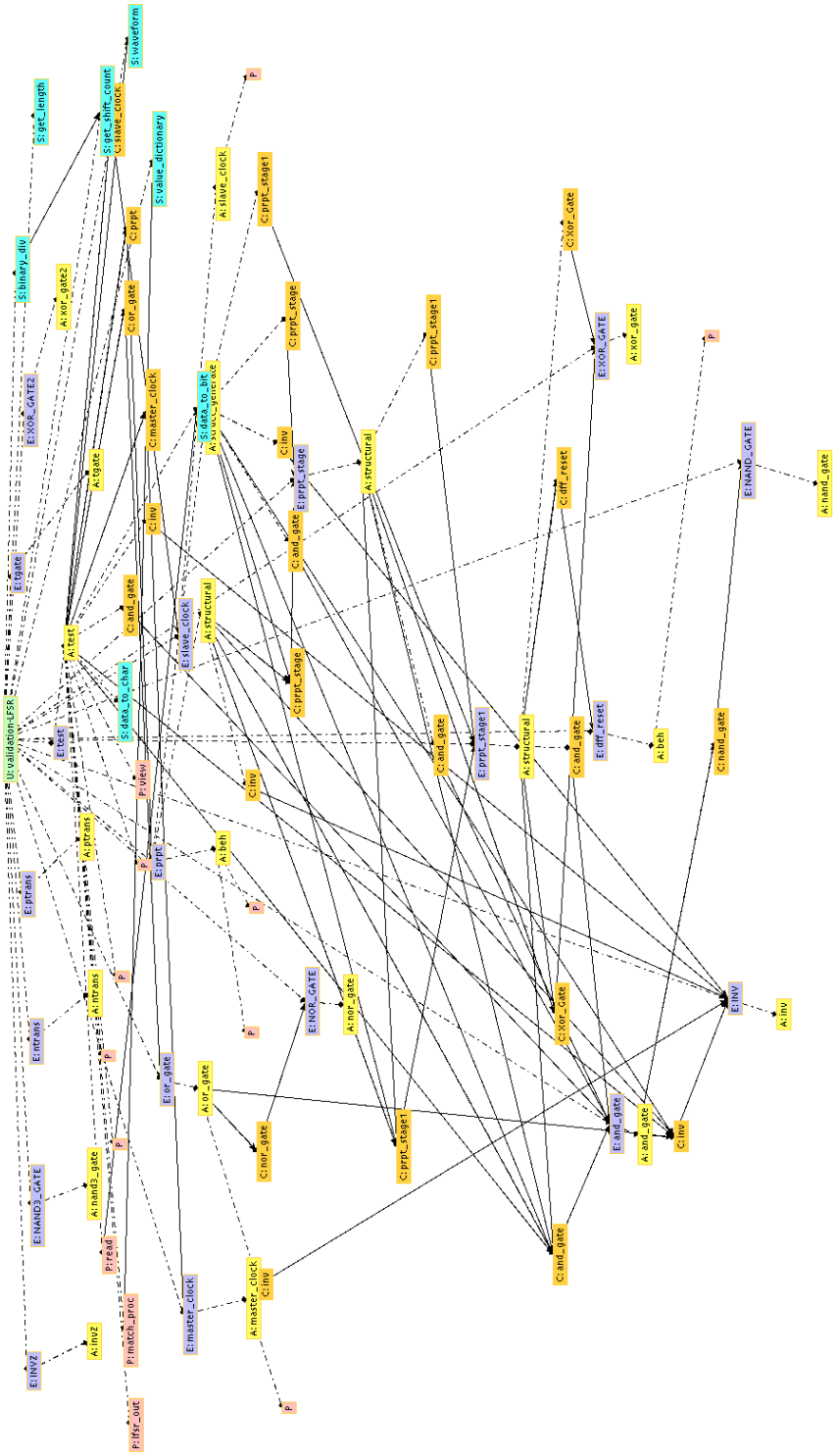


Figure 4. Schematic of 4-Stage **prpt**.

Syntax object graph:


U:validation-LFSR
S:binary_div
S:get_length
S:get_shift_count
S:slave_clock
S:waveform
S:value_dictionary
C:prpt
A:xor_gate2
E:XOR_GATE2
C:or_gate
C:master_clock
A:tgate
E:tgate
C:inv
C:and_gate
A:test
E:reset
S:data_to_char
P:view
E:prpt
A:ntrans
E:ntrans
A:ptrans
E:ptrans
A:beh
E:or_gate
A:or_gate
C:nor_gate
E:NOR_GATE
A:nor_gate
E:NAND3_GATE
A:nand3_gate
P:read
P:match_proc
P:lfsr_out
E:INVZ
A:invz
E:master_clock
A:master_clock
C:inv
P
A:slave_clock
C:prpt_stage1
C:prpt_stage
E:prpt_stage
A:structural
C:and_gate
E:prpt_stage1
C:diff_reset
E:diff_reset
C:xor_gate
E:XOR_GATE
A:xor_gate
E:NAND_GATE
A:nand_gate
C:nand_gate
E:INV
A:inv
C:xor_Gate
E:and_gate
A:and_gate

11.8 HDLLib

Project information	
Project name	HDLLib
Author	Eduardo Augusto Bezerra
Project Size	33 files, 123k
Archive	HDLLib.zip

From original documentation: HDLGen (c) 2000 by Eduardo Augusto Bezerra, version 0.1beta, 28/02/2000.

Block diagram:

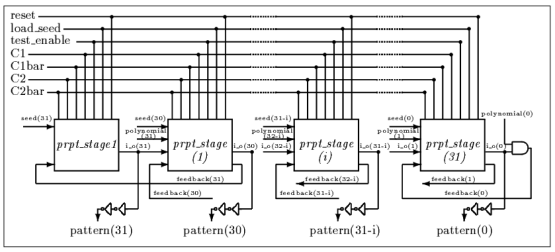


Figure 3. Schematic of **prpt**.

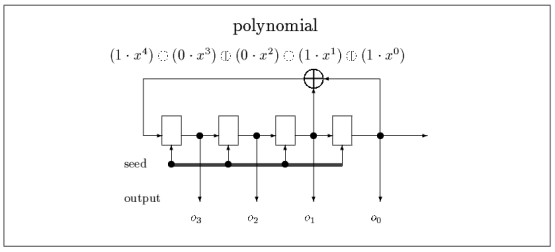
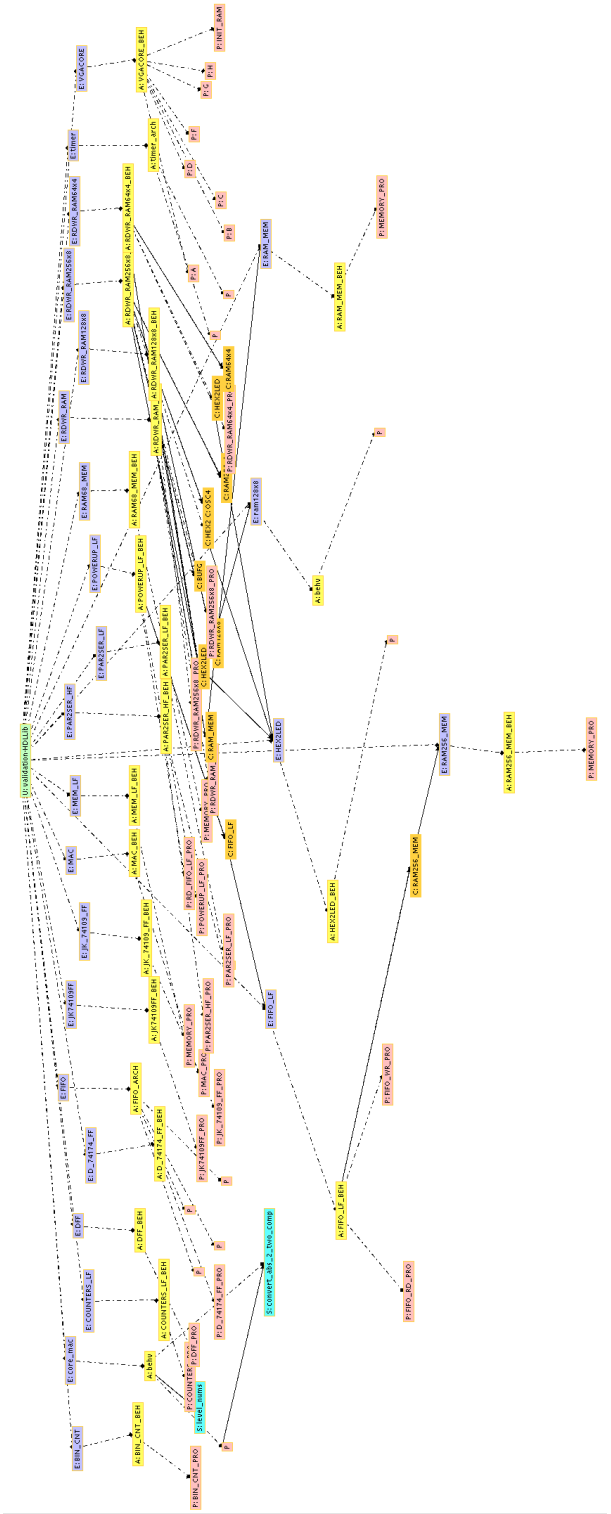


Figure 4. Schematic of 4-Stage **prpt**.

Syntax object graph:

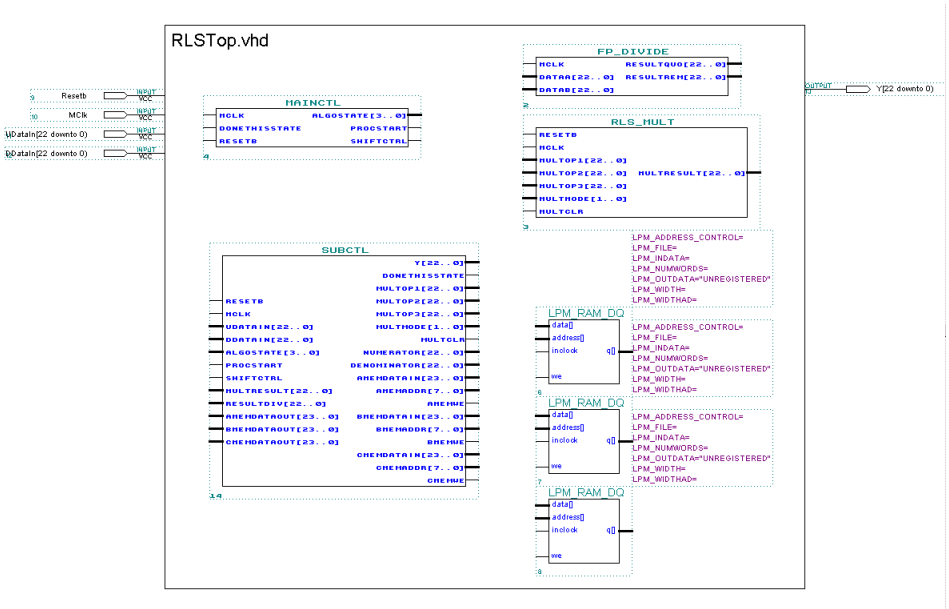


11.9 RLS filter

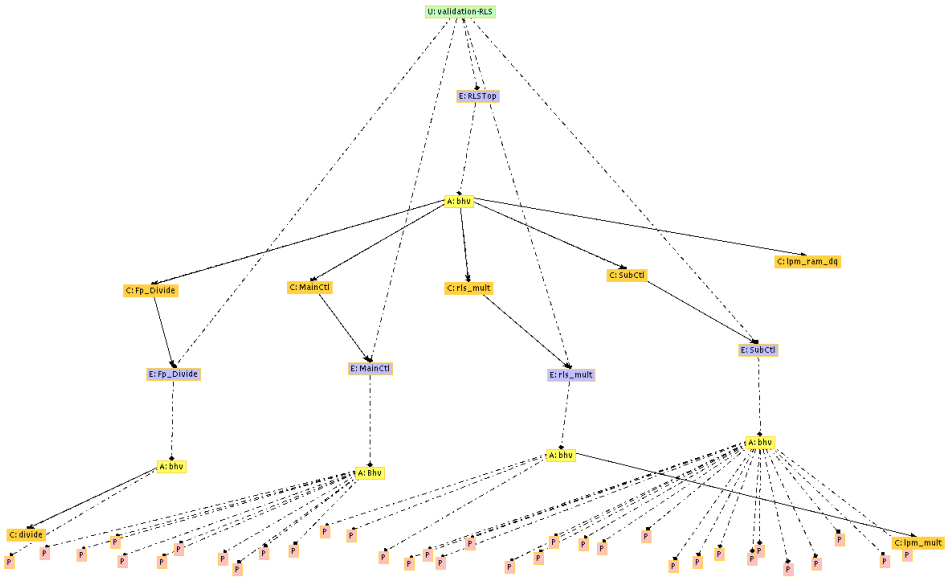
Project information	
Project name	RLS filter
Author	Kyeong Keol Ryu, Yeo-Sun Yoon, Taehyung Lee
Project Size	5 files, 54k
Archive	rls_filter.zip

From original documentation: RLS filter-based noise suppressor.

Block diagram:



Syntax object graph:

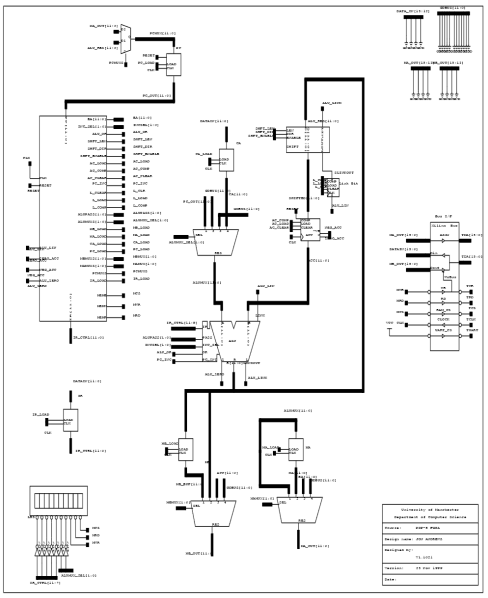


11.10 PDP-8

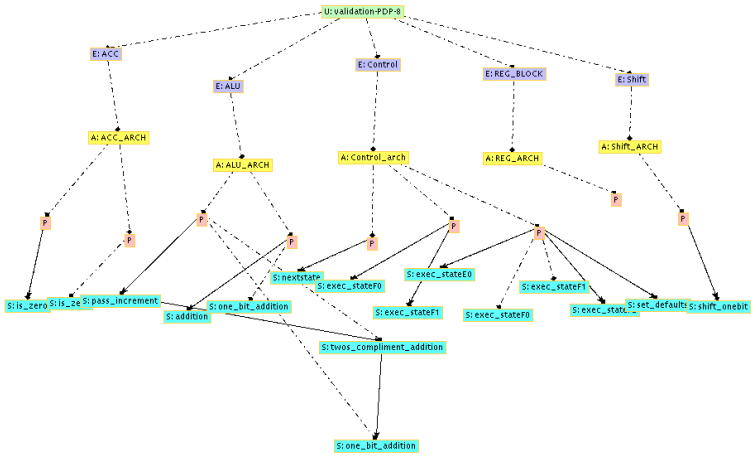
Project information	
Project name	PDP-8
Author	Jon Andrews, University of Manchester, UK
Project Size	8 files, 35.7k
Archive	pdp8.zip

From original documentation: a PDP-8 compatible processor implemented into a Xilinx 4000 Series FPGA (Field Programmable Gate Array). Instruction Format and Decode Ready, Datapath Complete, State Machine reduced to 5 states from 13, ALU currently being written in VHDL. Auto-Index state removed, this value is now calculated as part of the indirect path and then used conditionally, Microcoded Instructions now fit into existing 4 states and don't require looping in E0. Control Draft Made. TAD Z and TAD I Instructions execute successfully. The fully basic instruction set is working. The design in now running real PDP8 code at a speed of 5 MHz. This means that a TAD instruction will execute in 60ns. The refined state model means that all instructions complete under 100ns (for a skip instruction). Also all microcoded instruction run at 80ns, independent of the number of microcodes in that instruction.

Block diagram:



Syntax object graph:



11.11 RTC

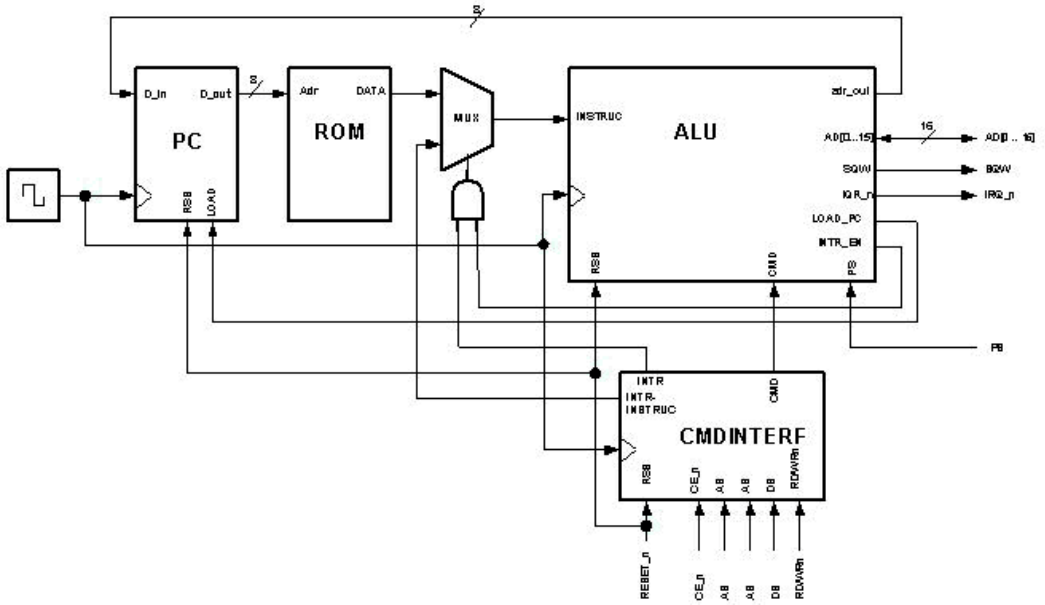
Project information	
Project name	RTC
Author	Bako Nebila
Project Size	173 files, 334k
Archive	pdp8.zip

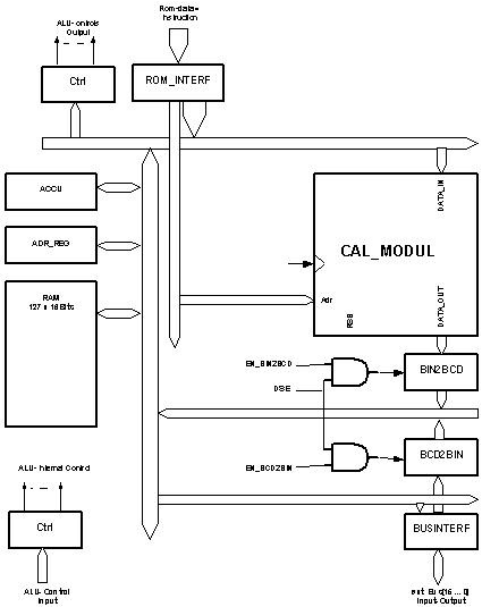
From original documentation (translated from German): a Real Time Clock compatible with Motorola MC146818 von Motorola, except for the Address/Data bus which is here 16 bit wide and for the quartz which is not required.

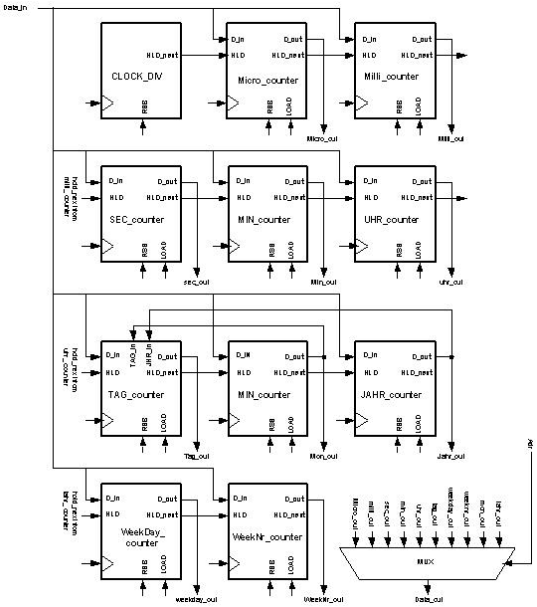
Additional features:

- micro and milliseconds can be read and written;
- the week number (1 through 52) can be read and written;
- the century can be read and written (store 20 for years like 2000, 19 for years like 1997);
- 57 Bytes available to the user;

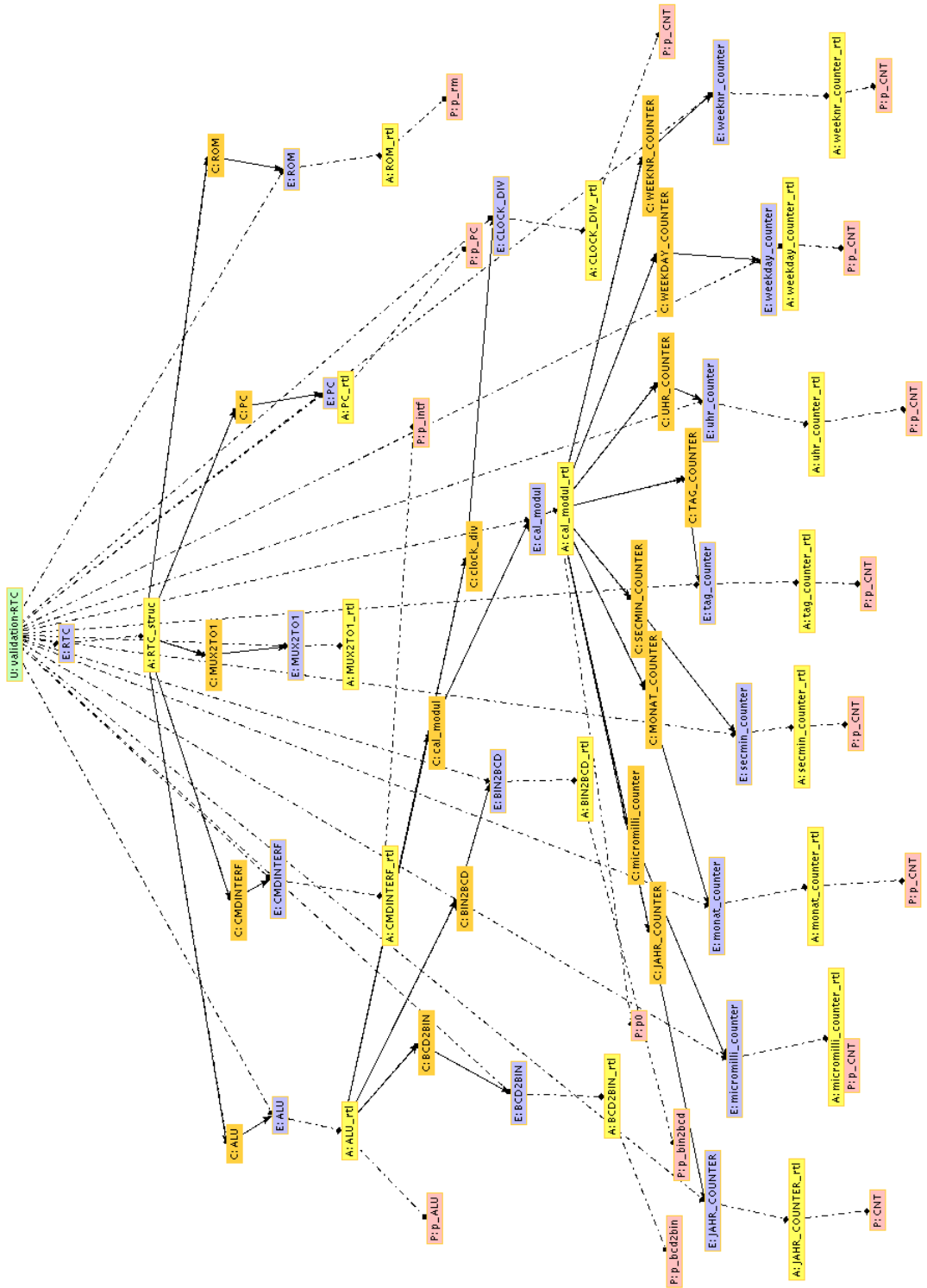
Block diagram:







Syntax object graph:

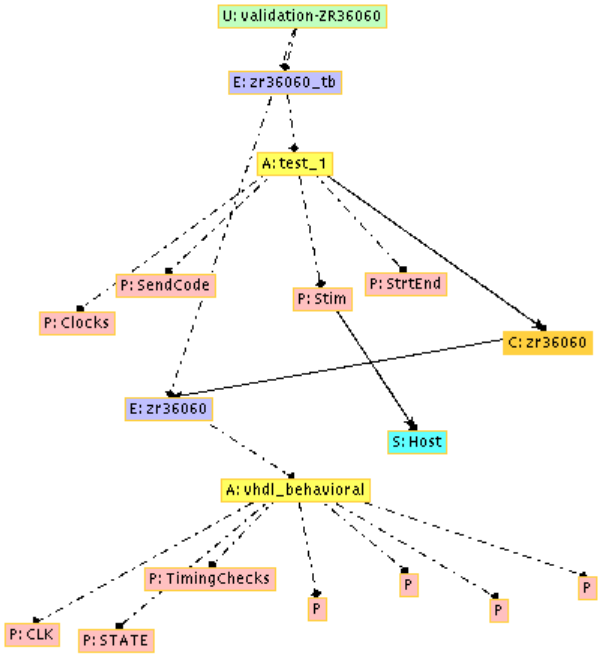

U: validation-RTC
E: RTC
A: RTC_struc
C: MUX2TO1
E: MUX2TO1
A: MUX2TO1_rtl
C: CMDINTERF
E: CMDINTERF
A: CMDINTERF_rtl
C: ALU
E: ALU
A: ALU_rtl
P: p_ALU
C: BCD2BIN
E: BCD2BIN
A: BCD2BIN_rtl
P: p_bcd2bin
C: BIN2BCD
E: BIN2BCD
A: BIN2BCD_rtl
C: cal_modul
C: clock_div
E: cal_modul
A: cal_modul_rtl
C: micromilli_counter
C: JAHR_COUNTER
E: micromilli_counter
A: micromilli_counter_rtl
P: p_CNT
P: p0
P: p_bin2bcd
E: JAHR_COUNTER
A: JAHR_COUNTER_rtl
P: CNT
C: MONAT_COUNTER
E: monat_counter
A: monat_counter_rtl
P: p_CNT
C: SECMIN_COUNTER
E: secmin_counter
A: secmin_counter_rtl
P: p_CNT
C: TAG_COUNTER
E: tag_counter
A: tag_counter_rtl
P: p_CNT
C: UHR_COUNTER
E: uhr_counter
A: uhr_counter_rtl
P: p_CNT
C: WEEKDAY_COUNTER
E: weekday_counter
A: weekday_counter_rtl
P: p_CNT
C: WEEKNR_COUNTER
E: weeknr_counter
A: weeknr_counter_rtl
P: p_CNT
P: p_intf
C: PC
E: PC
A: PC_rtl
P: p_PC
C: ROM
E: ROM
A: ROM_rtl
P: p_rm
E: CLOCK_DIV
A: CLOCK_DIV_rtl
P: p_CNT

11.12 ZR36060

Project information	
Project name	ZR36060
Author	Free Model Foundry, http://vhdl.org/fmf/
Project Size	3 files, 112k
Archive	zr36060.tar.gz

From original documentation: this is a partial model of the Zoran ZR36060 Integrated JPEG CODEC. This model simulates register loading and decompression bus cycles. When fed a JPEG file, it will find the start and end of image markers but will not decompress the image. It works in decompression mode only. The testbench became out of sync with the model once corrections started to be made based on hardware tests.

Syntax object graph:

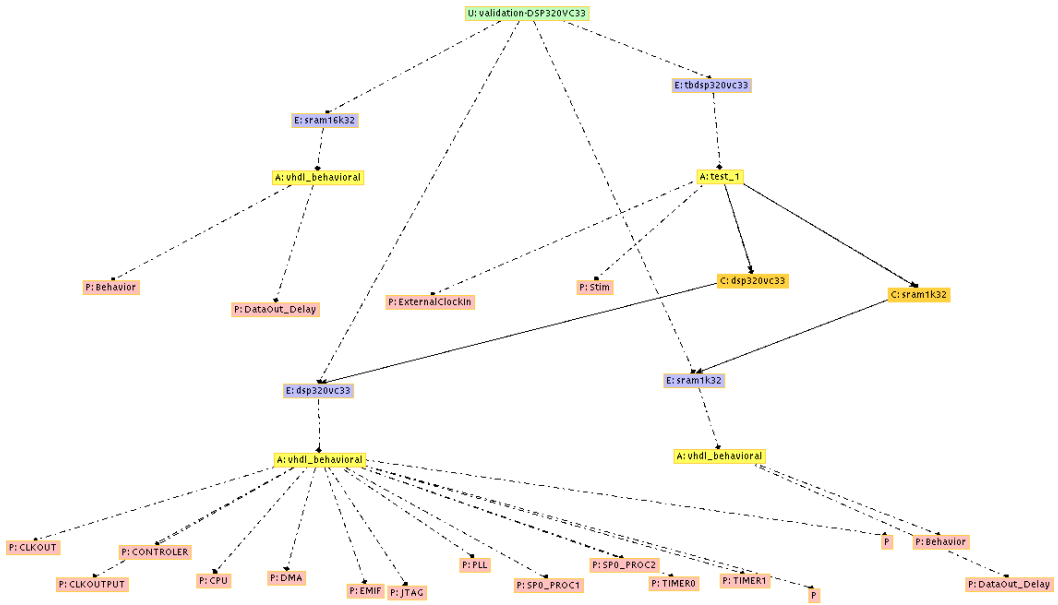


11.13 DSP320VC33

Project information	
Project name	DSP320VC33
Author	Free Model Foundry, http://vhdl.org/fmf/
Project Size	4 files, 253k
Archive	dsp320vc33_20020210.tar.gz

From original documentation: partial model of DSP320VC33.

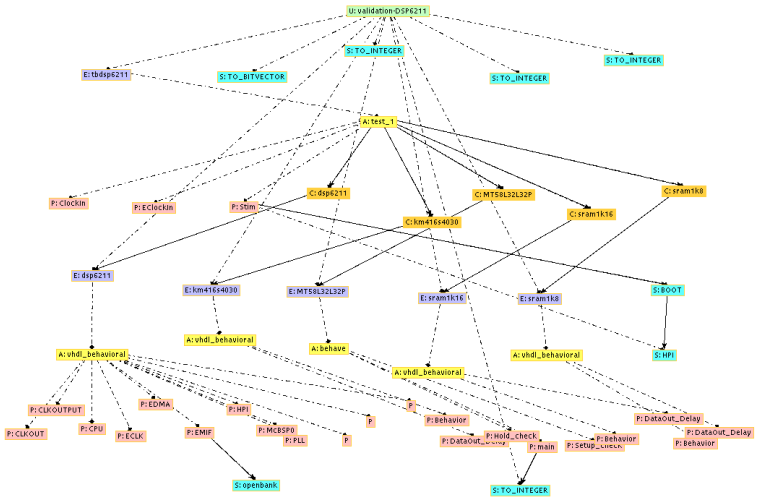
Syntax object graph:



11.14 DSP6211

Project information	
Project name	DSP6211
Author	Free Model Foundry, http://vhdl.org/fmf/
Project Size	8 files, 429k
Archive	dsp6211_20001105.tar.gz

From original documentation: model of DSP6211.
Syntax object graph:

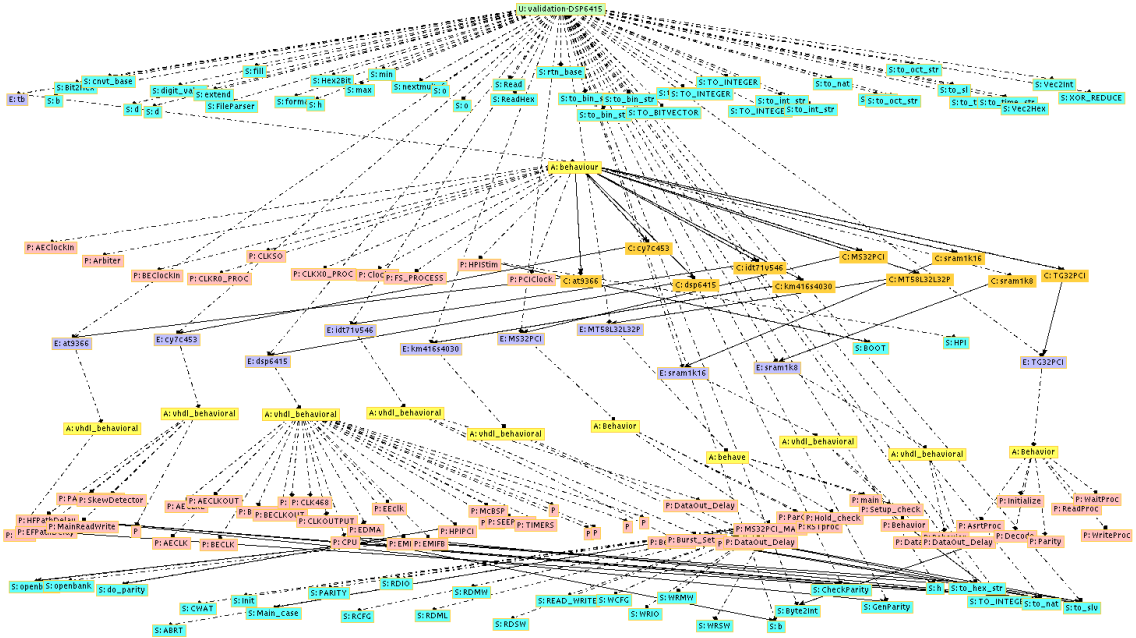


11.15 DSP6415

Project information	
Project name	DSP6415
Author	Free Model Foundry, http://vhdl.org/fmf/
Project Size	8 files, 429k
Archive	dsp6415_20020722.tar.gz

From original documentation: model of Fixed Point Digital Signal Processor, compatible with Texas Instruments TMS320C6415.

Syntax object graph:



(note: in the above SOG representation, $S \triangleright S$ links were not represented since it would have rendered the graph exageratedly complicated)

11.16 AMCC5933

Project information	
Project name	AMCC5933
Author	Papillon Research Corp.
Project Size	3 files, 58k
Archive	AMCC5933.zip

Behavioral model of AMCC S5933 PCI bus interface chip, designed for use in the Modeltech simulation environment.

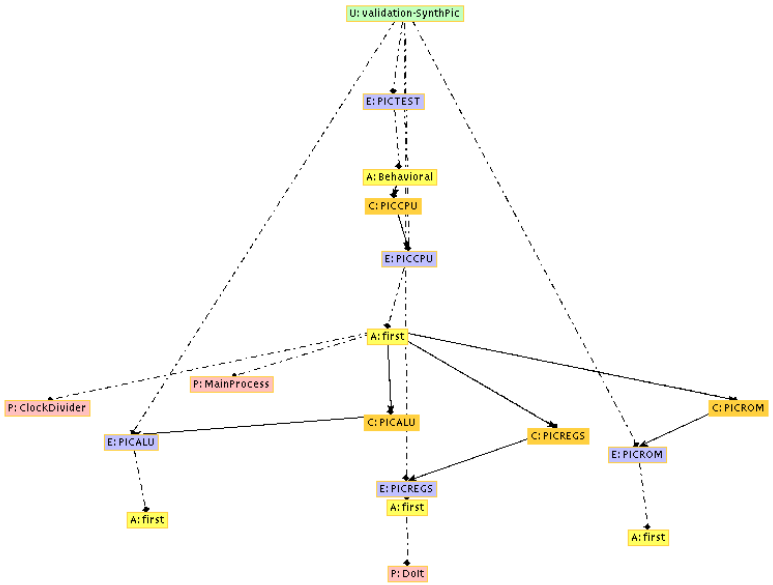
11.17 SynthPic

Project information	
Project name	SynthPic
Author	Thomas A. Coonan
Project Size	5 files, 38k
Archive	synthpic.zip

The Synthetic PIC is a synthesizable VHDL description of the basic Microchip PIC 16C5X microcontroller. It is written in the ViewLogic VHDL environment, it has successfully been synthesized to the XC4000 family, although it is not particular to Xilinx. The intent of the model is to provide a starting point for using the PIC architecture as a "core" for an FPGA, ASIC, etc. This model does not attempt to emulate the PIC with absolute fidelity, rather, it is a good starting point for spinning your own core.

This code has been successfully functionally simulated with Viewlogic's ViewSim and was successfully synthesized towards the XC4xxx family with the Vantage VHDL synthesizer. This code has not yet been actually embedded in an ASIC or FPGA, although that is its purpose.

Syntax object graph:

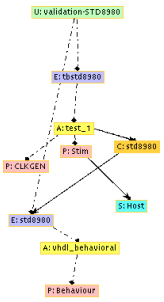


11.18 STD8980

Project information	
Project name	STD8980
Author	Free Model Foundry, http://vhdl.org/fmf/
Project Size	2 files, 78k
Archive	std8980_20010129.tar.gz

This is a model of a Texas Intruments Embedded Test-Bus Controller, JTAG Tap Master with 8-Bit Generic Host Interface

Syntax object graph:

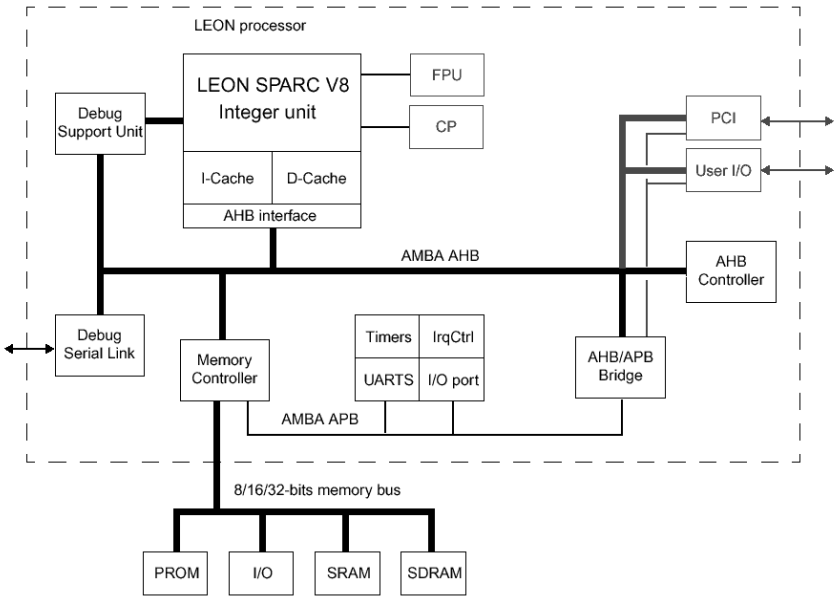


11.19 Leon 2

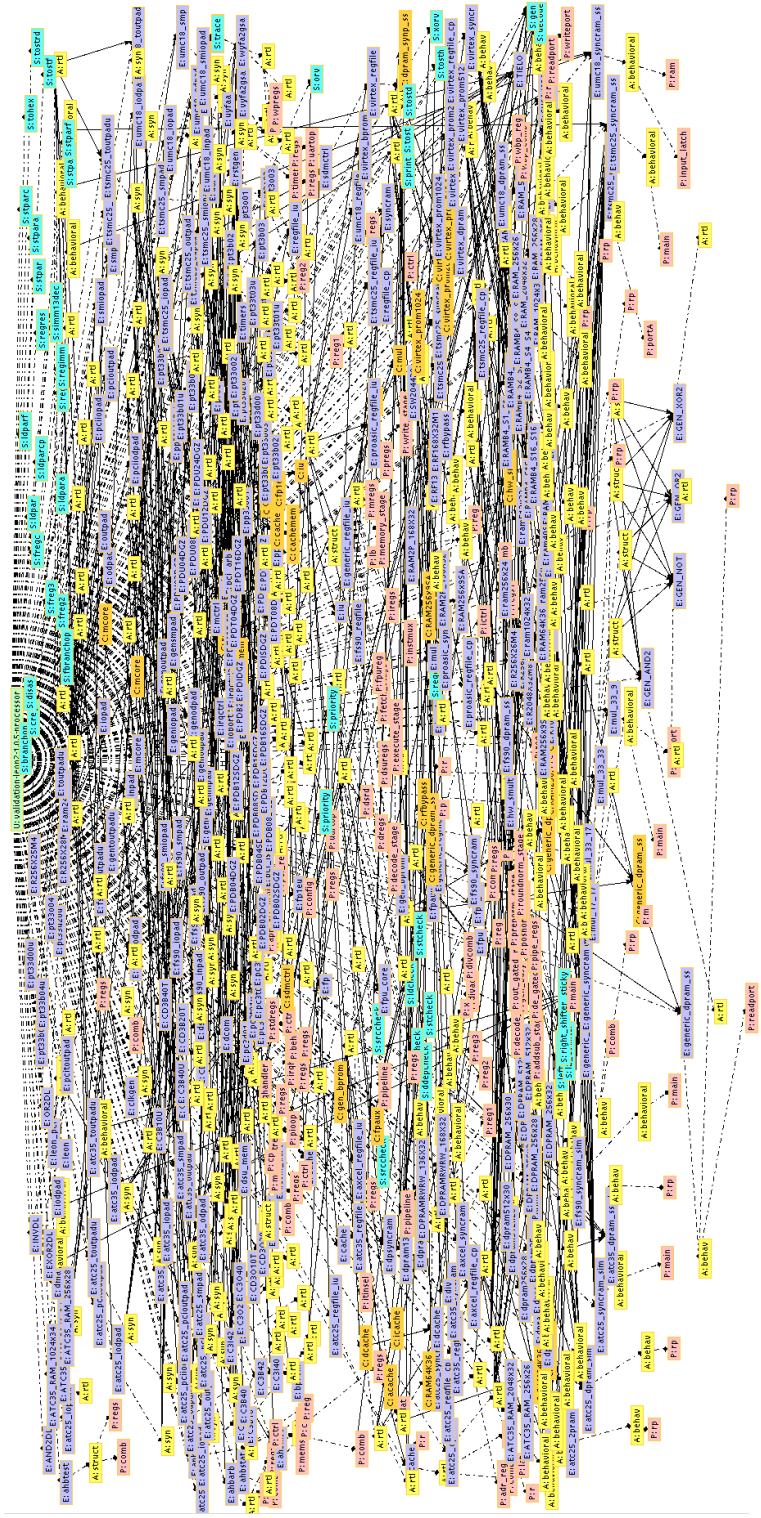
Project information	
Project name	Leon 2
Author	Jiri Gaisler, ESA/ETEC
Project Size	73 files, 6 845k
Archive	leon2-1.0.5.tar.gz

The LEON-2 VHDL model implements a 32-bit processor conforming to the SPARC V8 architecture. It is designed for embedded applications with the following features on-chip: separate instruction and data caches, hardware multiplier and divider, interrupt controller, debug support unit with trace buffer, two 24-bit timers, two UARTs, power-down function, watchdog, 16-bit I/O port and a flexible memory controller. New modules can easily be added using the on-chip AMBA AHB/APB buses. The VHDL model is fully synthesisable with most synthesis tools and can be implemented on both FPGAs and ASICs. Simulation can be done with all VHDL-87 compliant simulators.

Block diagram:



Syntax object graph:



Part IV

The Method

Chapter 12

Introduction to models

In part I, we introduced the abstractions of syntax object, bunch and SOG. We also proved that a SOG can be partitioned in bunches (thus being the length of a SOG equal to the sum of the lengths of its composing bunches), and that bunches themselves can be further partitioned in syntax objects, some of which are atomic from our point of view, whilst others can be further split in smaller syntax objects.

In order to reach the goal of this thesis, which is to estimate the size of a project (i.e., a estimate the size of a SOG), we will adopt a constructive approach, going from smallest elements which are known enough to give some estimate on them, to the largest element, which is the SOG itself.

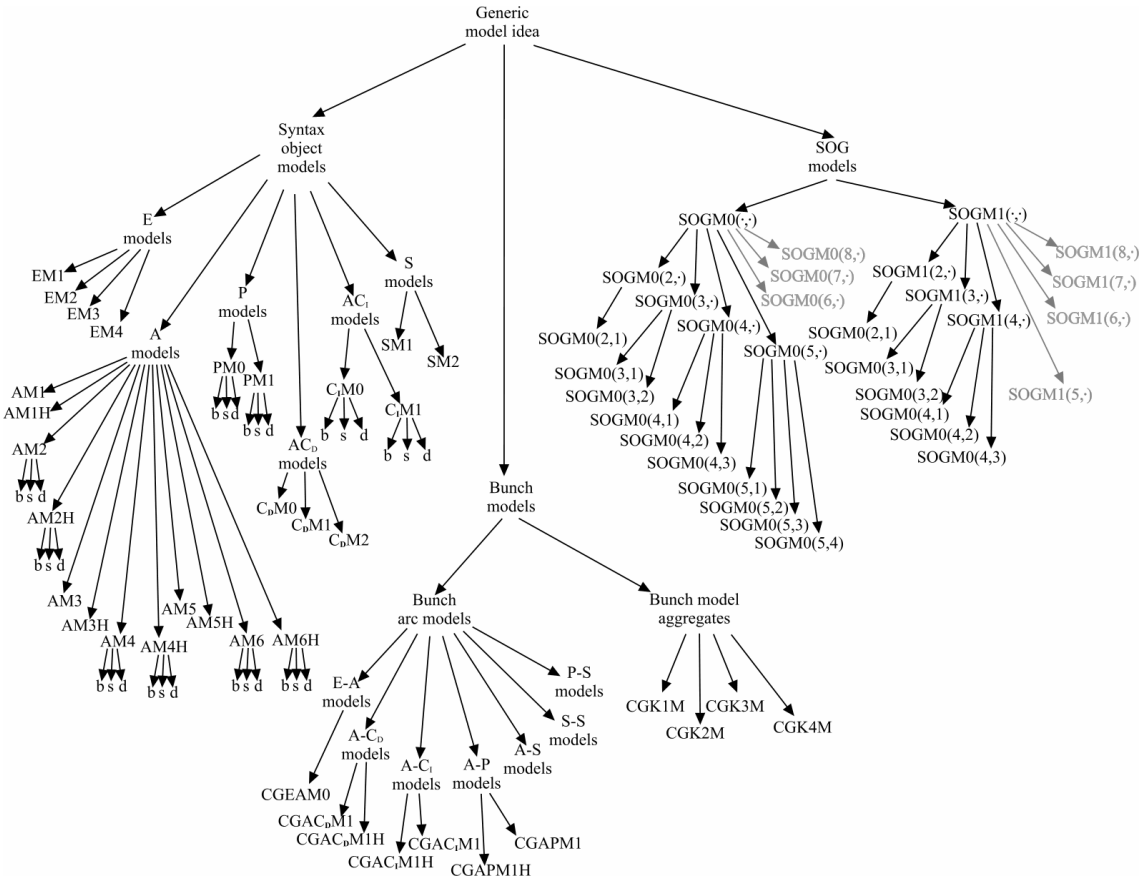
For each of the three level just mentioned (syntax object, bunch, project SOG), an appropriate methodology supported by suitable models will be proposed.

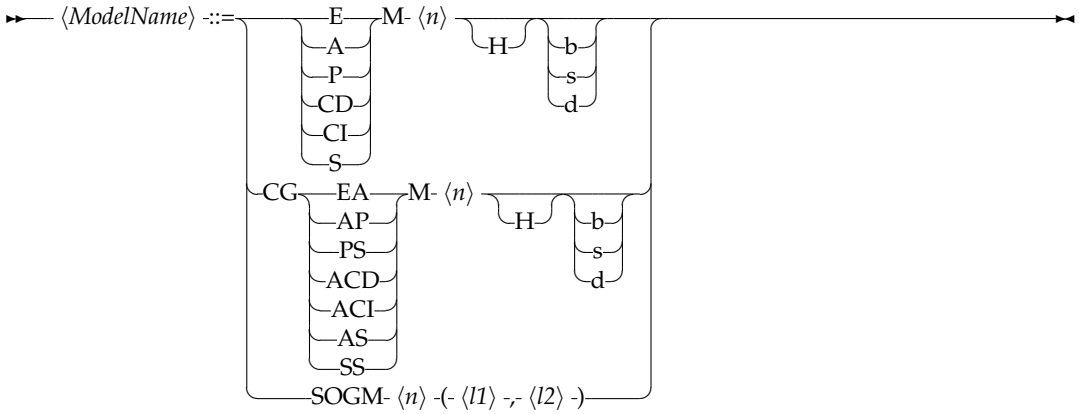
Each model tries to estimate at its best a quantity, which can be, for example, the length of an object, or the number of sub-objects of a given kind included in a given object. The variables used by each model are a subset of the variables available at that given knowledge condition of the KSOG. To be more precise, variables exhibiting the highest correlation coefficients with the data to be estimated are chosen.

Figure 12.1 represents in a tree view all the models presented in this thesis. All the models share some basic principles, that could be expressed by saying that they try to give an accurate size estimate of some object or aggregate, by knowing only a little information and exploiting it as much as possible (if information proves to be useful, i.e. correlated with the unknown variable). This general idea is represented by the root of the tree, "Generic model idea". From that root node, three different branches lead to models useful for estimation at syntax object level, bunch level and SOG level respectively.

For sake of clarity, each model is given a unique name, according to the following naming scheme:

Figure 12.1: The model tree





By looking at the above scheme, it should be easy to recognize three different horizontal lines which respectively generate syntax object-level model names, bunch-level model names and SOG-level model names. Syntax object-level model names have no prefix, and begin with a letter that identifies the type of the syntax object for which they estimate the length, e.g. E stands for entity, A for architecture, P for process, CD for component declara-

tion, CI for component instantiation and S for subprogram. Bunch-level model names have a "CG" prefix (it stands for containment graphs, which is an alternate name for bunches) and a middle part, which indicates the arc type for which they give an estimate of the cardinality of the set of contained elements. For example, when that middle part is "AP", that model tries to estimate $|R(a)|$ given the architecture a, where $R(a)$ is as as follows: $R(a) = \{p : P(p) \wedge a \supset p\}$; in practice that model estimates how many processes are contained in a given architecture. SOG-level model names have a "SOG" prefix, and a suffix with two positive numbers l_1 and l_2 which express the depth for which the size of the KSOG is known and the expected maximum depth of the SOG for which a size estimate is to be calculated.

Syntax object-level models, bunch-level models and SOG-level models will be thoroughly discussed in the next three chapters respectively.

All models have a identifying serial number (the n symbol); each model works on a different set of variables (input information). Despite common sense, in certain conditions it is reasonable to use trivial models, that is models not using any of the available information. Such trivial models are indicated with a serial number equal to 0; they are simple constants, whose value is defined thanks to some "a priori" considerations or is the average value of the considered population.

Sometimes information on the same characteristics of a given syntax object could be available in two flavors, namely as a simple count and as a sum of homogeneities. For example, information on entity ports could be expressed both as a port count or as the summation of homogeneity for the given ports. In those cases, models were prepared for both cases, and models using homogeneity information were tagged with a 'H' character, to distinguish them from models using simple count information.

Where possible, *mode* information is used. Here, with the *mode* word, we indicate how an architecture was described, namely in a behavioral, structural or data-flow fashion. Since we believe that when the designer has sufficient information to state the existence of an architecture in a KSOG, it should also have in mind a rough idea of how to implement it (i.e., its mode), we think that it would be a waste not to use mode information to develop better models. And mode information is indeed very valuable in many cases, e.g. the expected number of processes in an architecture marked by the designer as structural is close to zero. That estimate dramatically changes if the designer chooses to describe that architecture in a behavioral way, in this last case it is the expected number of component declarations and instantiations, to dramatically decrease with respect to the previous case. Models specialized for syntax objects described in a behavioral, structural or data-flow way (or syntax objects included in such objects) are respectively marked by a trailing 'b','s' or 'd' character.

By reading the following chapters and examining the mathematical structure of presented models, one could wonder why most of them are simple linear models. In most cases the answer is that modelled phenomena are so evidently linear, and so accurately approximated by linear models that it makes no sense to seek further (this happens for entity declarations). In other cases, modelled phenomena seem so complicated to be described by linear approximations, but linear models were chosen because more complicated ones (quadratic, logarithmic, exponential, mixed) did not any better in terms of accuracy. In a few cases a linear (or a trivial) model is proposed due to its simplicity, though there is experimental evidence that no model can do a good job at estimating the desired quantity with the available information, in practice when none of the available information exhibit any correlation with the desired quantity.

Each presented model will be accompanied by one or more specimens of a particular type of plot which is pretty common in this thesis: the *actual vs. estimated* plot. In this type of graph, actual values assumed by the quantity to be estimated are represented on

the X axis, while model-estimated values are represented on the Y axis. Reading this plot is an immediate operation: the accuracy of the model is good as much as the points on the plot distribute themselves on the quadrant bisector (which is always drawn). Points lying above the bisector represent overestimates, while points lying below are underestimates. Overestimates and underestimates are often due to coding style reasons (the designer tends to group more or less instructions per each line).

In addition, for each model, error distribution plots are given, which come in two flavours: an error distribution histogram, which mimics in a discrete way the error probability density function plot, and an error normal plot. This last plot deserves some additional explanation to be correctly understood: first of all the represented function is the error cumulative distribution function. Thus, a point located at coordinates $(10, 0.95)$ means that in 95% of the cases, the error assumes values less than or equal to 10. Secondly, it must be said that the cumulative distribution function is represented on a modified Y axis, plotted in such a way that a cumulative distribution function coming from a normal distribution, would appear on it as a straight line. Error normal plots were drawn using the `normplot`[1] function of the MATLAB package.

From an error normal plot it is easy to obtain error confidence intervals (such as "estimation error in 90% of the cases falls between -100 and +150 lines of code") and visually assess how closely the estimation error distribution approximates a normal distribution. When readability issues require it, two normal plots are reported at two different zoom levels; in that case, the first plot contains all the samples, whereas the zoomed version focuses on the most populated portion of the plot.

Please note that all the models here presented were tuned on human-written objects. They are not intended to be used in order to estimate size of test-benches and automatically-generated objects:

- **test benches** are so different in nature from normal embedded design code, that they require dedicated models. Test benches have more in common with conventional software listings for personal computer than with standard, embedded design, VHDL code.

  In fact, unlike normal entities, test bench entities have no ports, since they represent "the rest of the universe", and since there is nothing out of the universe by definition, there is no need to expose an interface towards nothing. On the contrary, all the other entities are thought to be instantiated in some larger picture, and need to talk to it in some way that is, in practice, a set of data lines, represented by an entity port.

  Test benches are thought to be useful debugging facilities, therefore they usually instantiate a small number of real embedded design entities which, inside the test bench, undergo a thorough test, in a sort of a question and answer conversation. Responsibility of the test bench is to mimic, as close as possible, the operating conditions under which the tested components will be actually used. The test bench will present a suitable series of inputs, give to the components the proper time to operate on it, then read

[1] the following text comes from original MATLAB documentation:

`normplot(X)` displays a normal probability plot of the data in X. For matrix X, `normplot` displays a line for each column of X. The plot has the sample data displayed with the plot symbol '+'. Superimposed on the plot is a line joining the first and third quartiles of each column of x.(A robust linear fit of the sample order statistics.) This line is extrapolated out to the ends of the sample to help evaluate the linearity of the data. If the data does come from a normal distribution, the plot will appear linear. Other probability density functions will introduce curvature in the plot.

what the components answer on their output data lines, and verify the correctness of the output.

Input series can be generated locally inside the test benches or are, in more complex cases, read from an external file, created by dedicated test case generation programs. The consequence of this is that test benches often use files, and file usage is a peculiarity that can never be found in real embedded design code.

Another, rather typical, feature exhibited by test benches is their attitude to communicate with their designer, thing that can be done by performing console I/O operation. In VHDL, it is easy to identify code working with the console by looking at its preamble and finding a

```
use std.textio.all;
```

clause. Almost any entity requiring the textio package is a test bench.

Avoiding the distinction between test benches and real embedded design code would result in biased models, and this claim was experimentally proven during the course of this thesis: all the models tuned indifferently on all objects show worse accuracy (that is, lower coefficients of correlation between actual and estimated data, and higher estimation error variance) with respect to models having the same mathematical structure but tuned on sets depurated from test benches.

Developing dedicated models for size estimation of test-bench objects is out of the scope of this thesis;

- **automatically-generated code** has, by definition, a cost which is equal to zero. In fact automatically-generated objects violate the fundamental hypothesis on which this thesis is based: human time required to develop a piece of code is proportional –on an average basis– to code length expressed in number of lines of code. Automatically-generated objects, instead, require a quantity of human time –that is, in practice, time to choose their dimensional design parameters– that is fixed, thus not proportional to generated code length. Moreover that time and is always negligible with respect to time required to implement the rest of the project. A good example of code that can be generated automatically are structural multipliers, to the discussion of which, one of the following paragraphs is devoted;
- **read-only memories** are used in several projects we analyzed. ROMs are memory objects for which the contents are defined, once for all, at design time; they have a number of applications in embedded design tasks, which we do not want to examine now. From the point of view of VHDL modelling, read only memory objects can be implemented mainly in two modes, and both are independent from the external parameters we can access, such as homogeneity of input/output signals. We decided to exclude ROM implementations from the scope of our models since their size is often remarkable, and it is not suitable to be estimated by models working on the external parameters, therefore they would have a non-negligible, negative effect on estimation error. ROMs will be better described later.

12.1 Homogeneity

Most of our models rely on this fundamental hypothesis: the complexity of a fragment of code (therefore, the effort and time required to develop it) is closely related to the complexity of data it processes.

This hypothesis is well known in the field of software engineering, and was developed as a well-established theory, properly supported by a large number of tools, under the name of *Function Points Methodology* (see [5], [6] and [7]). This thesis tries to port those results, with necessary adaptations, modifications and additions, to the field of embedded system design engineering.

In order to exploit the above hypothesis, we need to define a proper indicator of complexity of a given datum, that will be called *homogeneity*. In VHDL, data can appear in two form: signals and variables.

- **Signals** can be thought of as dynamically moving data; signals symbolize data lines where electrons physically move to transport information from a place to another one. Signals belong to the structural design paradigm, where functionalities are implemented as subsystems connected to each other thanks to signals, and working simultaneously in full parallelism.

- **Variables** can be thought of as statically persistent data; variables are *the* way *per antonomasia* to implement finite state machines, or to store a state of a sequential process, for any possible generalized meaning the word *state* could have. Variables belong to the behavioral design paradigm, where subsystems are thought of as machines performing one single task at a time in a sequential order, and no parallelism exists inside them;

Since signals and variables of the same type are interchangeable (that is, the value of a signal can be stored inside a variable, and a signal can be assigned the value of a variable), taking into account what we have said so far, it should become clear that signals and variables of the same type should have the same complexity, therefore the same homogeneity. Moreover, homogeneity value of a given signal or variable should depend only on its type. This is the reason why, from now on, we will speak of *homogeneity of type* instead of homogeneity of signal or of variable.

Given the above requirements, the first idea of homogeneity definition that comes in to one's head is the type width expressed in number of bits. In fact the idea that the complexity of a fragment of code depends on the number of bits it processes is not completely wrong (in fact it proves to be exact at least in the case of automatically generated structural multipliers, see § 12.2.2). Nevertheless, this hypothesis proves to be wrong (it fails to induce good models) for human-written code. Let's think of a behavioral or data-flow architecture which outputs the sum of a two integer input operands: that summation requires exactly one line of code, and to express the sum of two numbers you simply use the '+' operator of the VHDL language, without the need to specify any bit-level representation detail. Thus, summing two 32-bit integer operands or two 4096-bit integer operands requires exactly the same amount of code. This suggests that a positive integer number should have the same homogeneity, whatever its width is.

Homogeneity should express how much the information carried by a given type is complex to handle, that is, how many instructions are required to perform common operation on it.

After all the above considerations, we will go for the following formal definition of homogeneity:

Given a type T, we define the homogeneity $H(T)$ as follows:

- if T is a VHDL simple type[a], then $H(T) = 1$;
- if T is an array of a subtype S, thus it was declared in the following way:

```
type T is array (index_type) of S;
```

  then $H(T) = H(S)$;

- if T is a record of a collection of subtypes $S_i ... S_n$, thus it was declared in the following way:

```
type T is
  record
      v1 : S1;
      v2 : S2;
      ...
      vn : Sn;
  end record T;
```

  then $H(T) = \sum_{i=1}^{n} H(S_i)$;

[a] Simple types are all types except composite types, namely scalar types, access types and file types. Arrays and records are composite types, therefore not simple types.

Homogeneity is used often in our model as a type complexity indicator, but not always; it is used exactly in all those cases in which it proves to lead to greater accuracy than simpler count methods.

12.2 Application-based models

Syntax object models appearing in this thesis are classified by the kind of the syntax object whose size they estimate (e.g. entities, architectures, processes, ...), and this subdivision is widely justified by the fact that syntax object of different kinds exhibit such different characteristics, have such distant semantics, and make available such diversified information to the estimation algorithms, that it would make no sense to create a single model for all of them.

In addition, for each single syntax object kind, there exists a multiplicity of different models, and this is justified by two reasons: an experimental reason (we want to try more models working on different subsets of the available information, and select the ones that behave best), and a constraint due to knowledge conditions (since more knowledge condition are possible, and at a given knowledge condition only certain information are available, we need specialized models that exploit available information at their best).

Again, inside each model, there are often sub-models specifically differentiated on the basis of architecture mode (behavioral, structural, data-flow). This differentiation relies on the hypothesis (proven as true), that syntax objects contained in architectures developed according to a given design paradigm have more in common with objects developed according to the same paradigm than with objects developed under different paradigms. We have always verified this hypothesis: undifferentiated models operating on all architectures behave always worse than specialized models. The same is true also for processes and com-

ponents, thus suggesting the idea that mode induces not only a way of thinking but also a coding style.

Anyway, apart from all the above model specialization criteria, there could be at least one more good criterion, whose treatment is beyond the scope of this thesis: **application**. During our experimental researchs, we found reasonable findings showing that architectures designed to perform similar functions (i.e. designed to be used in similar applications, for example cache memories, timer, interrupt handlers, ALUs, etc...) often tend to group together, when plotted in *estimated vs. actual* charts. This suggests the idea that, knowing more about the application to which a given syntax object is devoted, could lead to improved, specialized models, able to estimate with greater accuracy the size of objects belonging to that class.

Design of such application-based models and theory is left to future developments; in the meanwhile we will examine two example study cases that could not help attracting our attention: read only memory components and integer multipliers implemented in a structural way.

12.2.1 Read only memories

As already said, read only memories occur quite often in the collected projects, especially in general purpose processors. We found 36 ROM entities in our tuning project base, and 9 more in our validation project base.

Estimating the size of ROM architectures could seem difficult at first sight, since the homogeneity of their ports is always equal to 2 (1 unit for the address input signals plus 1 unit for the data output) whereas their length in lines of code vary from 8 to 4104. Despite that, at a deeper look, it becomes evident that estimating the size of a ROM architecture is a trivial task.

ROMs are usually implemented in two ways:

- by explicitly specifying the content of each memory cell, like in the following example, coming from file `pic_rom.vhd`, in project "PIC16C5X":

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity PIC_ROM is
  port (
    Addr     : in   std_logic_vector(8 downto 0);
    Data     : out  std_logic_vector(11 downto 0)
  );
end PIC_ROM;

architecture first of PIC_ROM is
begin
  Data <=
        "110000100101" When to_integer(Addr) = 00000 Else
        "000000101000" When to_integer(Addr) = 00001 Else
        "110001110010" When to_integer(Addr) = 00002 Else
        "000111001000" When to_integer(Addr) = 00003 Else
        "000111101000" When to_integer(Addr) = 00004 Else
        "000111101000" When to_integer(Addr) = 00005 Else
        "001001001000" When to_integer(Addr) = 00006 Else
        "000011101000" When to_integer(Addr) = 00007 Else
        "001010101000" When to_integer(Addr) = 00008 Else
        "000100001000" When to_integer(Addr) = 00009 Else
        "000110101000" When to_integer(Addr) = 00010 Else
```

```
        "000101001000" When to_integer(Addr) = 00011 Else
        "001101101000" When to_integer(Addr) = 00012 Else
        "001101101000" When to_integer(Addr) = 00013 Else
        "001101101000" When to_integer(Addr) = 00014 Else
        "001100101000" When to_integer(Addr) = 00015 Else
        "001100101000" When to_integer(Addr) = 00016 Else
        "001100101000" When to_integer(Addr) = 00017 Else
        "001110101000" When to_integer(Addr) = 00018 Else
 [... cut ...]
        "101000000000" When to_integer(Addr) = 00511 Else
        "000000000000";
end first ;
```

- by instructing VHDL to get memory values from an external file, as in the following section of code, coming from file `bprom.vhd`, in project "Leon":

```
architecture behav of virtex_prom256 is
    component iram
        generic (
            index   :  integer  := 0;              -- Byte lane (0 - 3)
            Abits   :  Positive  := 10;            -- Default 10 address bits (1 Kbyte)
            echk    :  integer  := 0;              -- Generate EDAC checksum
            tacc    :  integer  := 10;             -- access time (ns)
            fname   :  string  :=  "ram.dat" );    -- File to read from
        port (
            A       : in  std_logic_vector;
            D       : inout std_logic_vector(7 downto 0);
            CE1     : in  std_logic;
            WE      : in  std_logic;
            OE      : in  std_logic);
    end component;

signal gnd          : std_logic  := '0';
signal vcc          : std_logic  := '1';
signal address      : std_logic_vector(7 downto 0);
signal data         : std_logic_vector(31 downto 0);
begin
    x : process(clk)
    begin
        if  rising_edge(clk) then
            address <= addr;
        end if;
    end process;

    romarr : for  i  in  0  to  3  generate
      rom0 : iram
        generic map (
            index => i,
            abits => 8,
            echk => 0,
            tacc => 10,
            fname => "tsource/bprom.dat")
        port map (
            A => address(7 downto 0),
            D => data((31 - i*8) downto (24-i*8)),
            CE1 => gnd,
            WE => VCC,
            OE => gnd);
    end generate;
    do <= data;
end;
```

In the first case, the number of lines of code required to describe a ROM architecture can be easily expressed as a constant (which value is around 4, representing the number of lines occupied by the beginning and ending lines) plus the number of cells for which the content is to be provided, one per line. This last number is equal to the used size of the ROM expressed in bytes or words, depending on its architecture, especially the width of the data bus.

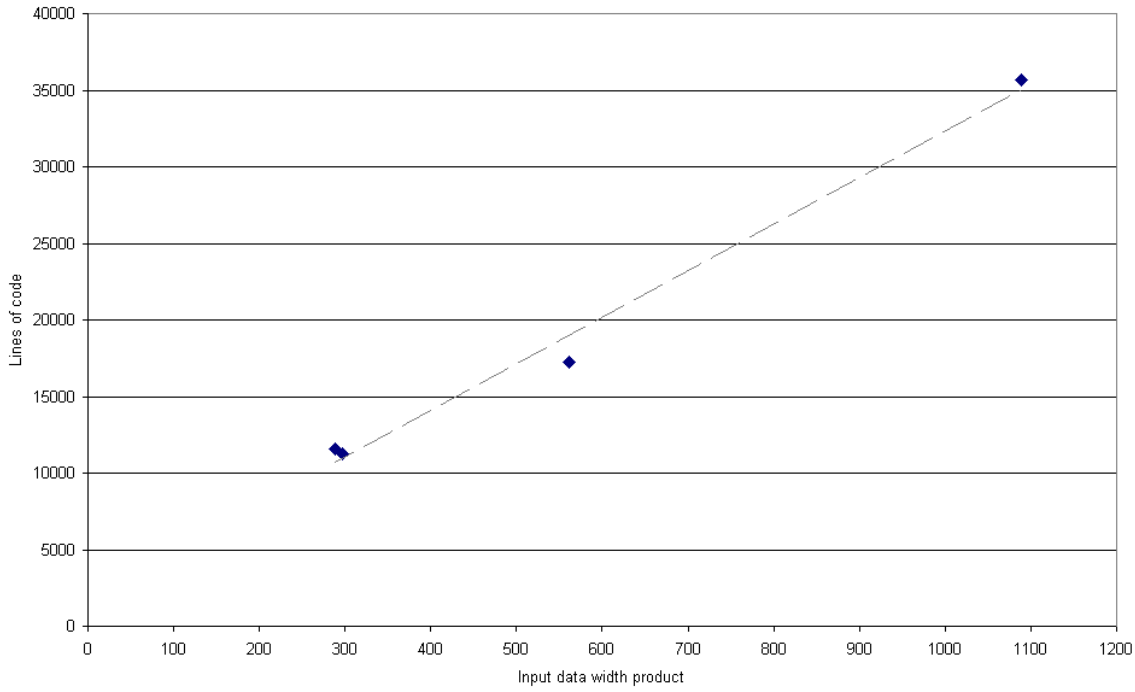
In both cases, especially in the first one, it would not be difficult to create suitable models able to estimate the number of lines of code by knowing the number of cells for which the content is to be provided.

The creation of such a model is left to future developments.

12.2.2 Structural multipliers

As said before, integer multipliers implemented as structural architectures are a good example of objects that can be generated automatically. In project "Leon" there are four such multipliers.

Figure 12.2: Structural multipliers: lines of code vs input width product.



In the following table, salient data regarding those multipliers are listed: L is, as usual, the length of the object expressed in lines of code; with a and b we indicate the two operands received by the multipliers as input, and $dim(a)$ and $dim(b)$ are their respective signal width expressed in number of bits. The fourth column contains $dim(a) \cdot dim(b)$, and we included it because we found that there is extraordinarily strong correlation between that data and L, and this is made evident by figure 12.2.

Multiplier name	$dim(a)$	$dim(b)$	$dim(a) \cdot dim(b)$	L
mul_17_17	17	17	289	11556
mul_33_9	33	9	297	11297
mul_33_17	33	17	561	17236
mul_33_33	33	33	1089	35663

Correlation coefficient between L and $dim(a) \cdot dim(b)$	0.994587

A deeper look at the internal structure of those multipliers, especially at the number of signals (n_s) and at the number of the instanced components (n_{ci}), confirms extreme regularity:

Multiplier name	L	n_s	L/n_s	n_{ci}	L/n_{ci}
mul_17_17	11556	5249	2.2016	5278	2.1895
mul_33_9	11297	7904	2.1807	7950	2.1681
mul_33_17	17236	16029	2.2249	16091	2.2163
mul_33_33	35663	5091	2.2190	5128	2.2030

Data in columns L/n_s and L/n_{ci}, which are almost invariant, suggest that, once the number of lines is known, it is easy to estimate the number of signals and component instantiations used inside the multiplier. This is confirmed by the following correlation coefficients:

Correlation coefficient between L and n_s	0.99991384
Correlation coefficient between L and n_{ci}	0.99990870

Now let's get back and focus again on the possibility of estimating the number of lines of code by knowing the bit width of the input operands. With the above data we performed a least squares identification and obtained the following model:

Model:

$$\hat{L} = k_d \cdot (dim(a) \cdot dim(b)) + k_0$$

Identified model coefficients:

Coefficient	Value
k_d	30.4320
k_0	1926.5246

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.994587235

(Obviously the correlation coefficient is the same as the first one cited in this paragraph, since correlation coefficients are invariant to linear transformations).

The above model was introduced for study reasons only: as already said, length of automatically generated code does not contribute to exploitation of human resources.

Chapter 13

Syntax object models

13.1 Entity models

13.1.1 Model variables

The following models try to estimate the size of an entity part by knowing only a small amount of information, such as the number of ports. Each information given to the models will be called *a variable* from now on; the problem of creating good entity length estimation models is basically reduced to choosing an appropriate set of variables and identifying a linear model on that variables. Except for L, which is the variable to estimate, this is the full list of variables available for the model generation:

- L is the total number of core[1] lines of code;
- n_p is the total number of ports;
- n_{ip} is the total number of `in` ports;
- n_{op} is the total number of `out` ports;
- n_{iop} is the total number of `inout` ports;
- n_{xp} is the total number of `linkage`, `buffer` and default-mode ports;
- h_p is the total homogeneity of all ports;
- h_{ip} is the total homogeneity of `in` ports;
- h_{op} is the total homogeneity of `out` ports;
- h_{iop} is the total homogeneity of `inout` ports;
- h_{xp} is the total homogeneity of `linkage`, `buffer` and default-mode ports;
- n_g is the total number of `generic` definitions;
- n_{upt} is the total number of unique port types used;
- n_{ugt} is the total number of unique `generic` types used;

[1] By *core* lines of code we mean the lines of code not occupied by subprograms or processes.

For sake of clarity, the following example is provided the declaration of entity `shift_adder` already listed in paragraph 8.5.8 (page 76) causes the above variables to assume the following values[2]:

Variable	Value
n_p	4
n_{ip}	3
n_{op}	1
n_{iop}	0
n_{xp}	0
h_p	4
h_{ip}	3
h_{op}	1
h_{iop}	0
h_{xp}	0
n_g	0
n_{upt}	2
n_{ugt}	0

The following table shows statistical properties of the above variables in our tuning project base. The last column shows the coefficient of correlation between each variable and L, which is the variable to be estimated. For two variables the correlation values are typed in bold, namely n_p and n_g: these variable exhibit the highest correlation coefficients with the length of the entities from which they come. We will see that models built on them show great accuracy.

Variable	Average value	Variance	Standard deviation	Correlation coefficient
h_p	11.076	289.260	17.008	0.5304
n_p	9.751	185.760	13.629	**0.6536**
h_{ip}	6.536	78.405	8.855	0.4548
h_{op}	3.933	86.900	9.322	0.4277
h_{iop}	0.563	13.125	3.623	0.2480
h_{xp}	0.044	0.131	0.361	0.3015
n_{ip}	5.805	52.572	7.251	0.5544
n_{op}	3.350	46.842	6.844	0.5670
n_{iop}	0.554	12.965	3.601	0.2495
n_{xp}	0.043	0.127	0.357	0.3066
n_g	2.780	44.468	6.668	**0.5737**
n_{upt}	1.689	1.985	1.409	0.1325
n_{ugt}	0.768	1.653	1.286	0.2996
L	14.154	255.090	15.972	(1.0000)

[2] Please note that, since all the considered port types have homogeneity equal to 1, $\forall s : h_s = n_s$, this is quite common.

13.1.2 Model EM1

This model is an example of how homogeneity, though being a more complex indicator with respect to simple port count, can be inappropriate to describe the effort required to handle given signals. Model EM2, though using much less information shows lower estimation error variance, and better coefficient of correlation between actual and estimated number of lines of code.

Model:

$$\hat{L} = k_{hip} \cdot h_{ip} + k_{hop} \cdot h_{op} + k_{iop} \cdot h_{iop} + k_{hxp} \cdot h_{xp} + k_0$$

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	14.236	14.236	0.000
Variance	256.017	79.430	176.588
Standard deviation	16.001	8.912	13.289

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.5570

Identified model coefficients:

Coefficient	Value
k_{hip}	0.4923
k_{hop}	0.3423
k_{hiop}	0.8373
k_{hxp}	7.9272
k_0	8.81462

Figure 13.1: Model EM1: Real vs. estimated lines of code.

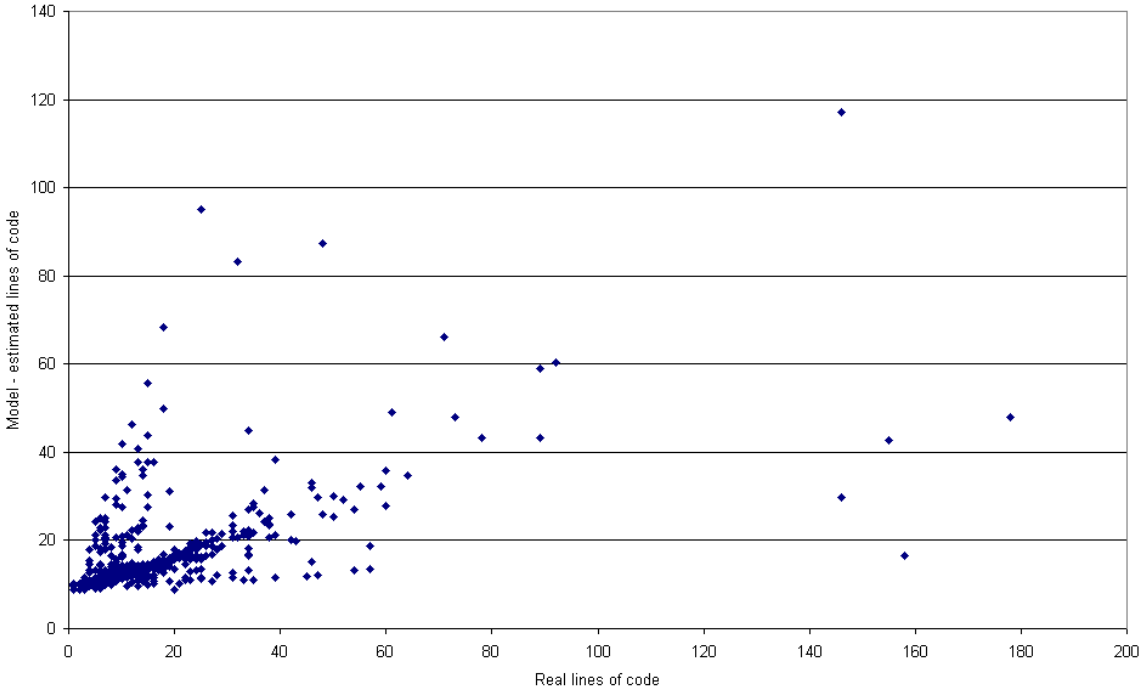


Figure 13.2: Model EM1: Error density distribution.

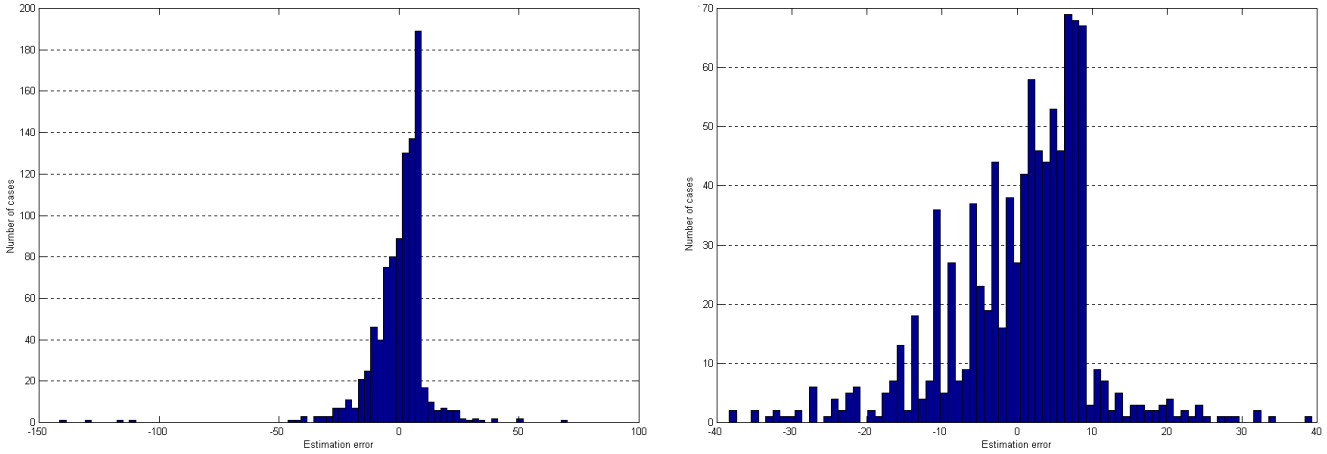
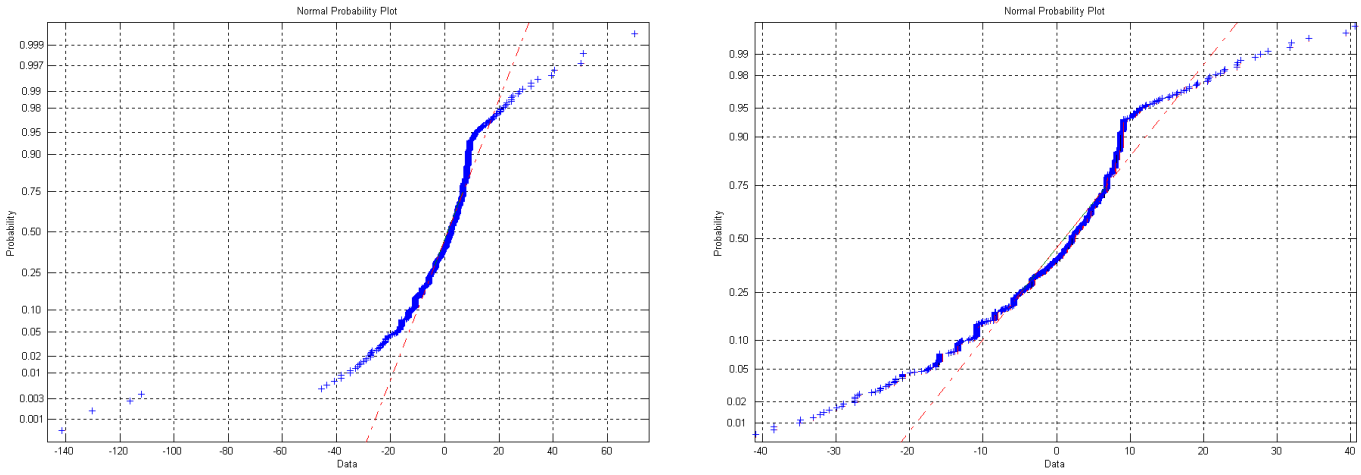


Figure 13.3: Model EM1: Error cumulative distribution.



13.1.3 Model EM2

Model EM2 is redesigned from scratch with respect to EM1, and it starts from the consideration that in the vast majority of cases, entity declarations comprise one line for each port.

Model:

$$\hat{L} = k_{np} \cdot n_p + k_0$$

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	14.236	14.236	0.000
Variance	256.017	108.954	147.063
Standard deviation	16.001	10.438	12.127

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.6536

Identified model coefficients:

Coefficient	Value
k_{np}	0.7643
k_0	6.7338

Figure 13.4: Model EM2: Real vs. estimated lines of code.

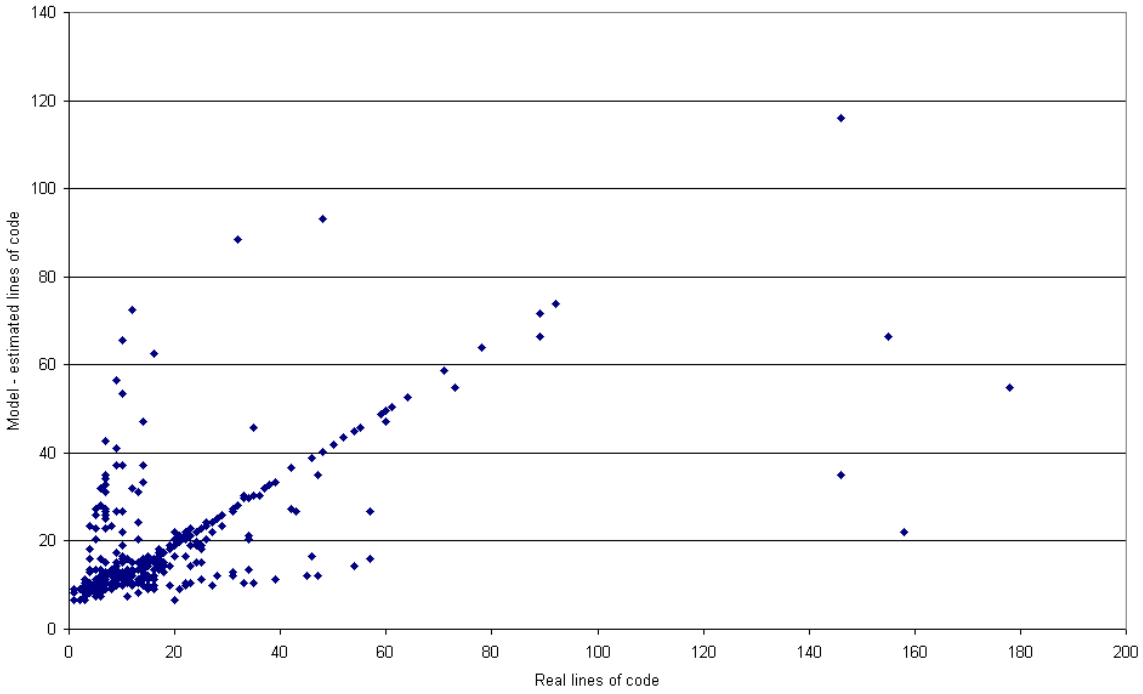


Figure 13.5: Model EM2: Error density distribution.

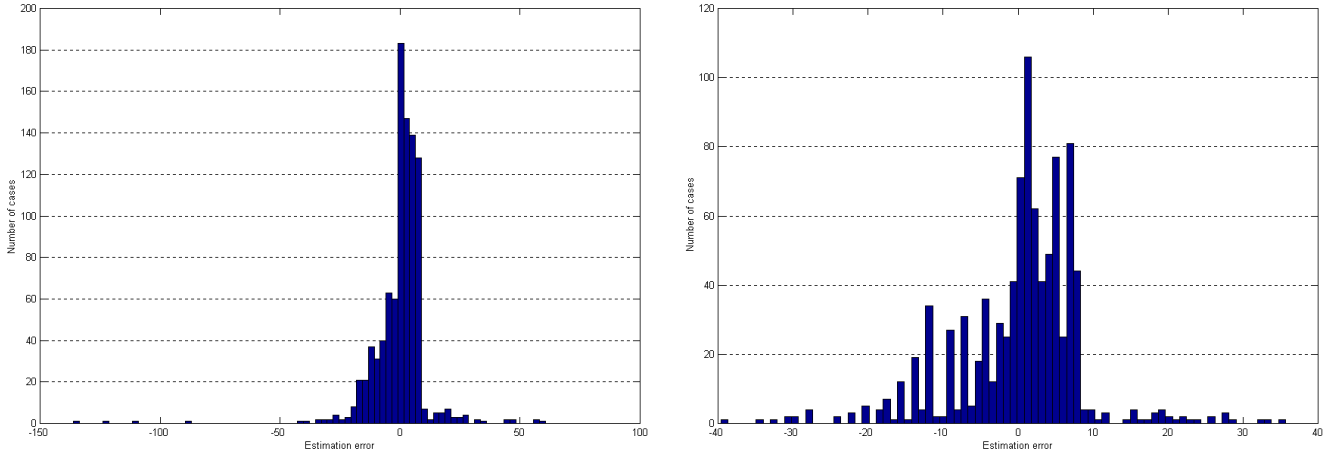
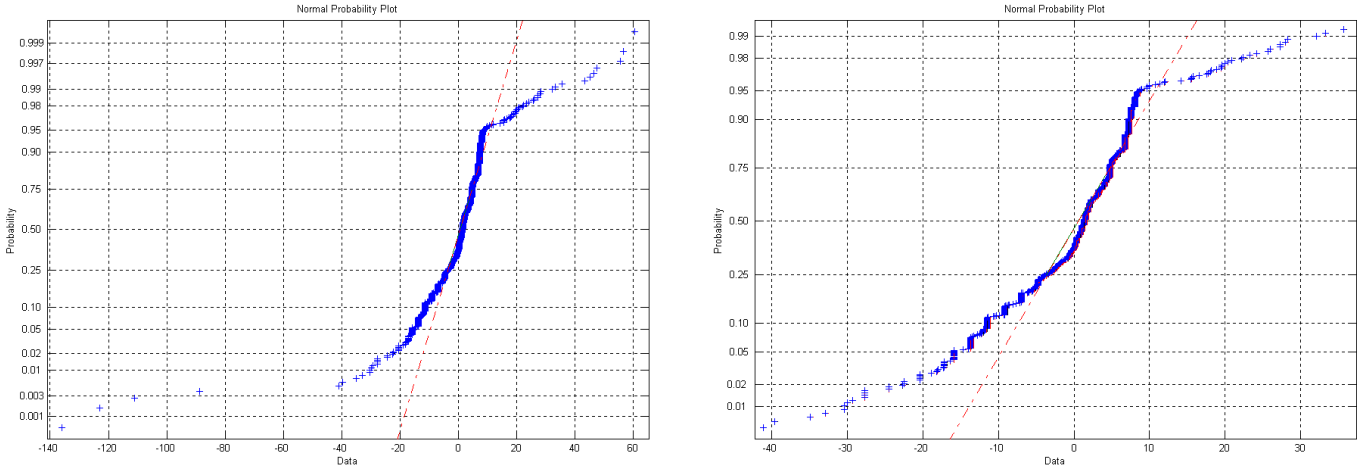


Figure 13.6: Model EM2: Error cumulative distribution.



13.1.4 Model EM3

Model EM3 is an improvement over model EM2, that tries to perform better in many cases where EM2 was underestimating L. By individually inspecting many such cases, it turned out that impact of generic constant declarations are not negligible as we were supposing at first sight. At the time in which this model was thought, our VHDL parser was not even programmed to annotated generic constant declarations inside the database, nor had the database a table to keep them. Both parser and database were modified in order to store generic constants and support model EM3, and EM3 turned out to be the best model for entities, except for the very minor improvements added by EM4 at the cost of more complex input information.

Model:

$$\hat{L} = k_{np} \cdot n_p + k_{ng} \cdot n_g + k_0$$

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	14.236	14.236	0.000
Variance	256.017	187.968	68.049
Standard deviation	16.001	13.710	8.249

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.8569

Identified model coefficients:

Coefficient	Value
k_{np}	0.7466
k_{ng}	1.3294
k_0	3.1834

Figure 13.7: Model EM3: Real vs. estimated lines of code.

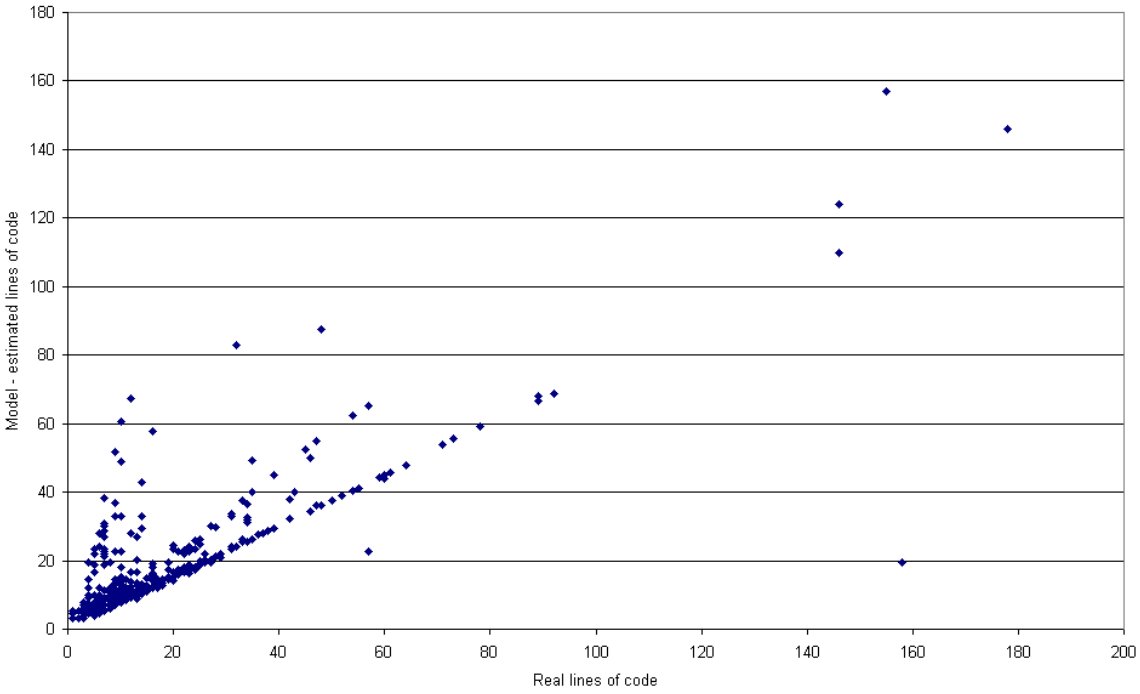


Figure 13.8: Model EM3: Error density distribution.

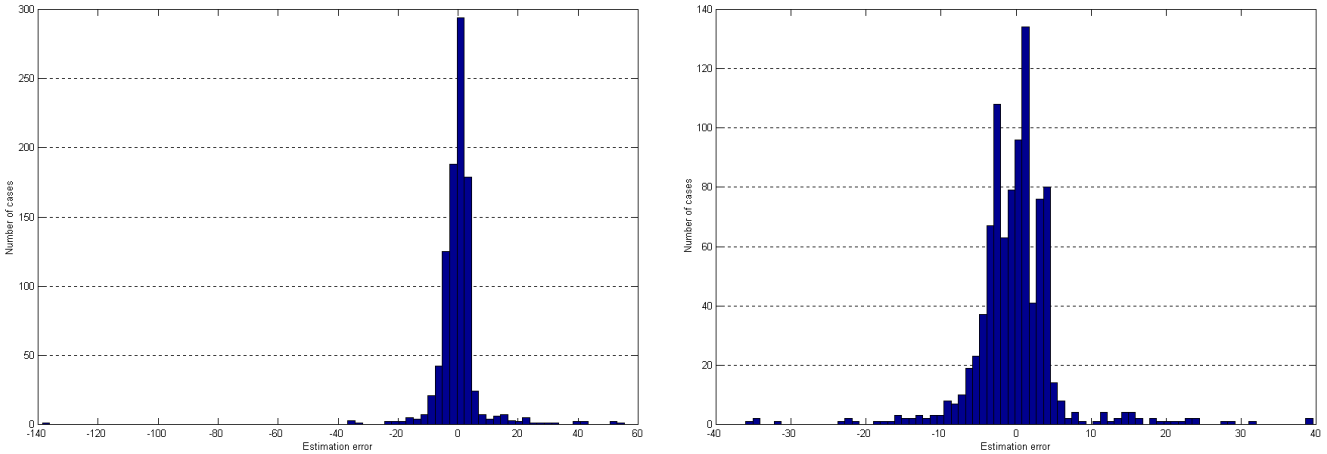
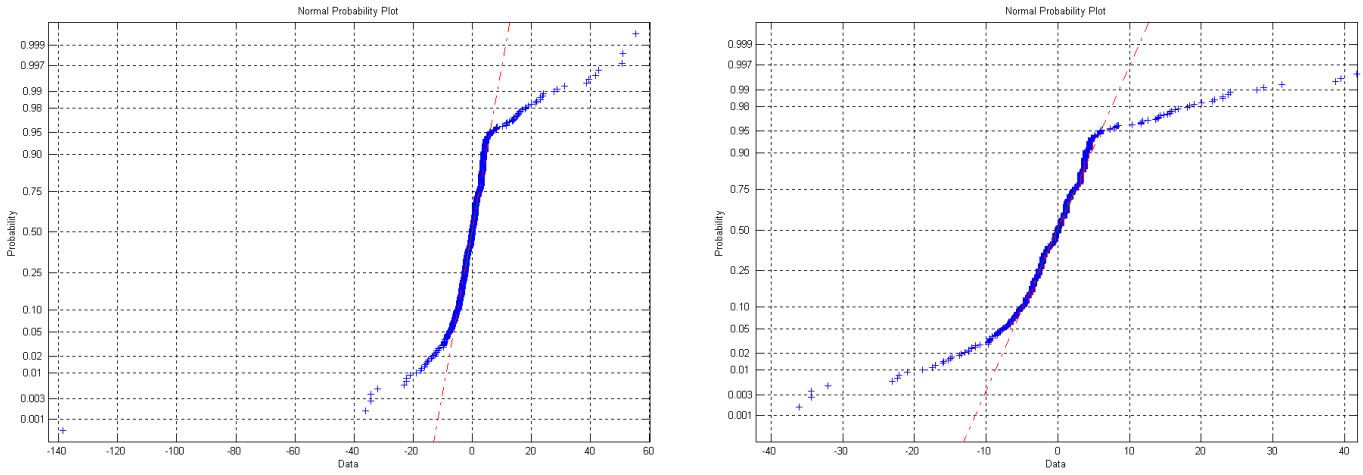


Figure 13.9: Model EM3: Error cumulative distribution.



13.1.5 Model EM4

Model EM4 is an attempt to increase the accuracy of model EM3, by taking into account the following observation: in most cases in which model EM3 overestimates the number of lines of code, this is due to stylistic reasons, in particular when the programmer declares more ports of the same type on a single line. For example the following code:

```
entity my_entity is
  port(
    a : inout std_logic;
    b : inout std_logic;
  );
```

could also be written in the following way:

```
entity my_entity is
  port( a, b : inout std_logic ; );
```

The same considerations apply to generic constant declarations too. Accuracy should be increased if the number of unique types used inside port and generic declaration is considered.

Model:

$$\hat{L} = k_{np} \cdot n_p + k_{nupt} \cdot n_{upt} + k_{ng} \cdot n_g + k_{nugt} \cdot n_{ugt} + k_0$$

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	14.236	14.236	0.000
Variance	256.017	190.038	65.980
Standard deviation	16.001	13.785	8.123

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.8616

Identified model coefficients:

Coefficient	Value
k_{np}	0.7244
k_{nupt}	1.0136
k_{ng}	1.4004
k_{nugt}	-0.3307
k_0	1.7359

Figure 13.10: Model EM4: Real vs. estimated lines of code.

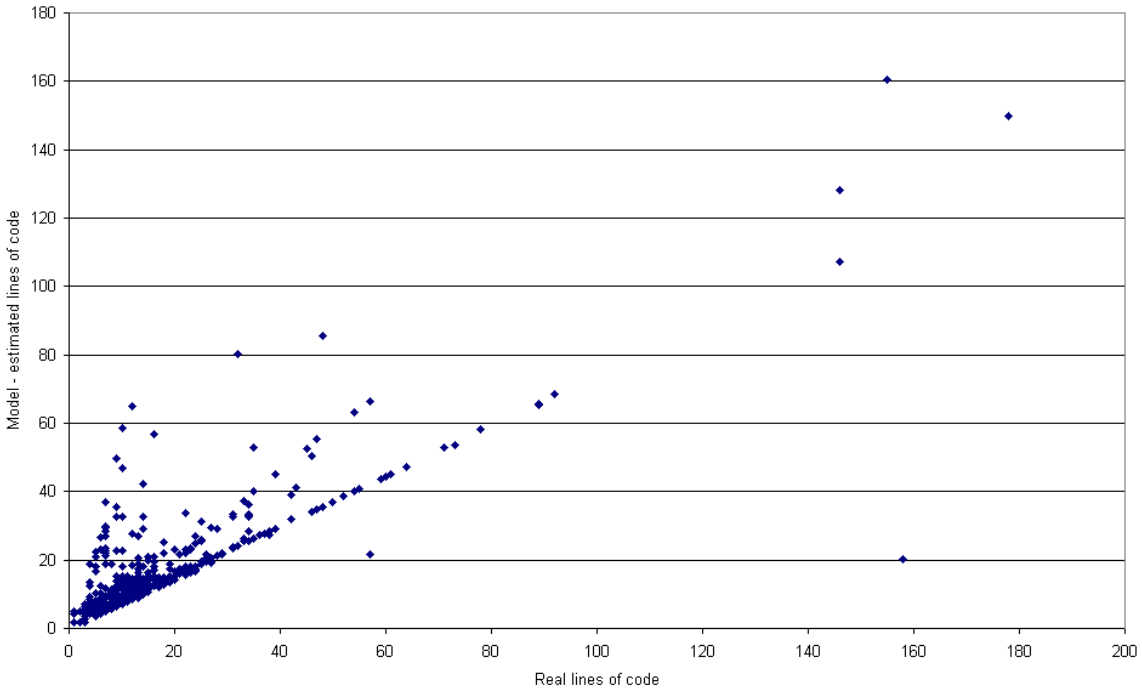


Figure 13.11: Model EM4: Error density distribution.

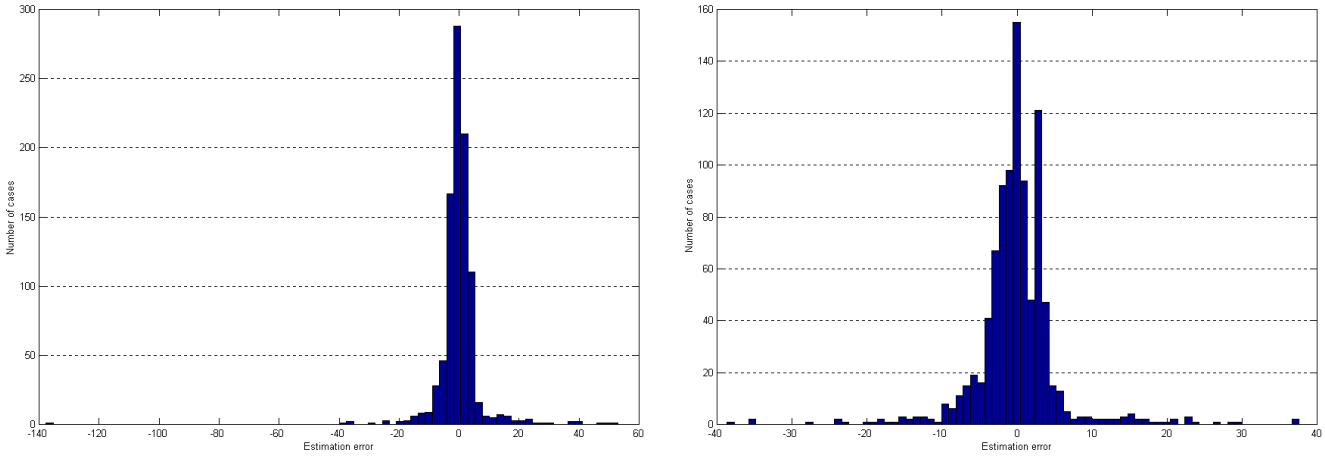
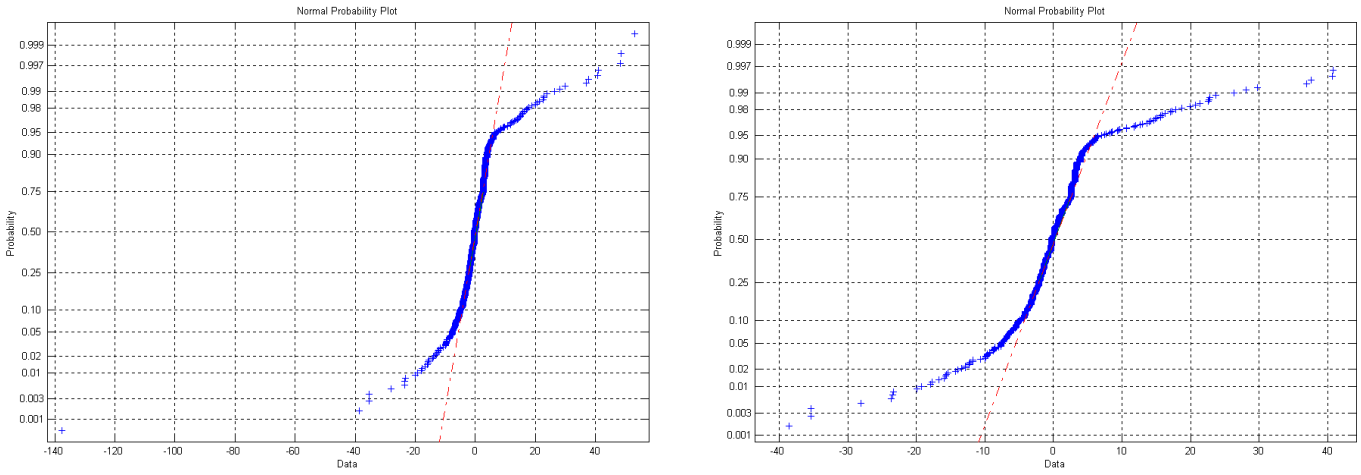


Figure 13.12: Model EM4: Error cumulative distribution.



13.1.6 Conclusions

Our experimental findings show that entities are suitable to be modelled in a very accurate way by linear models using primarily the number of declared ports and generic constants as input. Model EM4 shows an estimation error which is between -20 and $+20$ lines of code in 97% of the cases, and between -10 and $+10$ in 94% of the cases. Models EM3 shows an accuracy that is practically identical to EM4, and it uses less information. Models show a little tendency to overestimate some entity declarations, and this phenomenon is due to coding style reasons already examined.

As far as underestimation is concerned, there are still exactly two cases in which that phenomenon occurs, and it occurs in an evident way. We individually analyzed them to investigate on the reason of the underestimation: the VHDL source code of both entities is reported below.

The first fragment is the declaration of entity hc11cpu, coming from file `hc11rtl.vhd` in project "HC11", line 19 and following. The fragment is composed by 158 lines of code, but model EM4 estimates only 19.89 lines. The error is due to the large amount of constants.

Listing 13.1: Case 5

```
entity hc11cpu is
  generic (testmode : boolean := true);
  port (E, ph1                          : in  std_logic;
        reset                           : in  std_logic;
        ino                             : in  std_logic_vector (3 downto 0);
        iavail                          : in  std_logic;
        iaccept , CCR_X, CCR_I          : out std_logic;
        rw                              : out std_logic;
        address                         : out std_logic_vector (15 downto 0);
        data                            : in  std_logic_vector (7 downto 0);
        write_data                      : out std_logic_vector (7 downto 0);
        debug_cycle                     : out std_logic_vector (5 downto 0);
        debug_A, debug_B, debug_CCR : out std_logic_vector (7 downto 0);
        debug_X, debug_Y, debug_SP : out std_logic_vector (15 downto 0);
        debug_micro                     : out unsigned (3 downto 0));

  constant PREFIX_Y          : std_logic_vector (7 downto 0) := "00011000"; -- 18
  constant PREFIX_D          : std_logic_vector (7 downto 0) := "00011010"; -- 1A
  constant PREFIX_D_Y        : std_logic_vector (7 downto 0) := "11001101"; -- CD
  constant ABA               : std_logic_vector (7 downto 0) := "00011011"; -- 1B
  constant ABI               : std_logic_vector (7 downto 0) := "00111010"; -- 3A
  constant ADCA              : std_logic_vector (7 downto 0) := "10001001"; -- 89
  constant ADCB              : std_logic_vector (7 downto 0) := "11001001"; -- C9
  constant ADDA              : std_logic_vector (7 downto 0) := "10001011"; -- 8B
  constant ADDB              : std_logic_vector (7 downto 0) := "11001011"; -- CB
  constant ADDD              : std_logic_vector (7 downto 0) := "11000011"; -- C3
  constant ANDA              : std_logic_vector (7 downto 0) := "10000100"; -- 84
  constant ANDB              : std_logic_vector (7 downto 0) := "11000100"; -- C4
  constant ASLA              : std_logic_vector (7 downto 0) := "01001000"; -- 48
  constant ASLB              : std_logic_vector (7 downto 0) := "01011000"; -- 58
  constant ASL               : std_logic_vector (7 downto 0) := "01001000"; -- 48
  constant ASLD              : std_logic_vector (7 downto 0) := "00000101"; -- 05
  constant ASRA              : std_logic_vector (7 downto 0) := "01000111"; -- 47
  constant ASRB              : std_logic_vector (7 downto 0) := "01010111"; -- 57
  constant ASR               : std_logic_vector (7 downto 0) := "01000111"; -- 47
  constant BCC               : std_logic_vector (7 downto 0) := "00100100"; -- 24
  constant BCLR_DIR          : std_logic_vector (7 downto 0) := "00010101"; -- 15
  constant BCLR_IND          : std_logic_vector (7 downto 0) := "00011101"; -- 1D
  constant BCS               : std_logic_vector (7 downto 0) := "00100101"; -- 25
  constant BEQ               : std_logic_vector (7 downto 0) := "00100111"; -- 27
  constant BGE               : std_logic_vector (7 downto 0) := "00101100"; -- 2C
  constant BGT               : std_logic_vector (7 downto 0) := "00101110"; -- 2E
  constant BHI               : std_logic_vector (7 downto 0) := "00100010"; -- 22
  constant BITA              : std_logic_vector (7 downto 0) := "10000101"; -- 85
  constant BITB              : std_logic_vector (7 downto 0) := "11000101"; -- C5
  constant BLE               : std_logic_vector (7 downto 0) := "00101111"; -- 2F
  constant BLS               : std_logic_vector (7 downto 0) := "00100011"; -- 23
  constant BLT               : std_logic_vector (7 downto 0) := "00101101"; -- 2D
  constant BMI               : std_logic_vector (7 downto 0) := "00101011"; -- 2B
  constant BNE               : std_logic_vector (7 downto 0) := "00100110"; -- 26
  constant BPL               : std_logic_vector (7 downto 0) := "00101010"; -- 2A
  constant BRA               : std_logic_vector (7 downto 0) := "00100000"; -- 20
  constant BRCLR_DIR         : std_logic_vector (7 downto 0) := "00010011"; -- 13
  constant BRCLR_IND         : std_logic_vector (7 downto 0) := "00011111"; -- 1F
  constant BRN               : std_logic_vector (7 downto 0) := "00100001"; -- 21
  constant BRSET_DIR         : std_logic_vector (7 downto 0) := "00010010"; -- 12
  constant BRSET_IND         : std_logic_vector (7 downto 0) := "00011110"; -- 1E
  constant BSET_DIR          : std_logic_vector (7 downto 0) := "00010100"; -- 14
  constant BSET_IND          : std_logic_vector (7 downto 0) := "00011100"; -- 1C
  constant BSR               : std_logic_vector (7 downto 0) := "10001101"; -- 8D
  constant BVC               : std_logic_vector (7 downto 0) := "00101000"; -- 28
  constant BVS               : std_logic_vector (7 downto 0) := "00101001"; -- 29
  constant CBA               : std_logic_vector (7 downto 0) := "00010001"; -- 11
  constant CLC               : std_logic_vector (7 downto 0) := "00001100"; -- 0C
```

```
    constant CLI            : std_logic_vector (7 downto 0) := "00001110"; -- 0E
    constant CLV            : std_logic_vector (7 downto 0) := "00001010"; -- 0A
    constant CLRA           : std_logic_vector (7 downto 0) := "01001111"; -- 4F
    constant CLRB           : std_logic_vector (7 downto 0) := "01011111"; -- 5F
    constant CLR            : std_logic_vector (7 downto 0) := "01001111"; -- 4F
    constant CMPA           : std_logic_vector (7 downto 0) := "10000001"; -- 81
    constant CMPB           : std_logic_vector (7 downto 0) := "11000001"; -- C1
    constant COMA           : std_logic_vector (7 downto 0) := "01000011"; -- 43
    constant COMB           : std_logic_vector (7 downto 0) := "01010011"; -- 53
    constant COM            : std_logic_vector (7 downto 0) := "01000011"; -- 43
    constant CPI            : std_logic_vector (7 downto 0) := "10001100"; -- 8C
    constant DAA            : std_logic_vector (7 downto 0) := "00011001"; -- 19
    constant DECA           : std_logic_vector (7 downto 0) := "01001010"; -- 4A
    constant DECB           : std_logic_vector (7 downto 0) := "01011010"; -- 5A
    constant DEC            : std_logic_vector (7 downto 0) := "01001010"; -- 4A
    constant DES            : std_logic_vector (7 downto 0) := "00110100"; -- 34
    constant DEI            : std_logic_vector (7 downto 0) := "00001001"; -- 09
    constant EORA           : std_logic_vector (7 downto 0) := "10001000"; -- 88
    constant EORB           : std_logic_vector (7 downto 0) := "11001000"; -- C8
    constant INCA           : std_logic_vector (7 downto 0) := "01001100"; -- 4C
    constant INCB           : std_logic_vector (7 downto 0) := "01011100"; -- 5C
    constant INC            : std_logic_vector (7 downto 0) := "01001100"; -- 4C
    constant INS            : std_logic_vector (7 downto 0) := "00110001"; -- 31
    constant INI            : std_logic_vector (7 downto 0) := "00001000"; -- 08
    constant JMP            : std_logic_vector (7 downto 0) := "01001110"; -- 4E
    constant JSR            : std_logic_vector (7 downto 0) := "10001101"; -- 8D
    constant LDAA           : std_logic_vector (7 downto 0) := "10000110"; -- 86
    constant LDAB           : std_logic_vector (7 downto 0) := "11000110"; -- C6
    constant LDD            : std_logic_vector (7 downto 0) := "11001100"; -- CC
    constant LDS            : std_logic_vector (7 downto 0) := "10001110"; -- 8E
    constant LDI            : std_logic_vector (7 downto 0) := "11001110"; -- CE
    constant LSRA           : std_logic_vector (7 downto 0) := "01000100"; -- 44
    constant LSRB           : std_logic_vector (7 downto 0) := "01010100"; -- 54
    constant LSR            : std_logic_vector (7 downto 0) := "01000100"; -- 44
    constant LSRD           : std_logic_vector (7 downto 0) := "00000100"; -- 04
    constant MUL            : std_logic_vector (7 downto 0) := "00111101"; -- 3D
    constant NEGA           : std_logic_vector (7 downto 0) := "01000000"; -- 40
    constant NEGB           : std_logic_vector (7 downto 0) := "01010000"; -- 50
    constant NEG            : std_logic_vector (7 downto 0) := "01000000"; -- 40
    constant NOP            : std_logic_vector (7 downto 0) := "00000001"; -- 01
    constant ORA            : std_logic_vector (7 downto 0) := "10001010"; -- 8A
    constant ORB            : std_logic_vector (7 downto 0) := "11001010"; -- CA
    constant PSHA           : std_logic_vector (7 downto 0) := "00110110"; -- 36
    constant PSHB           : std_logic_vector (7 downto 0) := "00110111"; -- 37
    constant PSHI           : std_logic_vector (7 downto 0) := "00111100"; -- 3C
    constant PULA           : std_logic_vector (7 downto 0) := "00110010"; -- 32
    constant PULB           : std_logic_vector (7 downto 0) := "00110011"; -- 33
    constant PULI           : std_logic_vector (7 downto 0) := "00111000"; -- 38
    constant ROLA           : std_logic_vector (7 downto 0) := "01001001"; -- 49
    constant ROLB           : std_logic_vector (7 downto 0) := "01011001"; -- 59
    constant ROLc           : std_logic_vector (7 downto 0) := "01001001"; -- 49
    constant RORA           : std_logic_vector (7 downto 0) := "01000110"; -- 46
    constant RORB           : std_logic_vector (7 downto 0) := "01010110"; -- 56
    constant RORc           : std_logic_vector (7 downto 0) := "01000110"; -- 46
    constant RTI            : std_logic_vector (7 downto 0) := "00111011"; -- 3B
    constant RTS            : std_logic_vector (7 downto 0) := "00111001"; -- 39
    constant SBA            : std_logic_vector (7 downto 0) := "00010000"; -- 10
    constant SBCA           : std_logic_vector (7 downto 0) := "10000010"; -- 82
    constant SBCB           : std_logic_vector (7 downto 0) := "11000010"; -- C2
    constant SEC            : std_logic_vector (7 downto 0) := "00001101"; -- 0D
    constant SEI            : std_logic_vector (7 downto 0) := "00001111"; -- 0F
    constant SEV            : std_logic_vector (7 downto 0) := "00001011"; -- 0B
    constant STAA           : std_logic_vector (7 downto 0) := "10000111"; -- 87
    constant STAB           : std_logic_vector (7 downto 0) := "11000111"; -- C7
    constant STD            : std_logic_vector (7 downto 0) := "11001101"; -- CD
    constant STS            : std_logic_vector (7 downto 0) := "10001111"; -- 8F
    constant STI            : std_logic_vector (7 downto 0) := "11001111"; -- CF
    constant SUBA           : std_logic_vector (7 downto 0) := "10000000"; -- 80
    constant SUBB           : std_logic_vector (7 downto 0) := "11000000"; -- C0
    constant SUBD           : std_logic_vector (7 downto 0) := "10000011"; -- 83
    constant SWI            : std_logic_vector (7 downto 0) := "00111111"; -- 3F
    constant TAB            : std_logic_vector (7 downto 0) := "00010110"; -- 16
    constant TAP            : std_logic_vector (7 downto 0) := "00000110"; -- 06
    constant TBA            : std_logic_vector (7 downto 0) := "00010111"; -- 17
    constant TPA            : std_logic_vector (7 downto 0) := "00000111"; -- 07
    constant TSTA           : std_logic_vector (7 downto 0) := "01001101"; -- 4D
    constant TSTB           : std_logic_vector (7 downto 0) := "01011101"; -- 5D
    constant TST            : std_logic_vector (7 downto 0) := "01001101"; -- 4D
    constant TSI            : std_logic_vector (7 downto 0) := "00110000"; -- 30
    constant TIS            : std_logic_vector (7 downto 0) := "00110101"; -- 35
    constant WAI            : std_logic_vector (7 downto 0) := "00111110"; -- 3E
    constant XGDI           : std_logic_vector (7 downto 0) := "10001111"; -- 8F

    constant IMM            : std_logic_vector (1 downto 0) := "00";
    constant DIR            : std_logic_vector (1 downto 0) := "01";
    constant EXT            : std_logic_vector (1 downto 0) := "11";
    constant IND            : std_logic_vector (1 downto 0) := "10";

    constant SBIT           : integer  := 7;
    constant XBIT           : integer  := 6;
    constant HBIT           : integer  := 5;
    constant IBIT           : integer  := 4;
    constant NBIT           : integer  := 3;
    constant ZBIT           : integer  := 2;
    constant VBIT           : integer  := 1;
    constant CBIT           : integer  := 0;
end;
```

The following fragment represents the declaration of entity FPGA, present both in file `FPGA.vhd` of project "an-XC2S-XR16" and in file `xc2sFPGA.vhd` in project "an-XC2S-USB".

As one can easily notice, the estimation error is due to the large number of attribute declarations.

Listing 13.2: Case 5

```
entity FPGA is
    port(
        clk_a1    : in STD_LOGIC;
        init      : in STD_LOGIC;
        ion_a5    : in STD_LOGIC;
        ion_b2    : in STD_LOGIC;
        iop_a5    : in STD_LOGIC;
        iop_b3    : in STD_LOGIC;
        ion_a1    : out STD_LOGIC;
        ion_a2    : out STD_LOGIC;
        ion_a3    : out STD_LOGIC;
        ion_a4    : out STD_LOGIC;
        ion_a6    : out STD_LOGIC;
        ion_b1    : out STD_LOGIC;
        ion_b3    : out STD_LOGIC;
        ion_b4    : out STD_LOGIC;
        ion_b5    : out STD_LOGIC;
        ion_b6    : out STD_LOGIC;
        iop_a1    : out STD_LOGIC;
        iop_a2    : out STD_LOGIC;
        iop_a3    : out STD_LOGIC;
        iop_a4    : out STD_LOGIC;
        iop_a6    : out STD_LOGIC;
        iop_b1    : out STD_LOGIC;
        iop_b2    : out STD_LOGIC;
        iop_b4    : out STD_LOGIC;
        iop_b5    : out STD_LOGIC;
        iop_b6    : out STD_LOGIC
    );

    -- entity declarations --
    attribute LOC                : string ;
    attribute LOC of clk_a1      : signal is "P185";
    attribute LOC of iop_a1      : signal is "P191";
    attribute LOC of ion_a1      : signal is "P192";
    attribute LOC of iop_a2      : signal is "P193";
    attribute LOC of ion_a2      : signal is "P194";
    attribute LOC of iop_a3      : signal is "P195";
    attribute LOC of ion_a3      : signal is "P199";
    attribute LOC of iop_a4      : signal is "P200";
    attribute LOC of ion_a4      : signal is "P201";
    attribute LOC of iop_a5      : signal is "P202";
    attribute LOC of ion_a5      : signal is "P203";
    attribute LOC of iop_a6      : signal is "P204";
    attribute LOC of ion_a6      : signal is "P205";
    attribute LOC of iop_b1      : signal is "P4";
    attribute LOC of ion_b1      : signal is "P5";
    attribute LOC of iop_b2      : signal is "P6";
    attribute LOC of ion_b2      : signal is "P7";
    attribute LOC of iop_b3      : signal is "P8";
    attribute LOC of ion_b3      : signal is "P9";
    attribute LOC of iop_b4      : signal is "P10";
    attribute LOC of ion_b4      : signal is "P14";
    attribute LOC of iop_b5      : signal is "P15";
    attribute LOC of ion_b5      : signal is "P16";
    attribute LOC of iop_b6      : signal is "P17";
    attribute LOC of ion_b6      : signal is "P18";
    attribute LOC of init        : signal is "P107";
end FPGA;
```

Since this type of estimation error occurs only in the above two cases, we believe that it is not worth modifying the parser and the database in order to add annotation of constant and attribute declarations. Letting apart the fact that constants and attributes are **not** externally available, an improved model taking them into account would perform better than EM4 in a statistically irrelevant number of cases.

13.2 Architecture models

In order to create a method capable of estimating the length, expressed in lines of code, of a generic architecture, it is first necessary to examine which are its internals, and how they are arranged. During this discussion we will introduce two more syntax objects, that were not presented in chapter 3 for sake of simplicity.

An architecture can be roughly split into two parts:

- the declarative part, which can be intuitively thought of "everything comes before the `begin` keyword)" and syntactically corresponds to the expansion of the symbol `architecture_declarative_part`;
- the statement part, which can be thought of "everything comes after the `begin` keyword)" and syntactically corresponds to the expansion of the symbol `architecture_statement_part`;

The declarative part can contain type declarations, constant declarations, signal declarations, subprogram declarations and component declarations. The statement part can contain concurrent statements, processes and component instantiations. Our goal is to prove that estimating the length of an architecture by summing the lengths of each one of this constituents is easier (and leads to more accurate results) than estimating the length of an architecture as a whole.

To better illustrate how an architecture can be easily split into the above constituents, we will consider the following architecture code, which comes from project "xapp328", file `tst_dac3550a.vhd` (entity declaration is only reported for completeness, we will focus on the architecture only).

```
entity tst_dac3550a is
  port(
    -- I2C bus signals
    sda             : inout std_logic;
    scl             : in  std_logic;
    -- internal registers brought out on pins for viewing
    avol            : inout std_logic_vector(7 downto 0);  -- analog volume register
    gcfg            : inout std_logic_vector(7 downto 0);  -- config register
    sr              : inout std_logic_vector(7 downto 0);  -- sr_reg
    -- clock and reset
    sys_clk         : in  std_logic;
    porq            : in  std_logic
);  end tst_dac3550a;

architecture behave of tst_dac3550a is

-- ******************** CONSTANT DECLARATIONS ************************
constant RESET_ACTIVE : std_logic := '0';
constant DEV_WRITE  : std_logic_vector (7 downto 0) := "10011010"; -- 9A
constant GCFG_SUBADDR : std_logic_vector (7 downto 0) := "11000011"; -- C3
constant AVOL_SUBADDR : std_logic_vector (7 downto 0) := "11000010"; -- C2
constant SR_SUBADDR : std_logic_vector (7 downto 0) := "11000001"; -- C1
 [...  constants omitted  ...]

-- ******************** COMPONENT DECLARATIONS ************************
-- 8-bit serial load/parallel  shift  register  component SHIFT8
    port(
        clock       : in  std_logic;                        -- Clock
        reset       : in  std_logic;                        -- Active low clear
        data_ld     : in  std_logic;                        -- Data load enable
        data_in     : in  std_logic_vector (7 downto 0);    -- 8-bit data to load
        shift_in    : in  std_logic;                        -- Serial data in
        shift_en    : in  std_logic;                        -- Shift enable
        shift_out   : out std_logic;                        -- Bit to shift out
        data_out    : out std_logic_vector (7 downto 0));   -- 8-bit parallel out
end component;

-- Up counter - 4 bit component UPCNT4
    port(
        data        : in  std_logic_vector (3 downto 0);    -- Serial data in
        cnt_en      : in  std_logic;                        -- Count enable
        load        : in  std_logic;                        -- Load line enable
        reset       : in  std_logic;                        -- Active low clear
        clock       : in  std_logic;                        -- Clock
        qout        : out std_logic_vector (3 downto 0));
end component;
```

```
-- ********************* SIGNAL DECLARATIONS **********************
type state_type is (IDLE, HEADER, ACK_HEADER, RCV_SUBADDR, ACK_SUBADDR, RCV_DATA, ACK_DATA);
signal state          : state_type;
signal scl_in         : std_logic;    -- sampled version of scl
signal scl_in_d1      : std_logic;    -- delayed version of scl input
signal sda_in         : std_logic;    -- sampled version of sda
signal sda_in_d1      : std_logic;    -- delayed version of sda input
signal sda_out        : std_logic;    -- combinatorial sda output from scl generator state machine
 [... signals omitted ...]

begin
  -- set SDA
  sda <= '0' when sda_oe = '1' else 'Z';
  -- sda_oe is set slave and data to be output is 0
  sda_oe <= '1' when slave_sda = '0' else '0';

  -- these signals are here because Viewlogic won't allow assignment
  -- of a constant to a component port
  cnt_start <= ZERO_CNT;
  data_zero <= ZERO_DATA;

  -- invert SCL so that falling edge is used
  not_scl <= not(scl);
  -- invert power on reset
  not_porq <= not(porq);

-- ********************** Input Registers Process **********************
-- This process samples the incoming SDA and SCL with the system clock

   input_regs: process(sys_clk,porq)
   begin
      if porq = RESET_ACTIVE then
         sda_in <= '1';
         sda_in_d1 <= '1';
         scl_in <= '1';
         scl_in_d1 <= '1';
      elsif sys_clk'event and sys_clk = '1' then
         -- the following if, then, else clauses are used
         -- because scl may equal 'H' or '1'
         if scl = '0' then
            scl_in <= '0';
         else
            scl_in <= '1';
         end if;
         scl_in_d1 <= scl_in;
         if sda = '0' then
            sda_in <= '0';
         else
            sda_in <= '1';
         end if;
         sda_in_d1 <= sda_in;
      end if;
   end process;

-- ********************** Detect Start/Stop Process **********************
-- This process detects the start and stop conditions

   detect_start_stop: process(sys_clk,porq)
   begin
      if porq = RESET_ACTIVE then
         detect_start <= '0';
         detect_stop <= '0';
      elsif sys_clk'event and sys_clk = '1' then

         -- verify SCL is '1' then check for rising and falling edges of SDA
         --since SCL is open collector line, hard to check for '1', check for not 0
         if scl_in /= '0' and scl_in_d1 /= '0' then
            if sda_in = '0' and sda_in_d1 /= '0' then
               -- falling edge of SDA while SCL = 1 is a start condition
               detect_start <= '1';
            elsif state = HEADER then
               detect_start <= '0';
            else
```

```
                detect_start <= detect_start;
            end if;
            if sda_in /= '0' and sda_in_d1 = '0' then
                -- rising edge of SDA while SCL = 1 is a stop condition
                detect_stop <= '1';
            elsif detect_start = '1' then
                detect_stop <= '0';
            else
                detect_stop <= detect_stop;
            end if;
        end if;
    end if;
end process;

 [... processes omitted ...]

-- ********************** Address Match **********************
addr_match <= '1' when i2c_header(7 downto 1) = DEV_WRITE (7 downto 1) else '0';

-- ********************** Main State Machine Process **********************
-- The following process contains the main I2C state machine for both master and slave
-- modes. This state machine is clocked on the falling edge of SCL. DETECT_STOP must stay as
-- an asynchronous reset because once STOP has been generated, SCL clock stops.
state_machine: process (scl, porq, detect_stop)
begin
    if porq = RESET_ACTIVE or detect_stop = '1' then
        state <= IDLE;
    elsif scl'event and scl = '0' then

    case state is
            ---------------- IDLE STATE ----------------
            when IDLE =>
                if detect_start = '1' then
                    state <= HEADER;
                end if;
            ---------------- HEADER STATE ----------------
            when HEADER =>
                if bit_cnt = CNT_DONE then
                    state <= ACK_HEADER;
                end if;
            ---------------- ACK_HEADER STATE ----------------
            when ACK_HEADER =>
                if addr_match = '1' then
                    state <= RCV_SUBADDR;
                else
                    -- not addressed, go back to IDLE
                    state <= IDLE;
                end if;
             [... statements omitted ...]
            ---------------- ACK_DATA State ----------------
            when ACK_DATA =>
                state <= RCV_DATA;
        end case;
    end if;
end process;

 [... processes omitted ...]

-- ********************** I2C Data Shift Register **********************
I2CDATA_REG: SHIFT8
  port map (
            clock       => not_scl,
            reset       => not_porq,
            data_ld     => shift_reg_ld,
            data_in     => data_zero,
            shift_in    => sda_in,
            shift_en    => shift_reg_en,
            shift_out   => shift_out,
            data_out    => shift_reg );

i2cdata_reg_ctrl: process(sys_clk, porq)
begin
    if porq = RESET_ACTIVE then
        shift_reg_en <= '0';
```

```
195         elsif  sys_clk'event and sys_clk = '1' then
                if  (state = RCV_DATA) or (state = RCV_SUBADDR) then
                    shift_reg_en <= '1';
                else
                    shift_reg_en <= '0';
200             end if;
            end if;
        end process;

        shift_reg_ld <= '0';
205

        -- ************************ I2C Header Shift Register ************************
        -- Header/Address Shift Register
        I2CHEADER_REG: SHIFT8
          port map (
                    clock          => not_scl,
210                 reset          => not_porq,
                    data_ld        => i2c_header_ld,
                    data_in        => data_zero,
                    shift_in       => sda_in,
                    shift_en       => i2c_header_en,
215                 shift_out      => i2c_shiftout,
                    data_out       => i2c_header );

        i2cheader_reg_ctrl: process(sys_clk, porq)  begin
220         if  porq = RESET_ACTIVE then
                i2c_header_en <= '0';
            elsif  sys_clk'event and sys_clk = '1' then
                if  (detect_start  = '1')  or (state = HEADER) then
                    i2c_header_en <= '1';
225             else
                    i2c_header_en <= '0';
                end if;
            end if;
        end process;
230

        i2c_header_ld <= '0';

        -- ************************ Bit Counter ************************
        BITCNT : UPCNT4
235     port map( data         => cnt_start,
                  cnt_en       => bit_cnt_en,
                  load         => bit_cnt_ld,
                  reset        => not_porq,
                  clock        => not_scl,
240               qout         => bit_cnt );

        -- Counter control lines
        bit_cnt_en <= '1' when (state = HEADER) or (state = RCV_DATA)
                          or (state = RCV_SUBADDR) else '0';
245

        bit_cnt_ld <= '1' when (state = IDLE) or (state = ACK_HEADER)
                          or (state = ACK_DATA)
                       or (state = ACK_SUBADDR)
                       else  '0';
250 end behave;
```

We believe that the above example is rather interesting, since all of the possible constituents (except subprogram declarations) appear at least once[3].

[3] Note: in order to get a full listing of all the architectures containing components and processes, open a SQL shell and issue the command:
`SELECT DISTINCT P.PROJECT_NAME, P.ENTITY_NAME, P.ARCHITECTURE_NAME FROM VHDL.PROCESSES P, VHDL.COMPONENT_INSTANTIATIONS I WHERE P.PROJECT_NAME=I.PROJECT_NAME AND P.ENTITY_NAME=I.ENTITY_NAME AND P.ARCHITECTURE_NAME=I.ARCHITECTURE_NAME;`
you will obtain 98 such architectures.
If you want a full list of all the architectures satisfying all of the above conditions, and also containing subprograms declarations which are not included in processes, issue the following command:
`SELECT DISTINCT P.PROJECT_NAME, P.ENTITY_NAME, P.ARCHITECTURE_NAME FROM VHDL.PROCESSES P, VHDL.COMPONENT_INSTANTIATIONS I, VHDL.SUBPROGRAM_DECLARATIONS S WHERE P.PROJECT_NAME=I.ENTITY_NAME AND P.ENTITY_NAME=I.ENTITY_NAME AND`

In figure 13.13 the same architecture listing is split into its constituents. As said before, it is always possible to separate the lines of code of any architecture into type declarations, constant declarations, signal declarations, subprogram declarations, component declarations, concurrent statements, processes and component instantiations. Once the basic constituents were identified, it is easy to see whether the architecture shows a prevailing character of behavioral, structural or data-flow nature. In fact subprogram declarations and processes, being a collection of sequential statements and associated "encasings", constitute the behavioral part of the architecture; component declarations and instantiations are an expression of the structural part of an architecture; what else remains is the "proper part" of an architecture, strictly including its data-flow part (namely concurrent statements). This is better summarized by figure 13.14.

```
P.ARCHITECTURE_NAME=I.ARCHITECTURE_NAME AND S.PROJECT_NAME=P.PROJECT_NAME AND
S.ENTITY_NAME=P.ENTITY_NAME AND S.ARCHITECTURE_NAME=P.ARCHITECTURE_NAME AND
S.PROCESS_NAME='(null)' AND S.INSIDE_SUBPROGRAM='(null)';
```

In our project base there are no such architectures, otherwise we would have taken one of those in place of the above example architecture.

Figure 13.13: The example architecture split into its constituents.

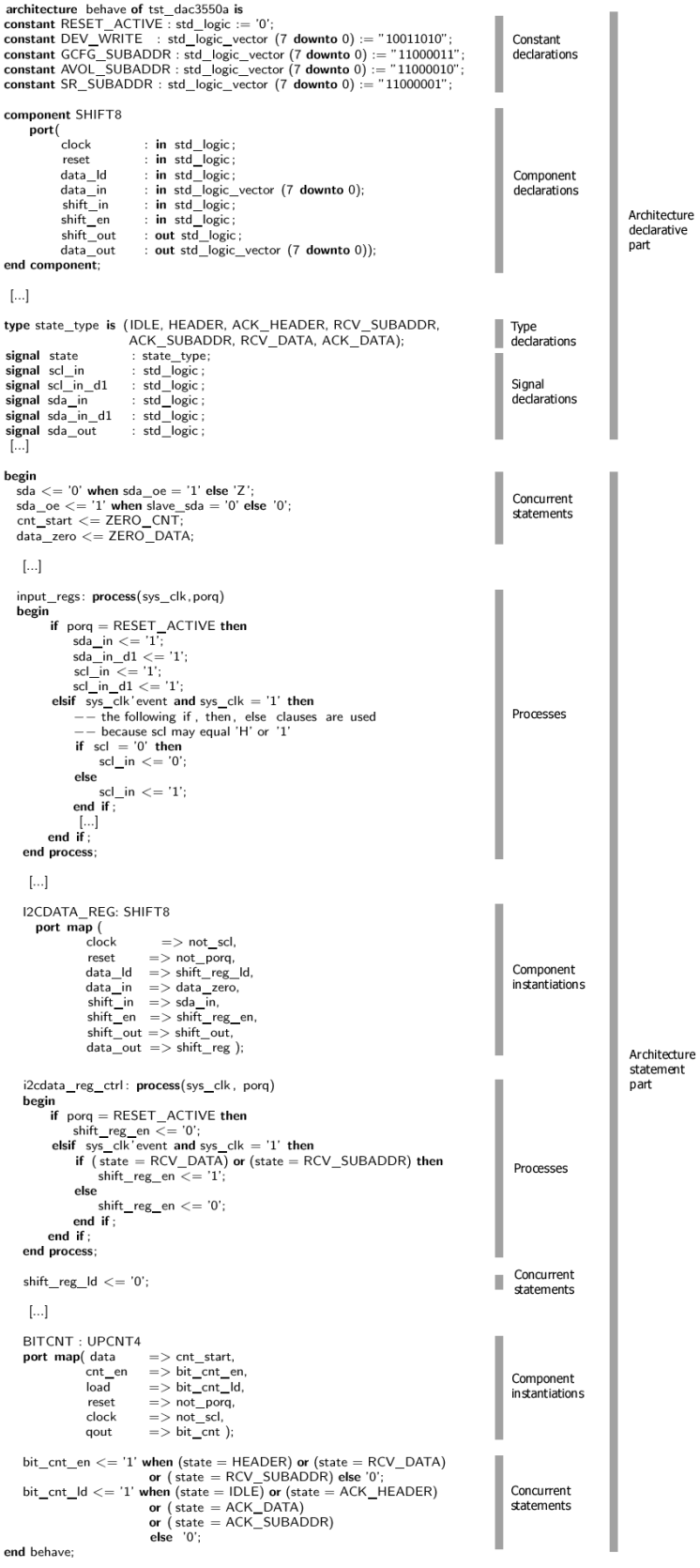
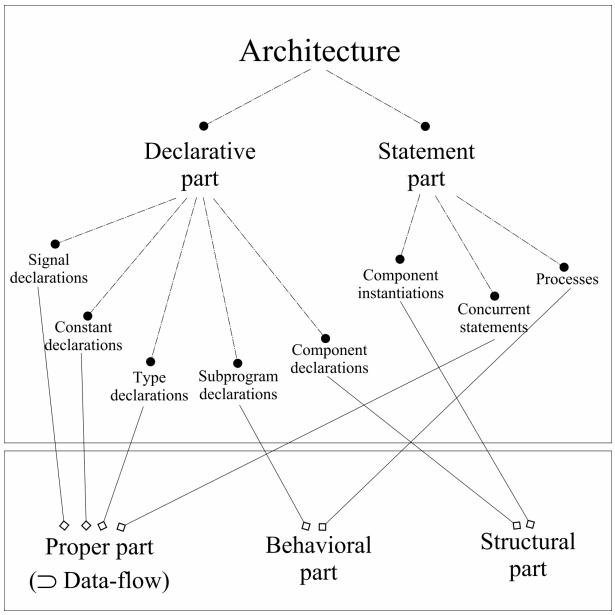


Figure 13.14: How architecture constituents contribute to its modality.



13.2.1 Model summary

The following models try to estimate the size of an architecture's proper part by knowing only a small amount of information, such as the number of ports or their homogeneity of the given architecture. Each information given to the models will be called *a variable* from now on; the problem of creating good architecture estimation models is basically reduced to choosing an appropriate set of variables and identifying a linear model on that variables. This is the full list of variables available for the model generation:

Variable	Symbol	Availability
Total number of ports[4]	n_p	Externally available
Total port homogeneity	h_p	Externally available
Number of ports per mode		Externally available
- number of `in` ports	n_{ip}	Externally available
- number of `out` ports	n_{op}	Externally available
- number of `inout` ports	n_{iop}	Externally available
- number of other ports	n_{xp}	Externally available
Sum of port homogeneity per mode		Externally available
- homogeneity of `in` ports	h_{ip}	Externally available
- homogeneity of `out` ports	h_{op}	Externally available
- homogeneity of `inout` ports	h_{iop}	Externally available
- homogeneity of other ports	h_{xp}	Externally available
Number of internal signals	n_s	Internally available
Homogeneity of internal signals	h_s	Internally available
Number of component instantiations	n_{ci}	Internally available

Part of the variables are declared as "externally available": this means that those variable are known when the architecture is externally known (as defined in the introductory chapters). "Internally available" variable instead require a higher degree of knowledge of the given architecture in order to be known. It is clear that our primary interest is for good models using externally available variables only. During this chapter we will examine both models using and not using internally available knowledge; we report the second ones only for study purposes, their usefulness in real estimation problems is limited to those conditions in which the used variables are actually known.

The following table gives a summary of all the models that will be studied in this section and the variable they use. Models indicated by an odd number use a very limited amount of variables and are not suitable to be actually used as estimators, they serve only as study models, for the assessment of the usefulness of the variables they use. These models were not completely developed in sub-models for behavioral, structural and data-flow architectures as we did for the even-numbered models. Even-numbered models are to be intended as real estimation models for production use and they were always developed in three flavors for the three architecture modes above, and respectively indicated with a "b", "s" or "d" tag.

Models indicated by an "H" use port homogeneity data, whereas their counterparts with the same number but no "H" use port count data.

Model:	Externally known variables				Internally known variables		
	n_p	h_p	n_{ip} ... n_{xp}	h_{ip} ... h_{xp}	n_s	h_s	n_{ci}
1	✓	·	·	·	·	·	·
1H	·	✓	·	·	·	·	·
2	·	·	✓	·	·	·	·
2H	·	·	·	✓	·	·	·
3	·	·	·	·	✓	·	·
3H	·	·	·	·	·	✓	·
4	·	·	✓	·	✓	·	·
4H	·	·	·	✓	·	✓	·
5	·	·	·	·	✓	·	✓
5H	·	·	·	·	·	✓	✓
6	·	·	✓	·	✓	·	✓
6H	·	·	·	✓	·	✓	✓

13.2.2 Model variable correlation

Unlike for entities, it is difficult to identify variables useful for creating good architecture length estimation models. In fact, all external variables show little or no correlation at all with L. The only variable exhibiting a high coefficient of correlation with L is n_s (i.e. the number of signals declared inside the architecture), which is an internally available variable.

Variable	Average value	Variance	Standard deviation	Correlation coefficient
Externally available variables				
n_p	10.7836	189.0364	13.7491	0.1629
n_{ip}	6.4738	59.0035	7.6814	0.1090
n_{op}	3.7849	54.8280	7.4046	0.1615
n_{iop}	0.4725	3.3316	1.8253	0.0932
n_{xp}	0.0525	0.1549	0.3936	0.0923
h_p	12.2522	312.3324	17.6729	0.1625
h_{ip}	7.2971	89.5168	9.4613	0.1067
h_{op}	4.4558	103.1279	10.1552	0.1645
h_{iop}	0.4456	3.3422	1.8282	0.0847
h_{xp}	0.0538	0.1586	0.3983	0.0905
n_g	3.3572	52.0325	7.2134	0.0127
Internally available variables				
n_s	26.0051	23912.8000	154.6376	0.1686
h_s	15.1536	2053.7174	45.3180	**0.6016**
n_{proc}	1.5262	21.3317	4.6186	0.2205
n_{cd}	0.4110	1.7578	1.3258	0.0839
n_{ci}	2.5608	80.4210	8.9678	0.0107
$\hat{L}$	23.9757	80.4210	8.9678	(1.0000)

Above considerations suggest that it will not be easy to create good architecture models from the available data.

13.2.3 Model AM1

The model AM1 is a very simple linear model, using only the total number of ports declared in the entity to which the current architecture belongs.

Model:

$$\hat{L} = k_{np} \cdot n_p + k_0$$

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	23.976	23.976	0.000
Variance	7453.542	197.738	7255.804
Standard deviation	86.334	14.062	85.181

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.1629

Identified model coefficients:

Coefficient	Value
k_{np}	1.0228
k_0	12.9467

Figure 13.15: Model AM1: Real vs. estimated lines of code.

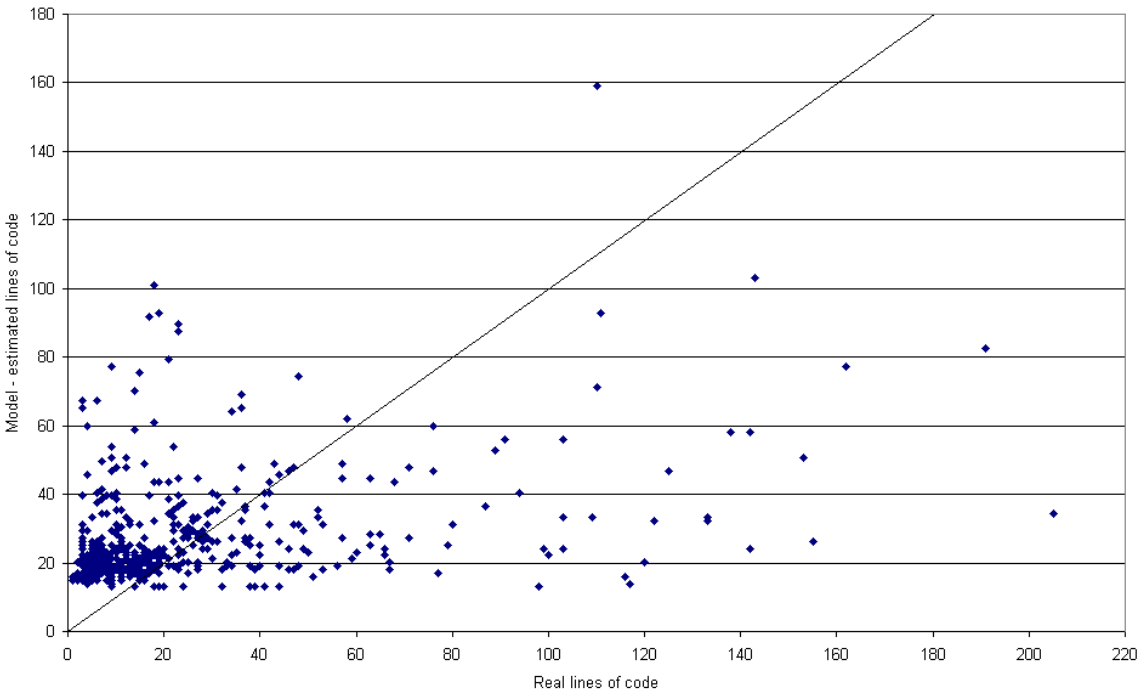
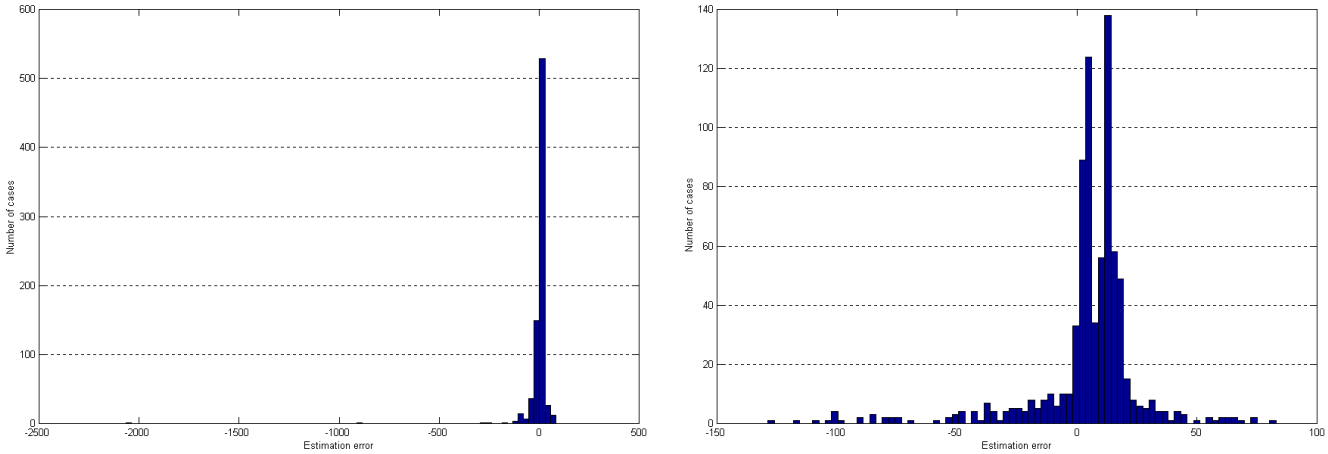
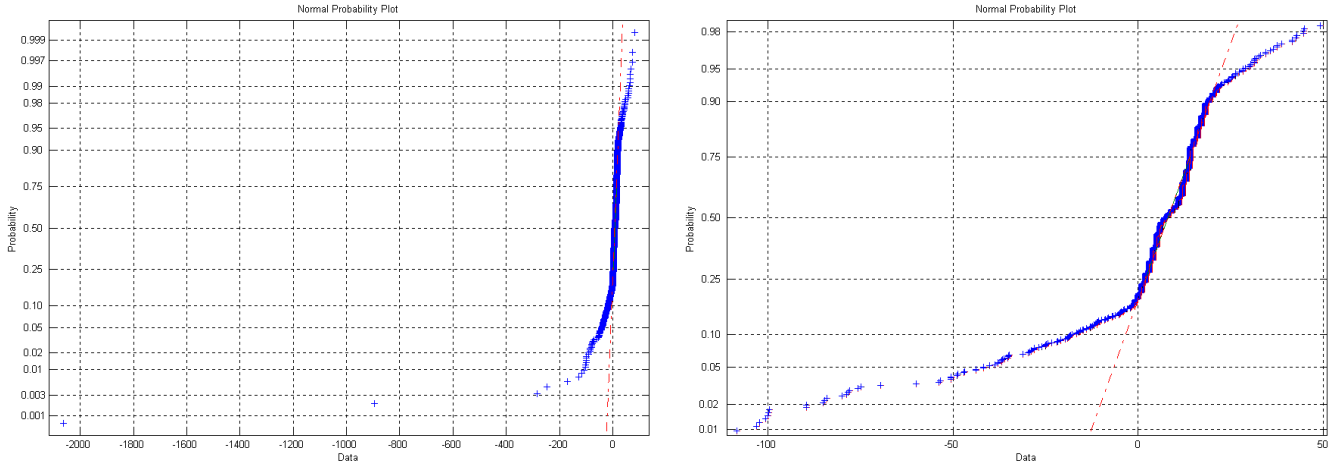


Figure 13.16: Model AM1: Error density distribution.



Remarks on error distribution: In 96% of the cases the error falls between -40 and +150 lines of code; in 93% of the cases the error falls between -40 and +100 lines of code; in 88% of the cases the error falls between -40 and +60 lines of code; in 50% of the cases the error falls between 0 and +40 lines of code.

Figure 13.17: Model AM1: Error cumulative distribution.



13.2.4 Model AM1H

Model AM1 is also a very simple linear model, using only the total homogeneity of the ports declared in the entity to which the current architecture belongs.

Model:

$$\hat{L} = k_{hp} \cdot h_p + k_0$$

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	23.976	23.976	0.000
Variance	7453.542	196.778	7256.764
Standard deviation	86.334	14.028	85.187

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.1625

Identified model coefficients:

Coefficient	Value
k_{np}	0.7937
k_0	14.2506

Figure 13.18: Model AM1H: Real vs. estimated lines of code.

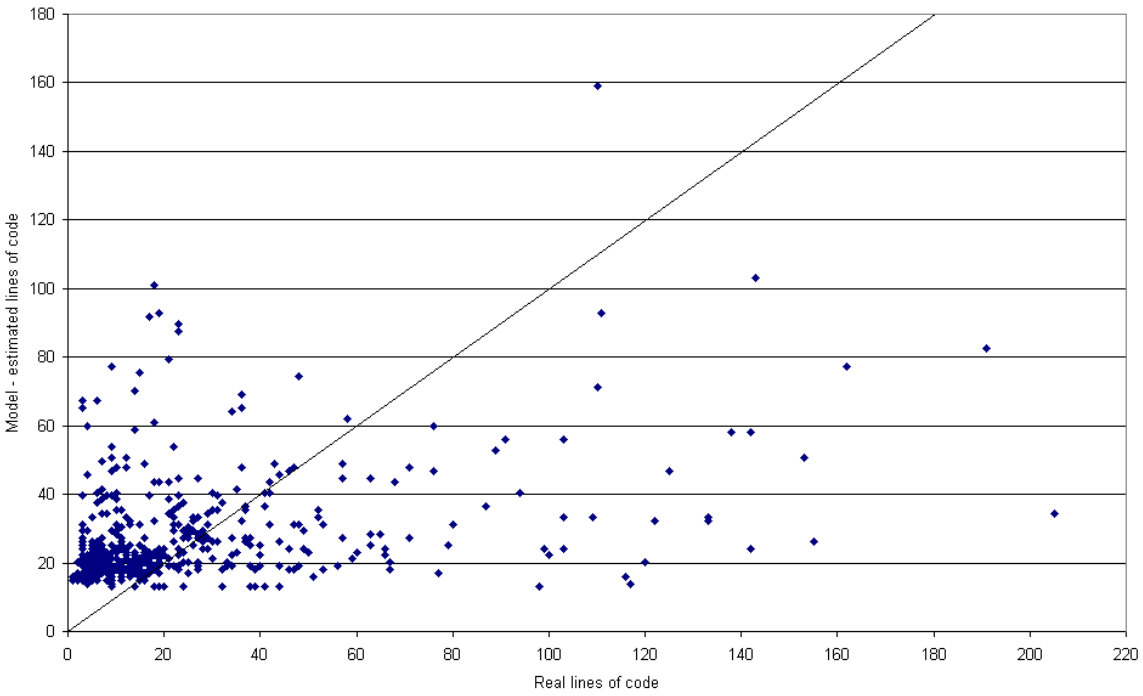
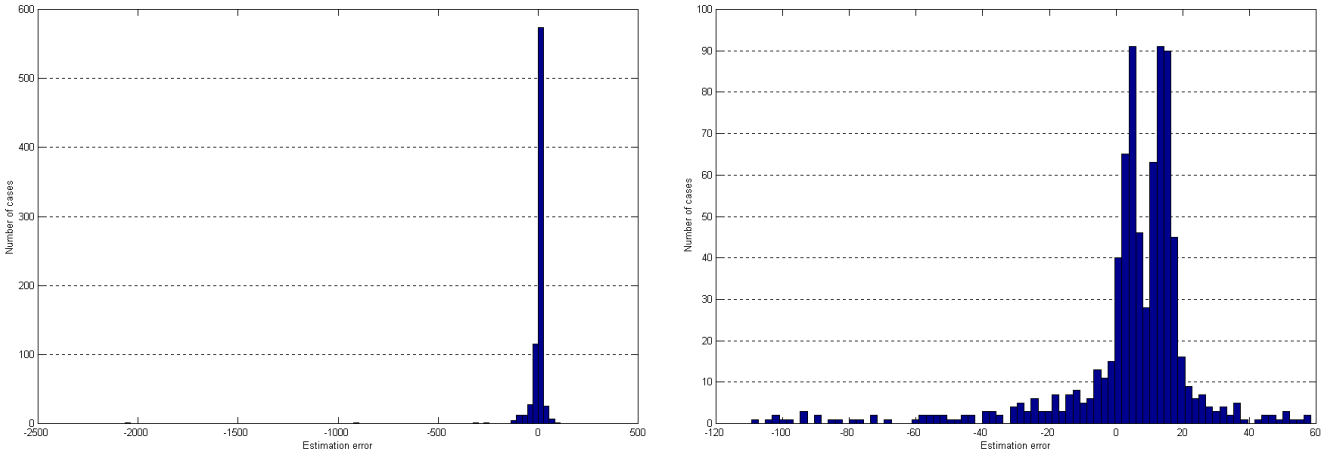
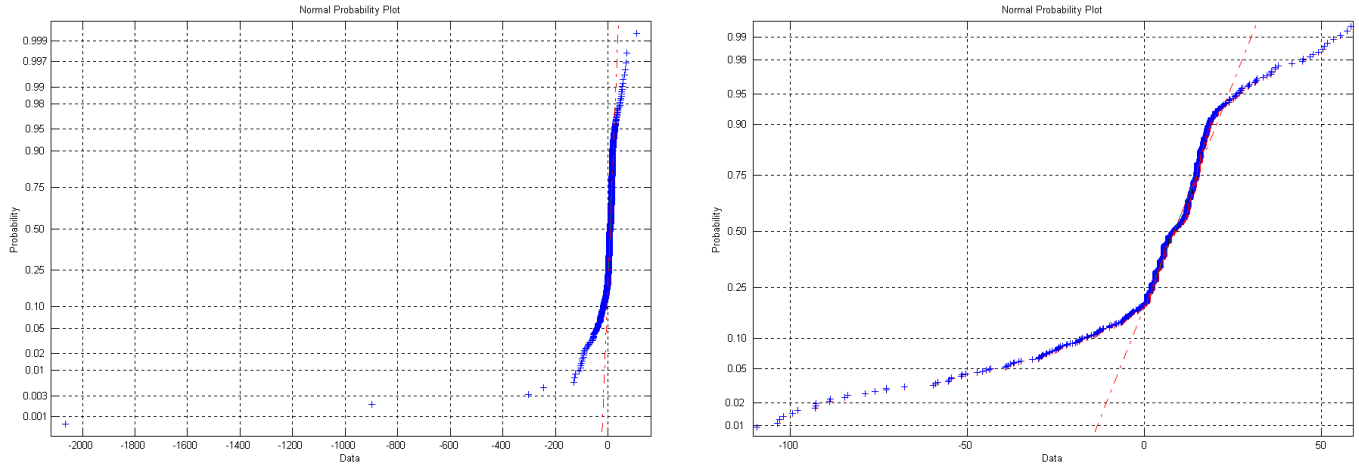


Figure 13.19: Model AM1H: Error density distribution.



Remarks on error distribution: In 98% of the cases the error falls between -110 and +60 lines of code; in 95% of the cases the error falls between -80 and +40 lines of code; in 90% of the cases the error falls between -40 and +25 lines of code; in 50% of the cases the error falls between +5 and +15 lines of code.

Figure 13.20: Model AM1H: Error cumulative distribution.



13.2.5 Model AM2

Model AM2 tries to improve accuracy by considering the partial count of ports per mode (the count of ports is given for `in`, `out`, `inout` and other ports.

Model:

$$\hat{L} = k_{nip} \cdot n_{ip} + k_{nop} \cdot n_{op} + k_{niop} \cdot n_{iop} + k_{nxp} \cdot n_{xp} + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	30.670	30.670	0.000
Variance	19931.806	395.991	19535.816
Standard deviation	141.180	19.900	139.771
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	11.173	11.173	0.000
Variance	136.887	52.738	84.149
Standard deviation	11.700	7.262	9.173
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	21.693	21.693	0.000
Variance	941.826	193.599	748.227
Standard deviation	30.689	13.914	27.354

Correlation between estimated and real values:

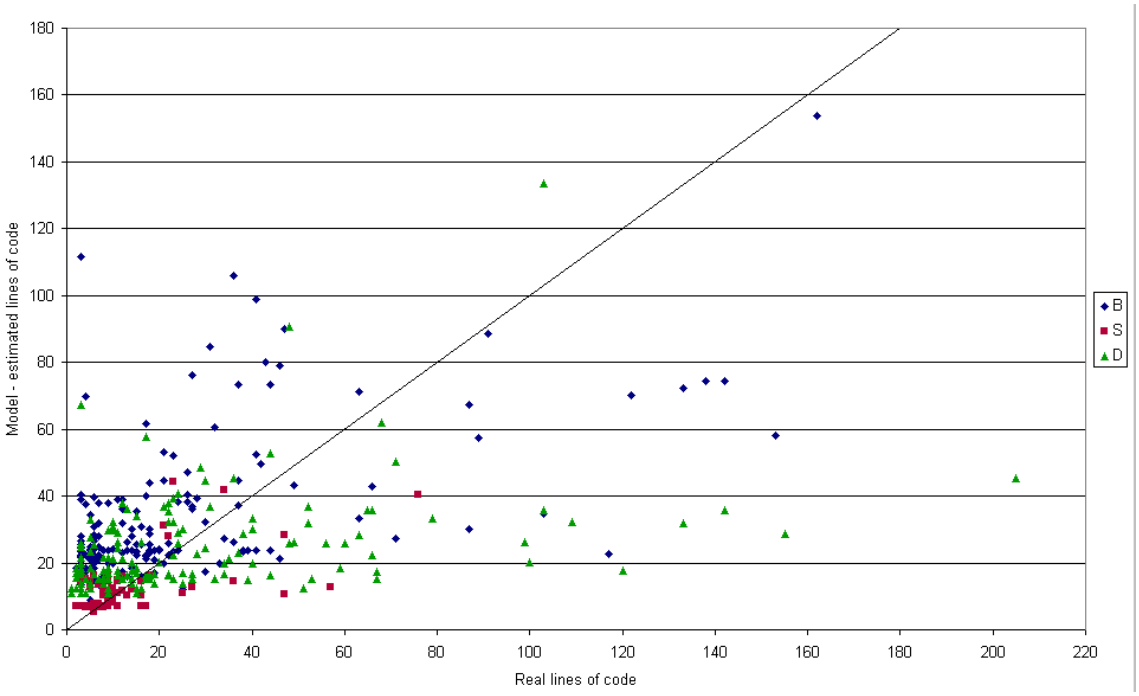
	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient (L,$\hat{L}$)	0.1410	0.6207	0.4534

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_{nip}	-1.2566	-0.0492	1.3448
k_{nop}	2.2719	0.8330	1.0738
k_{niop}	6.5459	7.3839	5.8286
k_{nxp}	-4.2766	-0.4115	11.7532
k_0	23.8559	6.3159	8.7304

Remarks on error distribution - Models AM2b, AM2s, AM2d: Except for 1 case, all the errors fall between -250 and +150 lines of code. In 98% of the cases the error falls between

Figure 13.21: Model AM2: Real vs. estimated lines of code.



-105 and +55 lines of code; in 96% of the cases the error falls between -75 and +40 lines of code; in 85% of the cases the error falls between -20 and +25 lines of code; in 50% of the cases the error falls between +0 and +20 lines of code.

Remarks on error distribution - behavioral architectures only: In 97% of the cases the error falls between -70 and +70 lines of code; in 93% of the cases the error falls between -55 and +55 lines of code; in 85% of the cases the error falls between -15 and +35 lines of code; in 50% of the cases the error falls between +5 and +20 lines of code.

Remarks on error distribution - structural architectures only: In 99% of the cases the error falls between -50 and +25 lines of code; in 96% of the cases the error falls between -37 and +12 lines of code; in 85% of the cases the error falls between -10 and +10 lines of code; in 50% of the cases the error falls between -3 and +5 lines of code.

Remarks on error distribution - data-flow architectures only: In 98% of the cases the error falls between -125 and +40 lines of code; in 93% of the cases the error falls between -50 and +30 lines of code; in 80% of the cases the error falls between -25 and +20 lines of code; in 50% of the cases the error falls between -5 and +15 lines of code.

Figure 13.22: Models AM2b, AM2s, AM2d: Real vs. estimated lines of code.

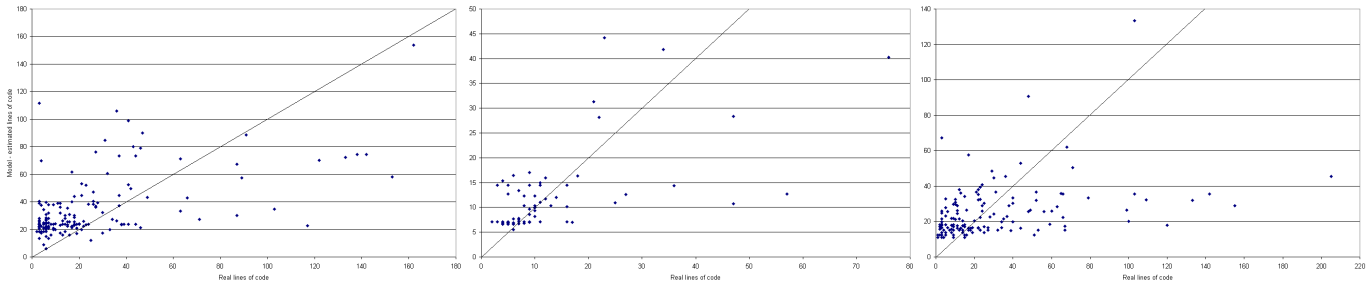


Figure 13.23: Model AM2: Error density distribution.

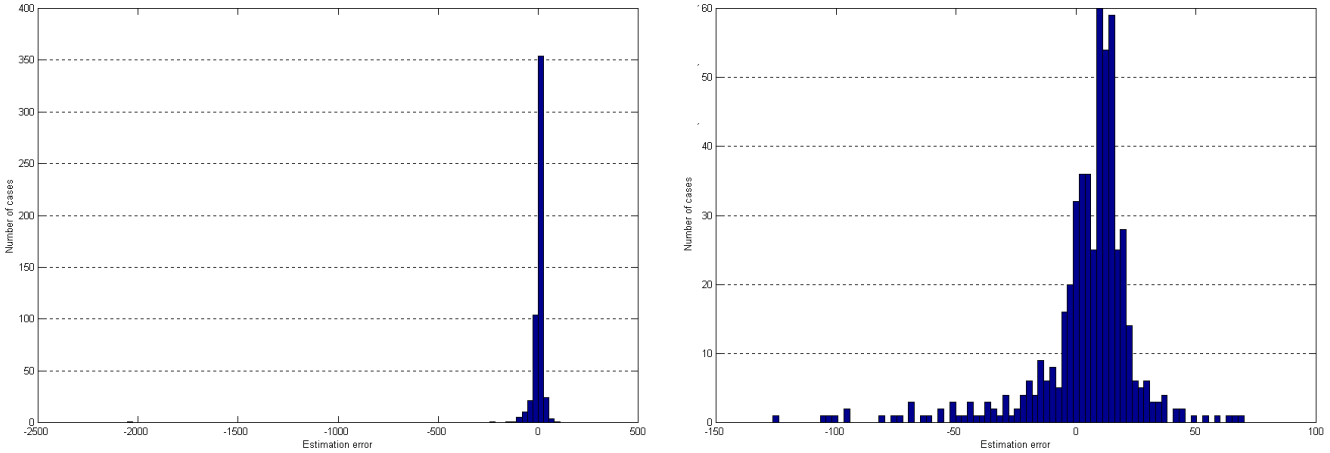


Figure 13.24: Model AM2: Error cumulative distribution.

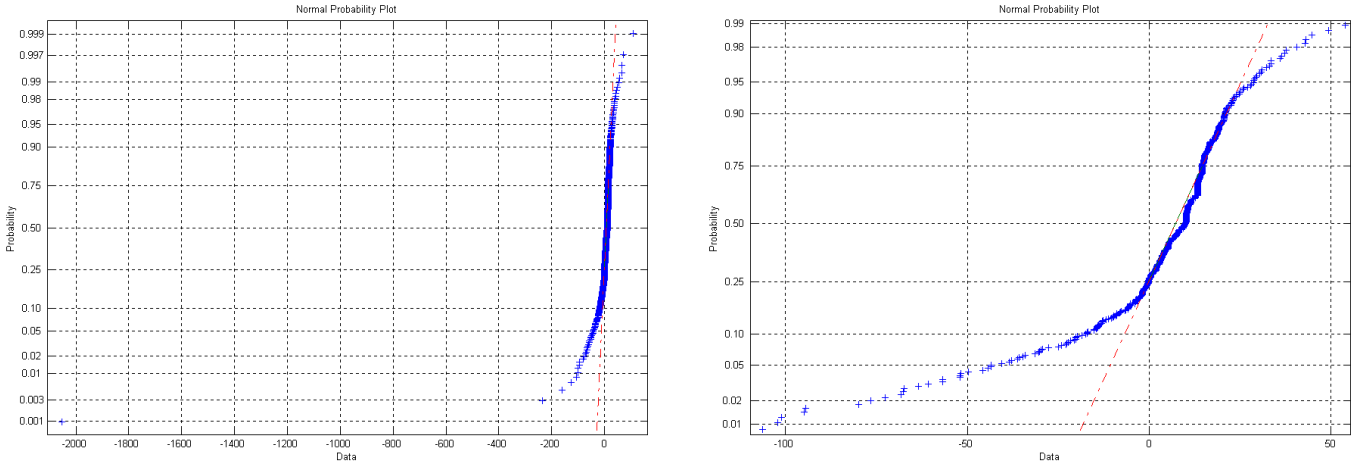
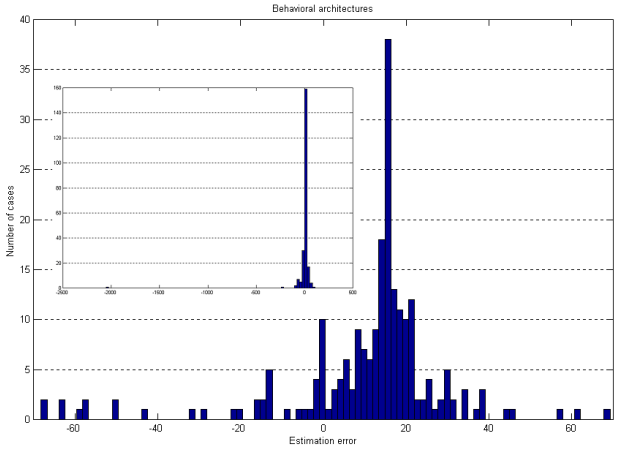
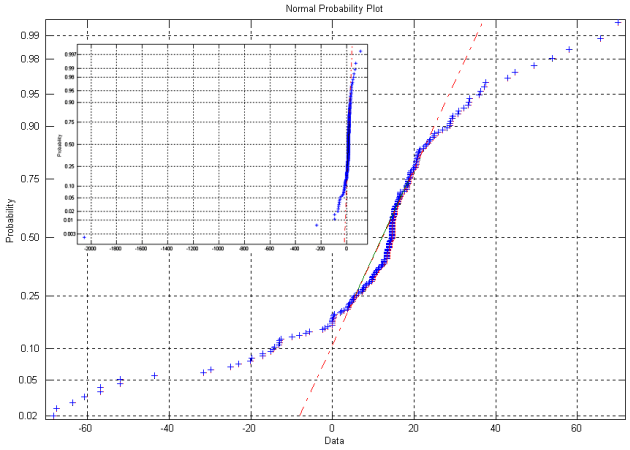
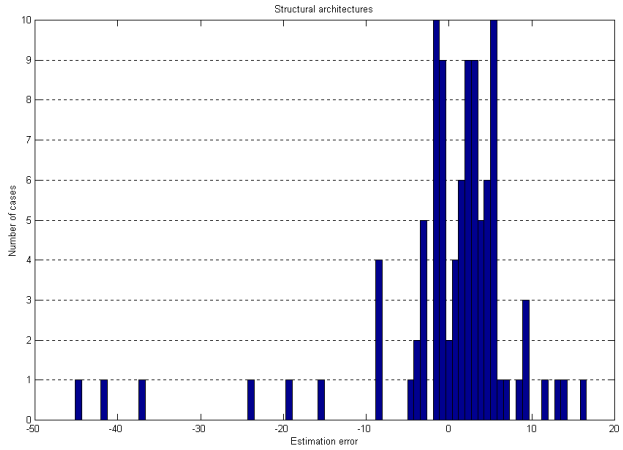
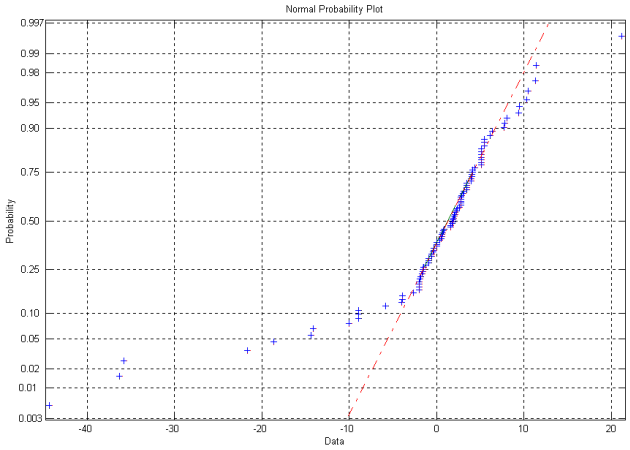


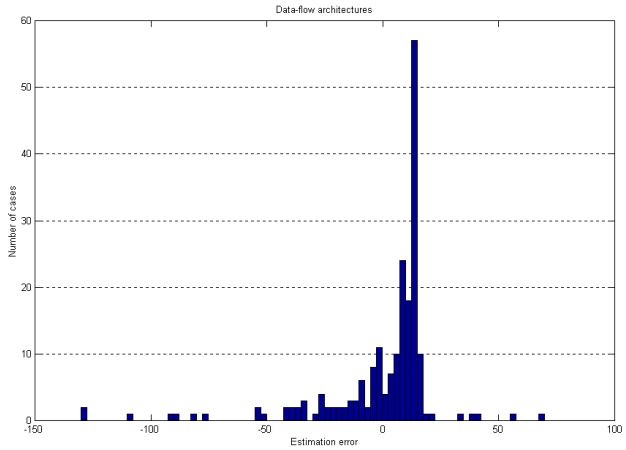
Figure 13.25: Models AM2b, AM2s, AM2d: Error density and cumulative distributions.

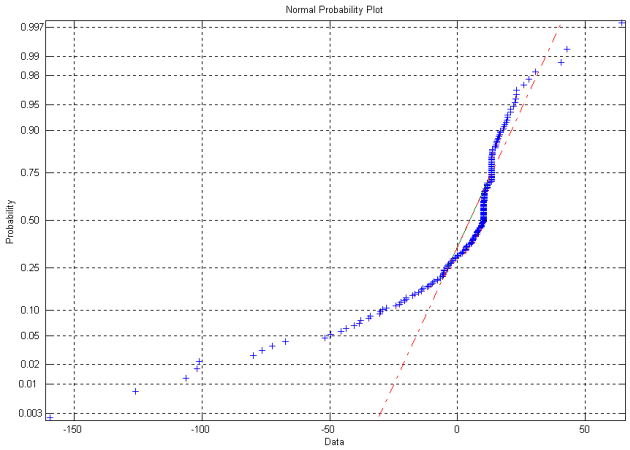












13.2.6 Model AM2H

Model AM2H is the dual version of model AM2, the difference consists in the fact that AM2H uses tries to improve accuracy by considering the partial count of ports per mode (the count of ports is given for `in`, `out`, `inout` and other ports.

Model:

$$\hat{L} = k_{hip} \cdot h_{ip} + k_{hop} \cdot h_{op} + k_{hiop} \cdot h_{iop} + k_{hxp} \cdot h_{xp} + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	30.6696	30.6696	0.0000
Variance	19931.8063	399.3456	19532.4607
Standard deviation	141.1800	19.9836	139.7586
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	11.173	11.173	0.000
Variance	136.887	48.382	88.505
Standard deviation	11.700	6.956	9.408
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	21.693	21.693	0.000
Variance	941.826	276.652	665.174
Standard deviation	30.689	16.633	25.791

Correlation between estimated and real values:

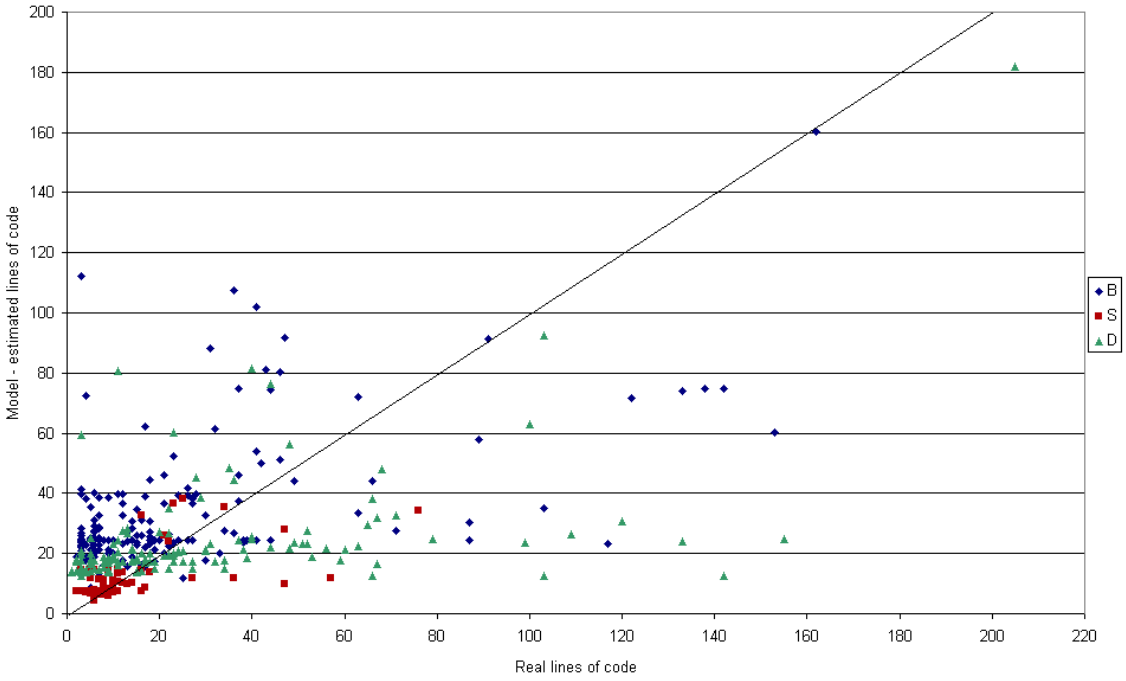
	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient (L,$\hat{L}$)	0.1415	0.5945	0.5420

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_{hip}	-1.3060	-0.0492	0.2981
k_{hop}	2.2869	0.8330	0.9575
k_{hiop}	6.8408	7.3839	3.9877
k_{hxp}	-4.3890	-0.4115	-0.5565
k_0	24.3908	6.3159	12.5590

Remarks on error distribution - Models AM2Hb, AM2Hs, AM2Hd: Except for 1 case, all the errors fall between -250 and +150 lines of code. In 97% of the cases the error falls

Figure 13.26: Model AM2H: Real vs. estimated lines of code.



between -90 and +45 lines of code; in 96% of the cases the error falls between -75 and +40 lines of code; in 85% of the cases the error falls between -25 and +30 lines of code; in 50% of the cases the error falls between +0 and +20 lines of code.

Remarks on error distribution - behavioral architectures only: In 99% of the cases the error falls between -100 and +115 lines of code; in 96% of the cases the error falls between -70 and +55 lines of code; in 85% of the cases the error falls between -15 and +35 lines of code; in 50% of the cases the error falls between +5 and +20 lines of code.

Remarks on error distribution - structural architectures only: In 98% of the cases the error falls between -45 and +15 lines of code; in 96% of the cases the error falls between -40 and +15 lines of code; in 85% of the cases the error falls between -8 and +10 lines of code; in 50% of the cases the error falls between -2 and +5 lines of code.

Remarks on error distribution - data-flow architectures only: In 99% of the cases the error falls between -130 and +70 lines of code; in 96% of the cases the error falls between -90 and +30 lines of code; in 85% of the cases the error falls between -30 and +15 lines of code; in 50% of the cases the error falls between -5 and +15 lines of code.

Figure 13.27: Models AM2Hb, AM2Hs, AM2Hd: Real vs. estimated lines of code.

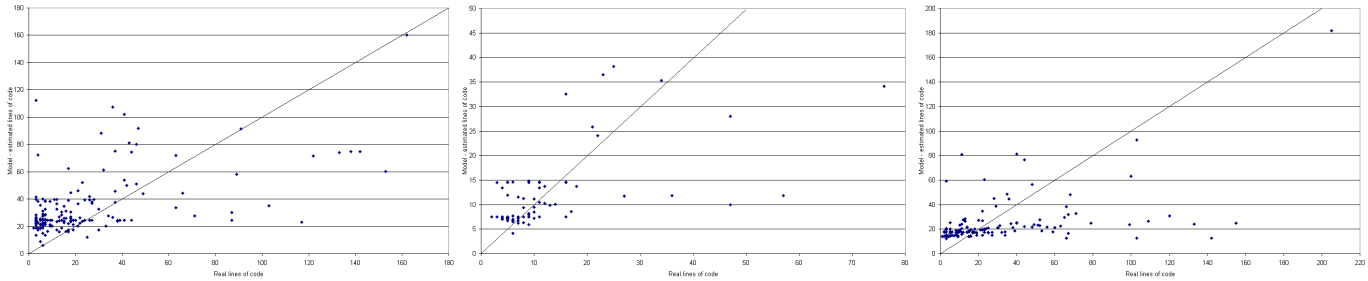


Figure 13.28: Model AM2H: Error density distribution.

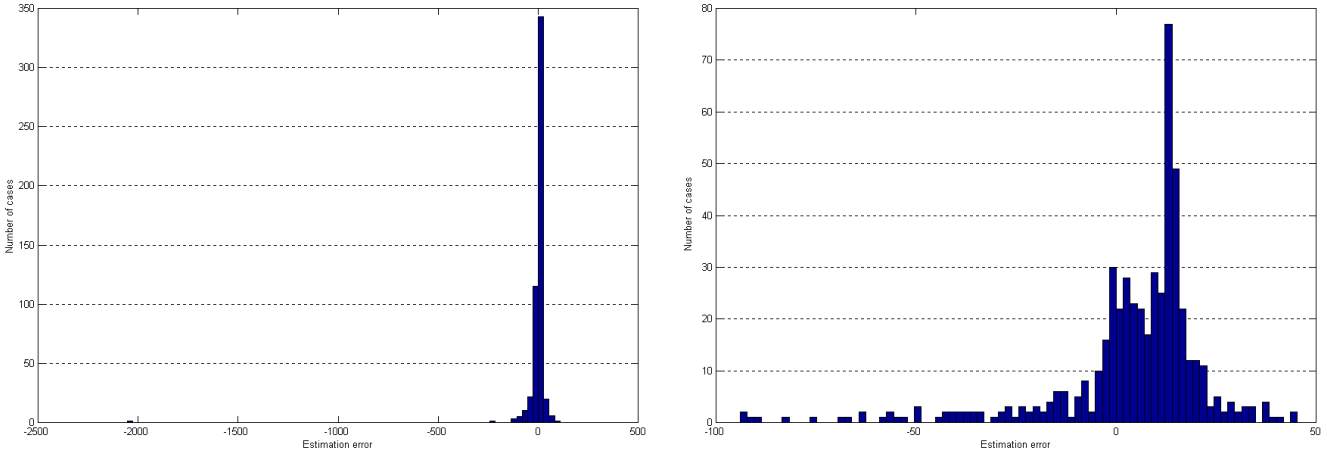


Figure 13.29: Model AM2H: Error cumulative distribution.

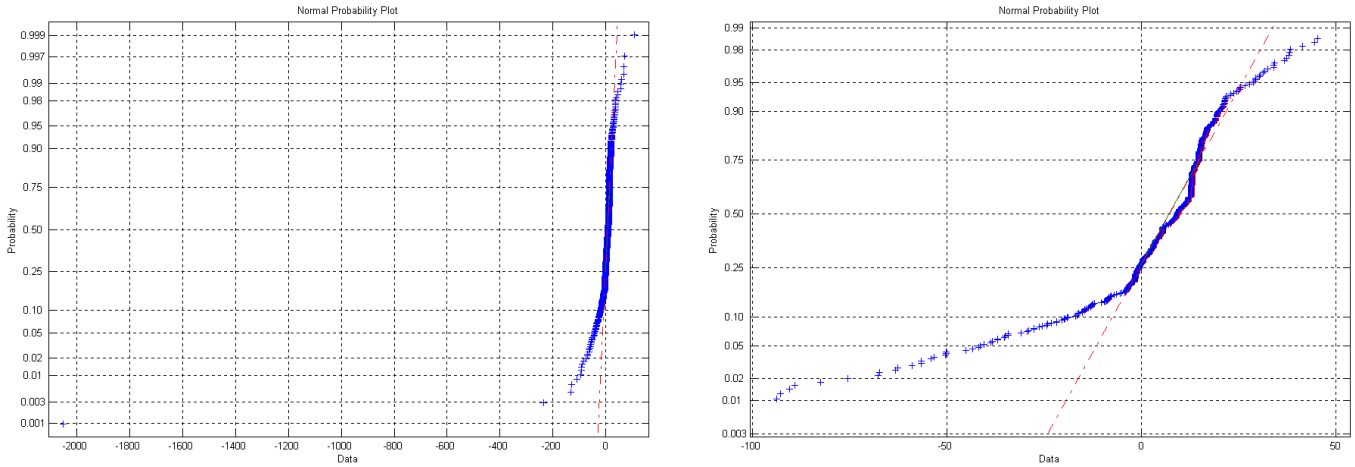
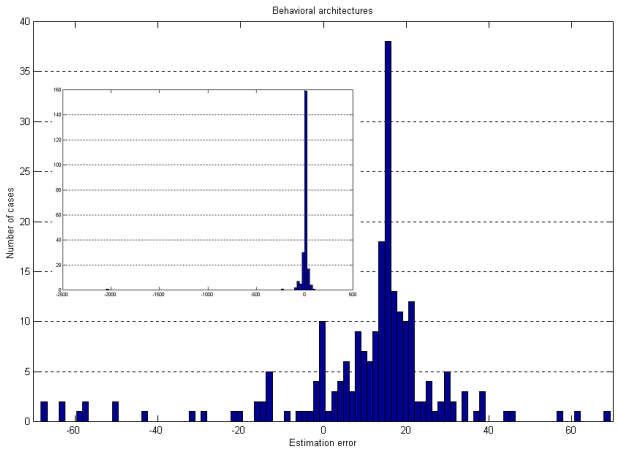
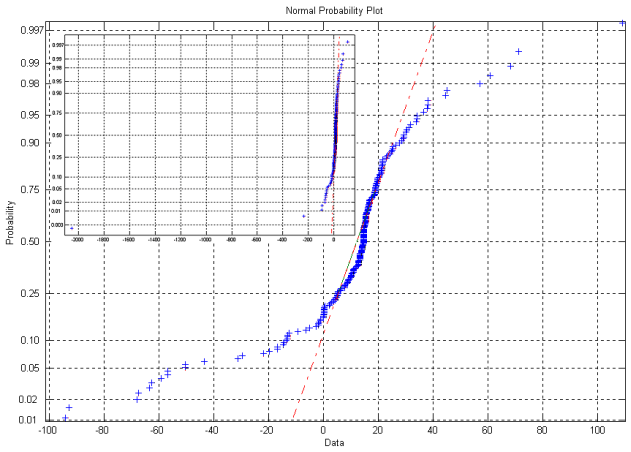
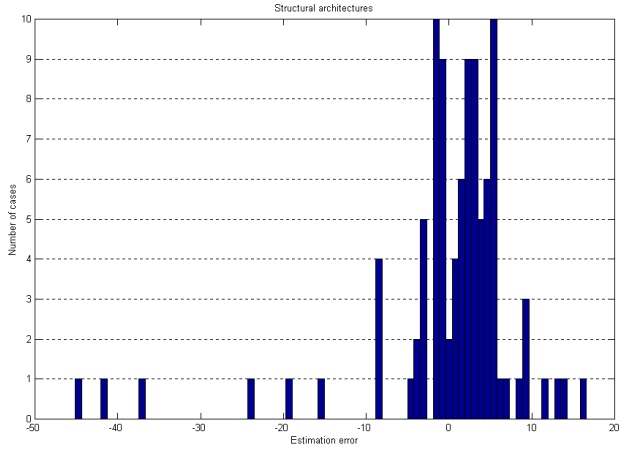
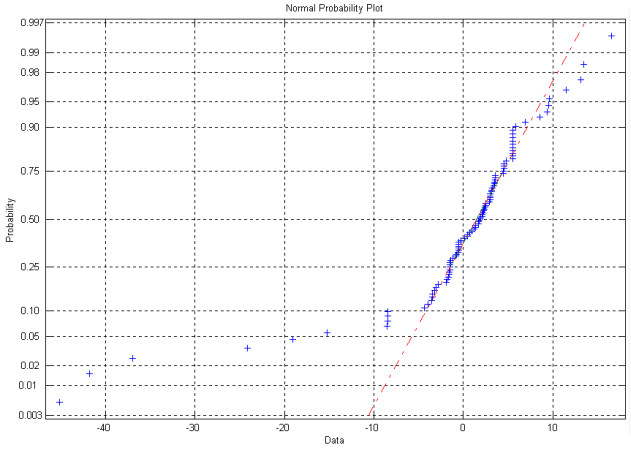


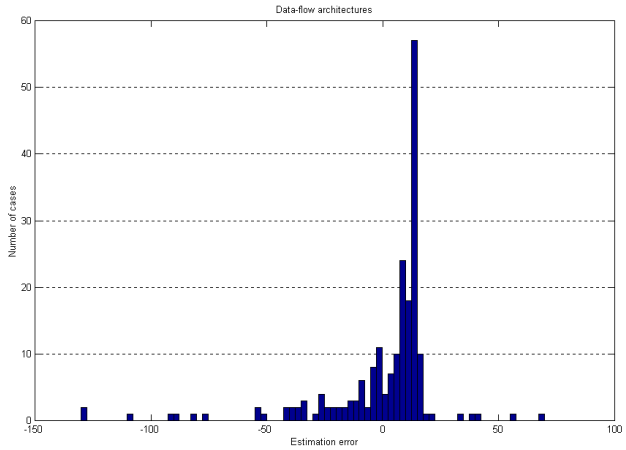
Figure 13.30: Models AM2Hb, AM2Hs, AM2Hd: Error density and cumulative distributions.

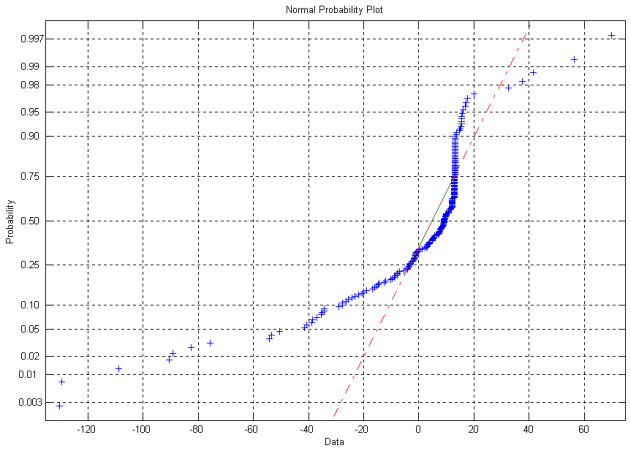












13.2.7 Model AM3

Model:

$$\hat{L} = k_{ns} \cdot n_s + k_0$$

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	23.976	23.976	0.000
Variance	7453.542	211.993	7241.548
Standard deviation	86.334	14.560	85.097

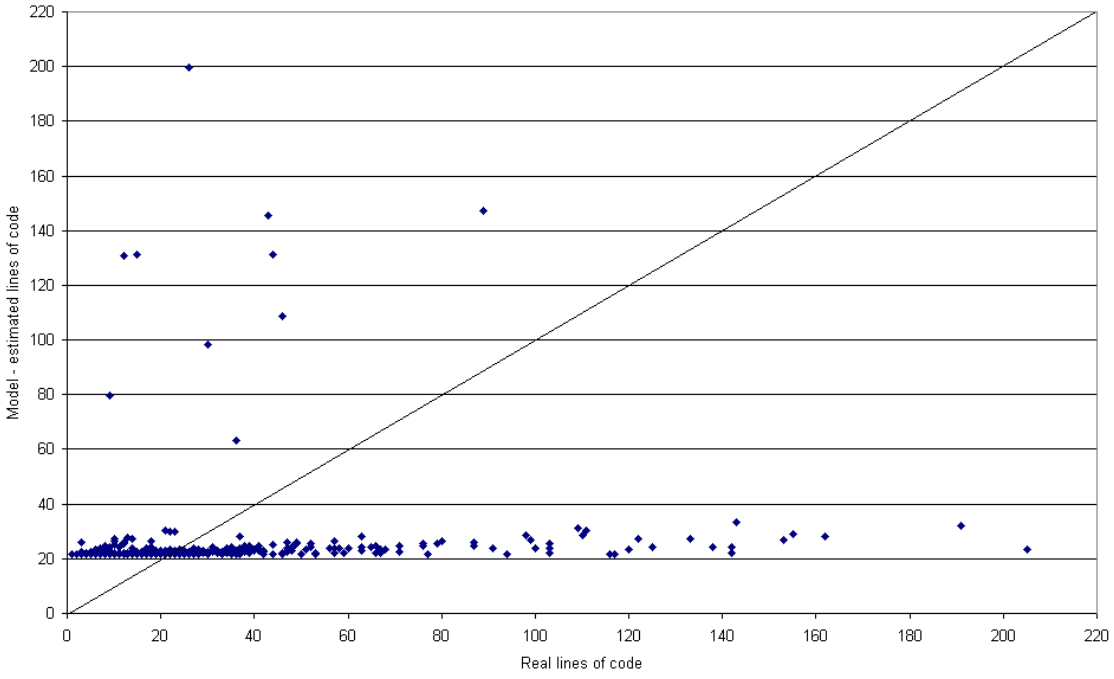
Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.1686

Identified model coefficients:

Coefficient	Value
k_{ns}	0.0942
k_0	21.5271

Figure 13.31: Model AM3: Real vs. estimated lines of code.



Remarks on error distribution: In 96% of the cases the error falls between -40 and +150 lines of code; in 93% of the cases the error falls between -40 and +100 lines of code; in 88% of the cases the error falls between -40 and +60 lines of code; in 50% of the cases the error falls between 0 and +40 lines of code.

Figure 13.32: Model AM3: Error density distribution.

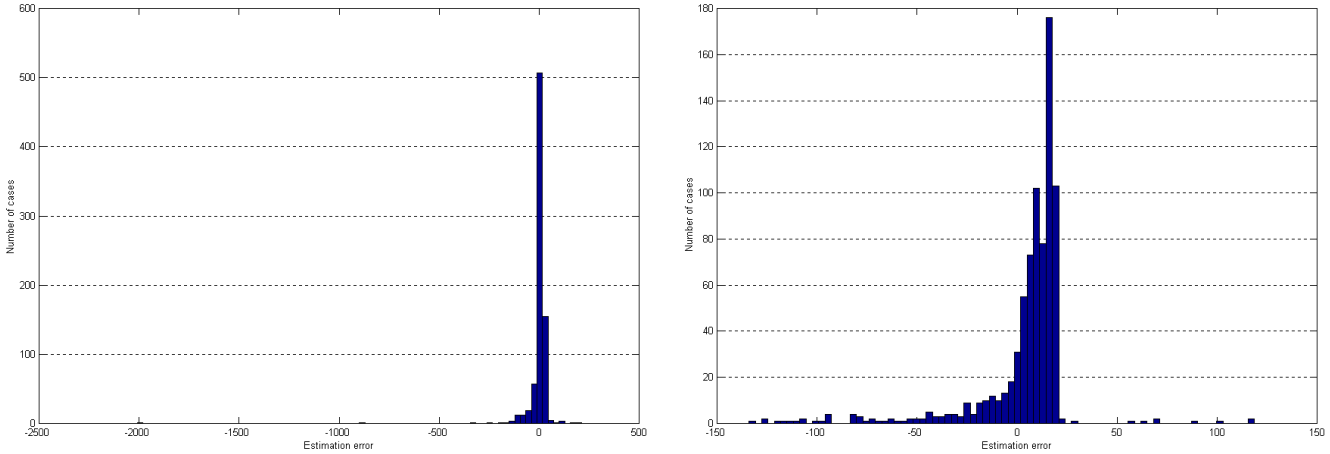
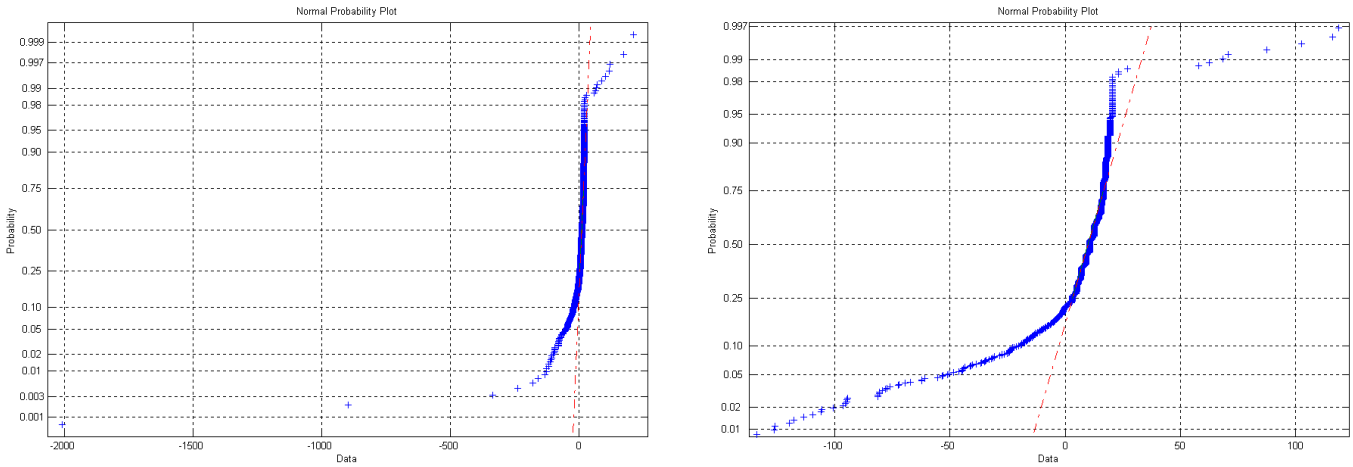


Figure 13.33: Model AM3: Error cumulative distribution.



13.2.8 Model AM3H

Model:

$$\hat{L} = k_{hs} \cdot h_s + k_0$$

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	23.976	23.976	0.000
Variance	7453.542	309.902	7143.640
Standard deviation	86.334	17.604	84.520

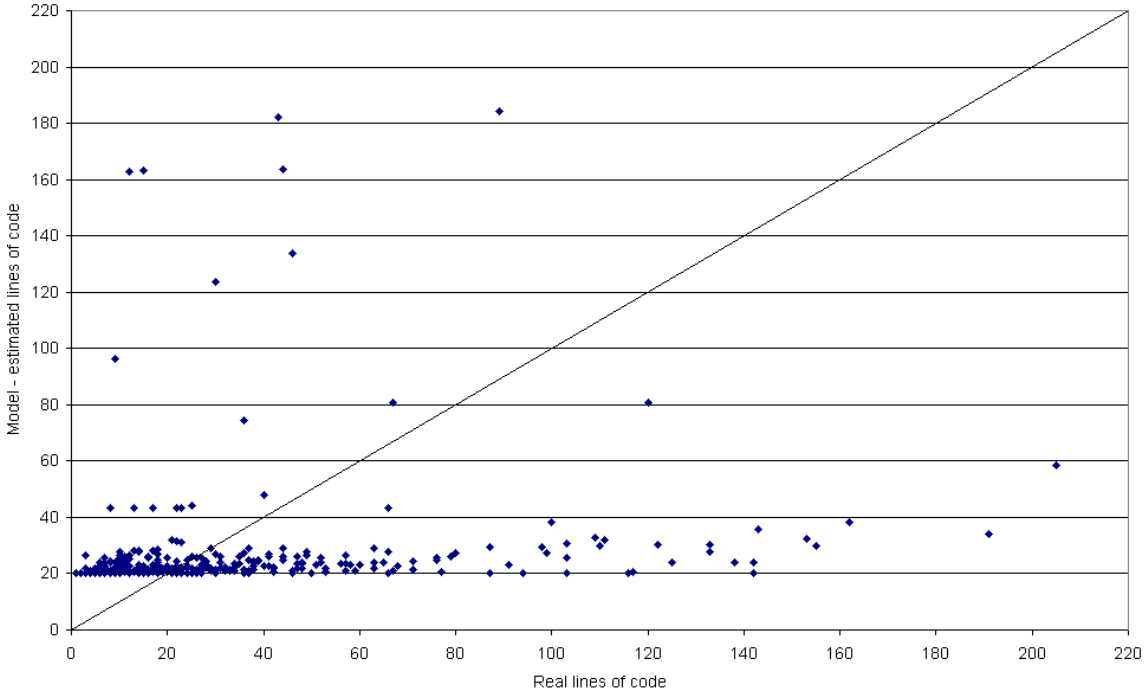
Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.2039

Identified model coefficients:

Coefficient	Value
k_{hs}	0.1229
k_0	20.3451

Figure 13.34: Model AM3H: Real vs. estimated lines of code.



Remarks on error distribution: In 98% of the cases the error falls between -125 and +80 lines of code; in 96% of the cases the error falls between -100 and +25 lines of code; in 88% of the cases the error falls between -20 and +25 lines of code; in 50% of the cases the error falls between +5 and +20 lines of code.

Figure 13.35: Model AM3H: Error density distribution.

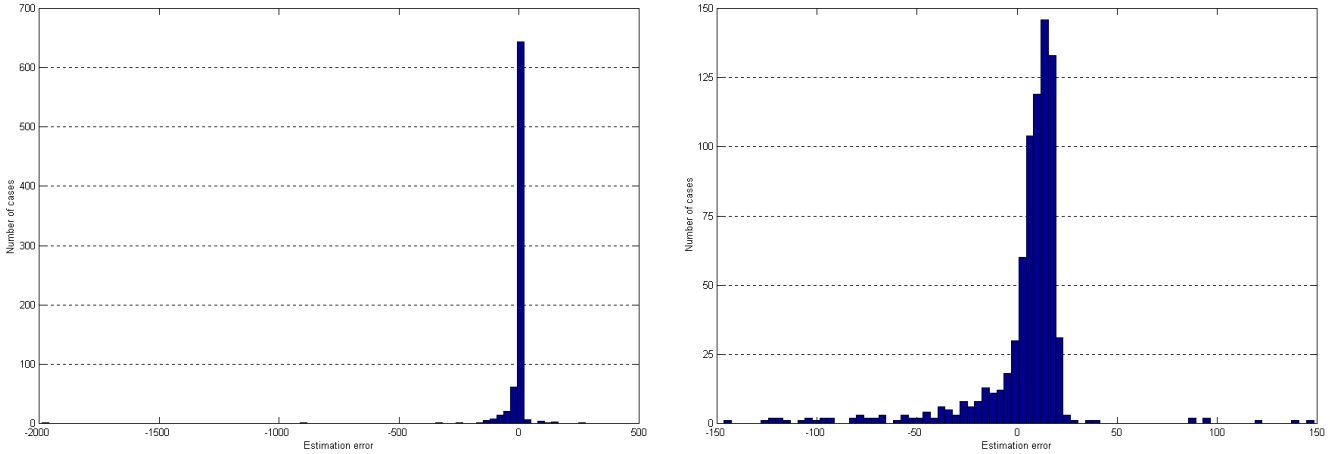
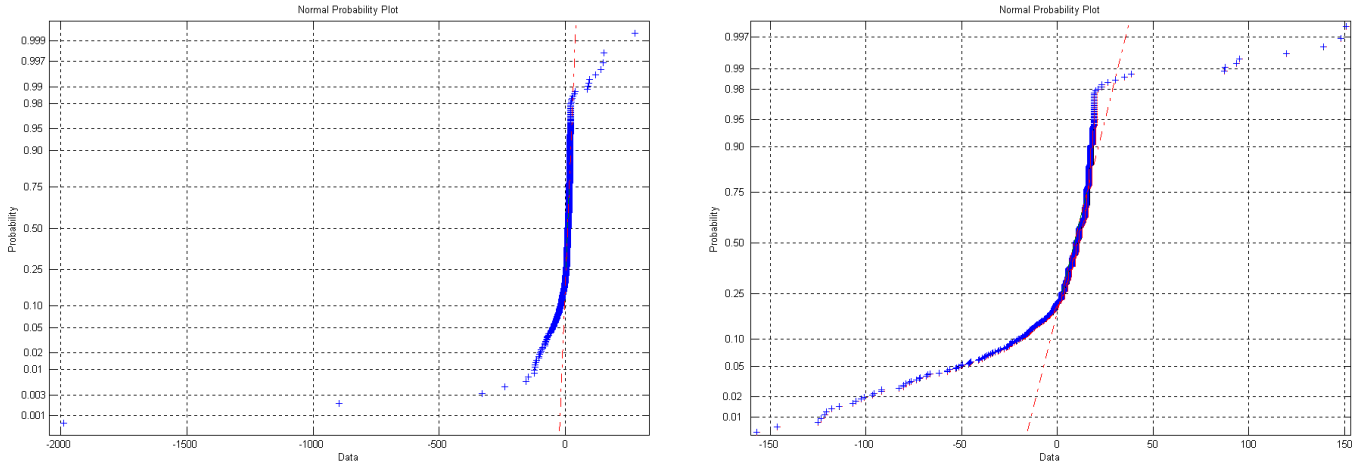


Figure 13.36: Model AM3H: Error cumulative distribution.



13.2.9 Model AM4

Model:

$$\hat{L} = k_{nip} \cdot n_{ip} + k_{nop} \cdot n_{op} + k_{niop} \cdot n_{iop} + k_{nxp} \cdot n_{xp} + k_{ns} \cdot n_s + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	30.670	30.670	0.000
Variance	19931.806	17359.456	2572.351
Standard deviation	141.180	131.755	50.718
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	11.173	11.173	0.000
Variance	136.887	54.510	82.377
Standard deviation	11.700	7.383	9.076
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	21.693	21.693	0.000
Variance	941.826	362.521	579.305
Standard deviation	30.689	19.040	24.069

Correlation between estimated and real values:

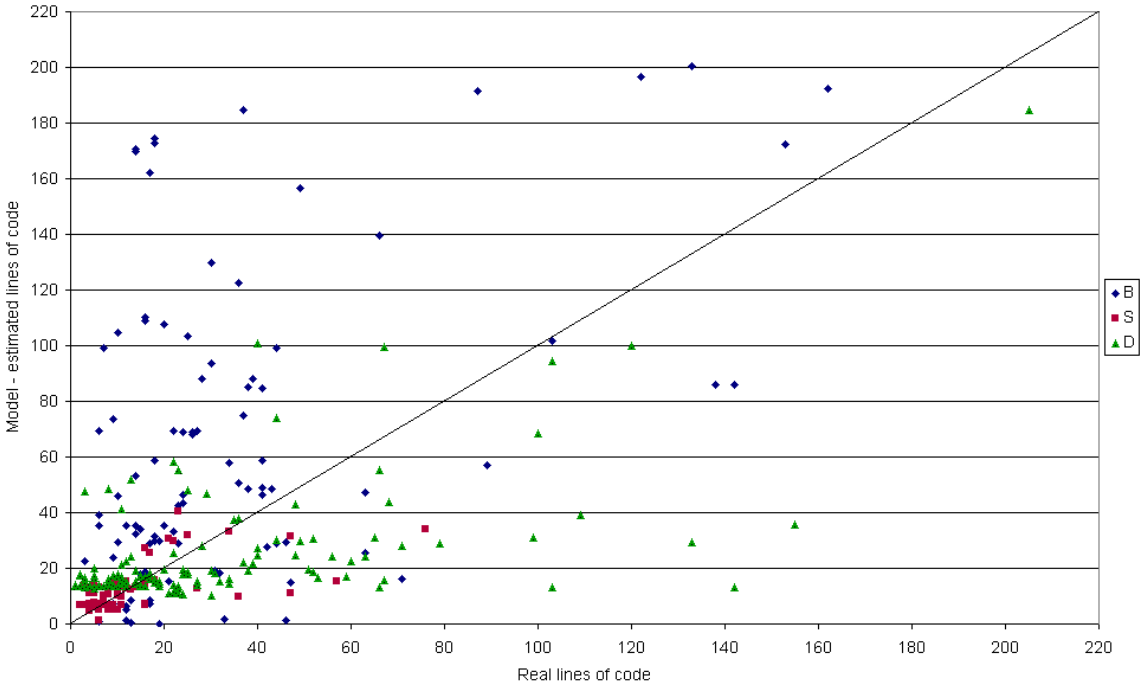
	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient (L,$\hat{L}$)	0.9332	0.6310	0.6204

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_{nip}	-1.3241	-0.1284	-0.1396
k_{nop}	0.9241	0.6074	0.7533
k_{niop}	-6.7543	6.9235	3.6139
k_{nxp}	3.1749	-2.5243	-0.4059
k_{ns}	2.7819	0.0989	0.1789
k_0	-6.5737	6.3198	13.3001

Remarks on error distribution - Models AM4b, AM4s, AM4d: All the errors fall between -270 and +160 lines of code. In 98% of the cases the error falls between -125 and +150 lines of code; in 96% of the cases the error falls between -70 and +90 lines of code; in 85% of the

Figure 13.37: Model AM4: Real vs. estimated lines of code.



cases the error falls between -25 and +60 lines of code; in 50% of the cases the error falls between -20 and +15 lines of code.

Remarks on error distribution - behavioral architectures only: In 99% of the cases the error falls between -270 and +160 lines of code; in 94% of the cases the error falls between -130 and +95 lines of code; in 85% of the cases the error falls between -30 and +95 lines of code; in 50% of the cases the error falls between -20 and +15 lines of code.

Remarks on error distribution - structural architectures only: In 98% of the cases the error falls between -43 and +15 lines of code; in 93% of the cases the error falls between -17 and +12 lines of code; in 85% of the cases the error falls between -5 and +9 lines of code; in 50% of the cases the error falls between -3 and +5 lines of code.

Remarks on error distribution - data-flow architectures only: In 99% of the cases the error falls between -130 and +62 lines of code; in 97% of the cases the error falls between -90 and +45 lines of code; in 85% of the cases the error falls between -25 and +17 lines of code; in 50% of the cases the error falls between -10 and +15 lines of code.

Figure 13.38: Models AM4b, AM4s, AM4d: Real vs. estimated lines of code.

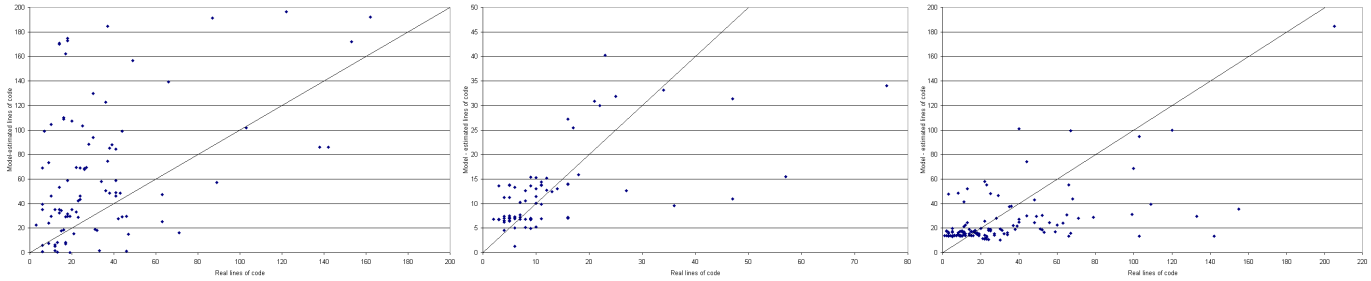


Figure 13.39: Model AM4: Error density distribution.

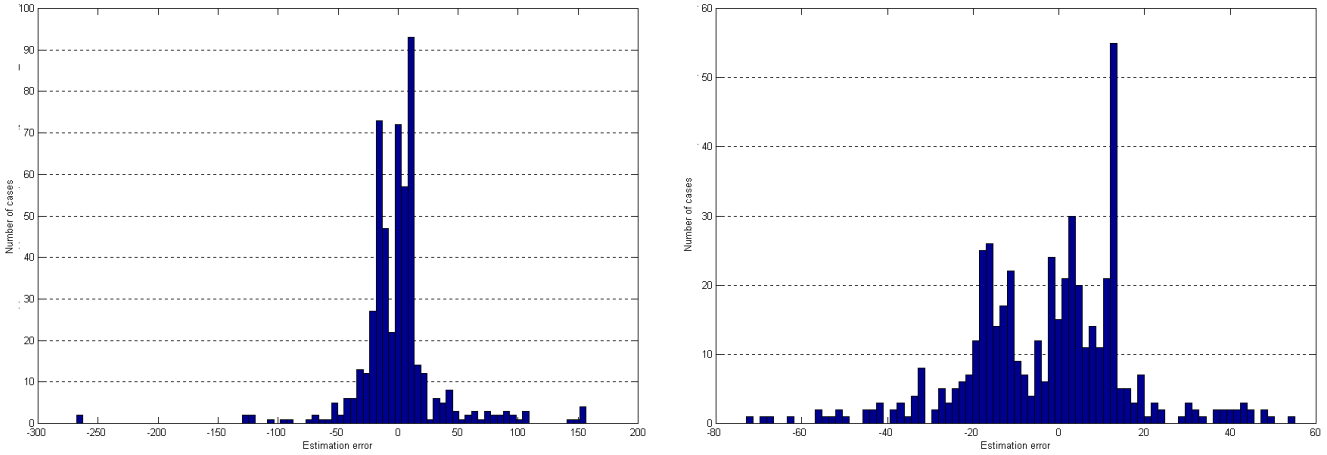


Figure 13.40: Model AM4: Error cumulative distribution.

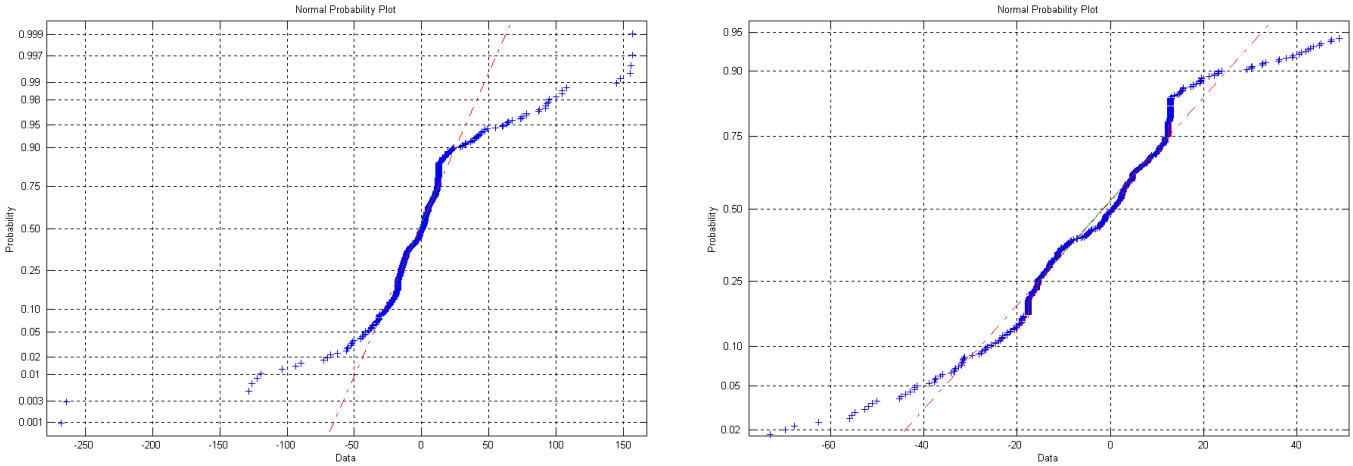
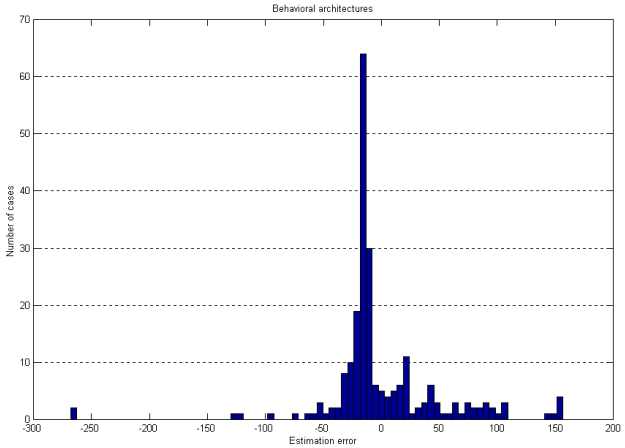
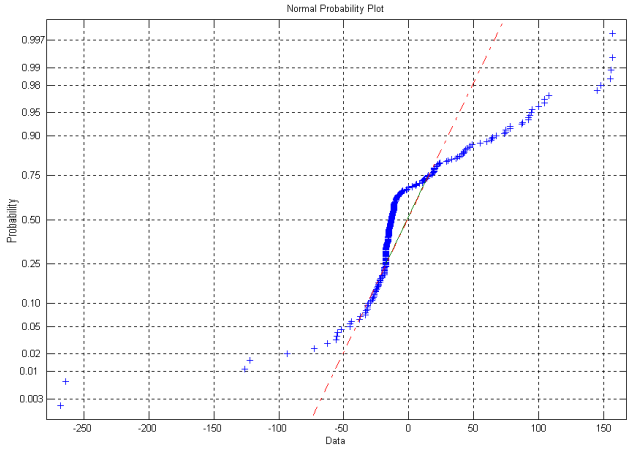
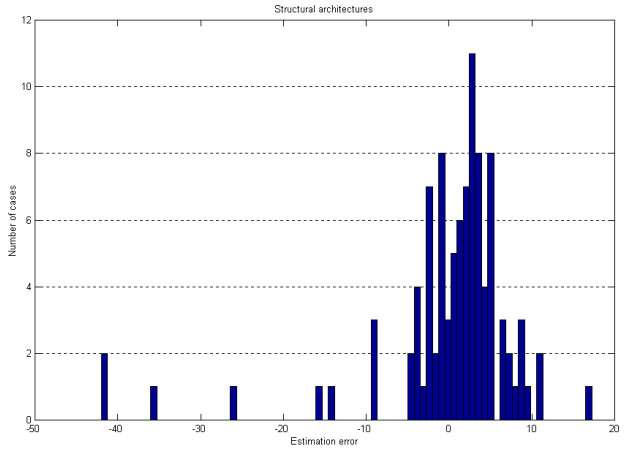
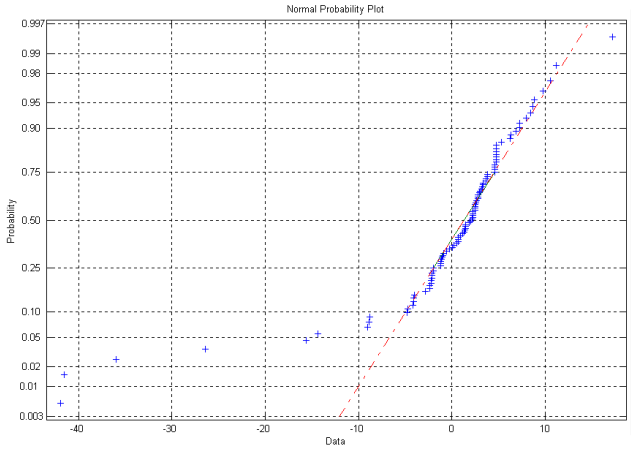


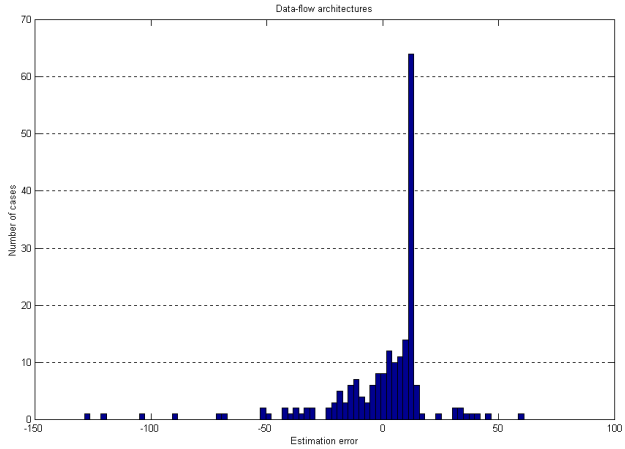
Figure 13.41: Models AM4b, AM4s, AM4d: Error density and cumulative distributions.

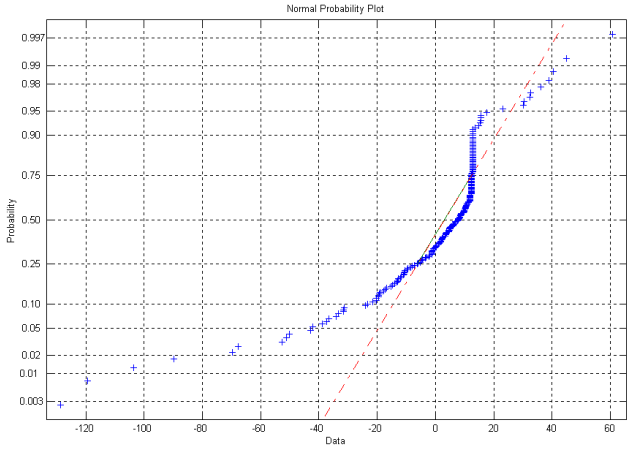












13.2.10 Model AM4H

Model:

$$\hat{L} = k_{hip} \cdot h_{ip} + k_{hop} \cdot h_{op} + k_{hiop} \cdot h_{iop} + k_{hxp} \cdot h_{xp} + k_{hs} \cdot h_s + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	30.670	30.670	0.000
Variance	19931.806	982.466	18949.341
Standard deviation	141.180	31.344	137.657
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	11.173	11.173	0.000
Variance	136.887	54.937	81.950
Standard deviation	11.700	7.412	9.053
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	21.693	21.693	0.000
Variance	941.826	280.762	661.064
Standard deviation	30.689	16.756	25.711

Correlation between estimated and real values:

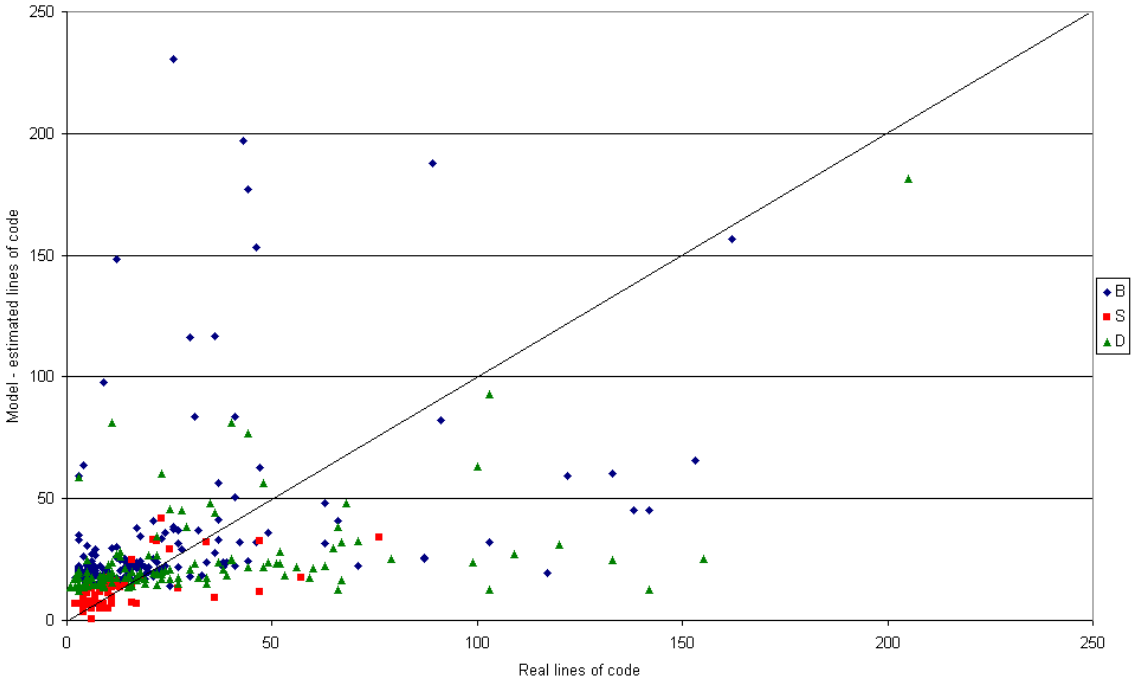
	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient (L,$\hat{L}$)	0.2220	0.6335	0.5460

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_{hip}	-0.6618	-0.1428	0.3034
k_{hop}	1.0274	0.5681	0.9531
k_{hiop}	5.8768	6.6407	3.9891
k_{hxp}	-3.9685	-3.8267	-0.4438
k_{hs}	0.1112	0.1426	0.0126
k_0	20.2192	6.3474	12.3116

Remarks on error distribution - Models AM4Hb, AM4Hs, AM4Hd: Except for 1 case, all the errors fall between -250 and +155 lines of code. In 97% of the cases the error falls between -90 and +155 lines of code; in 96% of the cases the error falls between -75 and +40

Figure 13.42: Model AM4H: Real vs. estimated lines of code.



lines of code; in 85% of the cases the error falls between -20 and +25 lines of code; in 50% of the cases the error falls between +0 and +20 lines of code.

Remarks on error distribution - behavioral architectures only: In 97% of the cases the error falls between -100 and +110 lines of code; in 93% of the cases the error falls between -60 and +110 lines of code; in 85% of the cases the error falls between -20 and +55 lines of code; in 50% of the cases the error falls between +5 and +20 lines of code.

Remarks on error distribution - structural architectures only: In 99% of the cases the error falls between -45 and +20 lines of code; in 96% of the cases the error falls between -40 and +12 lines of code; in 85% of the cases the error falls between -9 and +9 lines of code; in 50% of the cases the error falls between -3 and +5 lines of code.

Remarks on error distribution - data-flow architectures only: In 99% of the cases the error falls between -130 and +70 lines of code; in 96% of the cases the error falls between -90 and +35 lines of code; in 85% of the cases the error falls between -30 and +15 lines of code; in 50% of the cases the error falls between -5 and +12 lines of code.

Figure 13.43: Models AM4Hb, AM4Hs, AM4Hd: Real vs. estimated lines of code.

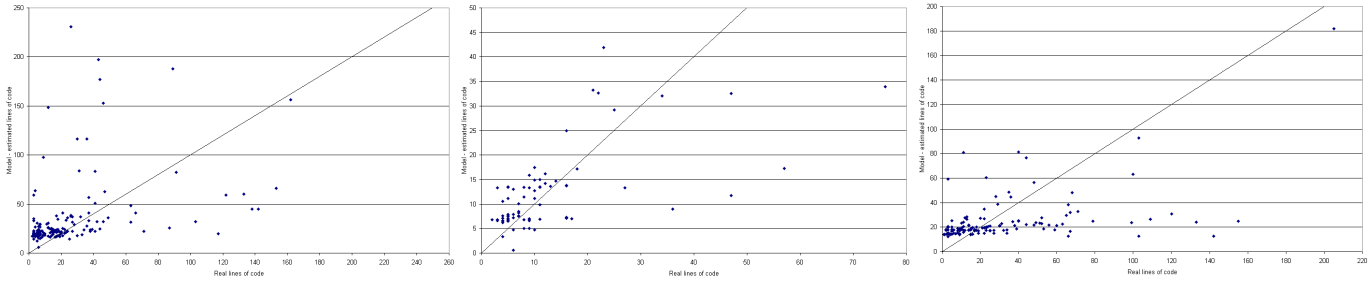


Figure 13.44: Model AM4H: Error density distribution.

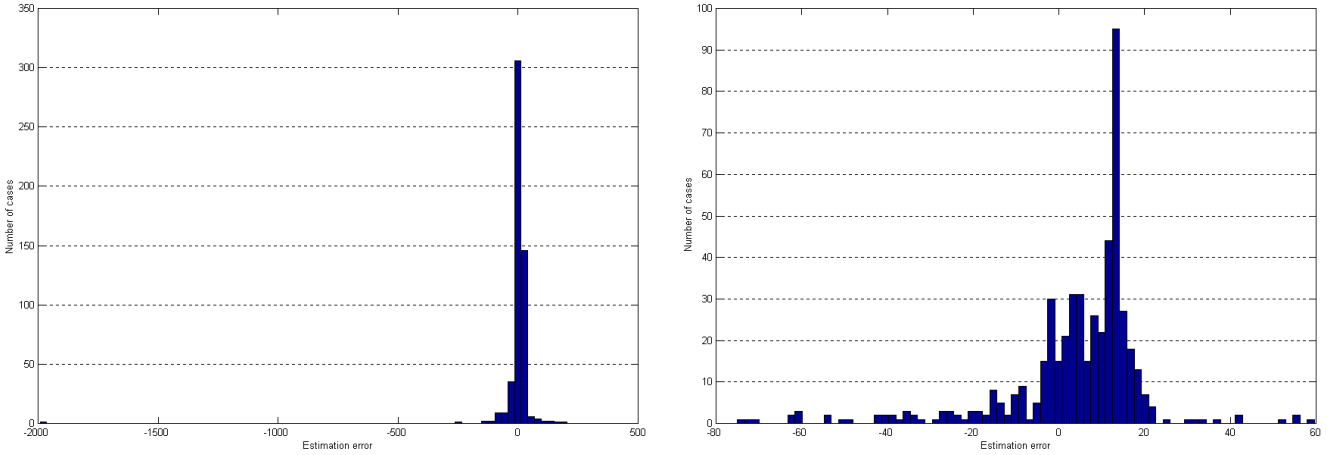


Figure 13.45: Model AM4H: Error cumulative distribution.

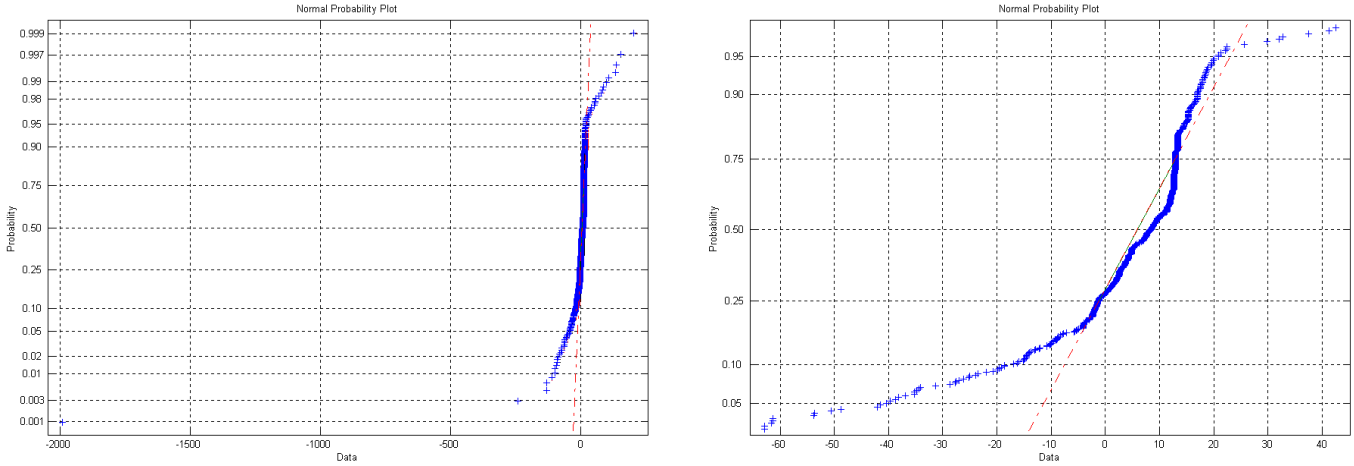
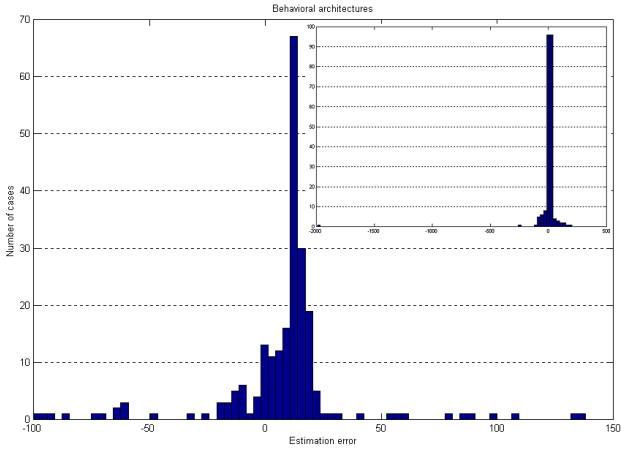
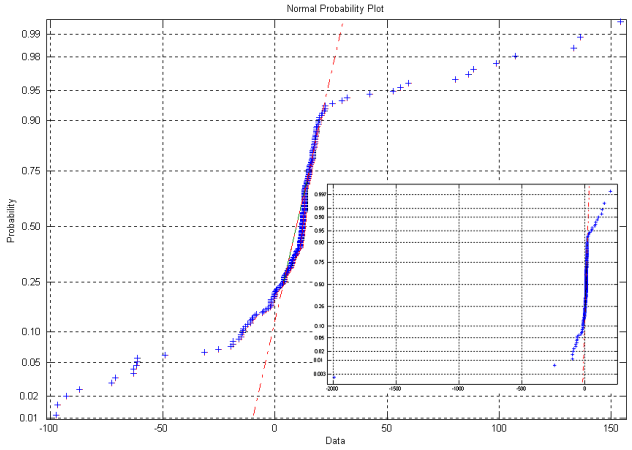
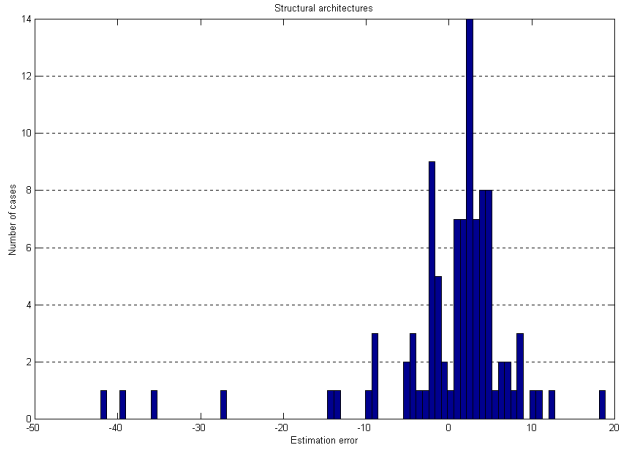
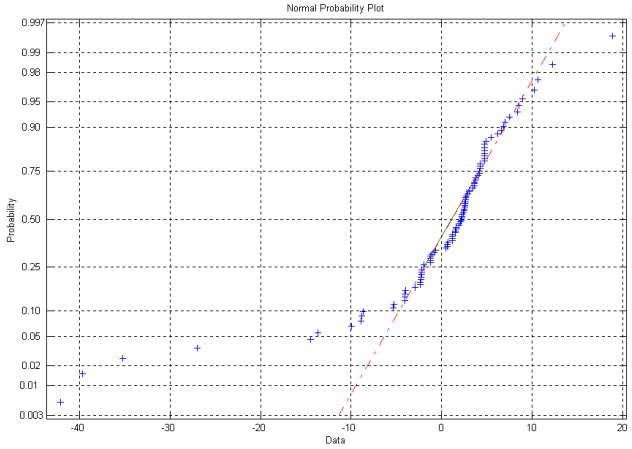


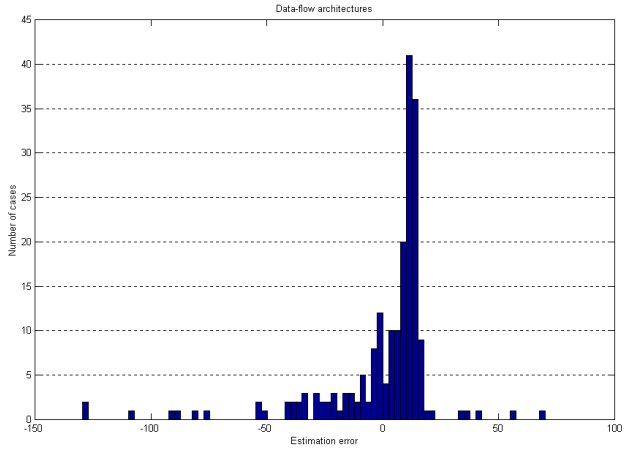
Figure 13.46: Models AM4Hb, AM4Hs, AM4Hd: Error density and cumulative distributions.

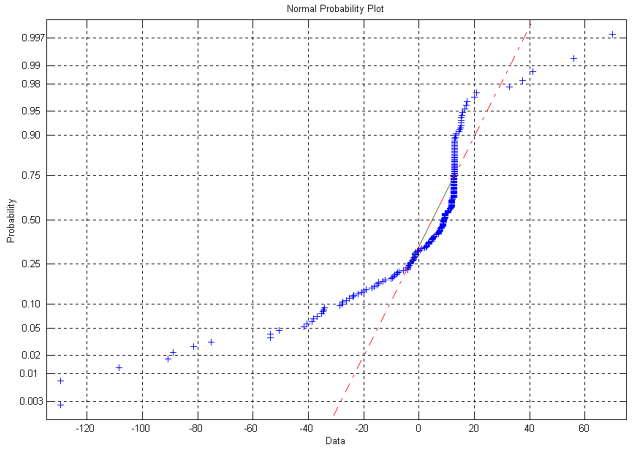












13.2.11 Model AM5

Model:

$$\hat{L} = k_{ns} \cdot n_s + k_{nci} \cdot n_{ci} + k_0$$

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	23.976	23.976	0.000
Variance	7463.098	212.604	7251.264
Standard deviation	86.389	14.581	85.154

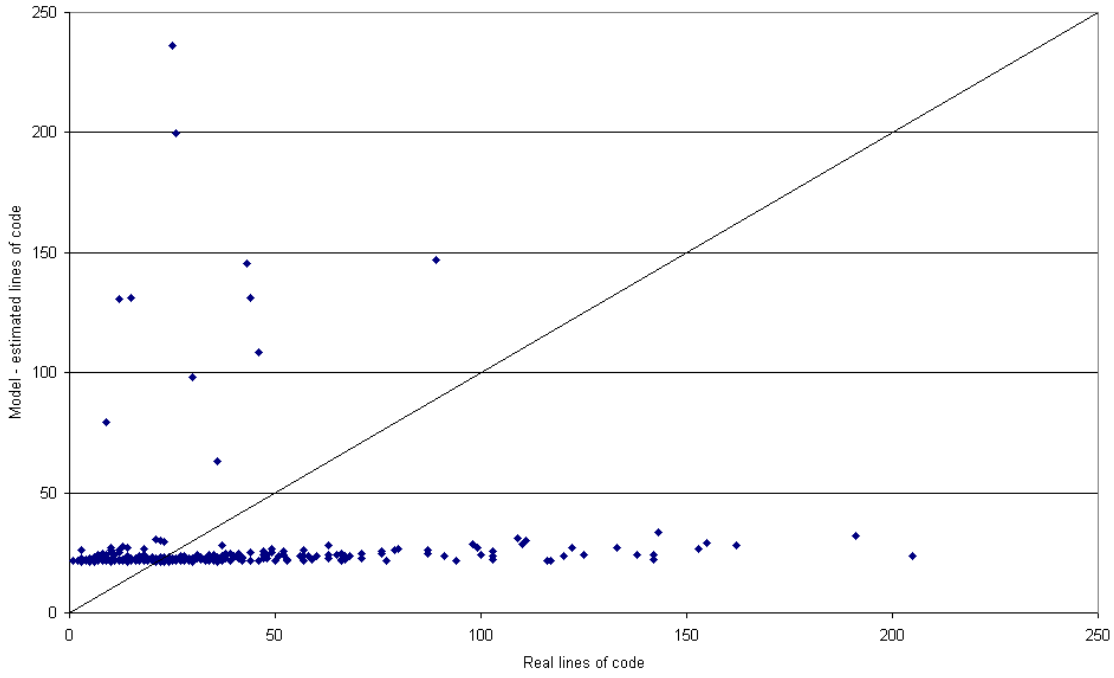
Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.1685

Identified model coefficients:

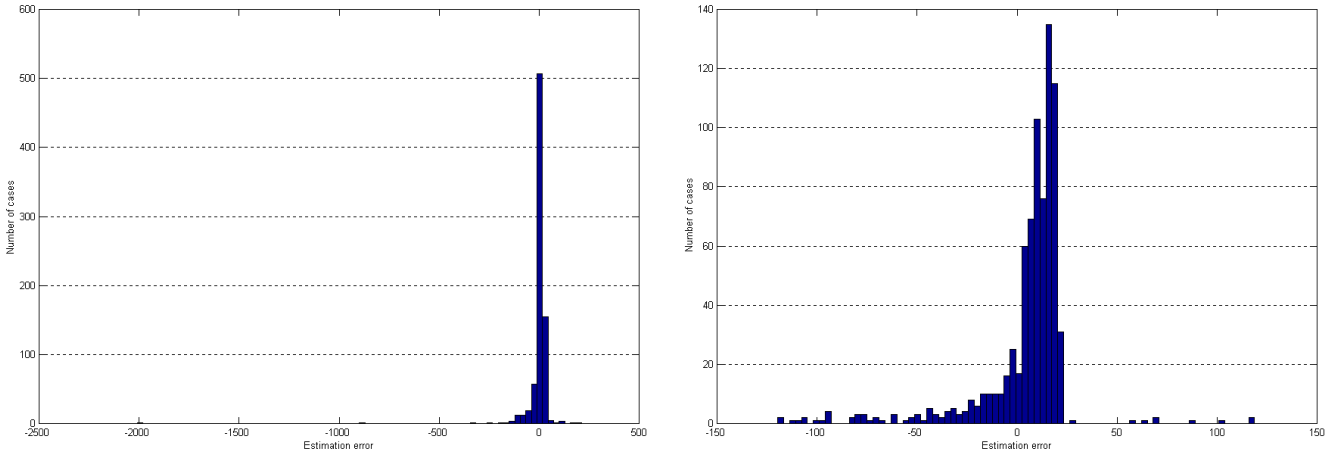
Coefficient	Value
k_{ns}	0.0941
k_{nci}	0.0387
k_0	21.4304

Figure 13.47: Model AM5: Real vs. estimated lines of code.



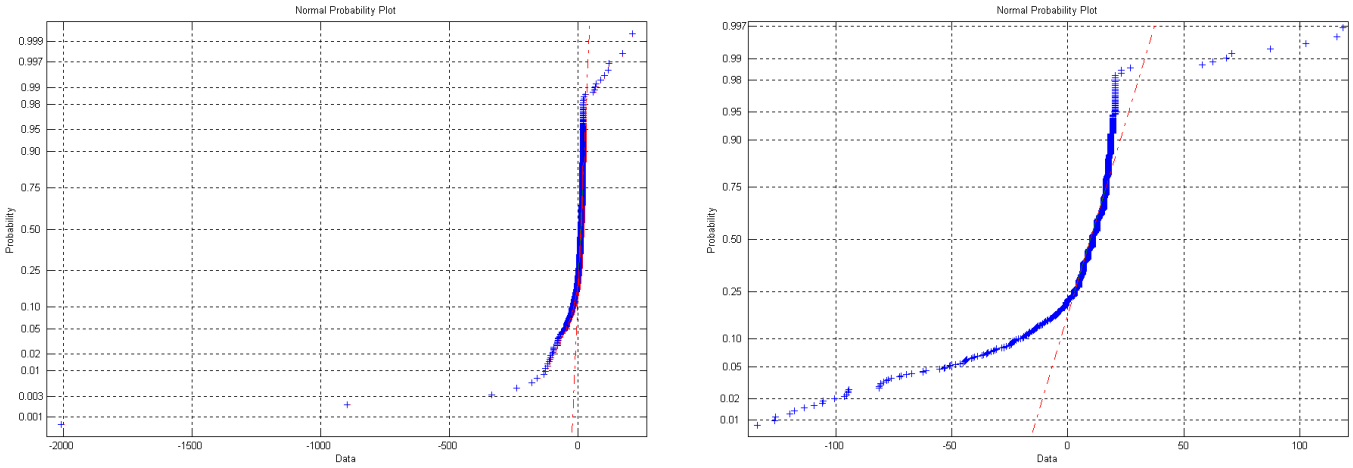
Remarks on error distribution: all the errors except for two cases fall between -350 and +250 lines of code. In 98% of the cases the error falls between -150 and +65 lines of code; in

Figure 13.48: Model AM5: Error density distribution.



93% of the cases the error falls between -50 and +25 lines of code; in 85% of the cases the error falls between -25 and +20 lines of code; in 50% of the cases the error falls between +5 and +15 lines of code.

Figure 13.49: Model AM5: Error cumulative distribution.



13.2.12 Model AM5H

Model:

$$\hat{L} = k_{hs} \cdot h_s + k_{nci} \cdot n_{ci} + k_0$$

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	23.976	23.976	0.000
Variance	7463.098	310.307	7152.759
Standard deviation	86.389	17.616	84.574

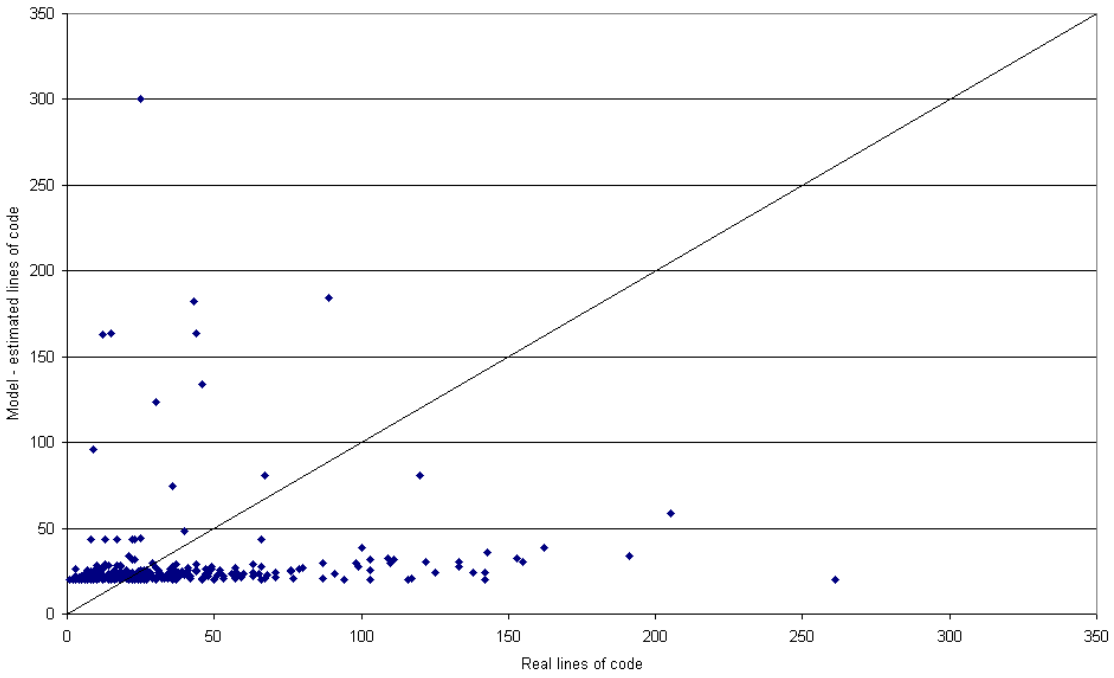
Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.2039

Identified model coefficients:

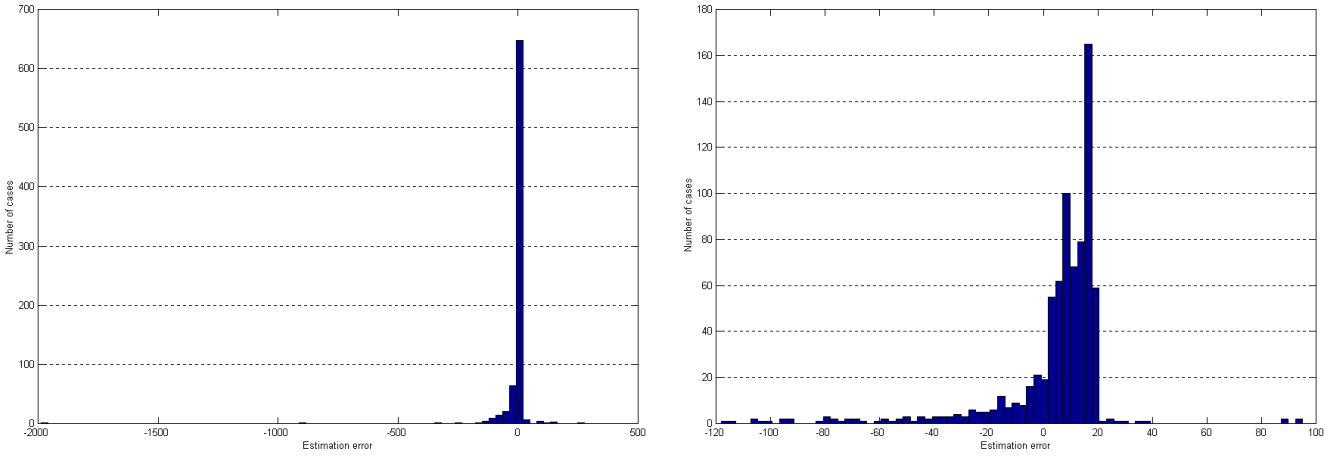
Coefficient	Value
k_{hs}	0.1228
k_{nci}	0.0136
k_0	20.3114

Figure 13.50: Model AM5H: Real vs. estimated lines of code.



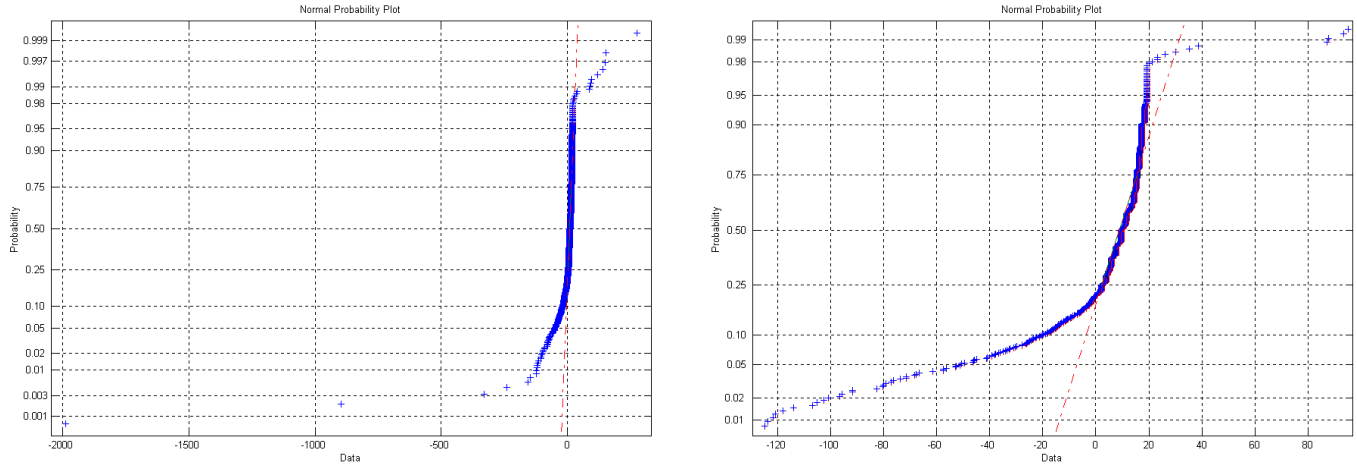
Remarks on error distribution: all the errors except for two cases fall between -350 and +250 lines of code. In 98% of the cases the error falls between -125 and +90 lines of code; in

Figure 13.51: Model AM5H: Error density distribution.



96% of the cases the error falls between -100 and +20 lines of code; in 85% of the cases the error falls between -20 and +20 lines of code; in 50% of the cases the error falls between +5 and +15 lines of code.

Figure 13.52: Model AM5H: Error cumulative distribution.



13.2.13 Model AM6

Model:

$$\hat{L} = k_{nip} \cdot n_{ip} + k_{nop} \cdot n_{op} + k_{niop} \cdot n_{iop} + k_{nxp} \cdot n_{xp} + k_{ns} \cdot n_s + k_{nci} \cdot n_{ci} + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	30.670	30.670	0.000
Variance	19931.806	1004.460	18927.347
Standard deviation	141.180	31.693	137.577
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	11.173	11.173	0.000
Variance	136.887	69.768	67.119
Standard deviation	11.700	8.353	8.193
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	21.693	21.693	0.000
Variance	941.826	229.091	712.735
Standard deviation	30.689	15.136	26.697

Correlation between estimated and real values:

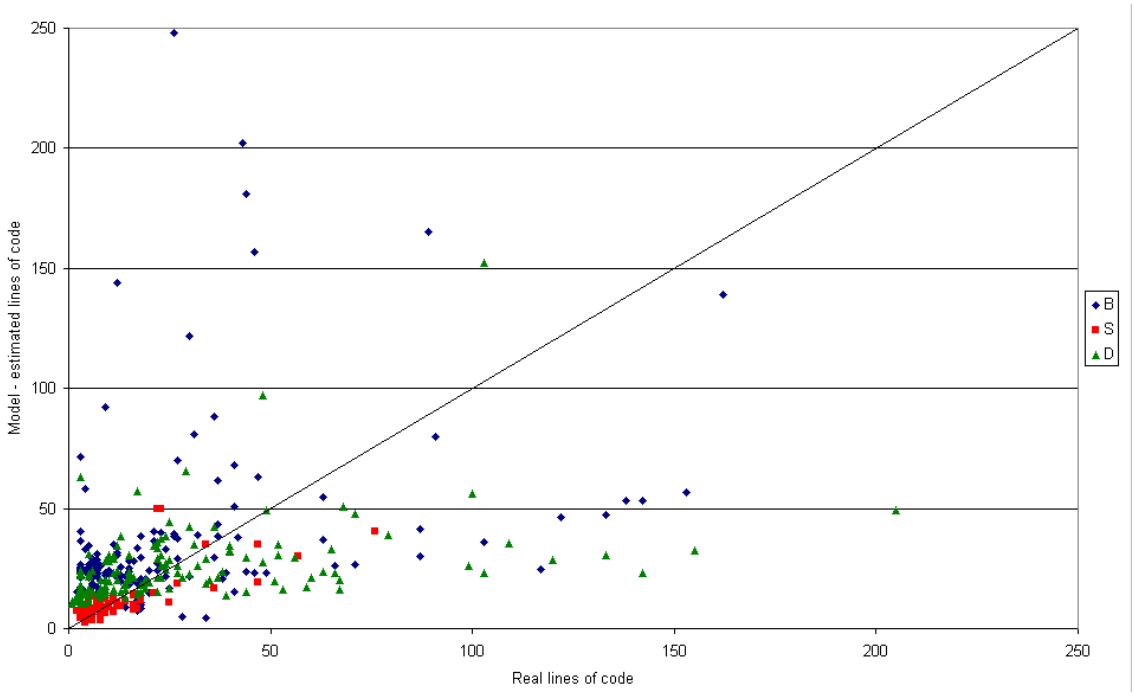
	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient (L,$\hat{L}$)	0.2245	0.7139	0.4932

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_{nip}	-0.9008	0.0613	1.2847
k_{nop}	1.2366	0.5187	1.0088
k_{niop}	5.5106	5.6032	2.0977
k_{nxp}	-5.3707	0.1466	13.2308
k_{ns}	0.1140	0.3876	0.0125
k_{nci}	-6.0553	-0.3753	1.9664
k_0	25.3735	7.0263	7.9990

Remarks on error distribution - Models AM6b, AM6s, AM6d: All the errors except for one case fall between -250 and +200 lines of code. In 98% of the cases the error falls between

Figure 13.53: Model AM6: Real vs. estimated lines of code.



-105 and +120 lines of code; in 96% of the cases the error falls between -80 and +55 lines of code; in 85% of the cases the error falls between -25 and +25 lines of code; in 50% of the cases the error falls between -5 and +10 lines of code.

Remarks on error distribution - behavioral architectures only: In 96% of the cases the error falls between -105 and +120 lines of code; in 93% of the cases the error falls between -90 and +50 lines of code; in 80% of the cases the error falls between -25 and +25 lines of code; in 50% of the cases the error falls between 0 and +20 lines of code.

Remarks on error distribution - structural architectures only: In 97% of the cases the error falls between -35 and +27 lines of code; in 93% of the cases the error falls between -28 and +7 lines of code; in 80% of the cases the error falls between -8 and +6 lines of code; in 50% of the cases the error falls between -3 and +4 lines of code.

Remarks on error distribution - data-flow architectures only: In 98% of the cases the error falls between -120 and +50 lines of code; in 96% of the cases the error falls between -105 and +40 lines of code; in 85% of the cases the error falls between -35 and +20 lines of code; in 50% of the cases the error falls between -5 and +10 lines of code.

Figure 13.54: Models AM6b, AM6s, AM6d: Real vs. estimated lines of code.

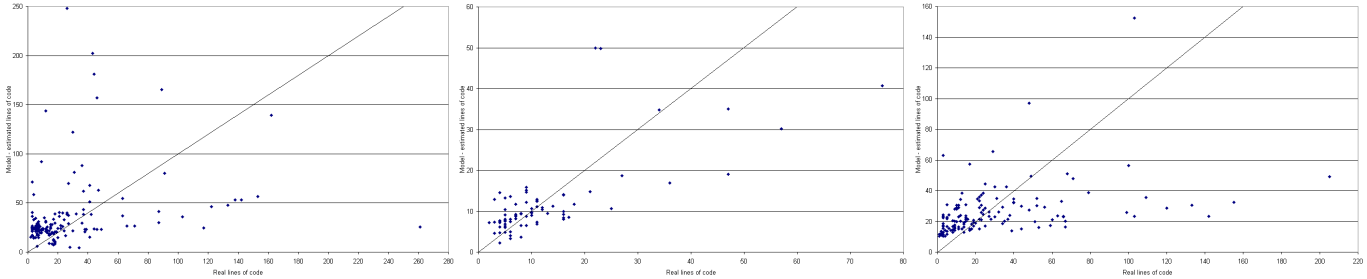


Figure 13.55: Model AM6: Error density distribution.

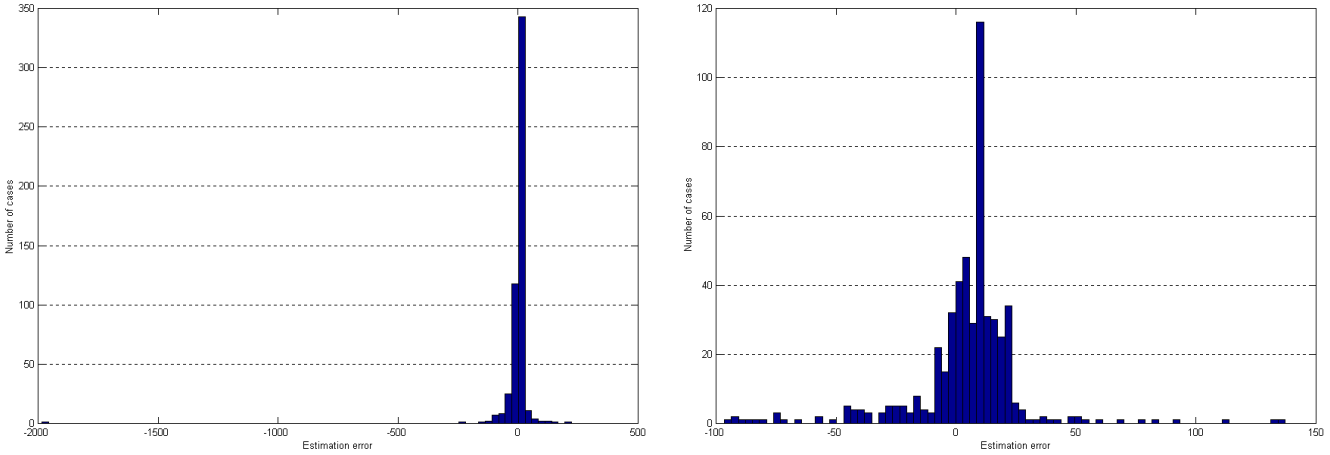


Figure 13.56: Model AM6: Error cumulative distribution.

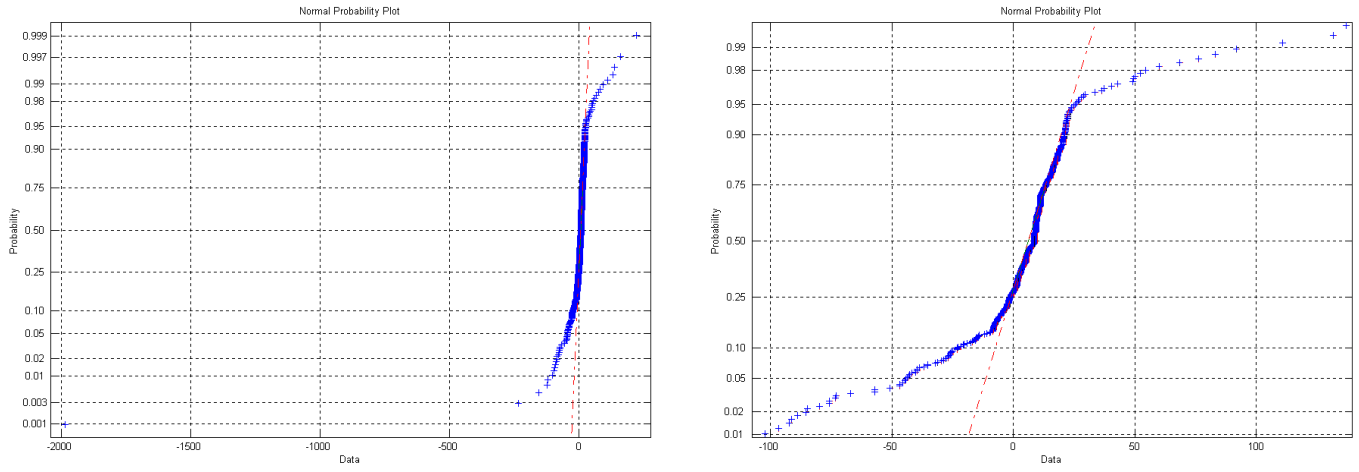
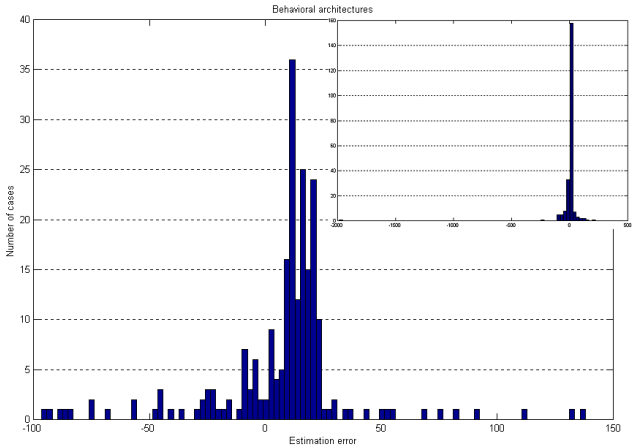
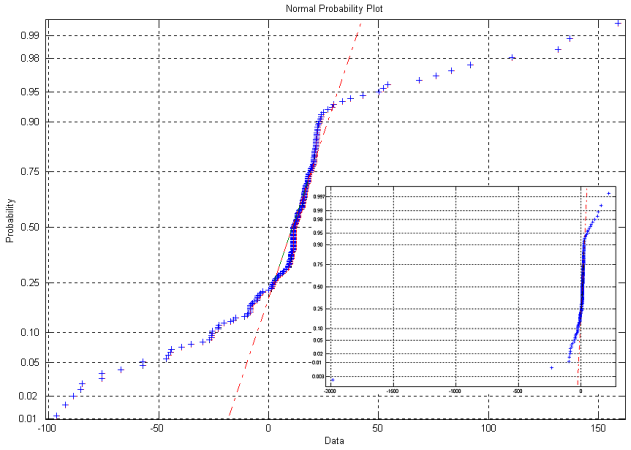
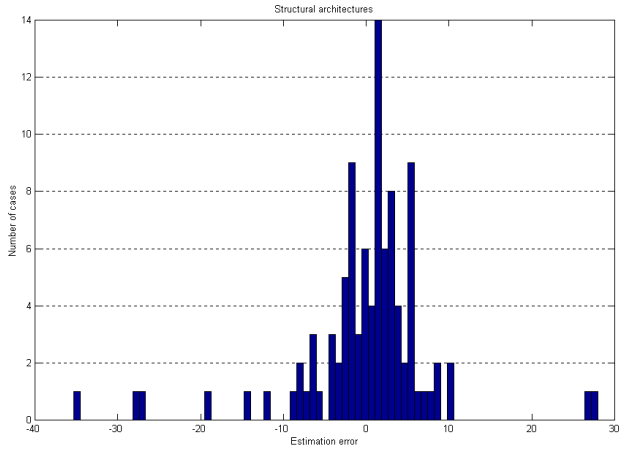
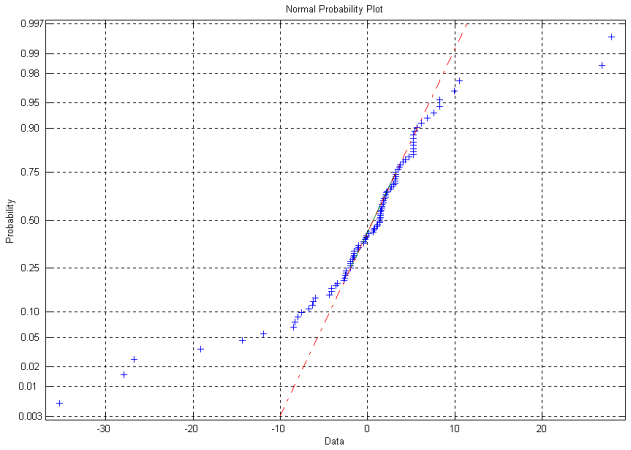


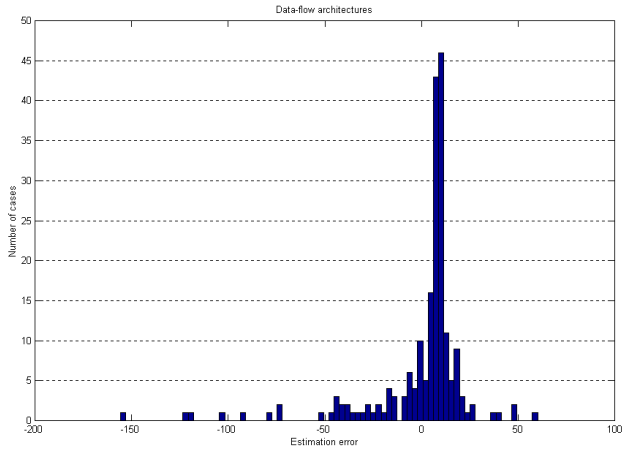
Figure 13.57: Models AM6b, AM6s, AM6d: Error density and cumulative distributions.

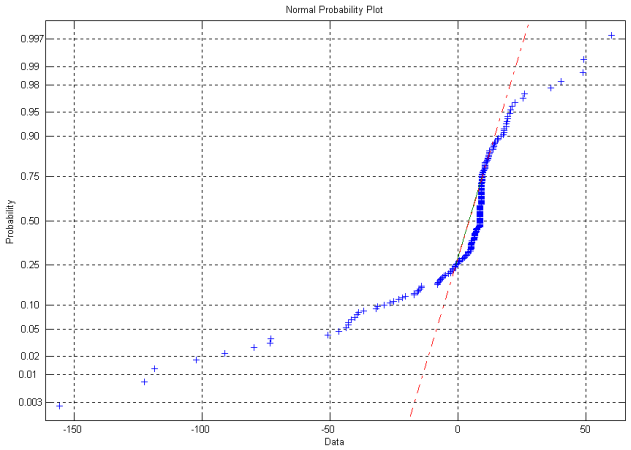












13.2.14 Model AM6H

Model:

$$\hat{L} = k_{hip} \cdot h_{ip} + k_{hop} \cdot h_{op} + k_{hiop} \cdot h_{iop} + k_{hxp} \cdot h_{xp} + k_{hs} \cdot h_s + k_{nci} \cdot n_{ci} + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	30.670	30.670	0.000
Variance	19931.806	1035.256	18896.550
Standard deviation	141.180	32.175	137.465
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	11.173	11.173	0.000
Variance	136.887	71.085	65.802
Standard deviation	11.700	8.431	8.112
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	21.693	21.693	0.000
Variance	941.826	316.346	625.480
Standard deviation	30.689	17.786	25.010

Correlation between estimated and real values:

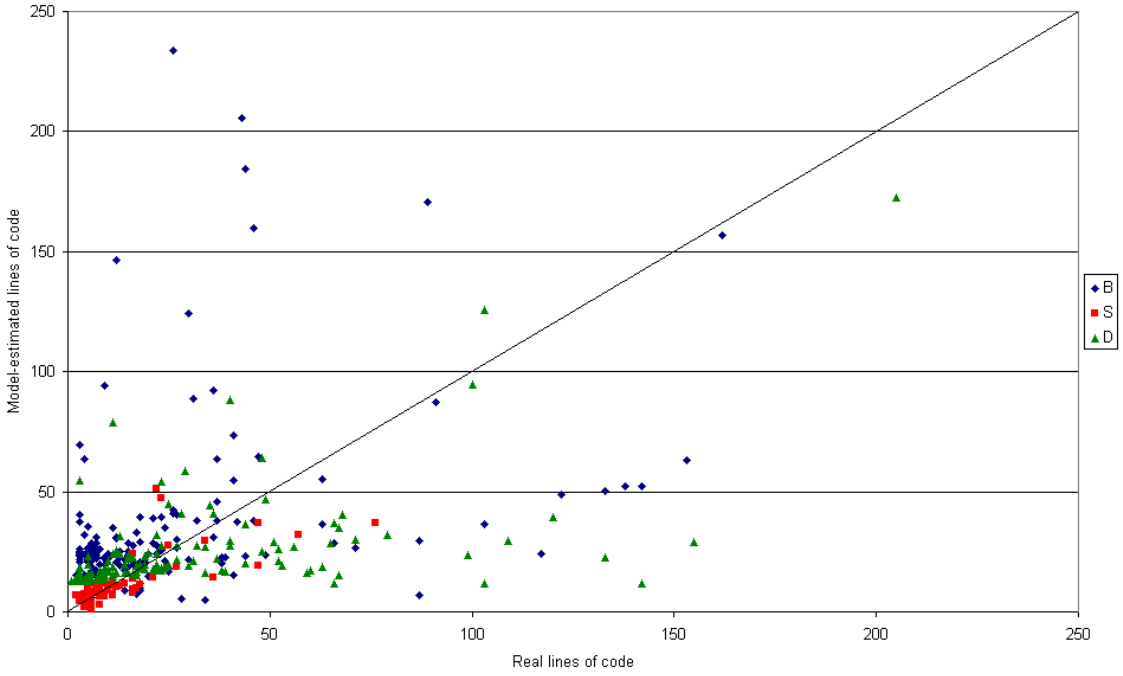
	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient (L,$\hat{L}$)	0.2279	0.7206	0.5796

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_{hip}	-0.8466	0.0240	0.2597
k_{hop}	1.1971	0.3795	0.8804
k_{hiop}	6.1510	5.3525	1.2256
k_{hxp}	-5.2613	-1.0433	1.3929
k_{hs}	0.1164	0.4479	0.0127
k_{nci}	-5.8549	-0.3784	1.9324
k_0	24.8214	6.9867	11.5704

Remarks on error distribution - Models AM6Hb, AM6Hs, AM6Hd: All the errors except for one case fall between -250 and +200 lines of code. In 98% of the cases the error falls

Figure 13.58: Model AM6H: Real vs. estimated lines of code.



between -90 and +90 lines of code; in 96% of the cases the error falls between -80 and +60 lines of code; in 85% of the cases the error falls between -25 and +25 lines of code; in 50% of the cases the error falls between -5 and +15 lines of code.

Remarks on error distribution - behavioral architectures only: In 96% of the cases the error falls between -80 and +120 lines of code; in 93% of the cases the error falls between -90 and +50 lines of code; in 85% of the cases the error falls between -25 and +55 lines of code; in 50% of the cases the error falls between 0 and +20 lines of code.

Remarks on error distribution - structural architectures only: In 96% of the cases the error falls between -27 and +25 lines of code; in 90% of the cases the error falls between -10 and +8 lines of code; in 80% of the cases the error falls between -7 and +6 lines of code; in 50% of the cases the error falls between -2 and +3 lines of code.

Remarks on error distribution - data-flow architectures only: In 97% of the cases the error falls between -130 and +30 lines of code; in 93% of the cases the error falls between -45 and +30 lines of code; in 85% of the cases the error falls between -30 and +15 lines of code; in 50% of the cases the error falls between -5 and +12 lines of code.

Figure 13.59: Models AM6Hb, AM6Hs, AM6Hd: Real vs. estimated lines of code.

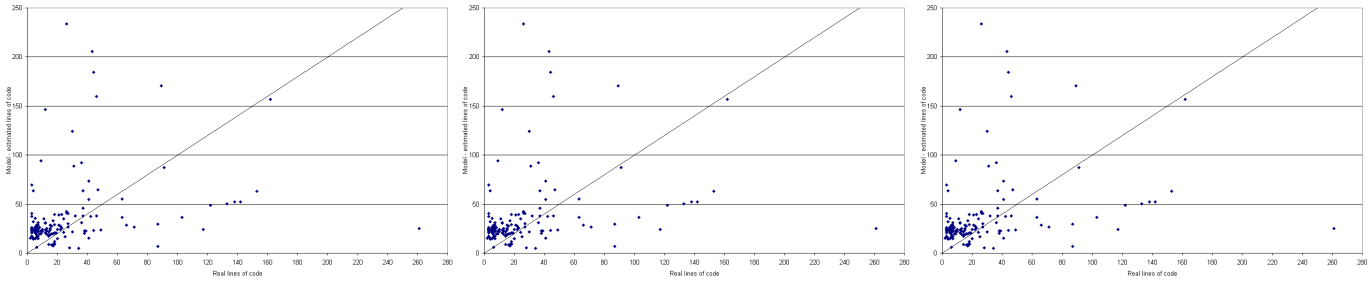


Figure 13.60: Model AM6H: Error density distribution.

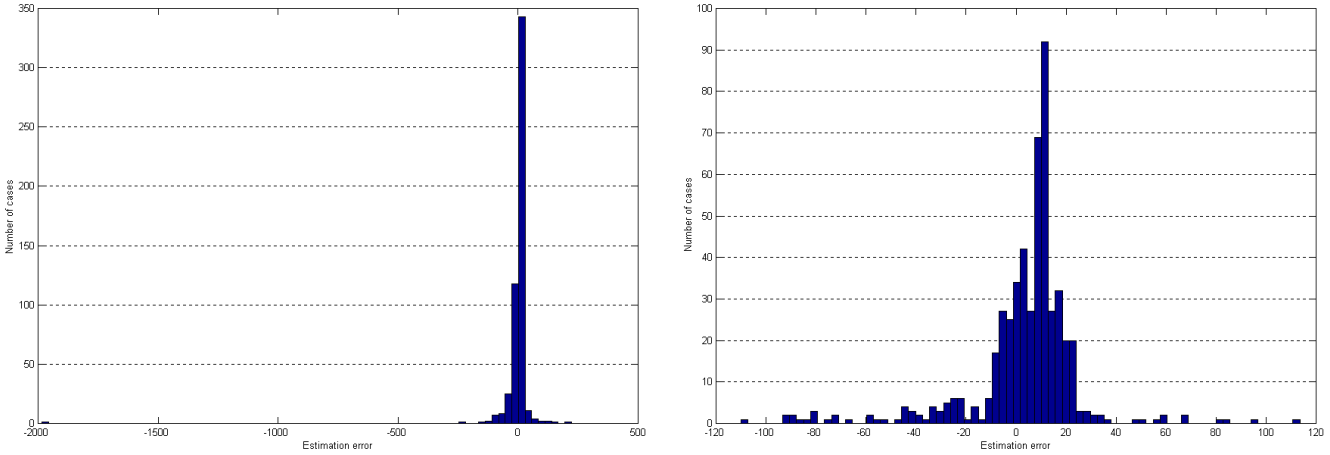


Figure 13.61: Model AM6H: Error cumulative distribution.

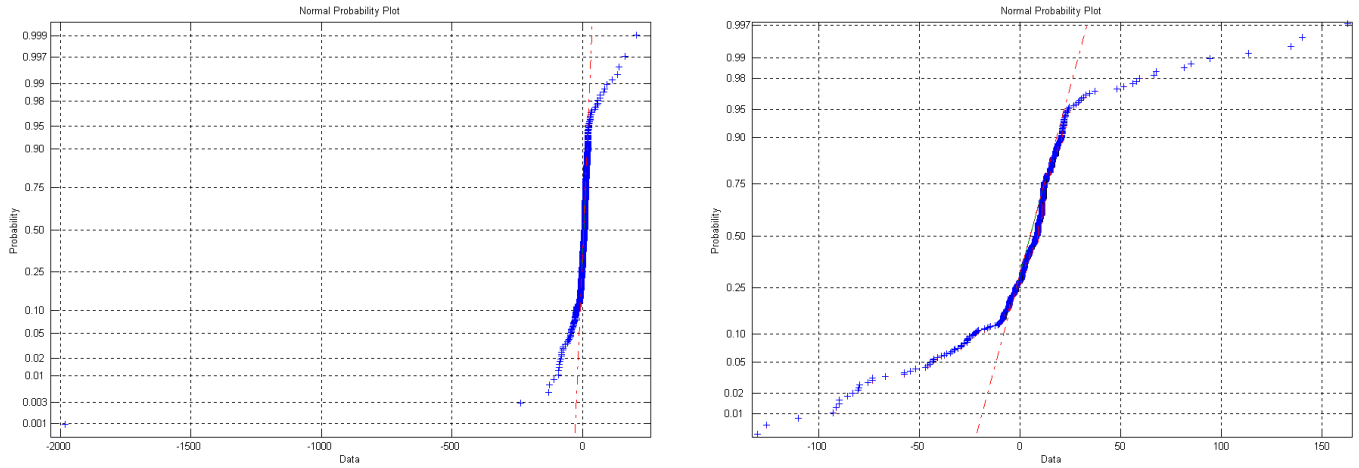
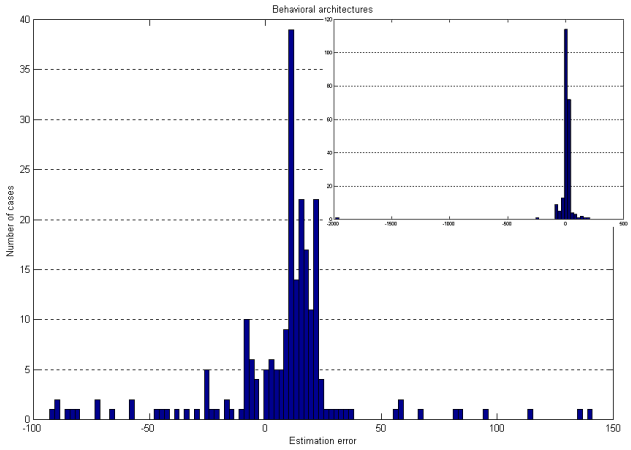
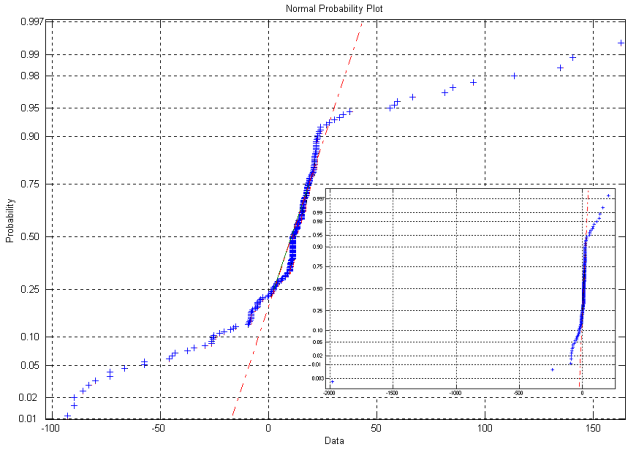
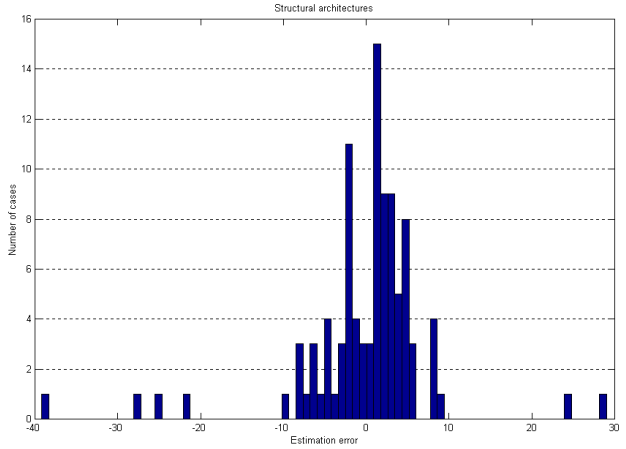
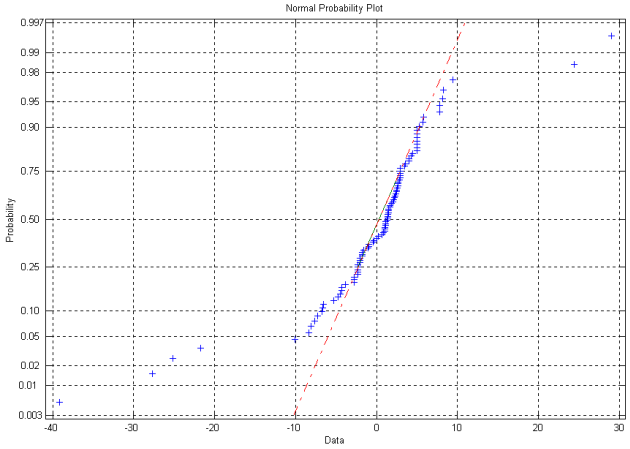


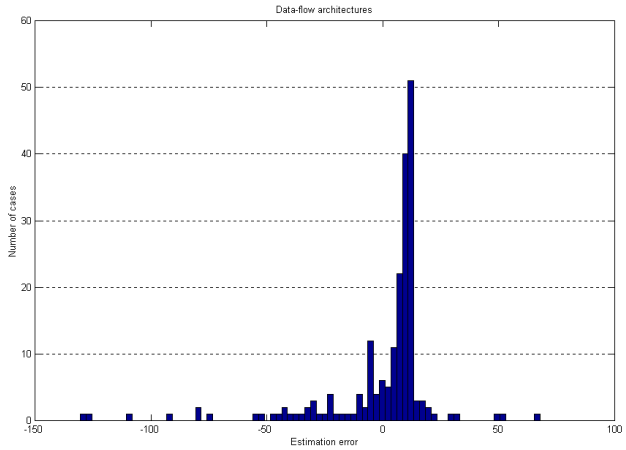
Figure 13.62: Models AM6Hb, AM6Hs, AM6Hd: Error density and cumulative distributions.

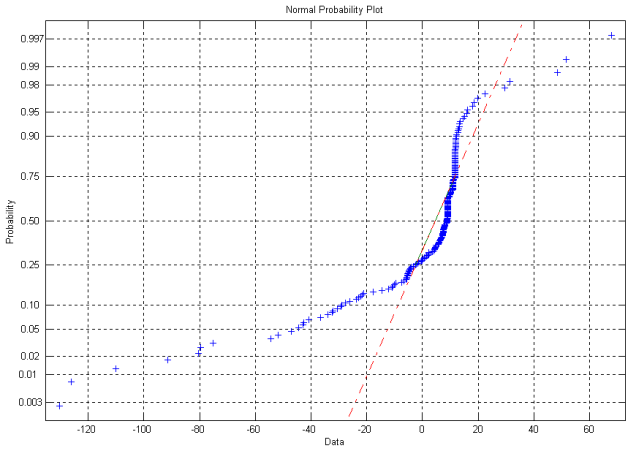












13.2.15 Conclusions

In complete honesty, we must admit that we were not able to find any models able to obtain estimates with a high coefficient of correlation between actual and estimated length. Nevertheless, thanks to results of preliminary correlation studies, we can hold the more than reasonable belief that such models do not exist.

Please note that by saying that we failed to reach good coefficient of correlation, we do not mean that our models are not suitable to be used for real estimation tasks in our methodology: in fact, most of them, though having poor correlations, exhibit low estimation error variance (mainly due to low original population variance), which make them still useful.

13.3 Component declaration models

Component declarations have been considered so far only as an architecture body constituent, nevertheless they can appear independently from any architectures, for example inside a package (packages are not considered in this thesis for reasons of simplicity; their content is conceptually ascribed to the project object, without any distinction based upon the package origin).

Anyway it must be said that, wherever they appear, component declarations show indifferently the same characteristics under all the considered circumstances. Thus we decided to consider all component declarations as instances of the same syntax object, disregarding the context in which they are found.

13.3.1 Variables and correlation

Since component declarations conceptually carry out, on a smaller scale, the same roles that entity declarations have (that is, describing how a thing *talks* to the rest of the world or, more formally, its interface), we decided to exploit the expertise developed for entites and immediately focus on two models, the first one using the port count only, and the second one using port count and generic constant count. We do not even take into account homogeneity data and port counts classified by port mode since they proved to be useless as far as entity/component declaration length estimation is concerned.

Correlation coefficients of the chosen variables are encouraging.

Variable	Average value	Variance	Standard deviation	Correlation coefficient
n_p	9.4905	154.6266	12.4349	0.7233
n_g	0.5672	15.1437	3.8915	0.5239
$\hat{L}$	13.5399	198.6504	14.0943	(1.0000)

13.3.2 Model CDM1

Model CDM1 starts from the hypothesis (proven fundamentally true) that component declarations are just a couple of lines more than a port list, usually written one per line. It seems therefore natural to create a simple linear model, which tries to estimate the declaration length disregarding port homogeneity, mode or type.

Model:

$$L = k_{np} \cdot n_p + k_0$$

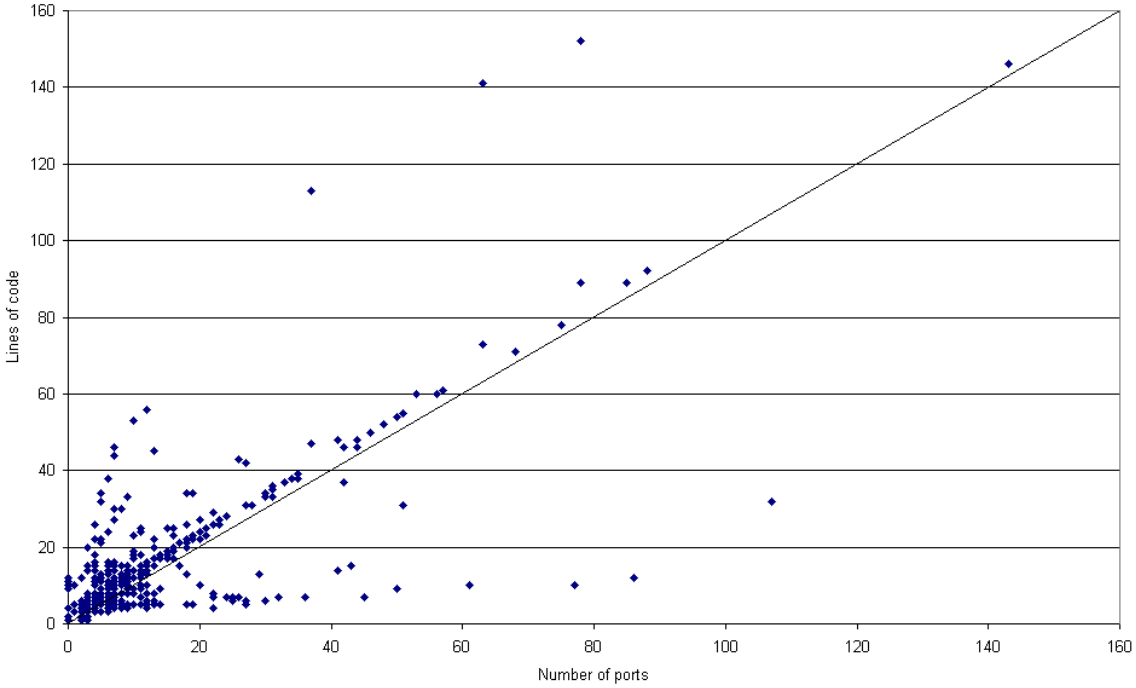
Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	13.540	13.540	0.000
Variance	198.650	103.912	94.738
Standard deviation	14.094	10.194	9.733

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.7233

Figure 13.63: Relationship between number of ports and lines of code.



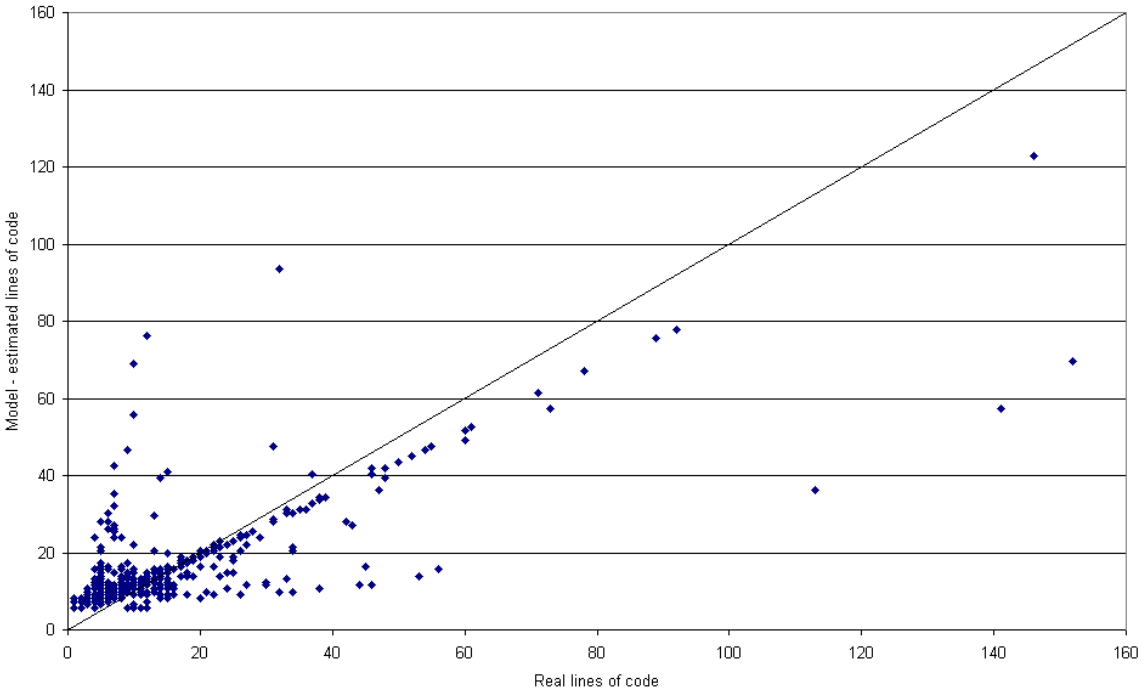
Identified model coefficients:

Coefficient	Value
k_{np}	0.8198
k_0	5.7599

By examining the model results, one can see a quite widespread overestimation phenomenon, due to coding style (the programmer wrote multiple port declarations per line, exactly as we noticed with entities), and less evident underestimation phenomenon, concentrated in a few samples diverging a lot from the desired bisector. We individually analyzed those component declarations, and basically found what we already were suspecting: in these cases, influence of generic constant declarations are not negligible. The code below is the declaration of component FPURTGeneric, from `fpurt_lib.vhd` in project "ERC32". Note how pervasive the presence of generic constants can be.

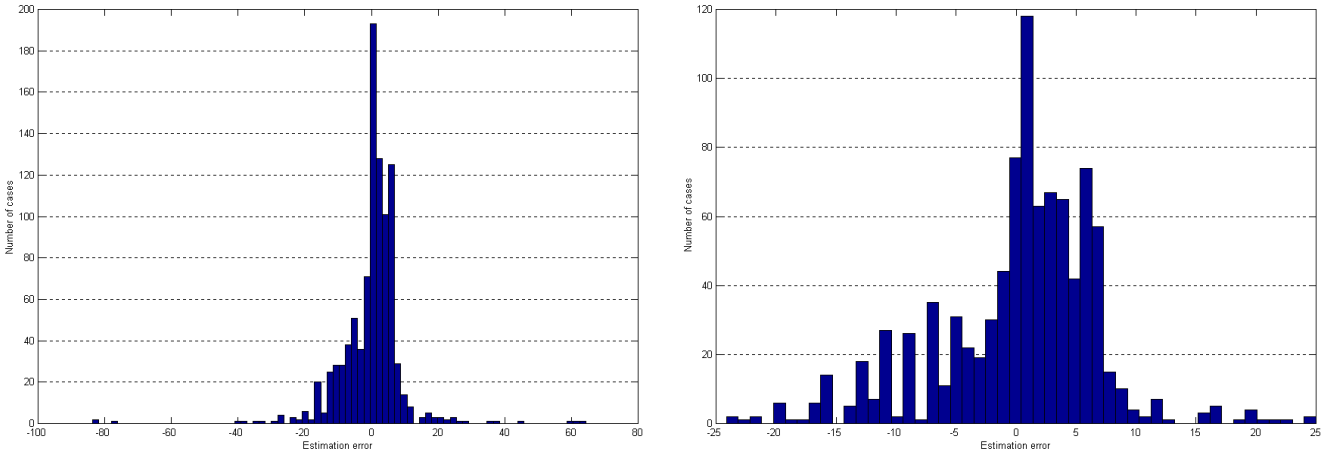
```
component FPURTGeneric
  generic( -- Fake default timing values
    tCY     : time := 50 ns;          -- Clock cycle
    tCHL    : time := 22 ns;          -- CLock High and Low
    tAS     : time := 5 ns;           -- A input setup
    tAH     : time := 1 ns;           -- A input hold
    tDIS    : time := 5 ns;           -- D input setup
    tDIH    : time := 1 ns;           -- D input hold
    tDOD    : time := 7 ns;           -- D output delay
    tDOH    : time := 6 ns;           -- D data valid
    tDOFFL  : time := 7 ns;           -- D output turn-off (FLUSH+)
    tDOHFL  : time := 6 ns;           -- D output valid (FLUSH+)
    tDOFOE  : time := 7 ns;           -- D output turn-off (DOE_N+)
    tDONOE  : time := 7 ns;           -- D output turn-on (DOE_N-)
    tDOHOE  : time := 6 ns;           -- D output valid (DOE_N-)
    tFIS    : time := 5 ns;           -- FINS1/2 input setup
    tFIH    : time := 1 ns;           -- FINS1/2 input hold
    tINS    : time := 5 ns;           -- INST input setup
    tINH    : time := 1 ns;           -- INST input hold
```

Figure 13.64: Model CDM1: Real vs. estimated lines of code.



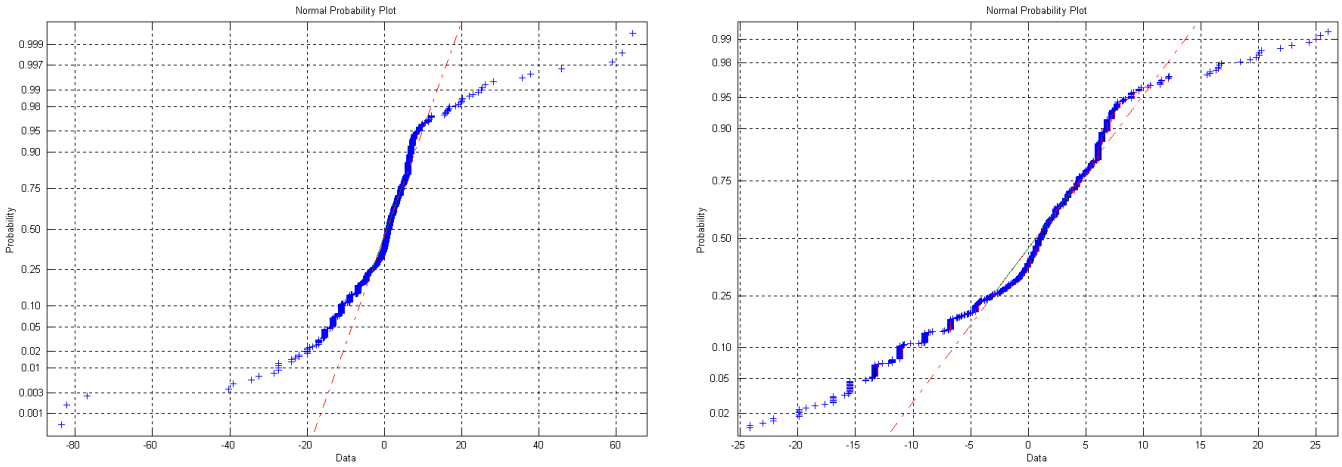

```
20          tFXS     : time := 5 ns;          -- FXACK input setup
            tFXH     : time := 1 ns;          -- FXACK input hold
            tFLS     : time := 5 ns;          -- FLUSH input setup
            tFLH     : time := 1 ns;          -- FLUSH input hold
            tRES     : time := 5 ns;          -- RESET_N input setup
25          tREH     : time := 1 ns;          -- RESET_N input hold
            tMHS     : time := 5 ns;          -- MHOLD_N input setup
            tMHH     : time := 1 ns;          -- MHOLD_N input hold
            tMDS     : time := 5 ns;          -- MDS_N input setup
            tMDH     : time := 1 ns;          -- MDS_N input hold
30          tFHD     : time := 7 ns;          -- FHOLD_N output delay
            tFHH     : time := 6 ns;          -- FHOLD_N output valid
            tFHDFI   : time := 7 ns;          -- FHOLD_N output delay (FINS1/2+)
            tFHDFL   : time := 7 ns;          -- FHOLD_N output delay (FLUSH+)
            tFHDMH   : time := 7 ns;          -- FHOLD_N output delay (MHOLD_N-)
35          tFCCVD   : time := 7 ns;          -- FCCV output delay
            tFCCVH   : time := 6 ns;          -- FCCV output valid
            tFCCVDFL : time := 7 ns;          -- FCCV output delay (FLUSH+)
            tFCCVDMH : time := 7 ns;          -- FCCV output delay (MHOLD_N-)
            tFCCD    : time := 7 ns;          -- FCC output delay
40          tFCCH    : time := 6 ns;          -- FCC output valid
            tFED     : time := 7 ns;          -- FEXC_N output delay
            tFEH     : time := 6 ns;          -- FEXC_N output valid
            tFND     : time := 7 ns;          -- FNULL output delay
            tFNH     : time := 6 ns;          -- FNULL output valid
45          tAPS     : time := 7 ns;          -- APAR input setup
            tAPH     : time := 6 ns;          -- APAR input hold
            tDPIS    : time := 7 ns;          -- DPAR input setup
            tDPIH    : time := 6 ns;          -- DPAR input hold
            tDPOD    : time := 7 ns;          -- DPAR output delay
50          tDPOH    : time := 6 ns;          -- DPAR output valid
            tIFS     : time := 7 ns;          -- IFPAR input setup
            tIFH     : time := 6 ns;          -- IFPAR input hold
            tFIPD    : time := 7 ns;          -- FIPAR output delay
            tFIPH    : time := 6 ns;          -- FIPAR output valid
55          tMCD     : time := 7 ns;          -- MCERR_N output delay
            tMCH     : time := 6 ns;          -- MCERR_N output valid
            tCMS     : time := 5 ns;          -- N602MODE_N, CMODE_N input setup
            tHAS     : time := 5 ns;          -- HALT_N input setup
            tHAH     : time := 1 ns;          -- HALT_N input hold
60          tHAD     : time := 7 ns;          -- HALT_N asserted to output disable delay
            tHAE     : time := 7 ns;          -- HALT_N asserted to output enable delay
            tERD     : time := 7 ns;          -- HWERROR_N output delay
            tERH     : time := 6 ns;          -- HWERROR_N output valid

65          tTCY     : time := 50 ns;         -- TCLK Clock Cycle
            tTMS     : time := 5 ns;          -- TMS setup
```

Figure 13.65: Model CDM1: Error density distribution.




```
    tTMH    : time := 1 ns;          -- TMS hold
    tTDIS   : time := 5 ns;          -- TDI setup
    tTDIH   : time := 1 ns;          -- TDI hold
    tTRS    : time := 5 ns;          -- TRST_N setup
    tTRH    : time := 1 ns;          -- TRST_N hold
    tTDOD   : time := 7 ns;          -- TDO output delay
    tTDOH   : time := 6 ns           -- TDO output valid
  );

  port(
    Clk           : in std_logic;                          -- clock signal

    -- Integer Unit Interface Signals
    FP_N          : inout std_logic;                       -- *Floating-point (Fp) Present
    FCC           : inout std_logic_vector(1 downto 0);    -- *Fp Condition Codes
    FCCV          : inout std_logic;                       -- *Fp Condition Codes Valid
    FHOLD_N       : inout std_logic;                       -- *Fp Hold
    FEXC_N        : inout std_logic;                       -- *Fp EXCeption
    FIPAR         : inout std_logic;                       -- *Fpu to Iu control PARity
    FXACK         : in std_logic;                          -- Fp eXception ACKnowledge
    INST          : in std_logic;                          -- INSTruction fetch
    FINS1         : in std_logic;                          -- Fp INStruction in buffer 1
    FINS2         : in std_logic;                          -- Fp INStruction in buffer 2
    FLUSH         : in std_logic;                          -- Fp instruction fLUSH
    IFPAR         : in std_logic;                          -- Iu to Fpu control PARity

    -- System/Memory Interface Signals
    A             : in std_logic_vector(31 downto 0);      -- Address bus
    APAR          : in std_logic;                          -- Address bus PARity
    D             : inout std_logic_vector(31 downto 0);   -- Data bus
    DPAR          : inout std_logic;                       -- Data bus PARity
    DOE_N         : in std_logic;                          -- Data Output Enable
    COE_N         : in std_logic;                          -- Control Output Enable
    MHOLDA_N      : in std_logic;                          -- Memory HOLD
    MHOLDB_N      : in std_logic;                          -- Memory HOLD
    BHOLD_N       : in std_logic;                          -- Bus HOLD
    MDS_N         : in std_logic;                          -- Memory Data Strobe
    FNULL         : inout std_logic;                       -- *Fpu NULLify cycle
    RESET_N       : in std_logic;                          -- Reset signal
    HWERROR_N     : out std_logic;                         -- Hardware error detected
    CMODE_N       : in std_logic;                          -- master/Checker MODE
    MCERR_N       : out std_logic;                         -- Comparison Error
    N602MODE_N    : in std_logic;                          -- Normal 602MODE Operation
    HALT_N        : in std_logic;                          -- Halt mode

    -- Coprocessor Interface Signals
    CHOLD_N       : in std_logic;                          -- Coprocessor hold.
    CCCV          : in std_logic;                          -- Coprocessor Condition Code Valid.

    -- Test Access Port (TAP) signals
    TCLK          : in std_logic;                          -- Test CLocK
    TRST_N        : in std_logic;                          -- Test ReSeT
    TMS           : in std_logic;                          -- Test Mode Select
    TDI           : in std_logic;                          -- Test Data In
    TDO           : out std_logic                          -- Test Data Out
  );
end component; -- FPURTGeneric
```

Figure 13.66: Model CDM1: Error cumulative distribution.



13.3.3 Model CDM2

Given the above considerations, a natural improvement over CDM1 would be to add a linear contribution associated to generic constants, thus obtaining CDM1.

Model:

$$L = k_{np} \cdot n_p + k_{ng} \cdot n_g + k_0$$

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	13.540	13.540	0.000
Variance	198.650	134.734	63.916
Standard deviation	14.094	11.608	
7.995			

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.8236

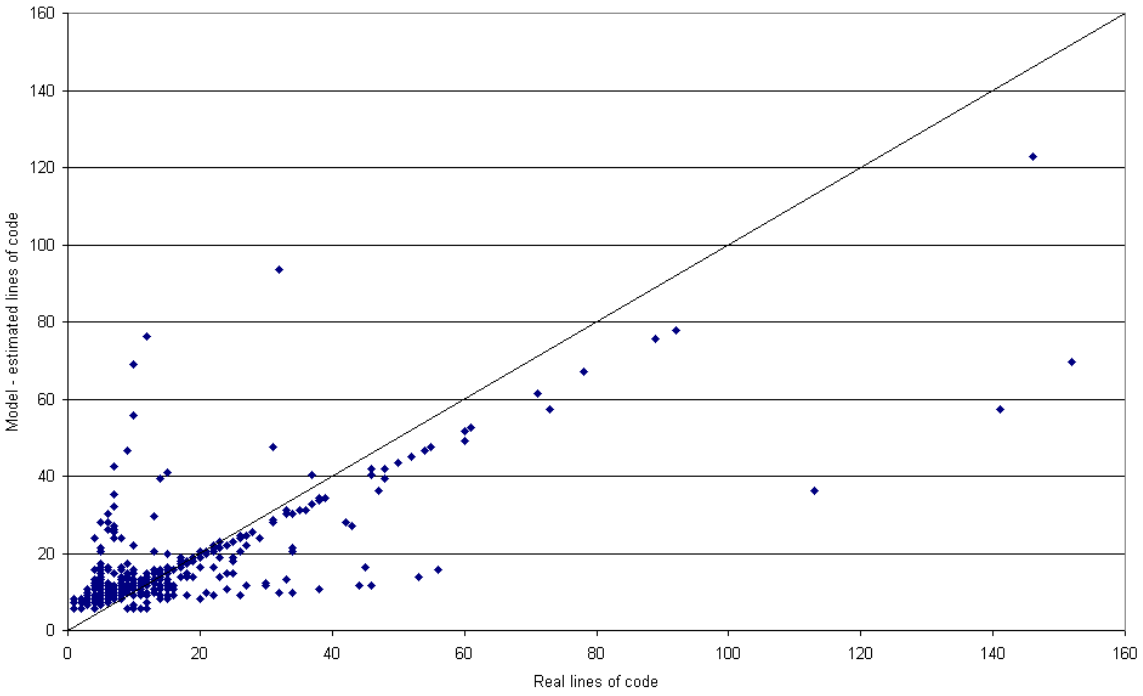
Identified model coefficients:

Coefficient	Value
k_{np}	0.7336
k_{ng}	1.4530
k_0	5.7537

13.3.4 Conclusions

Same considerations as for entities apply. Our results show that component declarations can be modelled in an accurate way by linear models using primarily the number of ports

Figure 13.67: Model CDM2: Real vs. estimated lines of code.



and generic constants as input. Model CDM2 shows an estimation error which is between -10 and +10 lines of code in 85% of the cases, and between -20 and +20 in 96% of the cases. Again, models show a little tendency to overestimate some declarations, but this phenomenon is due to coding style reasons, and there is no way to predict them.

Figure 13.68: Model CDM2: Error density distribution.

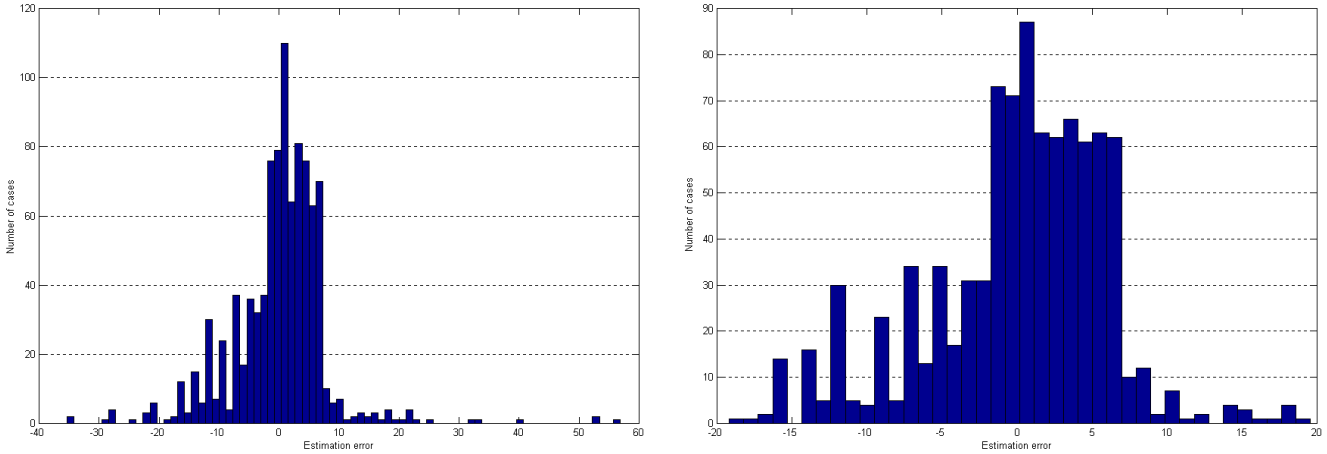
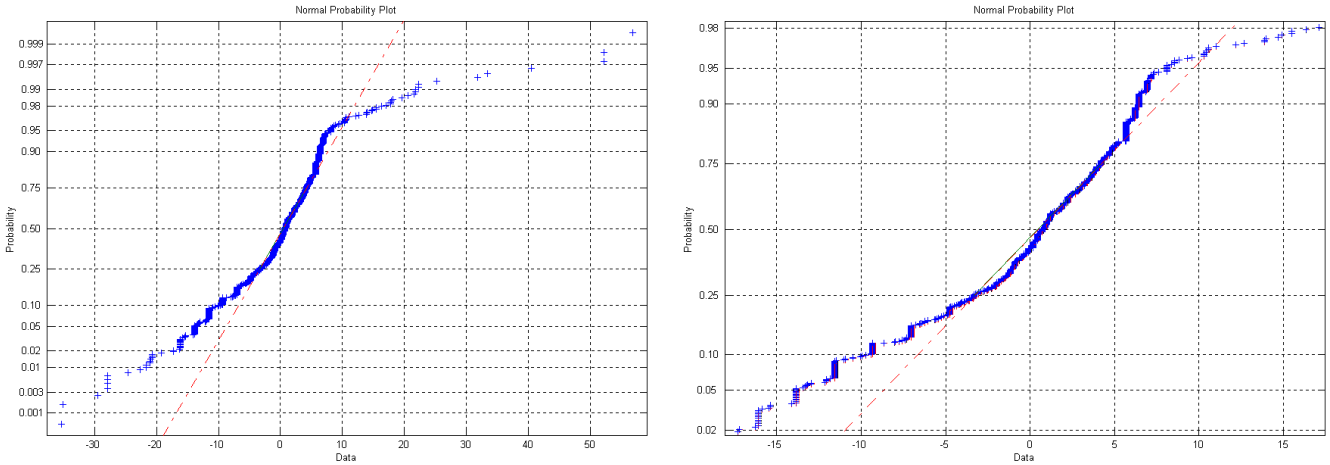


Figure 13.69: Model CDM2: Error cumulative distribution.



13.4 Architecture component instantiation models

13.4.1 Model CIM1

$$\hat{L} = k_{np} \cdot n_p + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	9.395	9.395	0.000
Variance	72.525	60.657	11.868
Standard deviation	8.516	7.788	3.445
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	1.638	1.638	0.000
Variance	14.383	5.453	8.921
Standard deviation	3.792	2.335	2.987
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	4.592	4.592	0.000
Variance	41.274	13.832	27.443
Standard deviation	6.425	3.719	5.239

Correlation between estimated and real values:

	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient $(L,\hat{L})$	0.915	0.616	0.579

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_{np}	1.0731	0.4097	0.3600
k_0	-1.3359	-0.3556	1.8573

Figure 13.70: Model CIM1: Real vs. estimated lines of code.

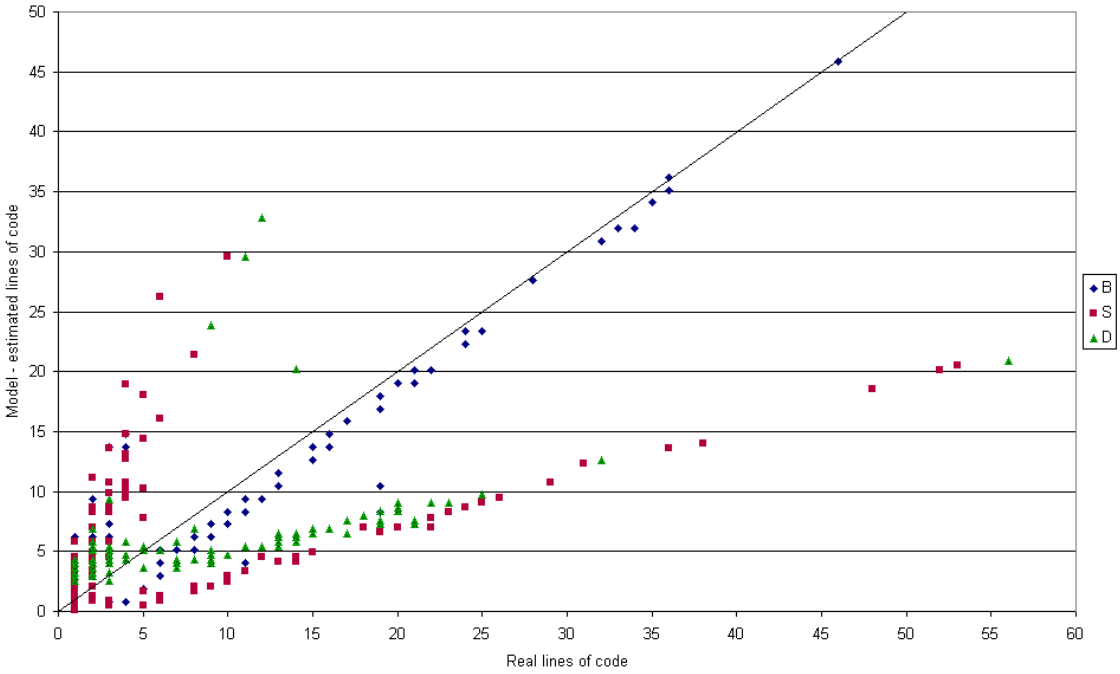


Figure 13.71: Models CIM1b, CIM1s, CIM1d: Real vs. estimated lines of code.

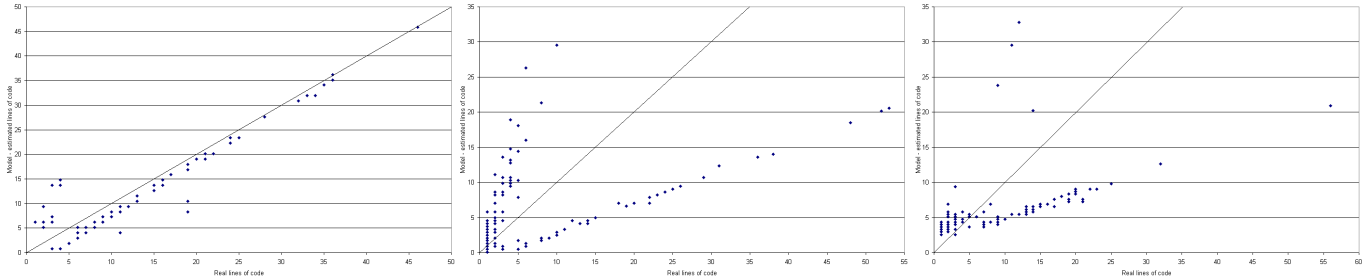


Figure 13.72: Model CIM1: Error density distribution.

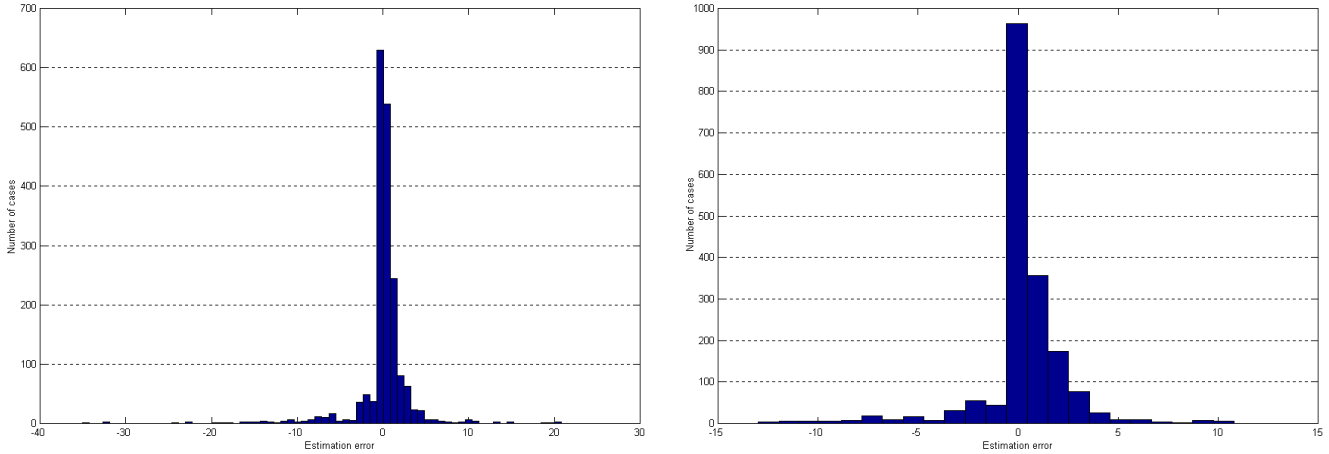


Figure 13.73: Model CIM1: Error cumulative distribution.

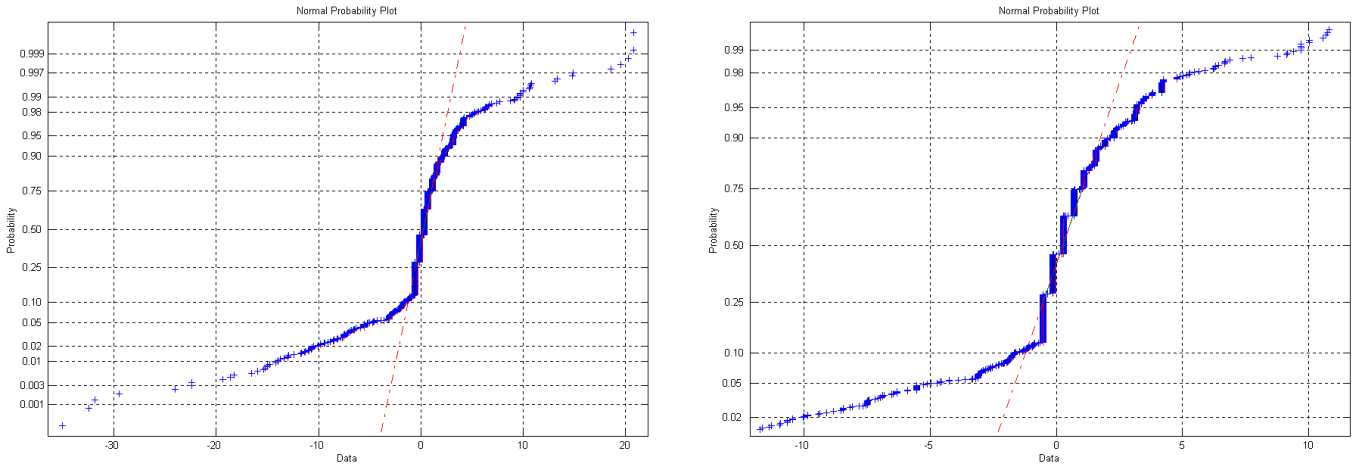
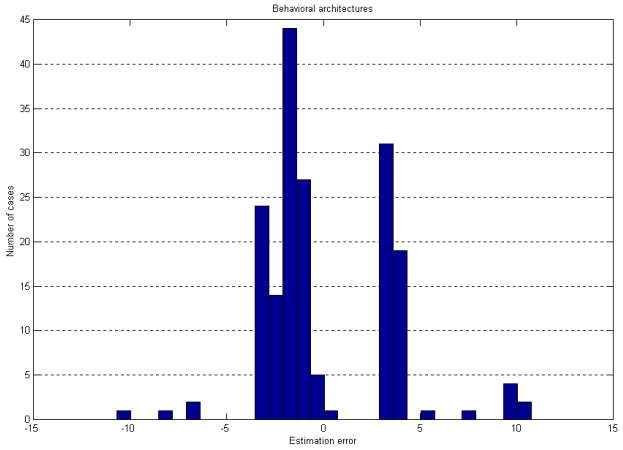
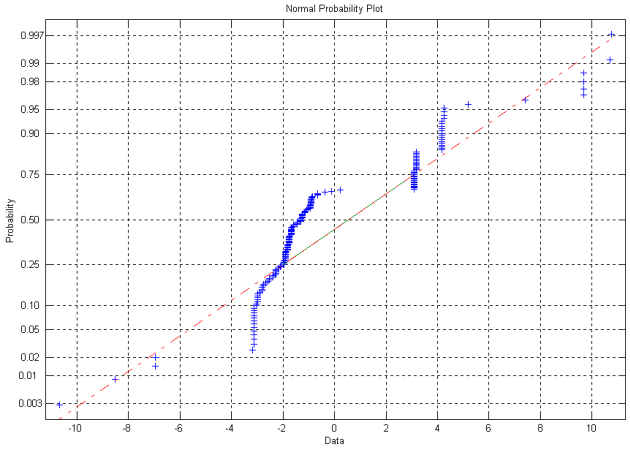
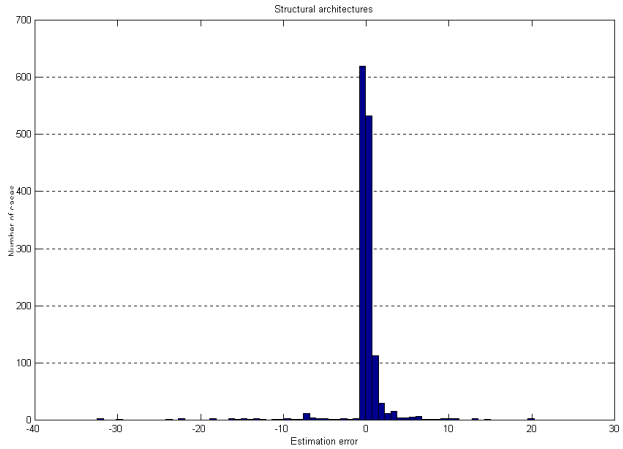
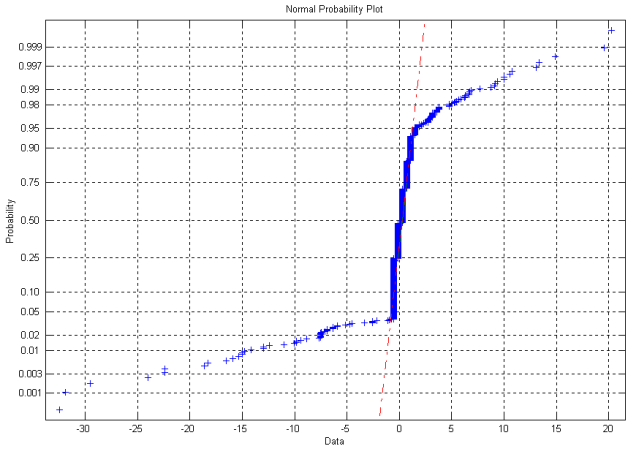


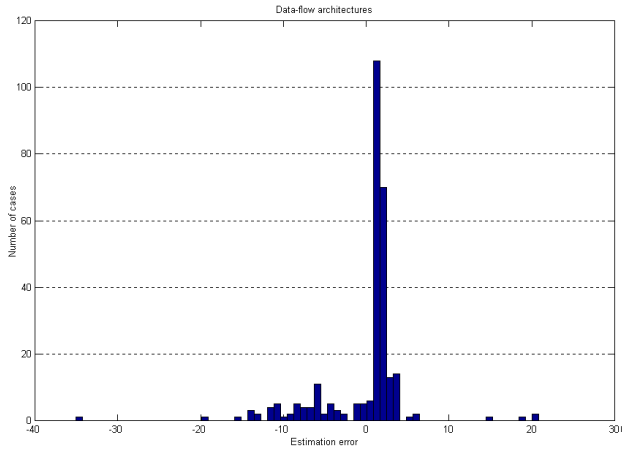
Figure 13.74: Models CIM1b, CIM1s, CIM1d: Error density and cumulative distributions.

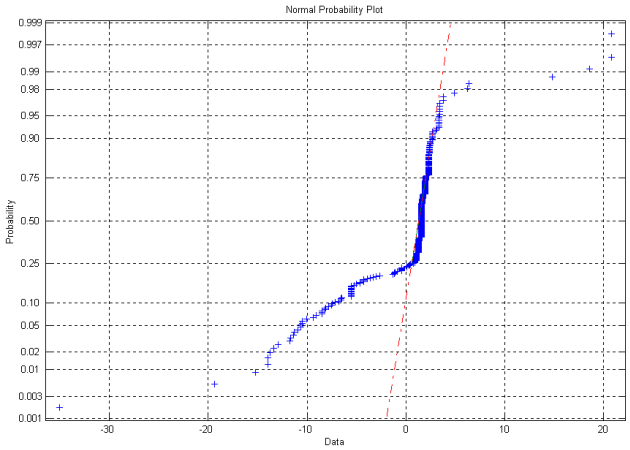












13.4.2 Conclusions

Findings emerged during the evaluation of model CIM1 confirm an idea that was already expressed at the beginning of part IV: the paradigm (i.e. the mode used to describe architectures) not only influences the way in which a programmer thinks, but also his coding style.

CIM1 is a good model, especially for behavioral architectures, where to coding style is uniform everywhere: all the designers write exactly one port per line (and this claim is confirmed in its crystalline simplicity by that $Corr(L, \hat{L}) = 0.915$ and $k_{np} = 1.0731$.

For structural and data-flow architectures, things are more complicated. It seems that two coding styles exist (the first one is one port per line, as above; the second one is multiple ports of the same type per line) and the model cannot do any better than returning an estimate which stands in between the two styles.

13.5 Process models

13.5.1 Variables

The following models try to estimate the number of core lines of code of a given process (that is, lines of code depurated from local subprogram declarations) by knowing little information, such as the number of signals in its sensitivity list, or their homogeneity. The following table gives a list of all the variables considered for model generation:

Externally available variables:

Variable	Symbol
Total number of ports in the entity associated to the architecture containing the current process	n_p
Number of sensitivity elements appearing in the process sensitivity list	n_{ps}
Total number of generic constants in the entity associated to the architecture containing the current process	n_g
Total homogeneity of signals declared in the architecture containing the current process	h_s
Number of signals declared in the architecture containing the current process	n_s
Number of component instantiations present in the architecture containing the current process	n_{ci}

Internally available variables:

Variable	Symbol
Number of variables declared in the current process	n_v
Homogeneity of variables declared in the current process	h_v

Unavailable variables:

Variable	Symbol
Number of distinct object references to signals or variables inside the current process body	n_{or}
Number of distinct object uses of signals or variables inside the current process body	n_{ou}
Number of distinct object definitions of signals or variables inside the current process body	n_{od}
Number of unique referenced objects inside the current process body	n_{uor}
Number of unique used objects inside the current process body	n_{uou}
Number of unique defined objects inside the current process body	n_{uod}

Variables grouped under the "externally available" category are known when the process is externally known (this means that the architecture to which it belongs should be at least internally). "Internally available" variables require in practice the knowledge of variables[5] declared inside the process.

It is clear that our primary interest is for good models using externally available variables only.

During this chapter we will examine both models using and not using internally available knowledge; we report the second ones only for study purposes, their usefulness in real estimation problems is limited to those conditions in which the used variables are actually known.

Our VHDL parser and database are designed to recognize and store each single instruction where a signal or a variable are read or written. In accordance with software engineering terms used for variable flow analysis, every time a variable or a signal is assigned a value, we say that there is a *definition*; when the value contained in a variable or a signal is used (for example it is given as input to a function call statement, or assigned to another object) we say that there is a *use*. Definitions and uses are *references*. References are here denoted as unavailable information because they are normally available to the designer only when the process in which they appear is completely known, that is, when implementation work is finished.

It is obvious that an estimation model requiring as input a piece of information which is available when at the end of development process is completely useless. Nevertheless, our parser is designed to store each reference in a table called `OBJECT_REFERENCES` (see § 8.5.12, page 79), and thanks to that table we will be able to draw some "a posteriori" conclusions that, though not resulting in a model, are yet of some interest.

13.5.2 Correlation study

Exactly like it happened with architectures, it is difficult to find variables useful for creating good process length estimation models. In fact, all external variables show little or no correlation at all with L. Internally available variable (number and homogeneity of process variables) exhibit a slightly better coefficient of correlation with L, yet still not encouraging.

[5] please note that in this context the term *variable* is overloaded. We call variables the input information used by our estimation models, but memory elements declared inside processes are also named variables. Here there are two a model variables, respectively containing the count and homogeneity of process variables.

Variable	Average value	Variance	Standard deviation	Correlation coefficient
		Externally available variables		
n_p	23.663	582.804	24.141	-0.0372
n_{ps}	2.828	10.051	3.170	0.0849
n_g	2.797	73.670	8.583	0.0771
h_s	52.907	7403.999	86.046	0.0104
n_s	43.578	5869.861	76.615	-0.0399
n_{ci}	1.644	8.002	2.829	-0.0670
		Internally available variables		
n_v	1.292	32.042	5.661	0.3558
h_v	2.363	199.258	14.116	0.2642
		Unavailable variables		
n_{or}	59.781	88348.876	297.235	**0.9617**
n_{ou}	23.651	19741.576	140.505	**0.9304**
n_{od}	36.130	25806.208	160.643	**0.9657**
n_{uor}	14.820	646.617	25.429	0.4218
n_{uou}	4.286	147.946	12.163	0.4059
n_{uod}	12.092	416.810	20.416	0.4256
L	46.135	33593.324	183.285	(1.0000)

The only indicators showing a very strong coefficient of correlation with the data to be estimated are object reference counts (their correlations are typed in bold), which are in practice inaccessible at the time in which project high-level specifications are prepared. Nevertheless, we would like to express our reasonable belief that a model using that data would estimate process lengths in a very accurate way. We leave to future developments the search for special cases and applications in which it is possible to know in advance the number of object references: for those cases such models will be practically useful.

13.5.3 Model PM0

As indicated by its null enumerating number, model PM0 is a trivial model, that is, a simple constant. Model PM0 does not make use of any available information on the given process (apart from mode of the architecture to which the current process belongs), and simply returns the average value of the length of processes for the given mode.

Being all externally available variables uncorrelated with process lengths, model PM0 is a forced choice under many circumstances.

Model:

$$\hat{L} = k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L}-L$
Average value	44.158	44.158	0.000
Variance	36328.511	0.000	36328.511
Standard deviation	190.600	0.000	190.600
	Structural architectures		
	L	$\hat{L}$	$\hat{L}-L$
Average value	24.333	24.333	0.000
Variance	1130.333	0.000	1130.333
Standard deviation	33.620	0.000	33.620
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L}-L$
Average value	58.584	58.584	0.000
Variance	18225.009	0.000	18225.009
Standard deviation	135.000	0.000	135.000

Correlation between estimated and real values:

	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient $(L,\hat{L})$	0	0	0

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient	Value		
k_0	44.158	24.333	58.584

Remarks: of course, for all PM0b, PM0s and PM0d models, correlation coefficients between real and estimated values are null, being the estimates constant values.

Figure 13.75: Model PM0: Real vs. estimated lines of code.

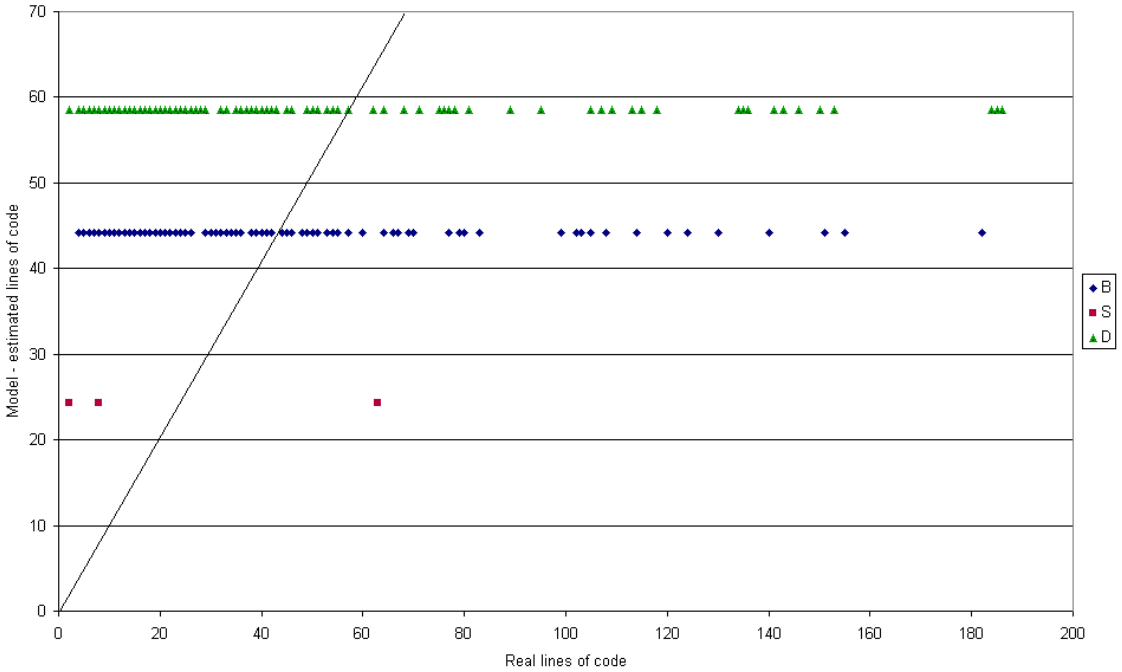


Figure 13.76: Model PM0: Error density distribution.

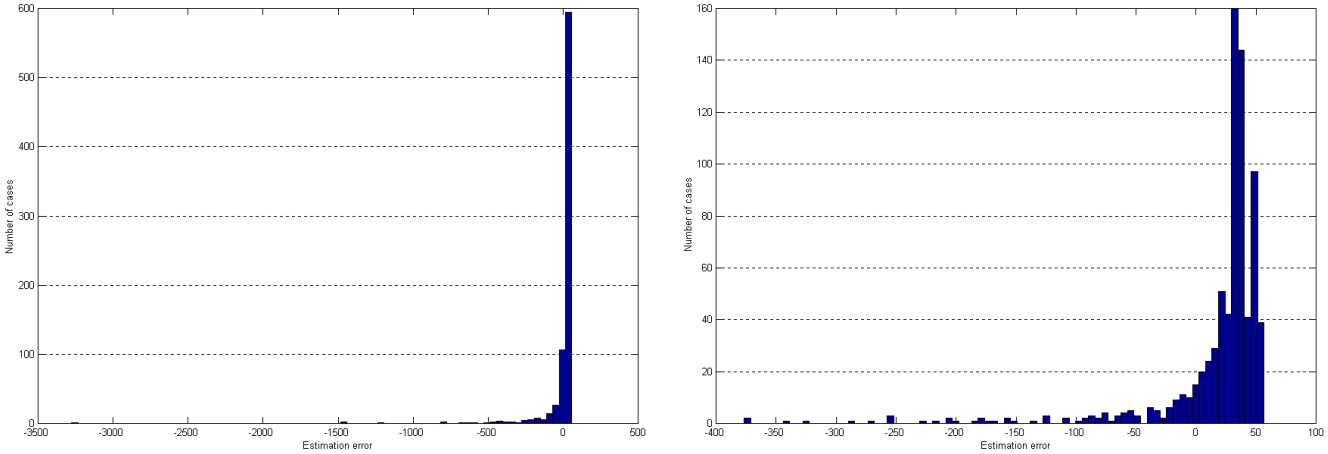


Figure 13.77: Model PM0: Error cumulative distribution.

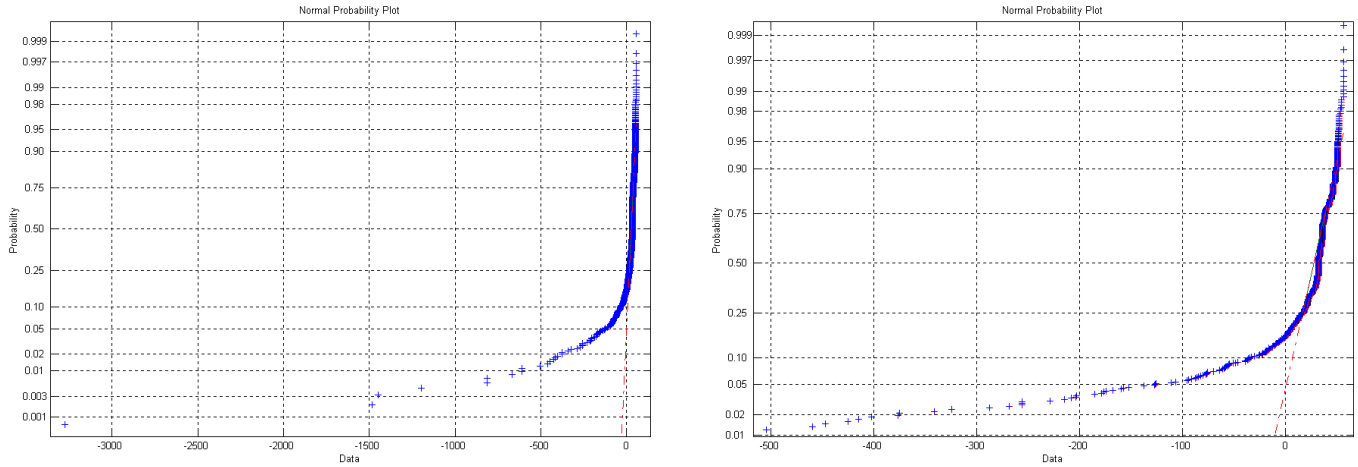
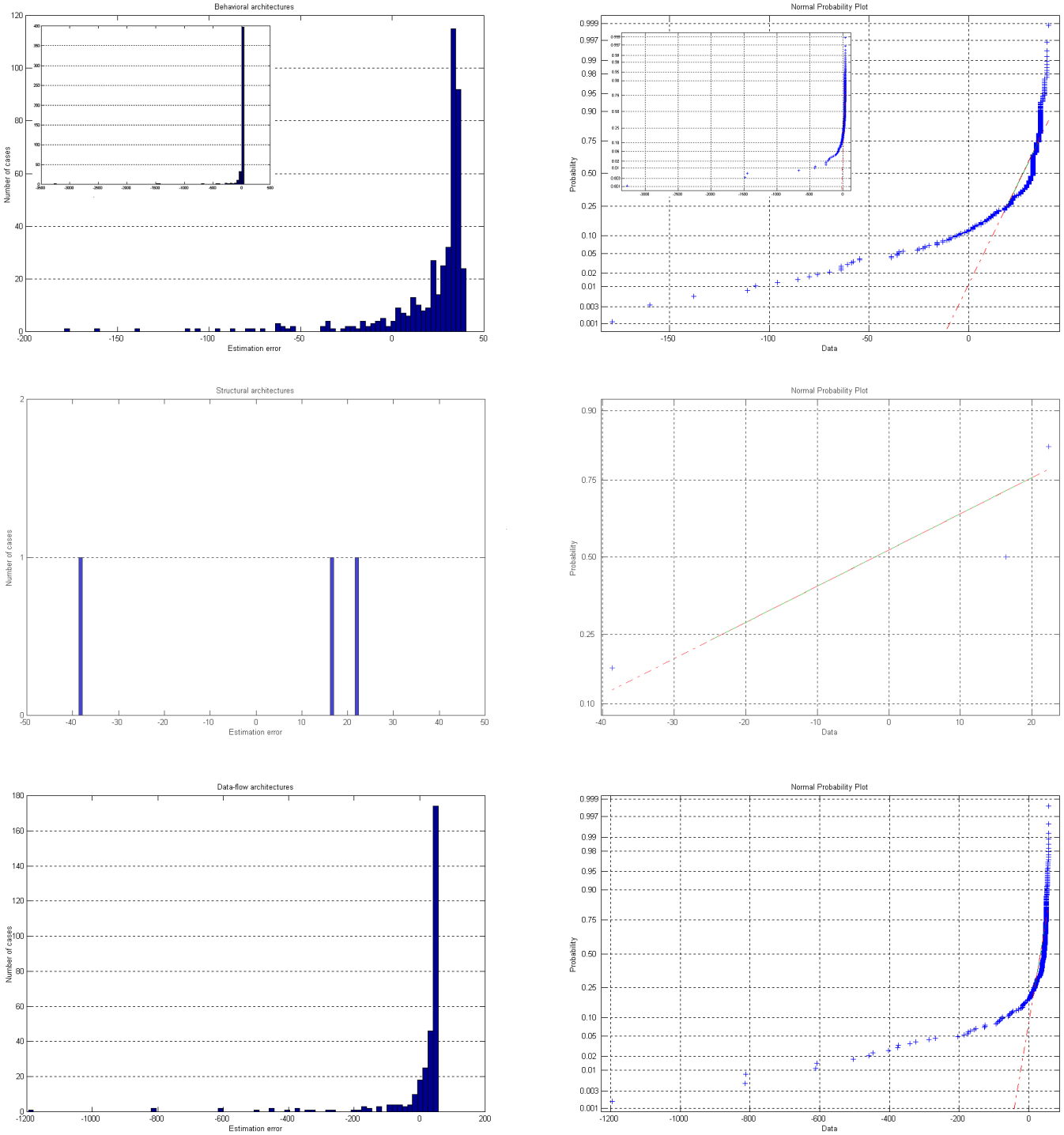


Figure 13.78: Models PM0b, PM0s, PM0d: Error density and cumulative distributions.



13.5.4 Model PM1

Model PM1 is an improvement over model PM0, using one internally available variable, namely the number of declared process variables. We did not prepare any PM1H model using the homogeneity of such process variables, since correlation coefficients suggest that it would exhibit worse accuracy than PM1.

Model:

$$\hat{L} = k_v \cdot n_v + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	44.158	44.158	0.000
Variance	36328.511	8066.131	28262.380
Standard deviation	190.600	89.812	168.114
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	24.333	24.333	0.000
Variance	1130.333	1121.333	9.000
Standard deviation	33.620	33.486	3.000
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	58.584	58.584	0.000
Variance	18225.009	4793.487	13431.521
Standard deviation	135.000	69.235	115.894

Correlation between estimated and real values:

	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient (L,$\hat{L}$)	0.4712	0.9960[6]	0.5129

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_v	14.863	19.333	9.772
k_0	30.839	5.000	41.941

Figure 13.79: Model PM1: Real vs. estimated lines of code.

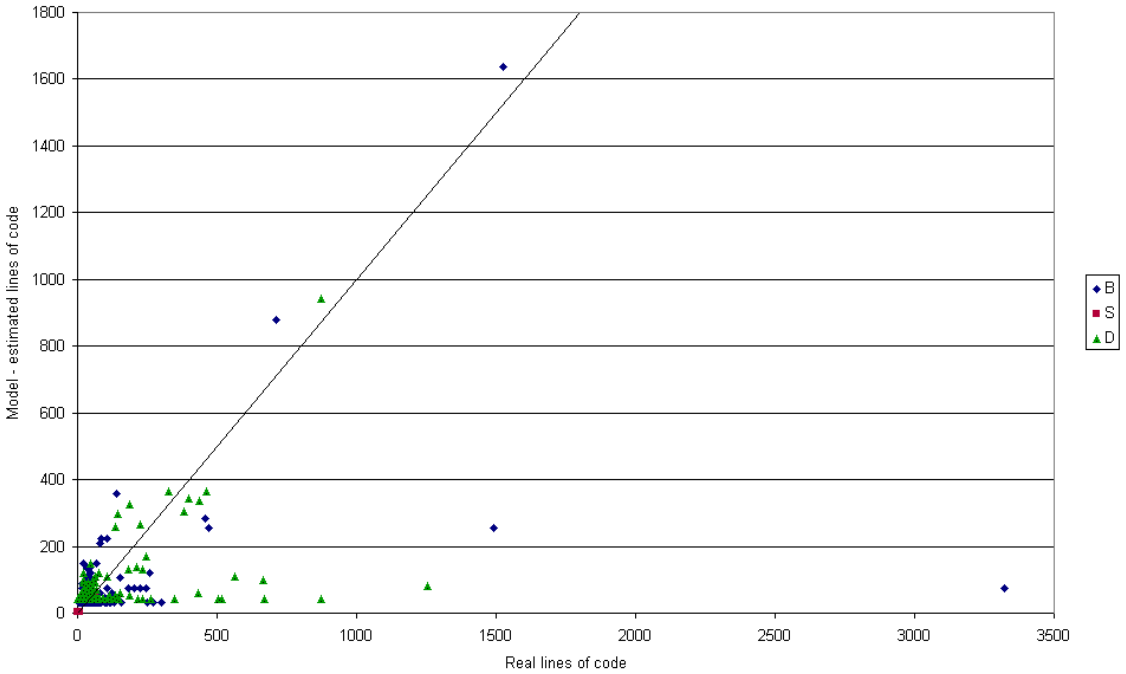


Figure 13.80: Model PM1: Error density distribution.

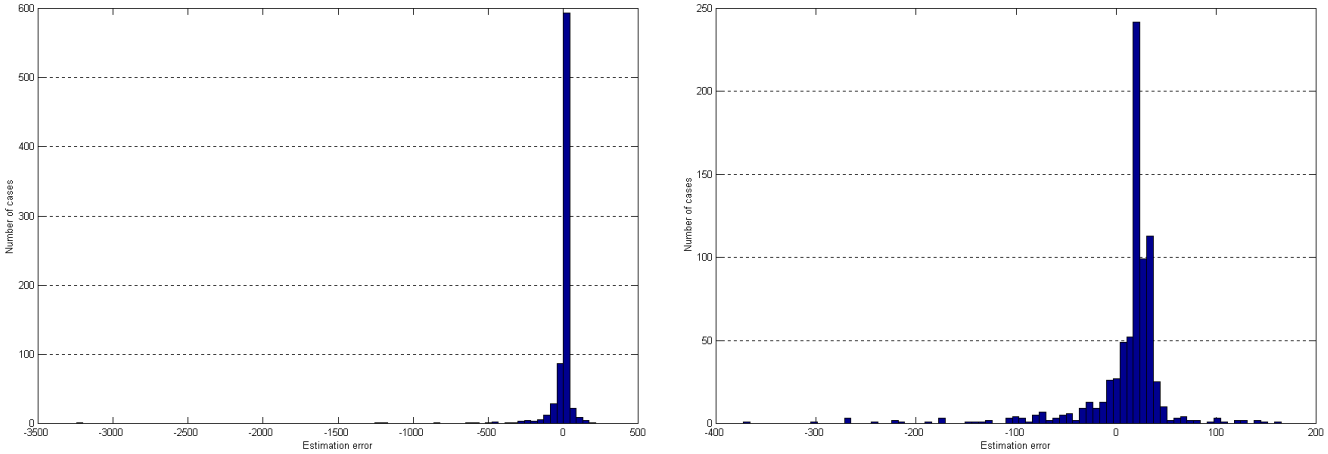


Figure 13.81: Model PM1: Error cumulative distribution.

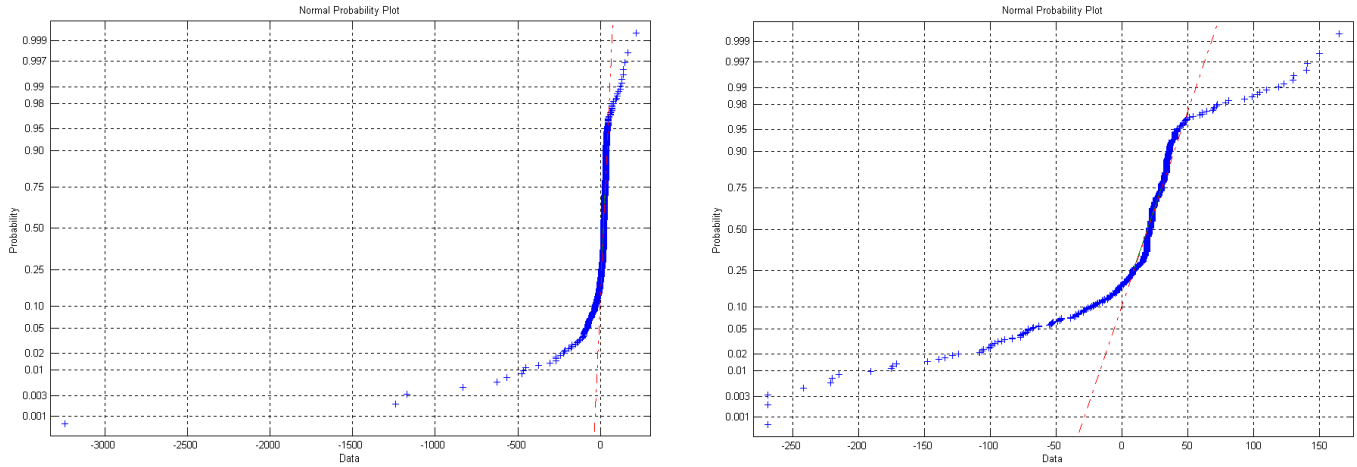
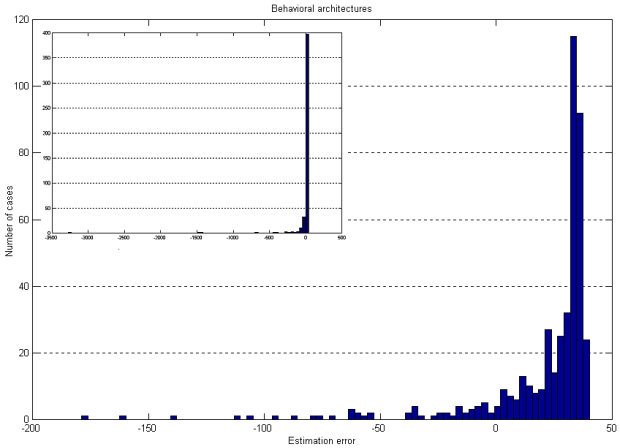
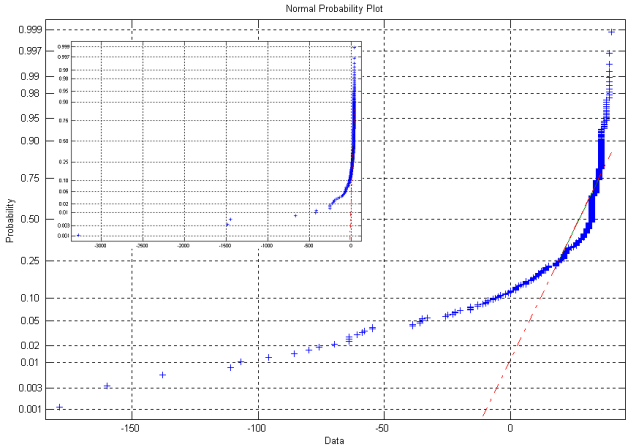
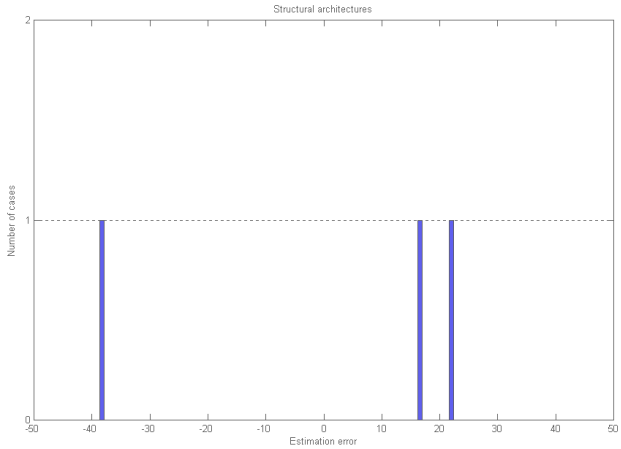
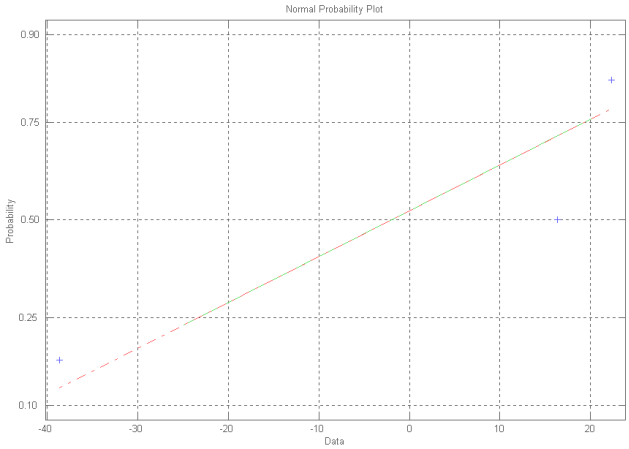


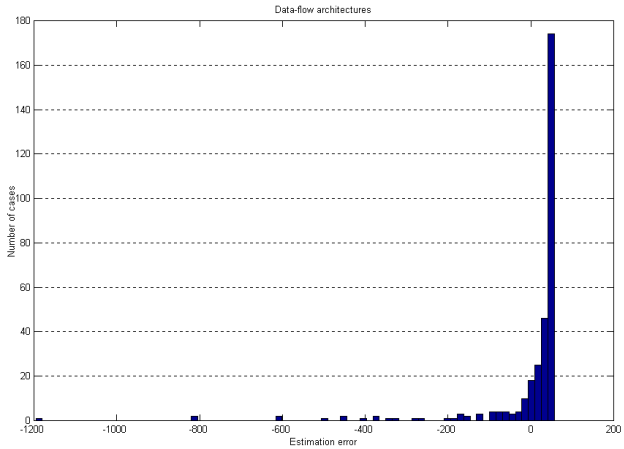
Figure 13.82: Models PM1b, PM1s, PM1d: Error density and cumulative distributions.

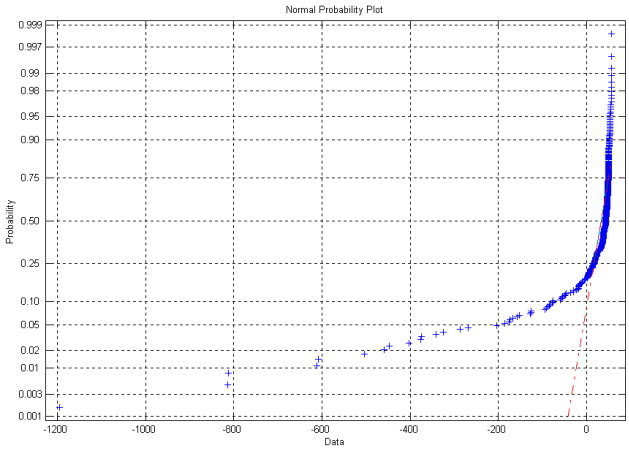












13.5.5 Conclusions

As for architectures, given the above correlation study results, we believe that there are no models that estimate process lengths, by using the available variables, exhibiting high coefficient of correlation between actual and estimated lengths. Again, failing to reach good correlations does not mean that above models are useless. On the contrary, error study results show that under many circumstances estimates provided by these models are acceptable, and thanks to limited variance of length in the original population, their error variance is not too high.

For example, model PM1 estimates the length of processes with an estimation error falling between -75 and +75 lines of code in 93% of the cases.

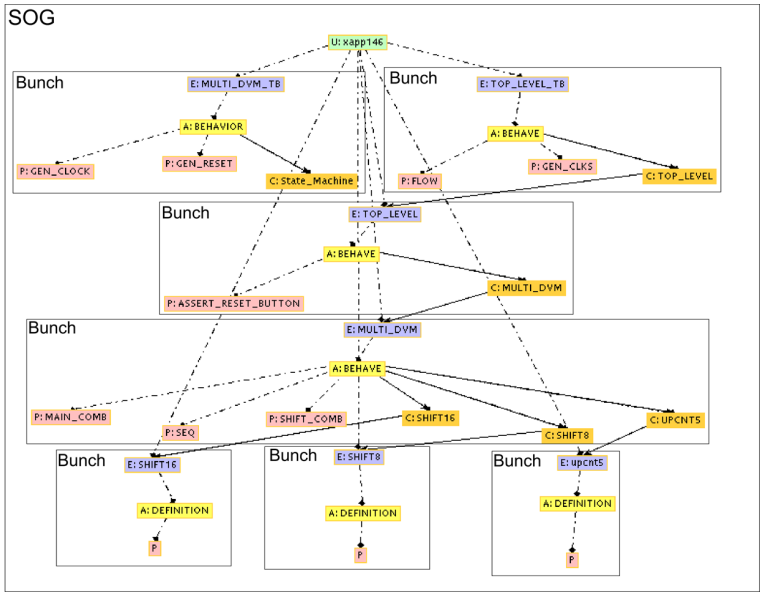
Chapter 14

Bunch models

Bunches were defined in the theory chapter, but in the meanwhile we introduced additional syntax objects (component declaration and instantiations) which were not defined; moreover bunch properties and purpose were not clarified so far. It is now time to do it.

First of all, it is easy to prove not only that every bunch is an acyclic graph, but also that it is a tree. In fact it makes use of *contains*-type only relation, and any DAG using only *contains*-type relations was proved to be a tree (see § 3.20, page 22).

Figure 14.1: Partition of a SOG into its bunches



It can be proven that, except for package objects and for the root (the unique U-class object, representing th project), all the nodes in a given SOG can be partitioned in bunches. Figure 14.1 contains an example of the partitioning operation just described, applied on a sample SOG.

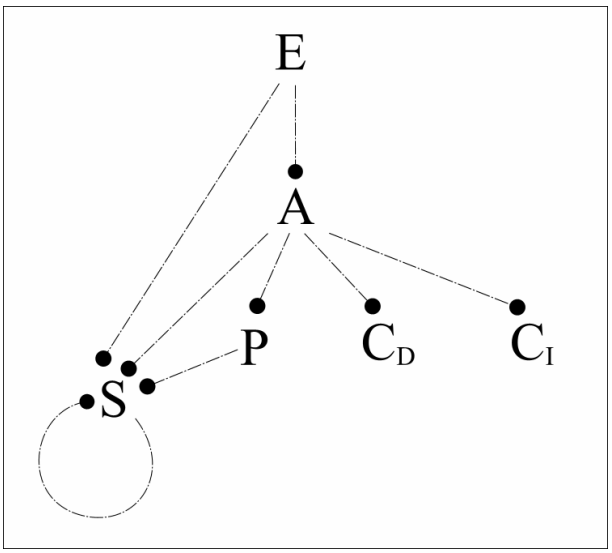
In order to simply verify SOG properties we introduced the SCG, we also do it for the

bunch (which is an ECOG), by introducing the bunch class graph (or, more formally, the entity containment class graph, ECCG for short). We must not forget that an ECCG is a theoretical useful object that does not represent any property specific to whatever real project. In fact, whereas there is an infinite number of possible bunches, there is only one possible ECCG, and it represents on a syntax class level the possible relations between syntax objects inside a bunch. An edge in a ECCG going from a class w_i to a class w_j means that it is possible, inside a bunch, for an object of class w_i to contains an object of class w_j.

An ECOG must contain exactly one entity and can contain one –and possibly more– architectures, zero or more component declarations and instantiations, zero or more processes, zero or more subprograms. An ECCG therefore must contains the entity, architecture, process, component declaration, component instantiation and subprogram classes.

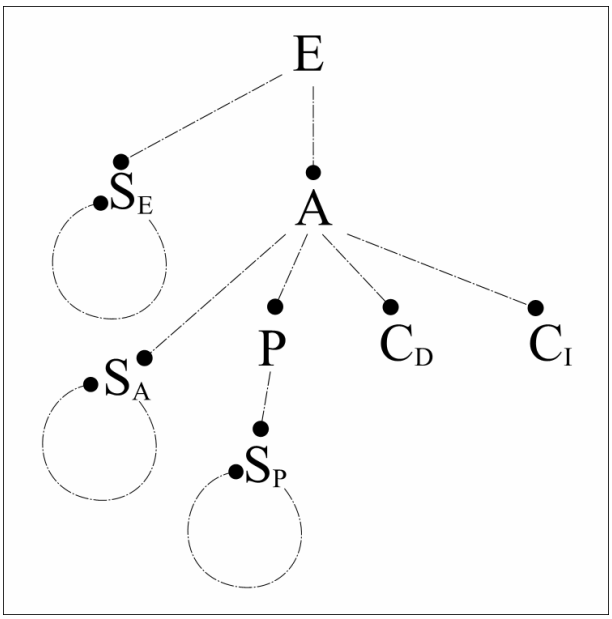
In an ECOG the one entity can contain architectures and subprograms, the architecture can contain processes, component declarations, component instantiations, and subprograms. Processes can contain subprograms. Subprograms can contain other subprograms.

The above considerations lead to the following ECCG:



Since our intent is to develop models to estimate cardinality of elements involved in each relationship appearing in that graph, and it cannot happen in the reality that there is an infinite number of recursively contained objects, we need to "unroll" loops and to distinguish objects of the same class contained in objects of different classes.

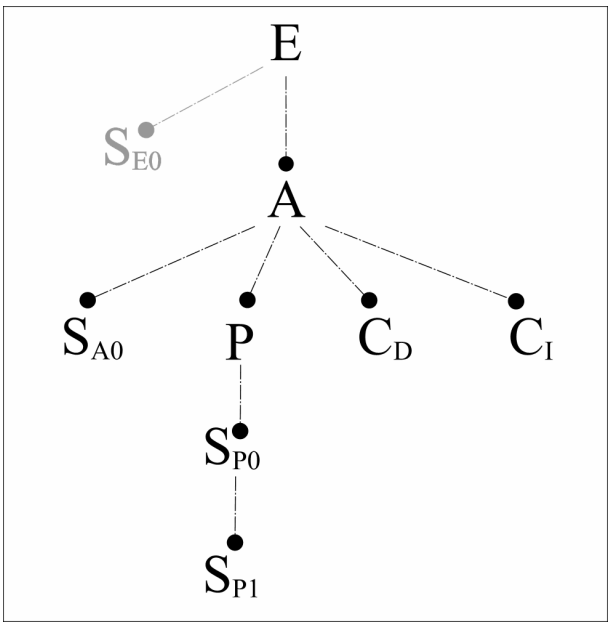
In order to do that, class S is replaced by classes S_E, S_A, S_P, which respectively represent subprograms contained in entities, architectures and processes, possibly nested inside subprograms of the same respective classes. After that transformation, which can be done without knowing anything about real projects, the ECCG appears as follows:



What remains to do now is to perform the unrolling operation, thus substituting subprogram containment loops with a finite number of arcs and new subprogram classes, one for each possible nesting level. For example S_{P0} represents the class of all subprograms immediately contained in processes, S_{P1} the class of all subprograms immediately contained in subprograms of class S_{P0}, and so on. Performing this unrolling operation would lead to an infinite chain of nodes, like the following:

$$S_{x0} \supset S_{x1} \supset S_{x2} \supset ... \qquad (\forall x \in \{E, A, P\})$$

In order to unroll up to useful nesting levels only (that is, levels which are actually populated by some node in the projects we examined), unlike the previous transformation we applied, a certain amount of knowledge about project base is needed, namely up to which nesting levels the various types of subprogram exist. In order to answer to that questions, inspection of the project base (that is, in practice, suitable SQL queries on the VHDL project database) is required. The result is reported in the following, definitive, ECCG.



14.1 Bunch nodes actually present

The actual reasoning and queries used to determine the existence of nodes and arcs in the definitive form of the ECCG are reported in the following pages. Most of the queries exclude test benches, automatically generated code and ROMs, as justified in §9. A suitable set of experimentally determined SQL conditions required to do so are the following:

```
     ARCHITECTURE_NAME <> '(null)' AND PROCESS_NAME = '(null)'
AND  INSIDE_SUBPROGRAM = '(null)'
AND  ENTITY_NAME NOT LIKE '\%test\%be\%'
AND  ARCHITECTURE_NAME NOT LIKE '\%test\%be\%'
AND  NOT (ENTITY_NAME LIKE 'mul\%' AND ARCHITECTURE_NAME='struct')
AND  FILE_NAME NOT LIKE '\%bench\%'
AND  FILE_NAME NOT LIKE '\%test_vectors\%'
AND  FILE_NAME NOT LIKE '\%time\%sim\%'
AND  FILE_NAME NOT LIKE '\%synth\%'
AND  ENTITY_NAME NOT LIKE '\%post\%'
AND  ENTITY_NAME NOT LIKE '\%cpld\%'
AND  ENTITY_NAME NOT LIKE '\%rom\%'
AND  ARCHITECTURE_NAME NOT LIKE '\%rom\%'
```

We will avoid repeating the above conditions every time we need them. In our place we will put three asterisks `'***'`.

14.1.1 S_{E0}

The $E \supset S_{E0}$ arc is drawn in grey since in our project base no subprogram declaration occur inside entities. In fact there are 4 cases in which subprograms are declared inside entities, but all these cases are inside test benches, and test benches are excluded from our model tuning sets, due to their particular nature.

In order to verify this claim, it is sufficient to invoke any suitable SQL query that lists all the subprograms included in some entity but not in any architecture, e.g. the following:

```
SELECT * FROM VHDL.SUBPROGRAM_DECLARATIONS
WHERE   ENTITY_NAME        <>   '(null)'
AND     ARCHITECTURE_NAME  =    '(null)';
```

According to these considerations, we will safely ignore the existence of S_{E0} nodes and $E \supset S_{E0}$ arcs.

14.1.2 S_{E1}

An S_{E1}-type node represents a subprogram declared inside another subprogram which is, in turn, declared inside an entity. This sort of elements do not exist, even considering objects like test benches and ROMs that we excluded.

In order to verify this claim, it is sufficient to invoke any suitable SQL query that lists all the subprograms included in some subprogram which are included in some entity but not in any architecture, e.g. the following:

```
SELECT * FROM VHDL.SUBPROGRAM_DECLARATIONS
WHERE   ENTITY_NAME        <>   '(null)'
AND     ARCHITECTURE_NAME  =    '(null)'
AND     INSIDE_SUBPROGRAM  <>   '(null)';
```

which returns the empty set.

14.1.3 S_{A0}

Nodes of this type represent subprograms included in architectures. There are 73 such cases in the considered architectures (and 91 in all architectures). In order to verify this claim, any query listing subprograms included in architectures but not in processes or other subprograms would do, e.g., the following one:

```
SELECT * FROM VHDL.SUBPROGRAM_DECLARATIONS
WHERE    ARCHITECTURE_NAME  <>   '(null)'
AND      PROCESS_NAME       =    '(null)'
AND      INSIDE_SUBPROGRAM  =    '(null)'
AND      ***;
```

14.1.4 S_{A1}

Nodes of type S_{A1} represent subprograms directly included in nodes of type S_{A0}. In all our projects, no such cases occur. This is the reason why the $S_{A0} \supset S_{A1} \supset S_{A2} \supset ...$ node chain stops early with S_{A0}.

To verify that there are no S_{A1}-type nodes, just issue a query listing subprograms included in architectures and other subprograms but not processes, like the one as follows:

```
SELECT * FROM VHDL.SUBPROGRAM_DECLARATIONS
WHERE  ARCHITECTURE_NAME   <>   '(null)'
AND    PROCESS_NAME        =    '(null)'
AND    INSIDE_SUBPROGRAM   <>   '(null)';
```

14.1.5 S_{P0}

Nodes of type S_{P0} represent subprograms immediately contained in processes. There are 84 such subprograms in all architectures, of which 84 in architecture which do not belong to "illegal" categories (ROMs, testbenches, automatically generated code). To get a list of them,

just issue a query that selects all subprograms included in processes but not in subprograms, like as follows:

```
SELECT * FROM VHDL.SUBPROGRAM_DECLARATIONS
WHERE     ENTITY_NAME        <>  '(null)'
AND       ARCHITECTURE_NAME  <>  '(null)'
AND       PROCESS_NAME       <>  '(null)'
AND       INSIDE_SUBPROGRAM  =   '(null)'
AND       ***;
```

14.1.6 S_{P1}

Nodes of type S_{P1} represent subprograms immediately contained in subprogram of type S_{P0}. There are 8 such cases in all "legal" architectures, and none among "illegal" ones. To get a list of them, just issue a query like as follows:

```
SELECT * FROM VHDL.SUBPROGRAM_DECLARATIONS A, VHDL.SUBPROGRAM_DECLARATIONS B
WHERE     A.PROJECT_NAME                 = B.PROJECT_NAME
AND       A.ENTITY_NAME                  = B.ENTITY_NAME
AND       A.ARCHITECTURE_NAME            = B.ARCHITECTURE_NAME
AND       A.PROCESS_NAME                 = B.PROCESS_NAME
AND       B.INSIDE_SUBPROGRAM            = A.SUBPROGRAM_NAME
AND       B.INSIDE_SUBPROGRAM_STARTLINE  = A.START_LINE
AND       A.PROCESS_NAME                 <> '(null)'
AND       ***;
```

To be exact, it must me said that the above query does not list only S_{P1} elements, but S_{P2} too. But since there are no S_{P2} nodes –and this is proven in the next paragraph– this simpler query will do as well.

14.1.7 S_{P2}

: As already anticipated, there are no S_{P2}-type subprograms, that is, subprograms directly included in S_{P1} nodes. Verify this claim by running the following query:

```
SELECT * FROM VHDL.SUBPROGRAM_DECLARATIONS A,
              VHDL.SUBPROGRAM_DECLARATIONS B,
              VHDL.SUBPROGRAM_DECLARATIONS C
WHERE  A.PROJECT_NAME    = B.PROJECT_NAME
AND    A.PROJECT_NAME    = C.PROJECT_NAME
AND    A.ENTITY_NAME     = B.ENTITY_NAME
AND    A.ENTITY_NAME     = C.ENTITY_NAME
AND    A.ARCHITECTURE_NAME = B.ARCHITECTURE_NAME
AND    A.ARCHITECTURE_NAME = C.ARCHITECTURE_NAME
AND    A.PROCESS_NAME    = B.PROCESS_NAME
AND    A.PROCESS_NAME    = C.PROCESS_NAME
AND    B.INSIDE_SUBPROGRAM = A.SUBPROGRAM_NAME
AND    C.INSIDE_SUBPROGRAM = B.SUBPROGRAM_NAME;
```

Again, the above query is even stronger than required: it does not only show that no S_{P2} elements exist, but also that no $S_{x_i} \longrightarrow S_{x_{i+1}} \longrightarrow S_{x_{i+2}}$ chains are possible, $\forall x, i$.

14.2 $E - A$ models

In this section we should propose models able to estimate N_a, which is the number of architectures inside a given bunch, i.e. the number of architectures per entity.

Gathered data show that the vast majority of entities have exactly one architecture; the average number of architectures per entity is 1.02004 (with a variance of 0.02177 and a standard deviation equal to 0.14756). Given the above findings, we will assume a trivial CGEAM model which identically returns 1 as an estimate.

14.3 $A - P$ models

Models described in this section to estimate the number of processes declared inside a given architecture. They are useful when a bunch size is to be estimated, and the number of processes belonging to one or more architectures contained in that bunch are not among available variables. Cumulative size due to processes declared in each architecture will be therefore estimated as the expected number of processes multiplied by their estimated average size.

14.3.1 Correlation study

Unlike syntax object models, bunch models do not estimate the length of a given object, instead they try to estimate how many sub-objects of a given kind are contained in the given object. In the case of CGAPM models, the variable to estimate is N_{pr}, that is, the number of processes contained in the given architecture.

The following table lists the statistical properties of all the available variables. Again, low coefficients of correlation between variables and N_{pr} are pretty common, thus implying that resulting estimation models will not exhibit exceptionally good accuracy.

Variable	Average value	Variance	Standard deviation	Correlation coefficient
		Externally available variables		
h_p	12.330	312.232	17.670	0.2613
n_p	10.889	191.527	13.839	0.3117
h_{ip}	7.285	88.448	9.405	0.2073
h_{op}	4.538	103.939	10.195	0.2408
h_{iop}	0.455	3.312	1.820	0.0413
h_{xp}	0.053	0.156	0.395	0.3456
n_{ip}	6.477	58.504	7.649	0.2555
n_{op}	3.879	56.667	7.528	0.2813
n_{iop}	0.481	3.302	1.817	0.0582
n_{xp}	0.052	0.152	0.390	0.3501
n_g	3.340	51.319	7.164	-0.0307
		Internally available variables		
n_s	32.054	30547.352	174.778	0.0648
h_s	21.295	8963.418	94.675	0.1137
n_{cd}	0.436	1.874	1.369	0.0860
n_{ci}	4.313	619.657	24.893	-0.0350
N_{pr}	1.511	619.657	24.893	(1.0000)

14.3.2 Model CGAPM1

Model CGAPM1 uses externally available information only and focuses on port count data per mode.

Model:

$$N_{pr} = k_{nip} \cdot n_{ip} + k_{nop} \cdot n_{op} + k_{niop} \cdot n_{iop} + k_{nxp} \cdot n_{xp} + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	1.608	1.608	0.000
Variance	4.992	0.724	4.268
Standard deviation	2.234	0.851	2.066
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	0.031	0.031	0.000
Variance	0.051	0.000	0.050
Standard deviation	0.225	0.020	0.224
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	1.351	1.351	0.000
Variance	7.344	1.050	6.294
Standard deviation	2.710	1.025	2.509

Correlation between estimated and real values:

	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient (L,$\hat{L}$)	0.3807	0.0885	0.3781

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_{nip}	-0.0281	-0.0011	0.0906
k_{nop}	0.0678	-0.0008	0.0885
k_{niop}	0.3101	-0.0362	0.0377
k_{nxp}	0.2926	-0.0422	1.1318
k_0	1.2345	0.0523	0.4210

Figure 14.2: Model CGAPM1: Real vs. estimated lines of code.

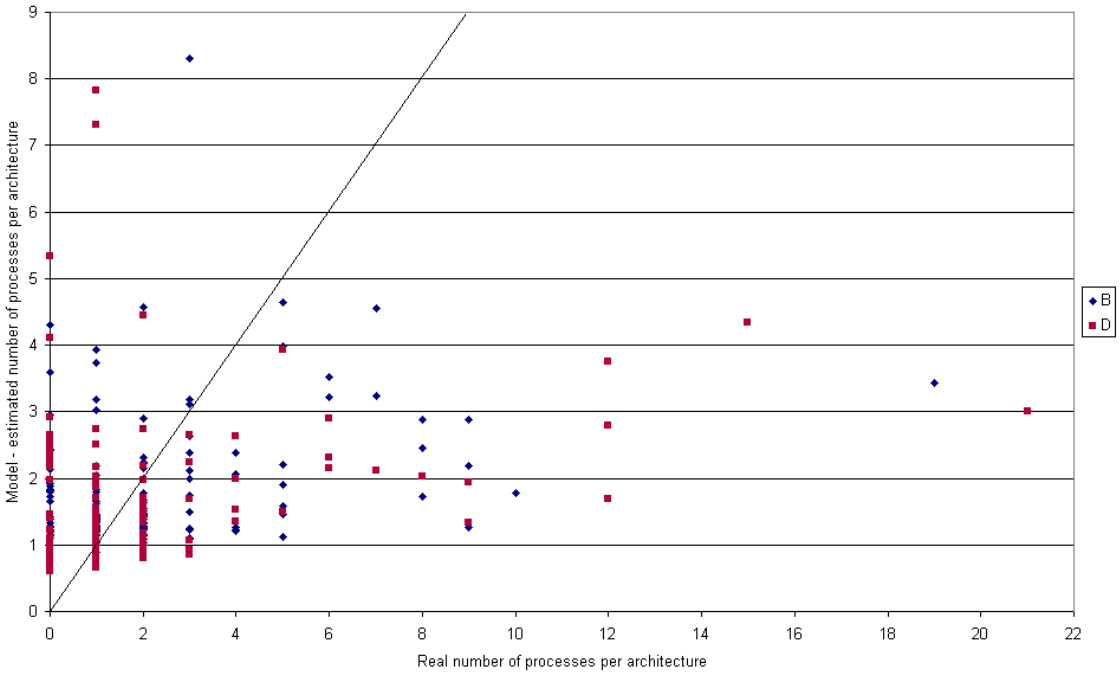


Figure 14.3: Models CGAPM1b, CGAPM1s, CGAPM1d: Real vs. estimated lines of code.

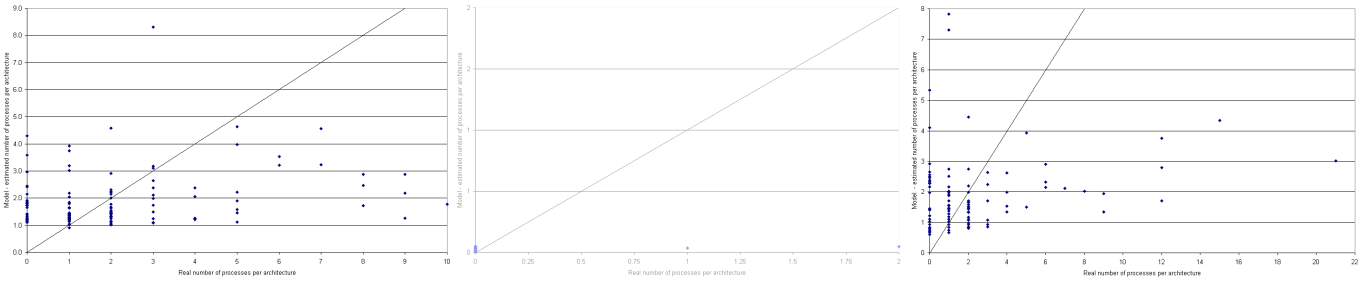


Figure 14.4: Model CGAPM1: Error density distribution.

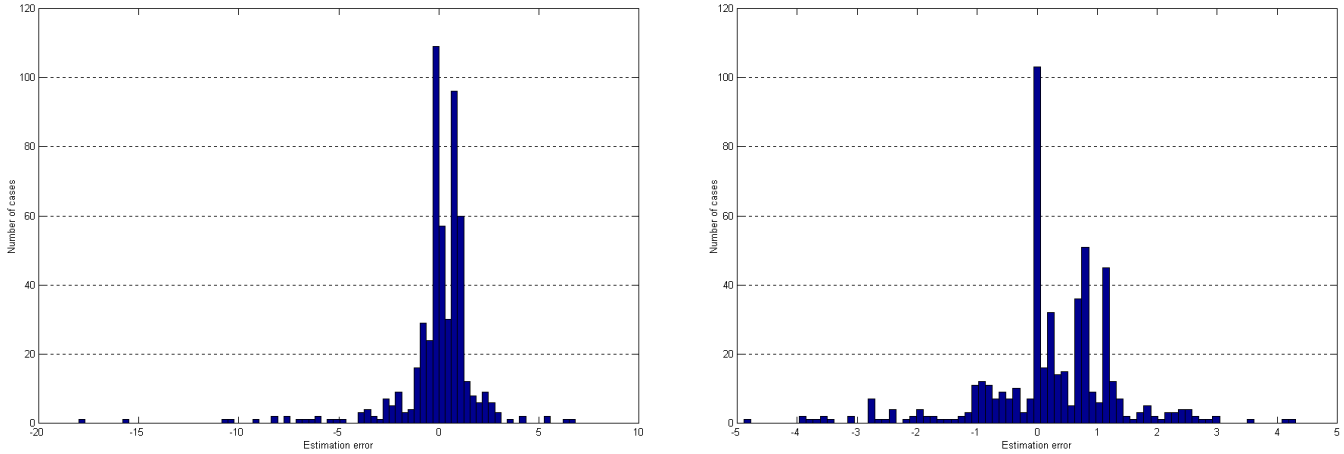


Figure 14.5: Model CGAPM1: Error cumulative distribution.

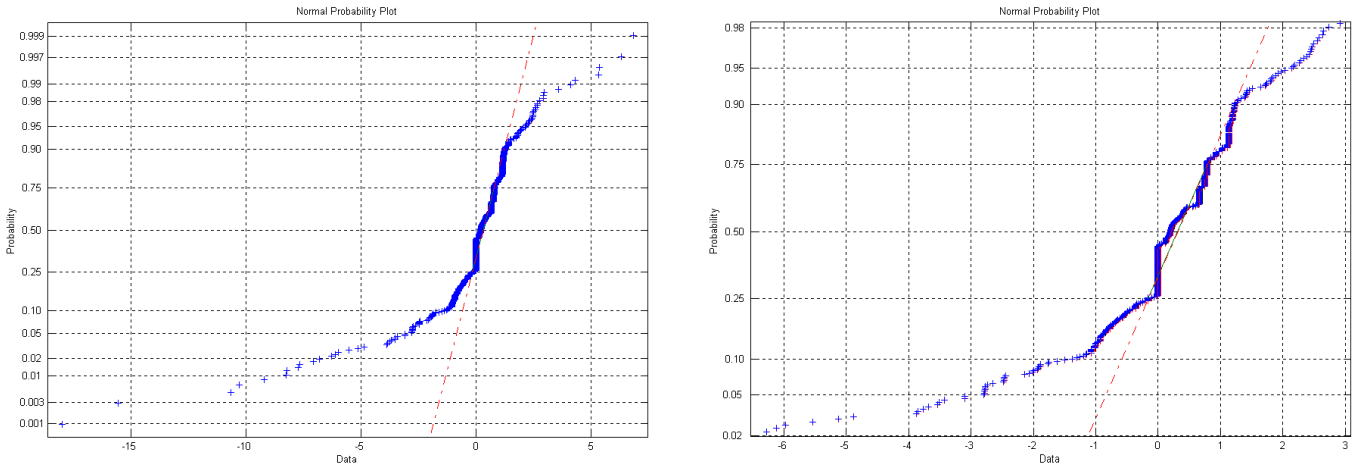
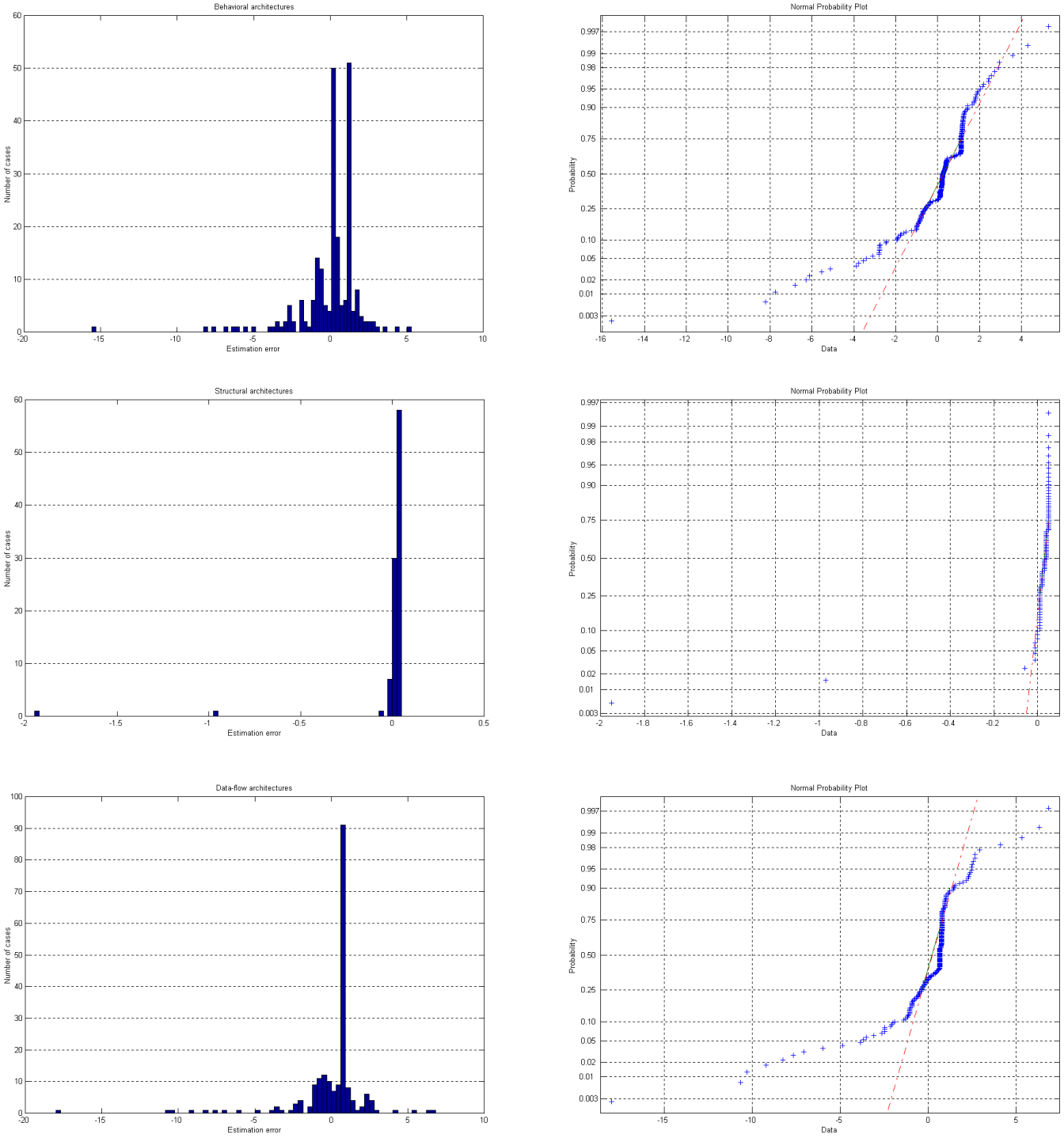


Figure 14.6: Models CGAPM1b, CGAPM1s, CGAPM1d: Error density and cumulative distributions.



14.3.3 Model CGAPM1H

Model CGAPM1H is mathematically structured as CGAPM1 except for the fact that homogeneity port data are used in place of simple port count data.

Model:

$$N_{pr} = k_{hip} \cdot h_{ip} + k_{hop} \cdot h_{op} + k_{hiop} \cdot h_{iop} + k_{hxp} \cdot h_{xp} + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	1.608	1.608	0.000
Variance	4.992	0.389	4.602
Standard deviation	2.234	0.624	2.145
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	0.031	0.031	0.000
Variance	0.051	0.000	0.050
Standard deviation	0.225	0.015	0.224
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	1.351	1.351	0.000
Variance	7.344	0.864	6.479
Standard deviation	2.710	0.930	2.545

Correlation between estimated and real values:

	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient (L,$\hat{L}$)	0.2793	0.0652	0.3431

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_{nip}	-0.0324	0.001	0.005
k_{nop}	0.0614	-0.001	0.060
k_{niop}	0.2280	-0.025	0.186
k_{nxp}	0.2493	-0.035	0.300
k_0	1.3815	0.035	0.909

Figure 14.7: Model CGAPM1H: Real vs. estimated lines of code.

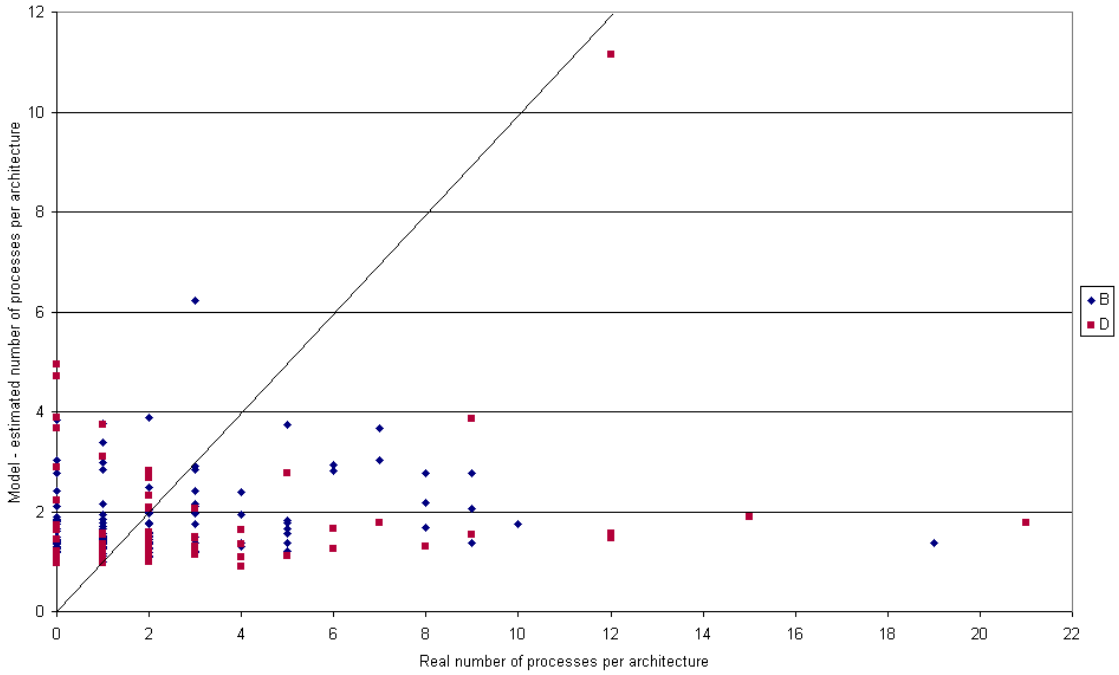


Figure 14.8: Models CGAPM1Hb, CGAPM1Hs, CGAPM1Hd: Real vs. estimated lines of code.

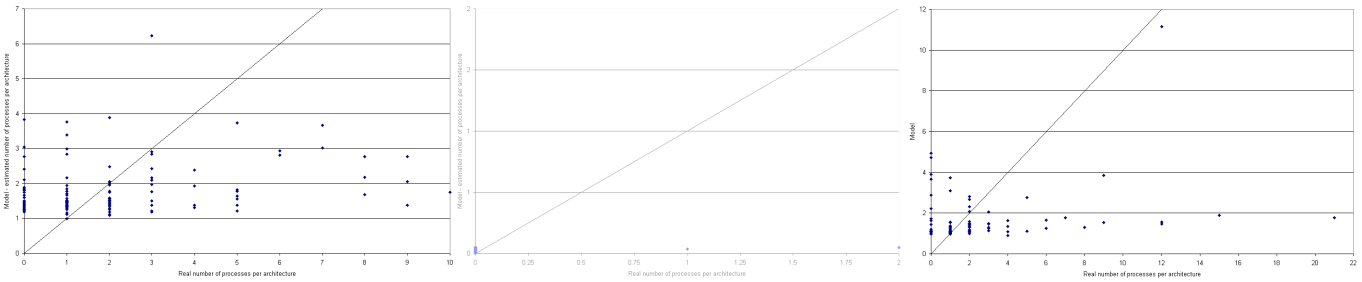


Figure 14.9: Model CGAPM1H: Error density distribution.

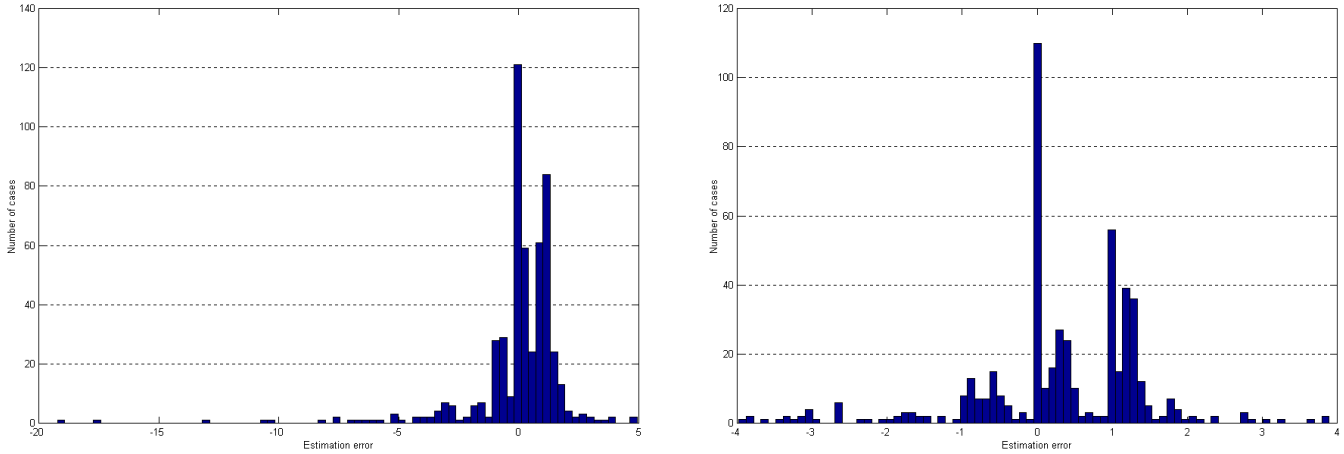


Figure 14.10: Model CGAPM1H: Error cumulative distribution.

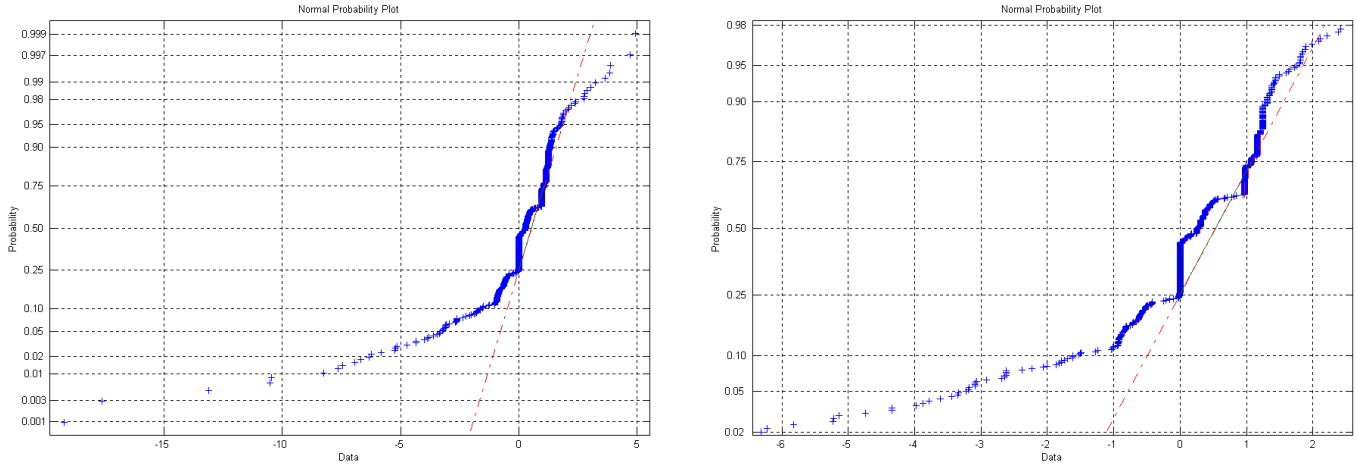
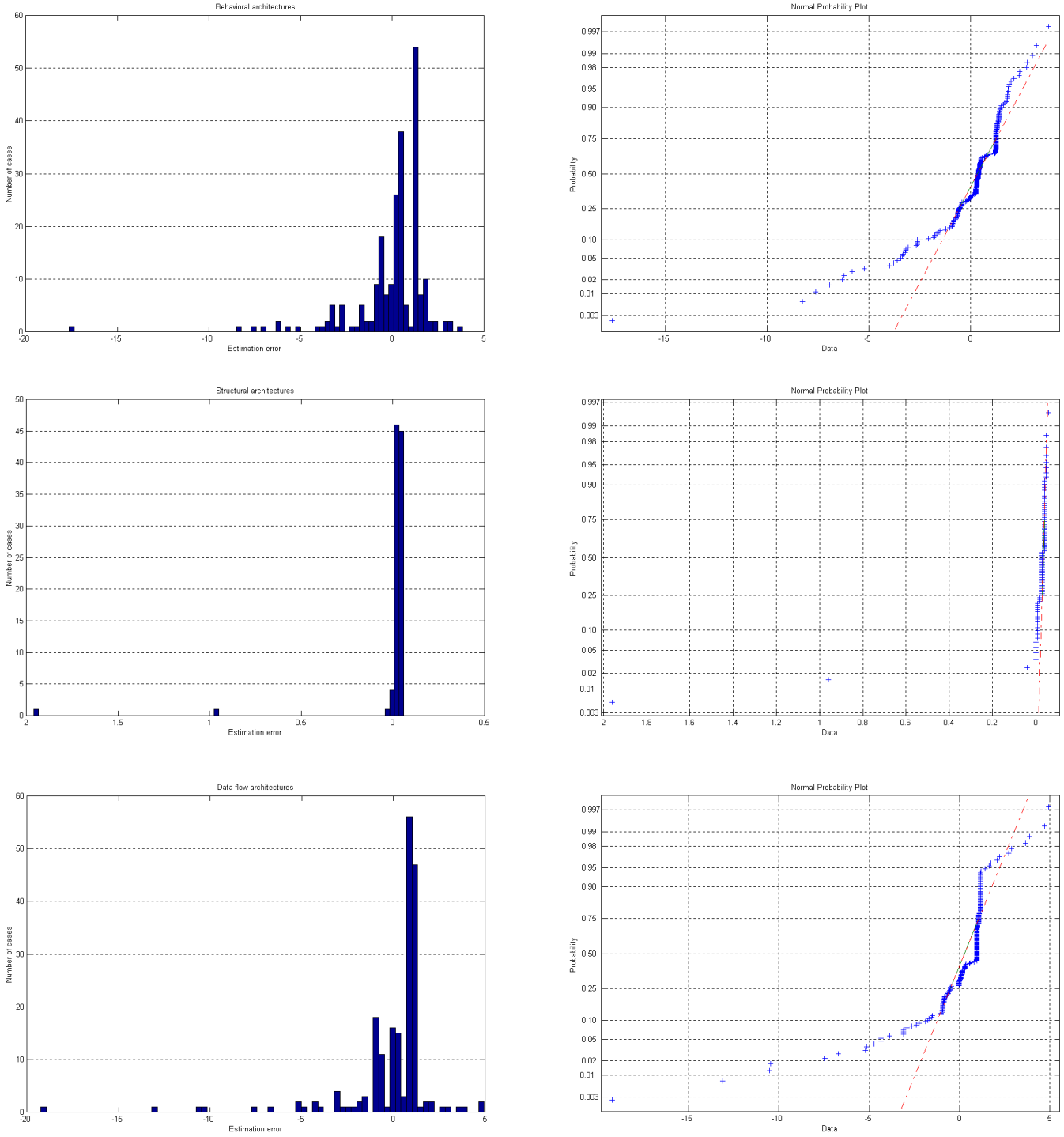


Figure 14.11: Models CGAPM1Hb, CGAPM1Hs, CGAPM1Hd: Error density and cumulative distributions.



14.3.4 Conclusions

Since the coefficients of correlation between internally available variables and N_{pr} are poor, we hold the reasonable belief that a model designed to take them into account would not make any better that already examined ones. Because of these motivations, no such models are presented in this section.

As far as the choice of the best model is concerned, we must say that CGAPM1 is better that CGAPM1H under all possible points of view, both when correlations are examined and when estimation error variance is considered.

Sub-models CGAPM1b and CGAPM1d show a correlation coefficient between N_{pr} and $\hat{N}_{pr}$ that is acceptable, and an error which falls between -3.5 and +2.5 units in 93% of the cases.

As far as sub-model CGAPM1s is concerned, since statistical properties of the analyzed population show that structural architectures containing processes are a negligible minority, we will replace that model with a trivial constant model, always returning a $\hat{N}_{pr}$ equal 0.

14.4 $A - C_D$ models

Models described in this section try to estimate the number of component declarations appearing inside a given architecture. Such models are useful when a bunch size is to be estimated, and the number of component declarations belonging to one or more architectures of the current bunch are not among available variables.

Cumulative size due to component declarations included in each architecture will be therefore estimated as the expected number of component declarations multiplied by their estimated average size.

14.4.1 Correlation study

CGACDM models try to estimate N_{cd} that is, the number of component declarations per architecture. The following table lists statistical properties of all the available variables. Coefficients of correlation between variables and N_{cd} are fairly good, thus suggesting that resulting models will show reasonably good estimation accuracy.

Variable	Average value	Variance	Standard deviation	Correlation coefficient
h_p	12.124	310.404	17.618	0.4174
n_p	10.750	195.606	13.986	0.4720
h_{ip}	7.131	85.513	9.247	0.3182
h_{op}	4.424	99.265	9.963	0.3954
h_{iop}	0.519	8.584	2.930	0.1385
h_{xp}	0.050	0.148	0.384	0.1721
n_{ip}	6.358	56.926	7.545	0.3662
n_{op}	3.799	54.356	7.373	0.4551
n_{iop}	0.544	8.570	2.927	0.1426
n_{xp}	0.049	0.144	0.380	0.1751
n_g	3.333	49.669	7.048	-0.0015
n_s	31.968	30853.192	175.651	0.2704
n_{pr}	1.456	19.946	4.466	0.0879
n_{ci}	4.542	747.412	27.339	0.3397
h_s	21.820	10404.069	102.000	0.3447
N_{cd}	0.418	747.412	27.339	(1.0000)

14.4.2 Model CGACDM1

Model CGACDM1 uses externally available information only, and focuses on port count data per mode.

Model:

$$N_{cd} = k_{nip} \cdot n_{ip} + k_{nop} \cdot n_{op} + k_{niop} \cdot n_{iop} + k_{nxp} \cdot n_{xp} + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	0.4537	0.4537	0.0000
Variance	0.9835	0.1698	0.8136
Standard deviation	0.9917	0.4121	0.9020
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	1.0612	1.0612	0.0000
Variance	4.4911	1.7004	2.7907
Standard deviation	2.1192	1.3040	1.6705
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	0.2624	0.2624	0.0000
Variance	1.3189	0.1650	1.1539
Standard deviation	1.1484	0.4061	1.0742

Correlation between estimated and real values:

	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient $(L,\hat{L})$	0.4156	0.6153	0.3537

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_{nip}	-0.0665	0.0800	0.0393
k_{nop}	0.0623	0.0660	0.0508
k_{niop}	0.0992	2.0416	0.0878
k_{nxp}	-0.0959	-0.3800	0.0411
k_0	0.5537	-0.3708	-0.1251

Figure 14.12: Model CGACDM1: Real vs. estimated lines of code.

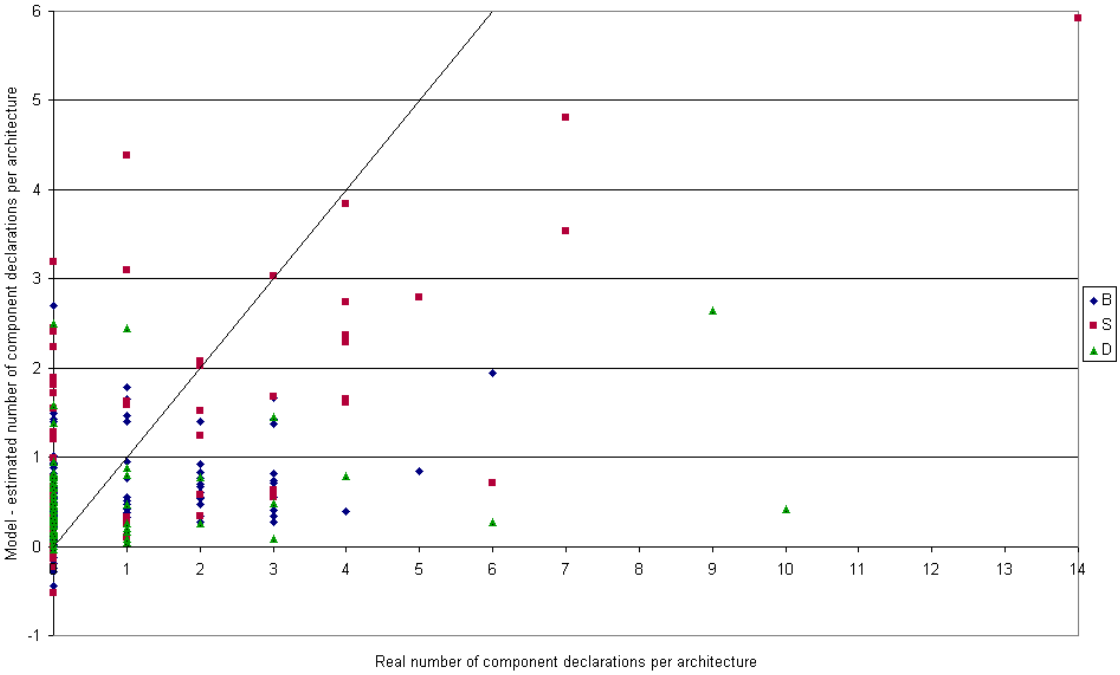


Figure 14.13: Models CGACDM1b, CGACDM1s, CGACDM1d: Real vs. estimated lines of code.

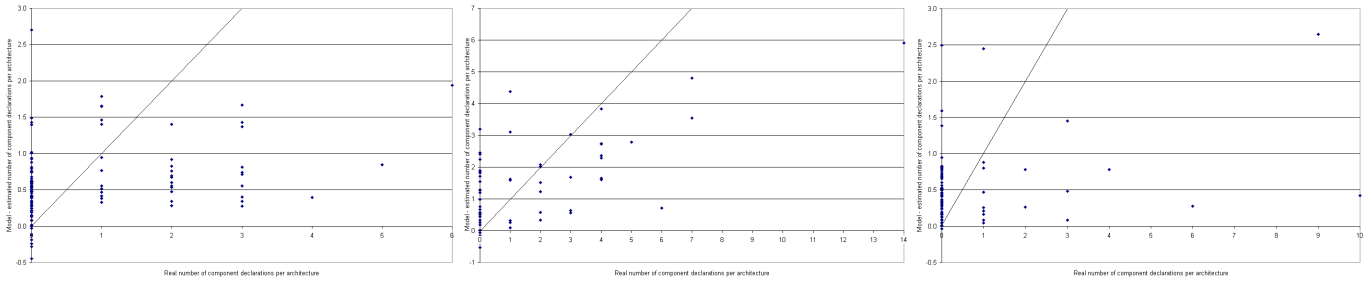


Figure 14.14: Model CGACDM1: Error density distribution.

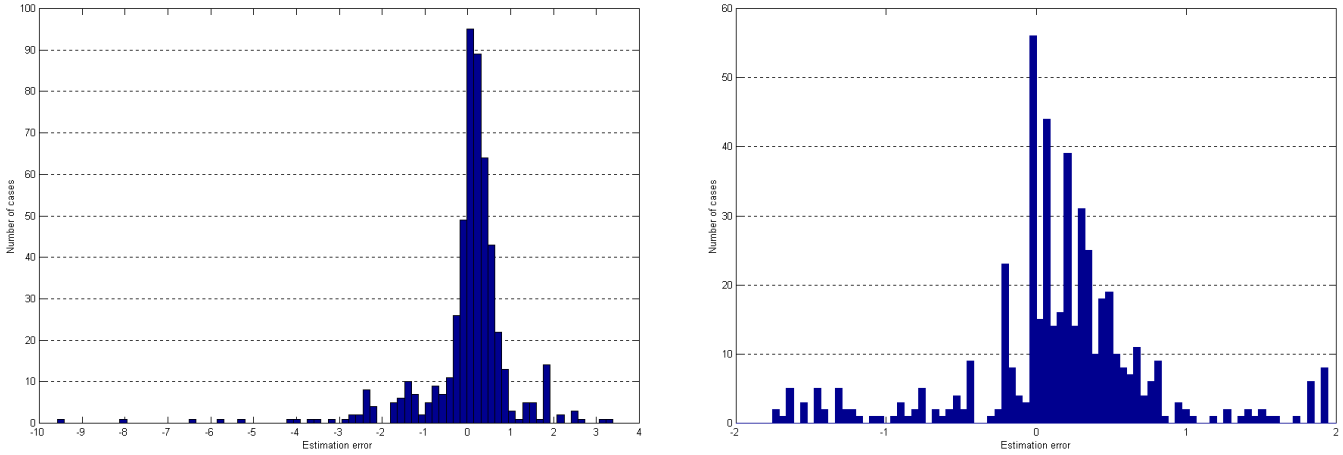


Figure 14.15: Model CGACDM1: Error cumulative distribution.

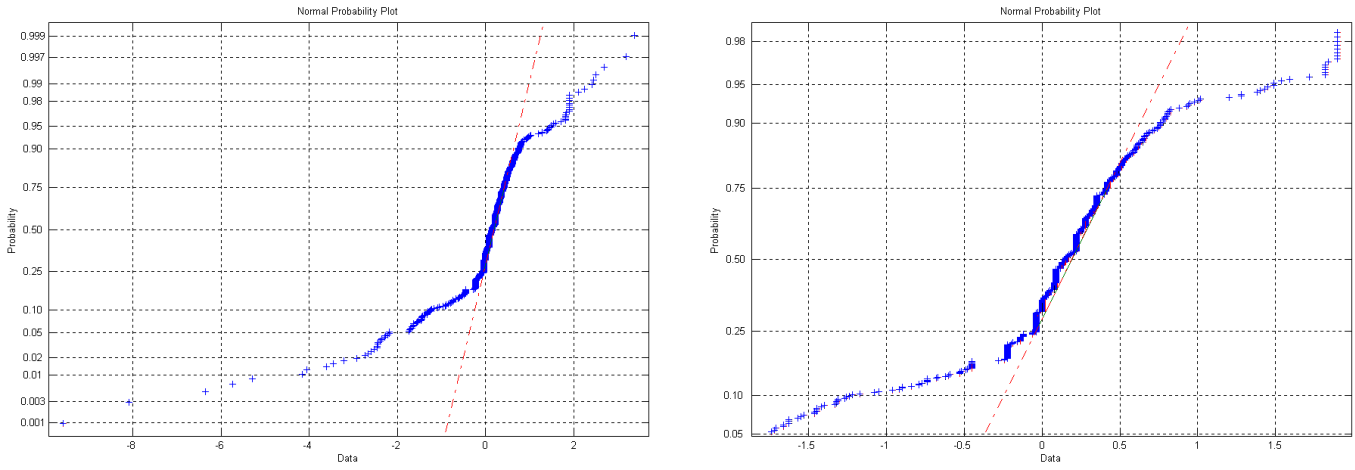
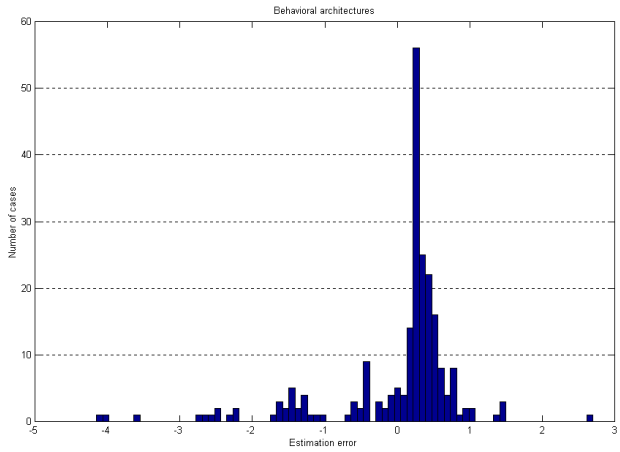
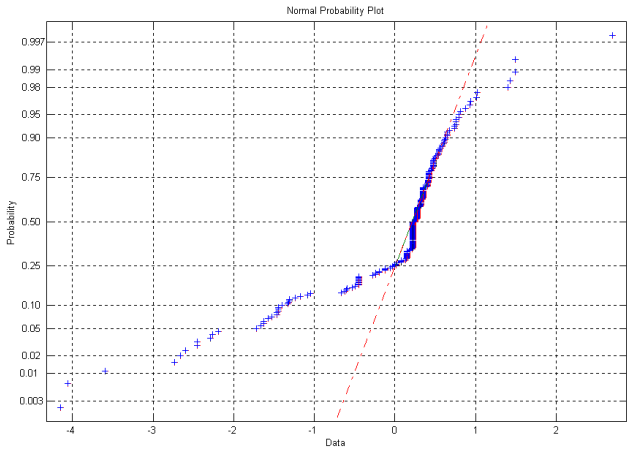
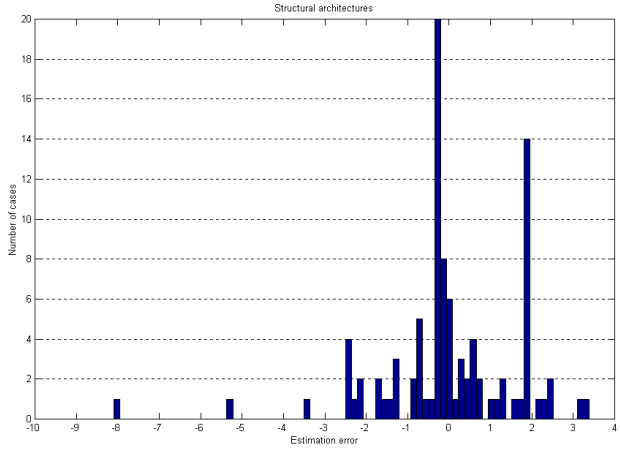
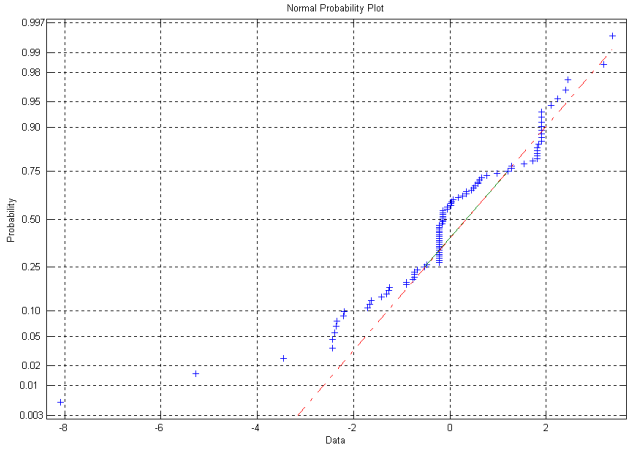


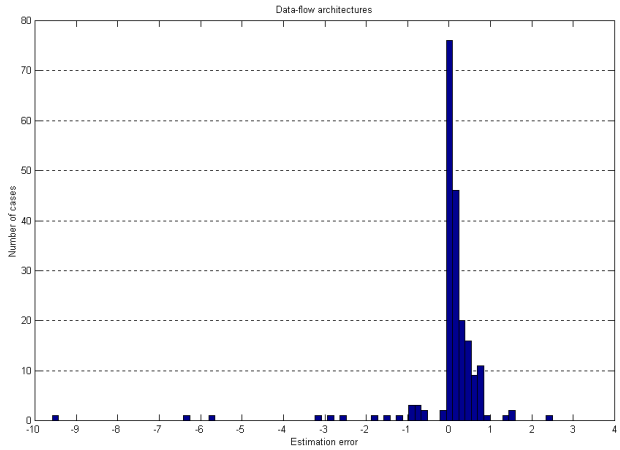
Figure 14.16: Models CGACDM1b, CGACDM1s, CGACDM1d: Error density and cumulative distributions.

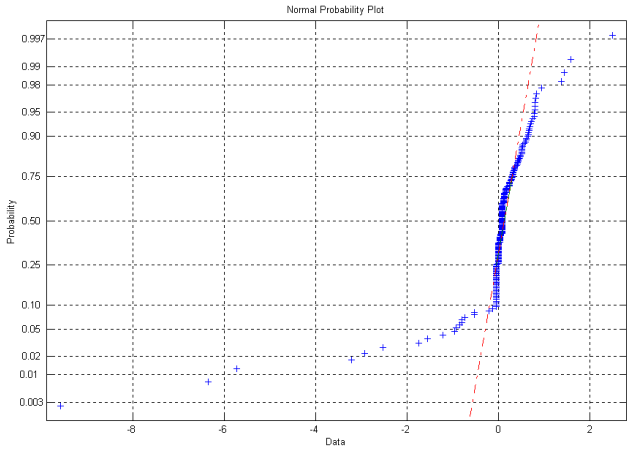












14.4.3 Model CGACDM1H

Model CGACDM1H is mathematically structured as CGACDM1 except for the fact that homogeneity port data are used in place of simple port count data.

Model:

$$N_{cd} = k_{hip} \cdot h_{ip} + k_{hop} \cdot h_{op} + k_{hiop} \cdot h_{iop} + k_{hxp} \cdot h_{xp} + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	0.4537	0.4537	0.0000
Variance	0.9835	0.1649	0.8186
Standard deviation	0.9917	0.4061	0.9048
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	1.0612	1.0612	0.0000
Variance	4.4911	1.1806	3.3105
Standard deviation	2.1192	1.0865	1.8195
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	0.2624	0.2624	0.0000
Variance	1.3189	0.1963	1.1225
Standard deviation	1.1484	0.4431	1.0595

Correlation between estimated and real values:

	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient (L,$\hat{L}$)	0.4095	0.5127	0.3858

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_{nip}	-0.0658	0.0663	0.0089
k_{nop}	0.0616	0.0237	0.0258
k_{niop}	0.0931	1.8692	0.0969
k_{nxp}	-0.0990	-0.4377	-0.2324
k_0	0.5600	-0.0764	0.0225

Figure 14.17: Model CGACDM1H: Real vs. estimated lines of code.

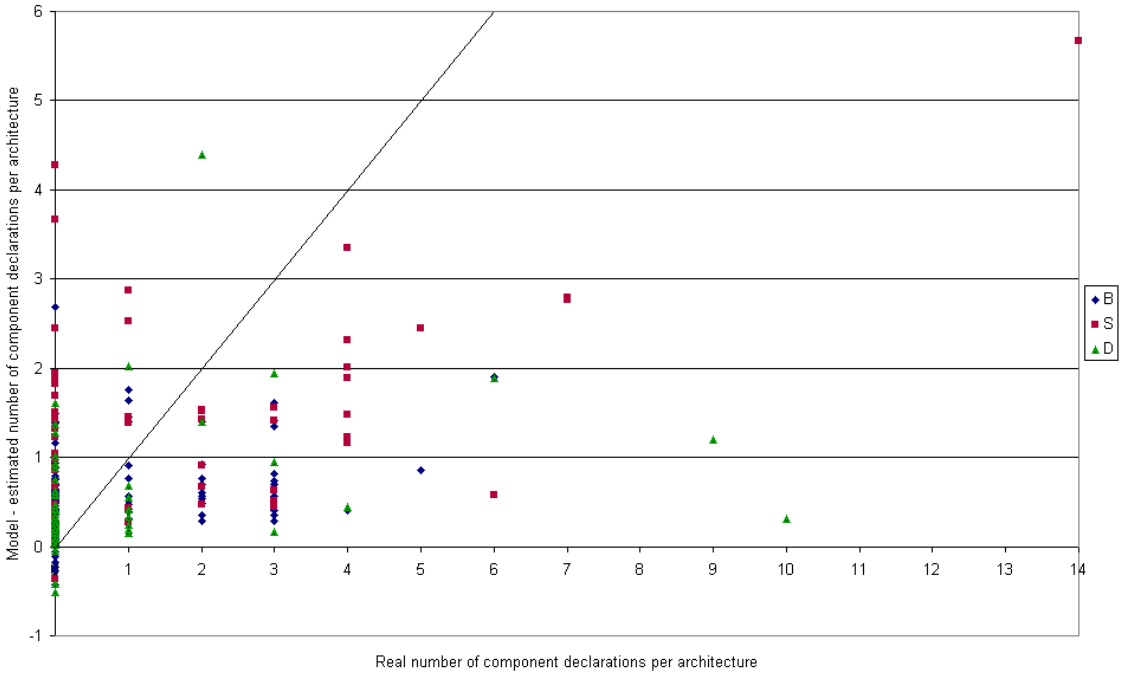


Figure 14.18: Models CGACDM1Hb, CGACDM1Hs, CGACDM1Hd: Real vs. estimated lines of code.

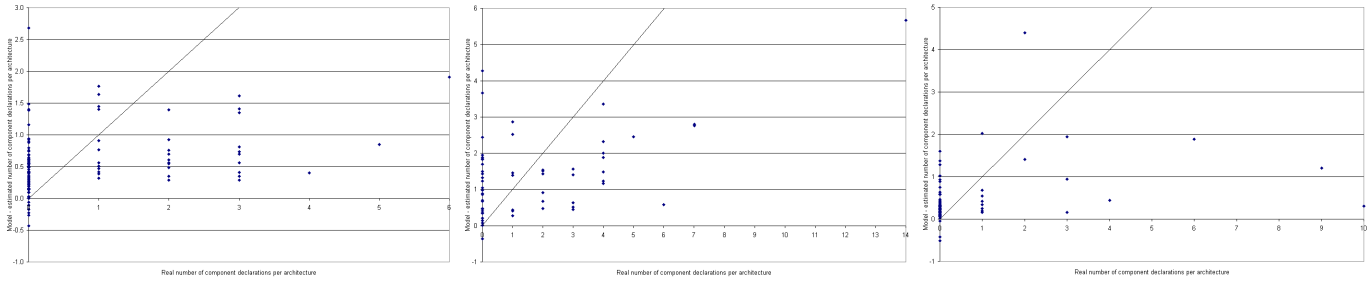


Figure 14.19: Model CGACDM1H: Error density distribution.

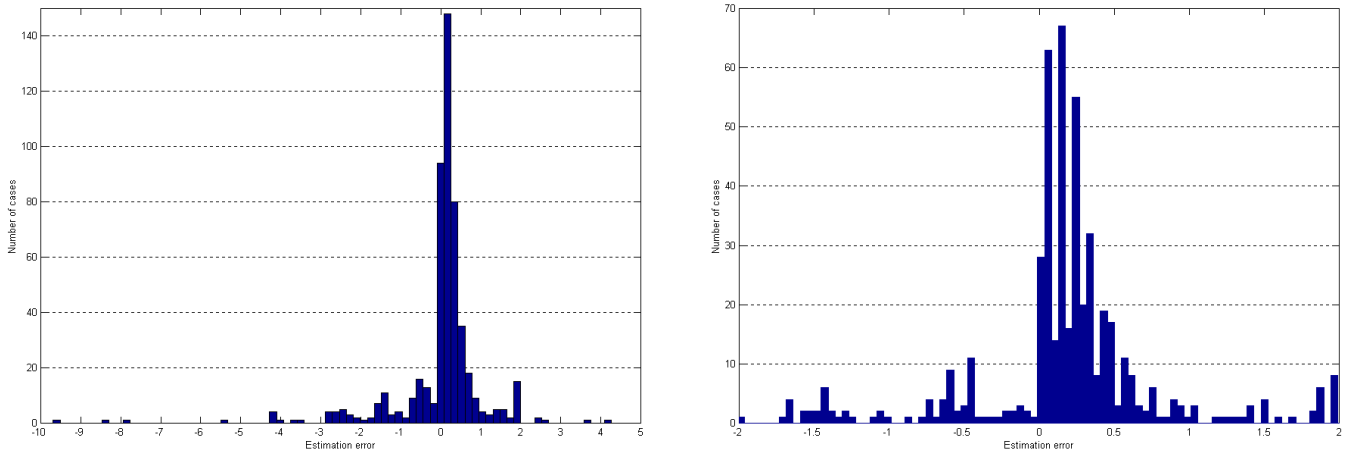


Figure 14.20: Model CGACDM1H: Error cumulative distribution.

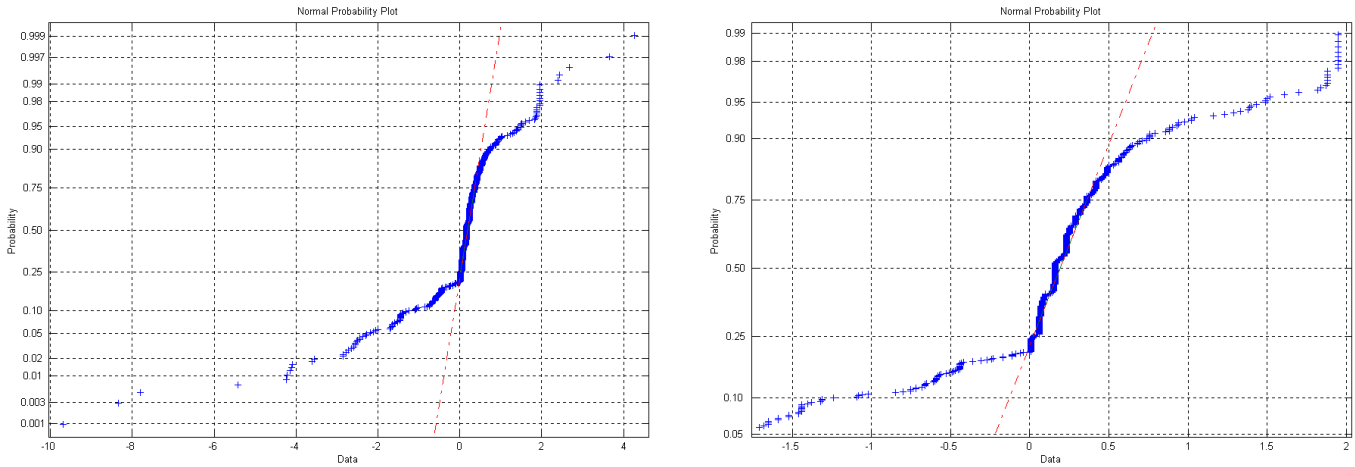
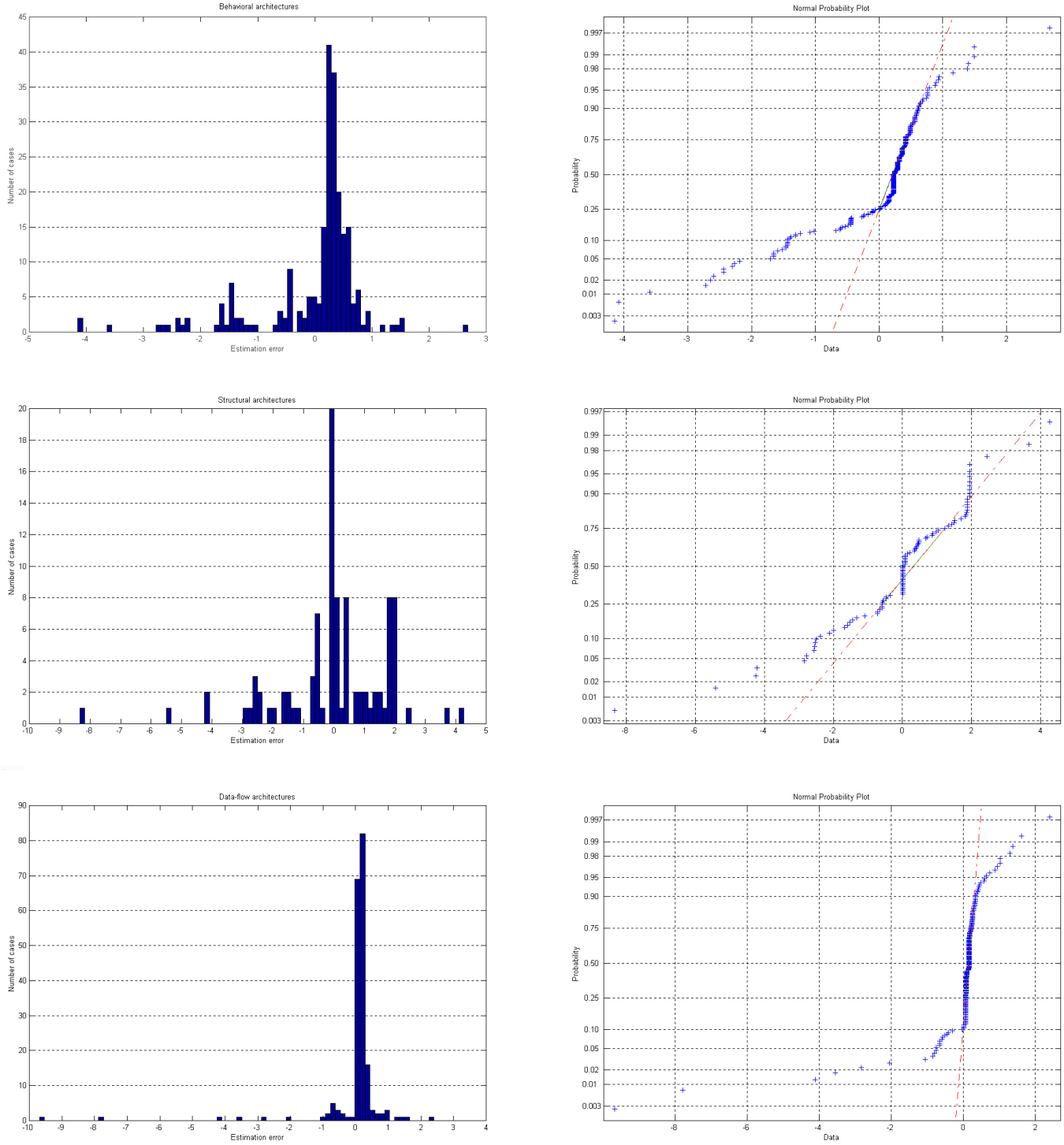


Figure 14.21: Models CGACDM1Hb, CGACDM1Hs, CGACDM1Hd: Error density and cumulative distributions.



14.4.4 Conclusions

As far as the choice of the best model is concerned, models CGACDM1 and CGACDM1H are practically indistinguishable from each other, under all possible points of view; CGACDM1 shows an hardly appreciable superiority, especially when behavioral architectures are considered, and it therefore our preferred choice in the next model aggregates.

14.5 $A - C_I$ models

Models described in this section try to estimate the number of component instantiations appearing inside a given architecture. Such models come handy when a bunch size estimated is desired, and the actual number of component instantiations belonging to one or more architectures of the current bunch are not known.

In such cases, size due to component instantiations, included in each architecture will be calculated as the product of number of component instantiations (estimated with the following numbers) multiplied by their expected average size.

14.5.1 Correlation study

CGACIM models try to estimate N_{ci} that is, the number of component instantiations per architecture. The following table lists statistical properties of all the available variables. Coefficients of correlation between variables and N_{ci} are acceptable, thus suggesting that resulting models will show reasonably good estimation accuracy.

Variable	Average value	Variance	Standard deviation	Correlation coefficient
		Externally available variables		
h_p	12.252	312.332	17.673	0.2535
n_p	10.784	189.036	13.749	0.3236
h_{ip}	7.297	89.517	9.461	0.2866
h_{op}	4.456	103.128	10.155	0.1684
h_{iop}	0.446	3.342	1.828	0.0254
h_{xp}	0.054	0.159	0.398	0.0296
n_{ip}	6.474	59.003	7.681	0.3566
n_{op}	3.785	54.828	7.405	0.2234
n_{iop}	0.472	3.332	1.825	0.0240
n_{xp}	0.052	0.155	0.394	0.0309
n_g	3.357	52.032	7.213	-0.1006
n_s	26.005	23912.800	154.638	0.0394
h_s	15.154	2053.717	45.318	0.1732
n_{pr}	1.526	21.332	4.619	-0.0322
n_{cd}	0.411	1.758	1.326	0.2243
N_{ci}	2.561	80.421	8.968	(1.0000)

14.5.2 Model CGACIM1

Model CGACIM1 uses externally available information only and focuses on port count data per mode.

Model:

$$N_{ci} = k_{nip} \cdot n_{ip} + k_{nop} \cdot n_{op} + k_{niop} \cdot n_{iop} + k_{nxp} \cdot n_{xp} + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	0.780	0.780	0.000
Variance	1.642	0.076	1.565
Standard deviation	1.281	0.276	1.251
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	14.378	14.378	0.000
Variance	439.042	181.196	257.846
Standard deviation	20.953	13.461	16.058
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	1.406	1.406	0.000
Variance	15.257	7.135	8.123
Standard deviation	3.906	2.671	2.850

Correlation between estimated and real values:

	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient (L,$\hat{L}$)	0.2154	0.6424	0.6838

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_{nip}	-0.0395	0.9512	0.0342
k_{nop}	0.0414	0.5739	0.0236
k_{niop}	0.0687	1.0747	1.9066
k_{nxp}	-0.2187	28.2954	-0.7187
k_0	0.8144	1.5134	0.2644

Figure 14.22: Model CGACIM1: Real vs. estimated lines of code.

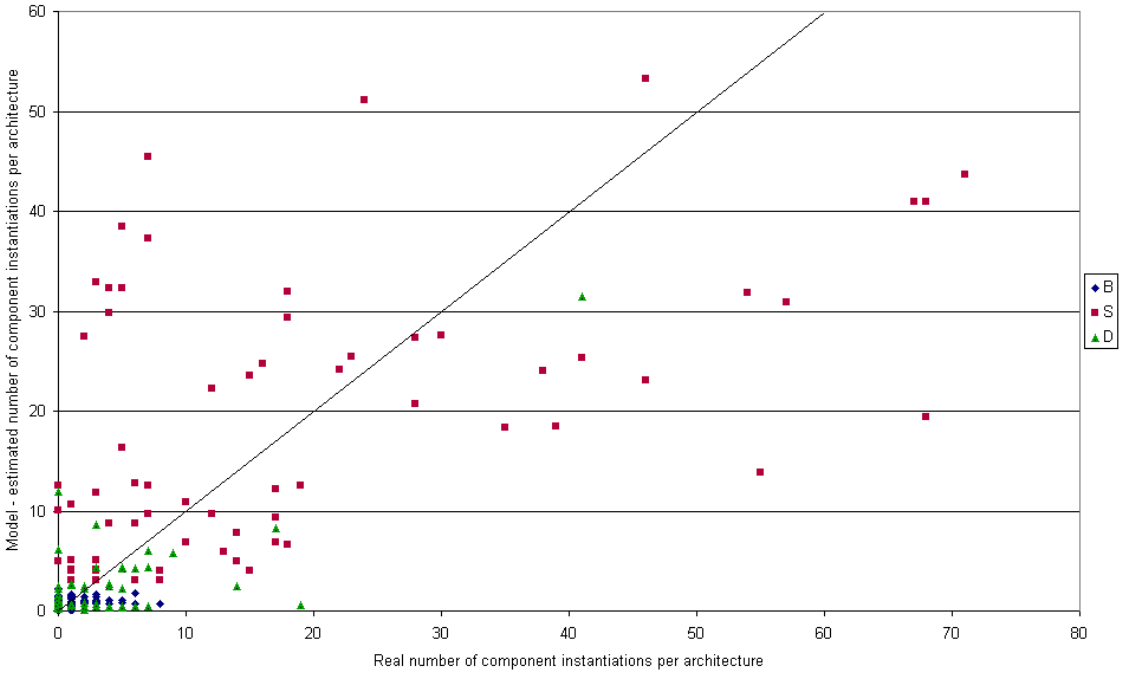


Figure 14.23: Models CGACIM1b, CGACIM1s, CGACIM1d: Real vs. estimated lines of code.

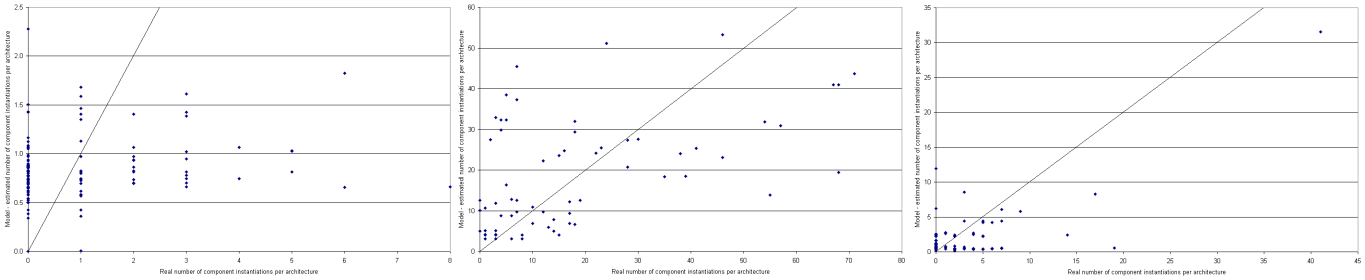


Figure 14.24: Model CGACIM1: Error density distribution.

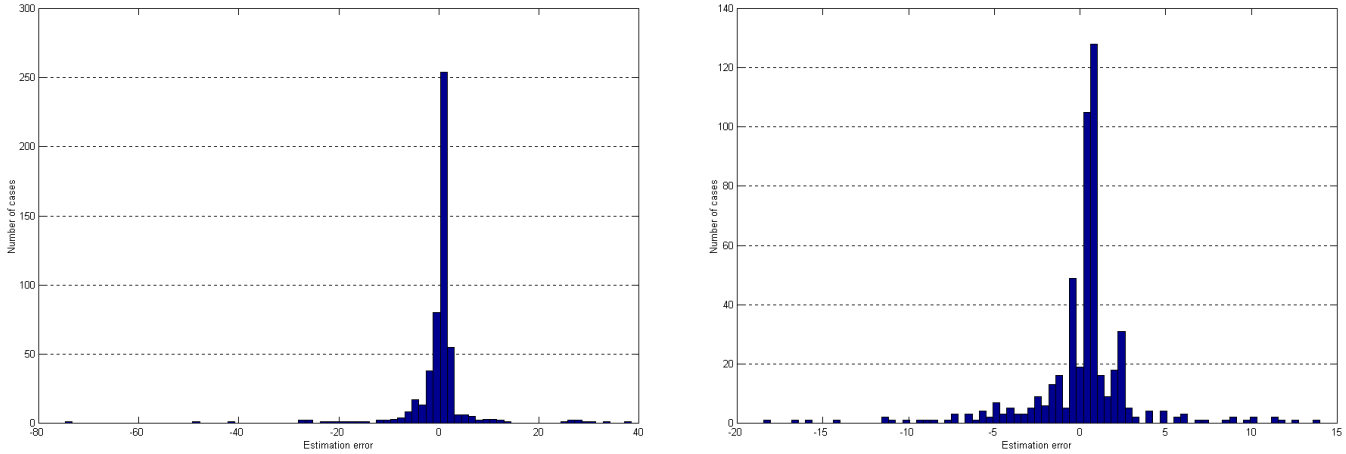


Figure 14.25: Model CGACIM1: Error cumulative distribution.

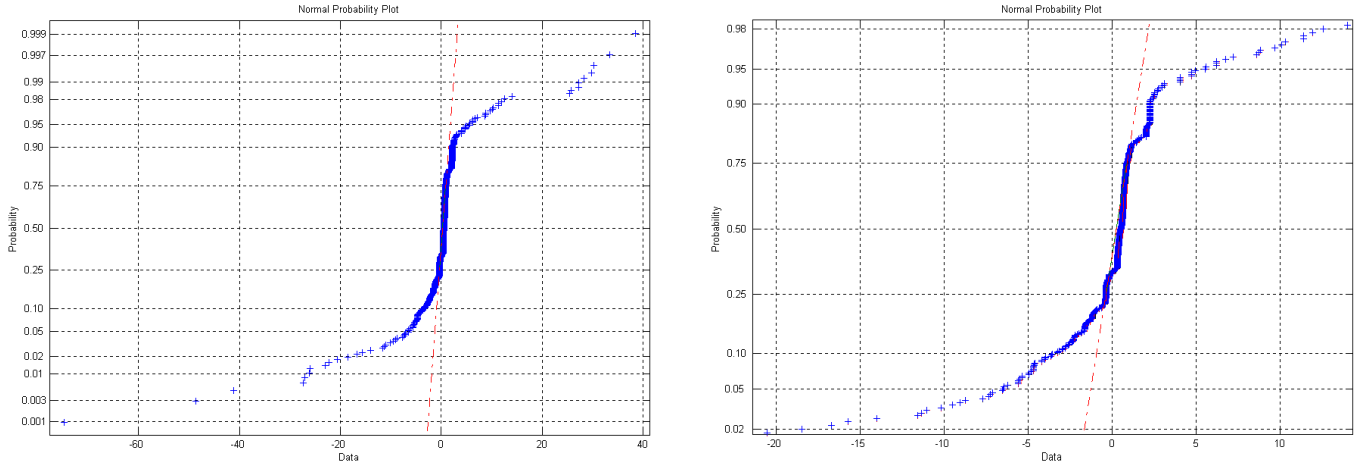
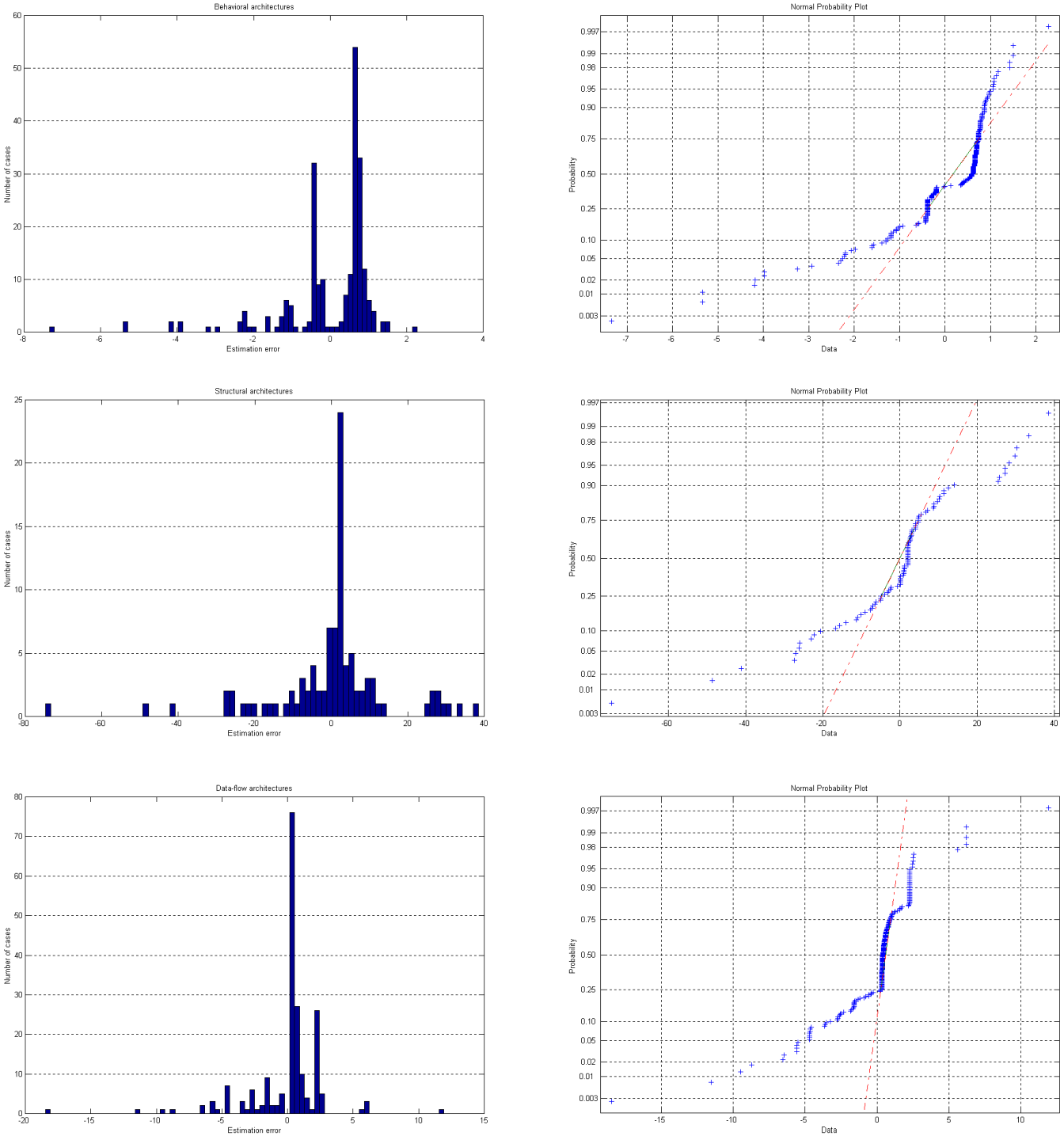


Figure 14.26: Models CGACIM1b, CGACIM1s, CGACIM1d: Error density and cumulative distributions.



14.5.3 Model CGACIM1H

Model CGACIM1H is mathematically structured as CGACIM1 except for the fact that homogeneity port data are used in place of simple port count data.

Model:

$$N_{ci} = k_{hip} \cdot h_{ip} + k_{hop} \cdot h_{op} + k_{hiop} \cdot h_{iop} + k_{hxp} \cdot h_{xp} + k_0$$

Population statistical properties and model accuracy:

	Behavioral architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	0.4537	0.4537	0.0000
Variance	0.9835	0.1649	0.8186
Standard deviation	0.9917	0.4061	0.9048
	Structural architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	1.0612	1.0612	0.0000
Variance	4.4911	1.1806	3.3105
Standard deviation	2.1192	1.0865	1.8195
	Data-flow architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	0.2624	0.2624	0.0000
Variance	1.3189	0.1963	1.1225
Standard deviation	1.1484	0.4431	1.0595

Correlation between estimated and real values:

	Behavioral architectures	Structural architectures	Data-flow architectures
Correlation coefficient (L,$\hat{L}$)	0.4095	0.5127	0.3858

Identified model coefficients:

	Behavioral architectures	Structural architectures	Data-flow architectures
Coefficient		Value	
k_{nip}	-0.0658	0.0663	0.0089
k_{nop}	0.0616	0.0237	0.0258
k_{niop}	0.0931	1.8692	0.0969
k_{nxp}	-0.0990	-0.4377	-0.2324
k_0	0.5600	-0.0764	0.0225

Figure 14.27: Model CGACIM1H: Real vs. estimated lines of code.

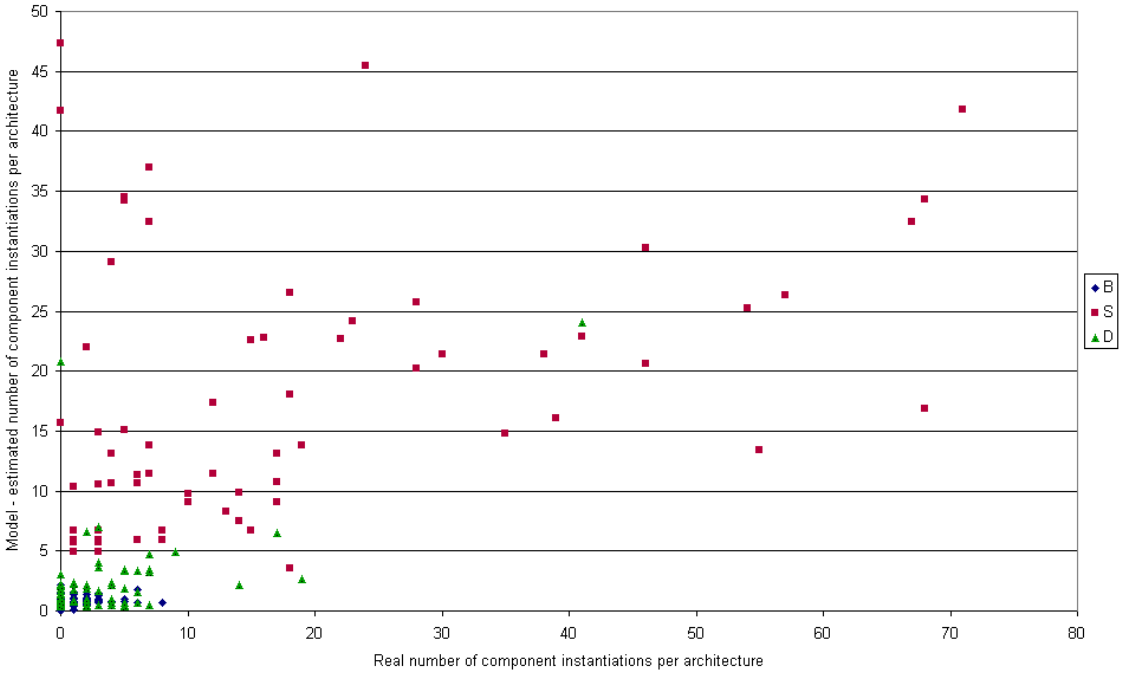


Figure 14.28: Models CGACIM1Hb, CGACIM1Hs, CGACIM1Hd: Real vs. estimated lines of code.

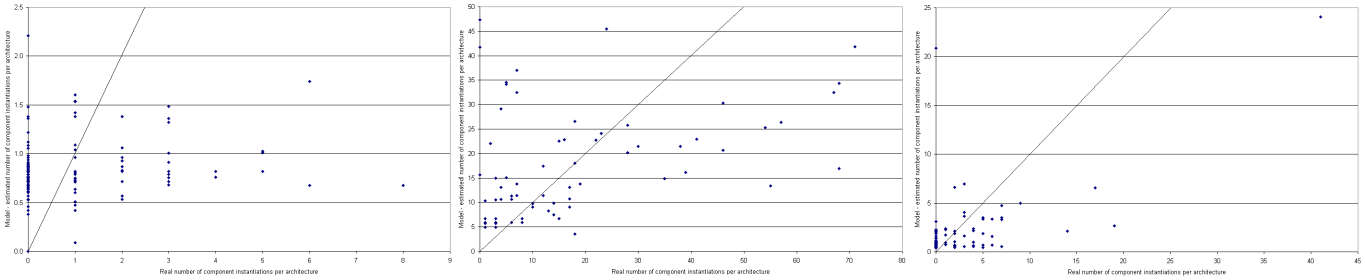


Figure 14.29: Model CGACIM1H: Error density distribution.

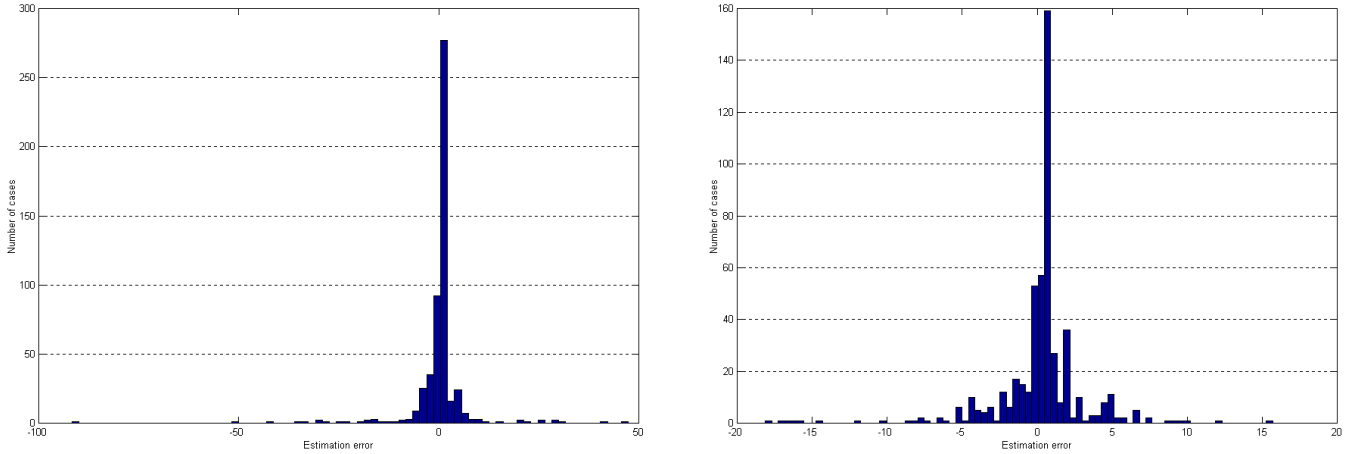


Figure 14.30: Model CGACIM1H: Error cumulative distribution.

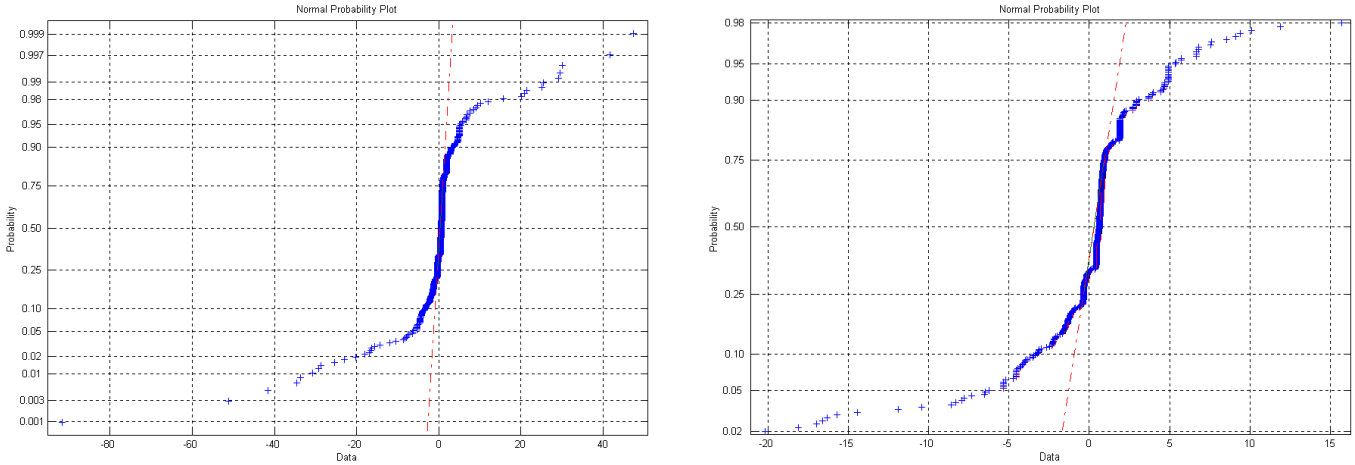
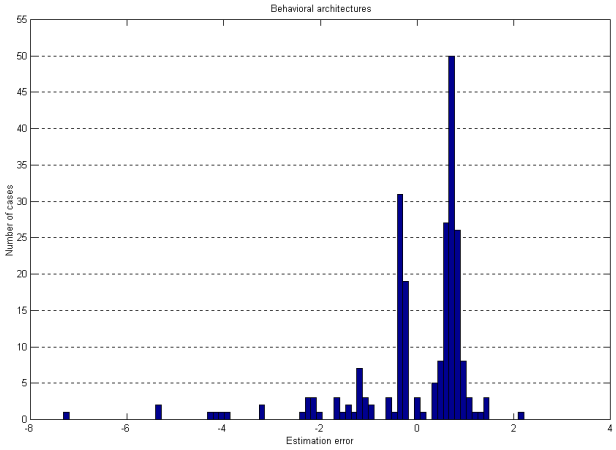
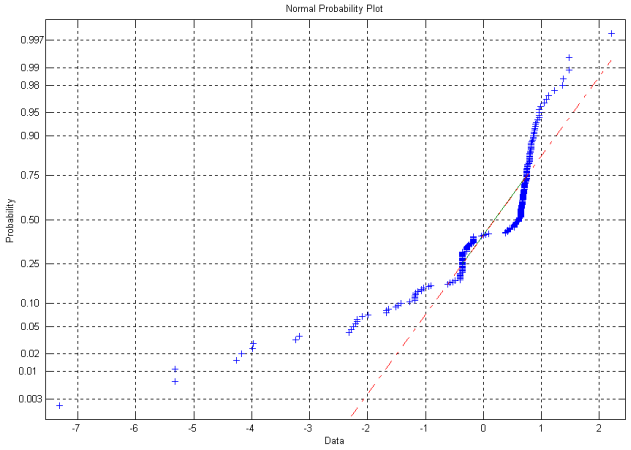
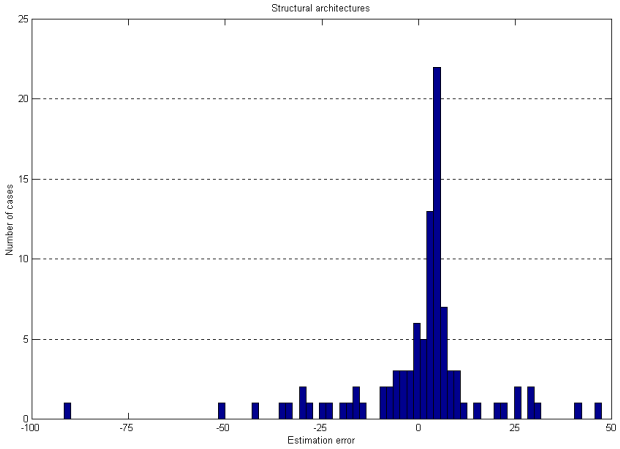
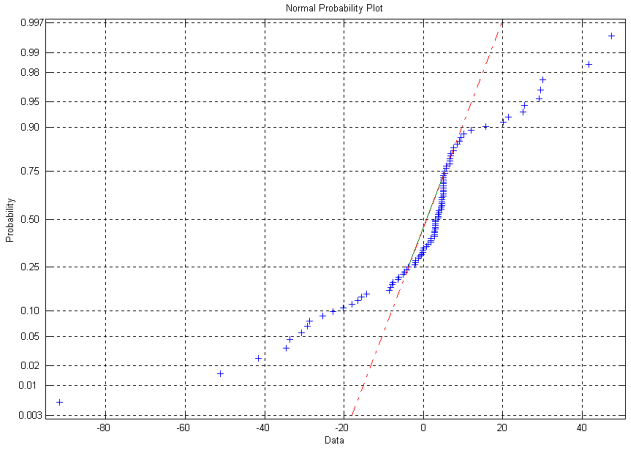


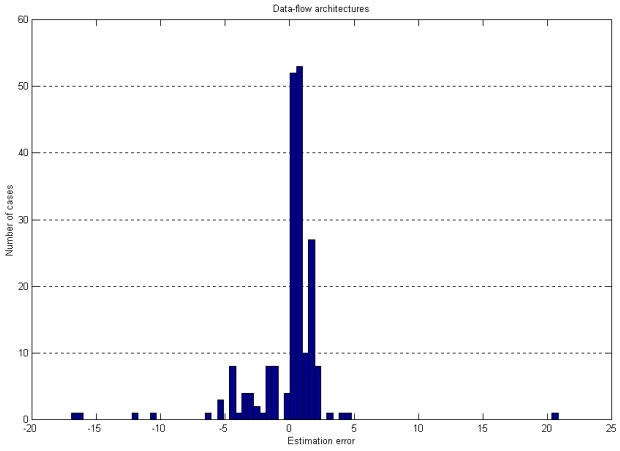
Figure 14.31: Models CGACIM1Hb, CGACIM1Hs, CGACIM1Hd: Error density and cumulative distributions.

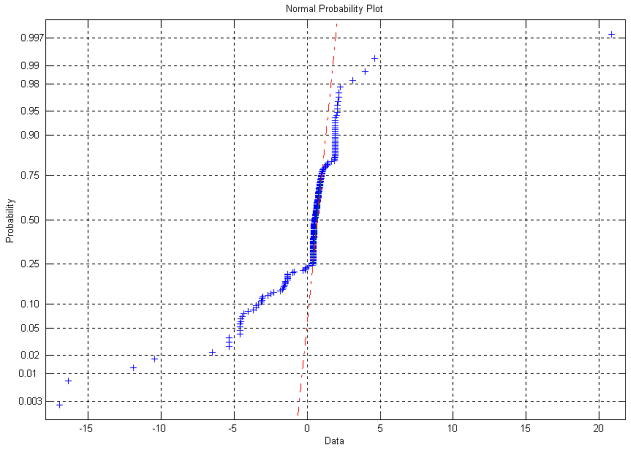












14.5.4 Conclusions

Models CGACIM1 is by far more accurate than CGACIM1H, therefore it will be our preferred choice for the construction of model aggregates to which following chapters are dedicated. N_{ci} tends to be estimated by model CGACIM1 in a quite accurate way. The estimation error falls between -7 and +5 units in the 90% of the internal validation cases.

14.6 Other models

We believe that adding models designed to estimate the length of subprograms and their cardinality inside bunches would not add any conceptual innovation with respect to the rest of this thesis. Moreover, our experimentally gathered data show that the impact of subprograms onto the overall size of each project is negligible. Therefore, creation of such models is left to future developments.

14.7 Bunch-level model validation

In the following pages, models presented so far (both syntax object models and bunch models) are validated. Validation is performed on the tuning set (that is, bunches belonging to one of the 41 tuning projects; we will call this operation *internal* validation) and on the test set (that is, bunches belonging to one of the 19 test projects; we will call this operation *external* validation.

In order to match as close as possible the real circumstances and operating conditions in which our models should be actually used, models are not tested individually. Instead, they are grouped in models aggregates suitable to be used for the length estimation of a given bunch, when certain knowledge conditions are met. The next paragraphs are devoted to better illustrate these concepts.

14.7.1 Knowledge conditions

Given a single bunch populated with a reasonable number of nodes (one architecture, several processes and components, several signals and variables), the number of possible different knowledge conditions that could occur in a given project refinement step is usually remarkable. It is therefore impractical (and of dubious usefulness) to prepare and validate model aggregates for each possible knowledge condition.

Instead, we established four discrete conditions, associated with respective sets of rules that are to be applied to assess whether a bunch in a given refinement state qualifies or not for a given knowledge state. Such states are called K1, K2, K3 and K4, and the associated rules are illustrated in the table below. For each cell, the presence of a tick mark (✓) means that variables indicated in that row must be known in order to qualify for the knowledge state indicated in that column.

	K1	K2	K3	K4
Entity interface	✓	✓	✓	✓
Entity mode	✓	✓	✓	✓
Number of declared components	✓	✓	✓	·
Declared component interface	✓	✓	·	·
Number of instanced components	✓	✓	✓	·
Instanced component interface	✓	✓	·	·
Number of architecture signals	✓	·	·	·
Number of processes	✓	✓	·	·
Process variables	✓	·	·	·

The following table illustrates all the involved variables: as usual, they can appear with or without a hat ($\hat{}$), which denotes estimated quantities and distinguishes them from actual (measured) ones.

For each estimated variable, the name of the model chosen to estimate it, is reported. The chosen model is, in each case, the model with the lowest error variance among all the models suitable to be used with the available information.

Variable	Symbol
Total number of ports	n_p
Total port homogeneity	h_p
Number of ports per mode	
- number of `in` ports	n_{ip}
- number of `out` ports	n_{op}
- number of `inout` ports	n_{iop}
- number of other ports	n_{xp}
Sum of port homogeneity per mode	
- homogeneity of `in` ports	h_{ip}
- homogeneity of `out` ports	h_{op}
- homogeneity of `inout` ports	h_{iop}
- homogeneity of other ports	h_{xp}
Number of internal signals	n_s
Homogeneity of internal signals	h_s
Number of component instantiations	n_{ci}
Number of component instantiations	n_{cd}
Number of processes	n_{pr}
Number of variables in the i-th process	n_{pv_i}
Length of entity declaration in lines of code	L_e
Length of architecture core in lines of code	L_{ac}
Length of the i-th component declaration in lines of code	L_{cd_i}
Number of the i-th component instantiation in lines of code	L_{ci_i}
Number of the i-th process in lines of code	L_{pr_i}

14.7.2 Model aggregates

In the following paragraphs you will find the formulae and the actual model aggregates used to validate the methodology. Since there will be exactly one model aggregate for each

of the previously described knowledge conditions, we find it natural to indicate model aggregates with the same names of their knowledge levels, confident that the reader will not be confused. Paragraphs denoted by a final 'I' character denote internal validation conditions, on the other hand, a final 'E' denotes external validation cases.

K1

Model:

$$\begin{array}{rcll} \hat{L} & = & \hat{L}_e(n_p, n_g) & \text{(EM3)} \\ & + & \hat{L}_{ac}(h_{ip}, h_{op}, h_{iop}, h_{xp}, h_s) & \text{(AM4H)} \\ & + & \sum_{i=1}^{n_{cd}} \hat{L}_{cd_i}(n_{p_i}, n_{g_i}) & (\text{C}_\text{D}\text{M2}) \\ & + & \sum_{i=1}^{n_{ci}} \hat{L}_{ci_i}(n_{p_i}) & (\text{C}_\text{I}\text{M1}) \\ & + & \sum_{i=1}^{n_{pr}} \hat{L}_{pr_i}(n_{pv_i}) & \text{(PM1)} \end{array}$$

K2

Model:

$$\begin{array}{rcll} \hat{L} & = & \hat{L}_e(n_p, n_g) & \text{(EM3)} \\ & + & \hat{L}_{ac}(h_{ip}, h_{op}, h_{iop}, h_{xp}) & \text{(AM2H)} \\ & + & \sum_{i=1}^{n_{cd}} \hat{L}_{cd_i}(n_{p_i}, n_{g_i}) & (\text{C}_\text{D}\text{M2}) \\ & + & \sum_{i=1}^{n_{ci}} \hat{L}_{ci_i} & (\text{C}_\text{I}\text{M1}) \\ & + & n_{pr} \cdot \hat{L}_{pr} & \text{(PM0)} \end{array}$$

K3

Model:

$$\begin{array}{rcll} \hat{L} & = & \hat{L}_e(n_p, n_g) & \text{(EM3)} \\ & + & \hat{L}_{ac}(h_{ip}, h_{op}, h_{iop}, h_{xp}) & \text{(AM2H)} \\ & + & n_{cd}(n_{ip}, n_{op}, n_{iop}, n_{xp}) \cdot \hat{L}_{cd} & (\text{C}_\text{D}\text{M0}) \\ & + & n_{ci}(n_{ip}, n_{op}, n_{iop}, n_{xp}) \cdot \hat{L}_{ci} & (\text{C}_\text{I}\text{M0}) \\ & + & \hat{n}_{pr} \cdot \hat{L}_{pr}(n_{ip}, n_{op}, n_{iop}, n_{xp}) & \text{(CGAPM1)} \cdot \text{(PM0)} \end{array}$$

K4

Model:

$$\begin{array}{rcll} \hat{L} & = & \hat{L}_e(n_p, n_g) & \text{(EM3)} \\ & + & \hat{L}_{ac}(h_{ip}, h_{op}, h_{iop}, h_{xp}) & \text{(AM2H)} \\ & + & \hat{n}_{cd}(n_{ip}, n_{op}, n_{iop}, n_{xp}) \cdot \hat{L}_{cd} & (\text{CGAC}_\text{D}\text{M1}) \cdot (\text{C}_\text{D}\text{M0}) \\ & + & \hat{n}_{ci}(n_{ip}, n_{op}, n_{iop}, n_{xp}) \cdot \hat{L}_{ci} & (\text{CGAC}_\text{I}\text{M1}) \cdot (\text{C}_\text{I}\text{M0}) \\ & + & \hat{n}_{pr} \cdot \hat{L}_{pr}(n_{ip}, n_{op}, n_{iop}, n_{xp}) & \text{(CGAPM1)} \cdot \text{(PM0)} \end{array}$$

14.8 K1I

14.8.1 Result summary

Population statistical properties and model accuracy:

	All architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	121.865	125.814	3.949
Variance	82132.638	42720.235	38047.713
Standard deviation	286.588	206.689	195.058
	Behavioral architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	148.815	134.588	-14.227
Variance	121245.948	50276.013	64691.291
Standard deviation	348.204	224.223	254.345
	Structural architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	54.806	107.909	53.103
Variance	5960.385	16378.653	9404.351
Standard deviation	77.204	127.979	96.976
	Data-flow architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	124.114	124.641	0.527
Variance	72713.624	47117.252	20728.851
Standard deviation	269.655	217.065	143.975

Correlation between estimated and real values:

	Correlation coefficient $(L, \hat{L})$
All architectures	0.7327
Behavioral architectures only	0.6842
Structural architectures only	0.6546
Data-flow architectures only	0.8466

Figure 14.32: Model K1I: Real vs. estimated lines of code.

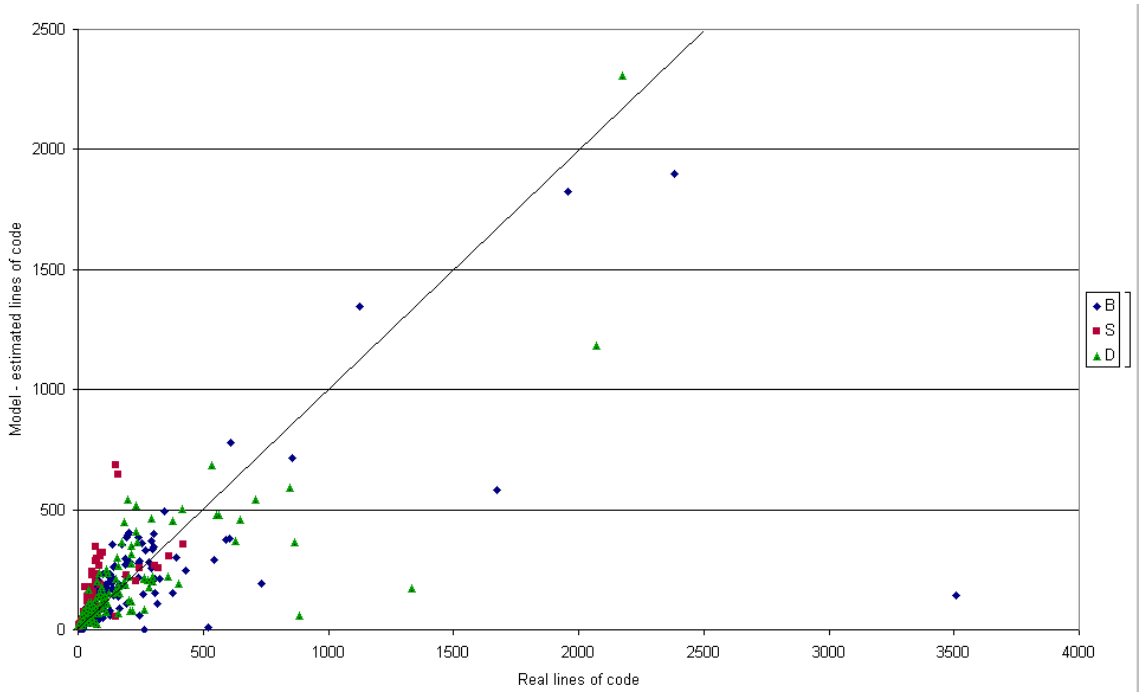


Figure 14.33: Models K1Ib, K1Is, K1Id: Real vs. estimated lines of code.

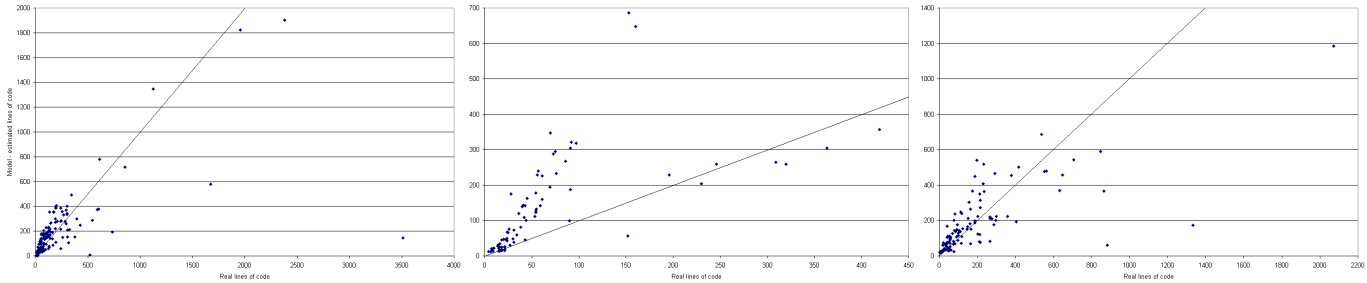


Figure 14.34: Model K1I: Error density distribution.

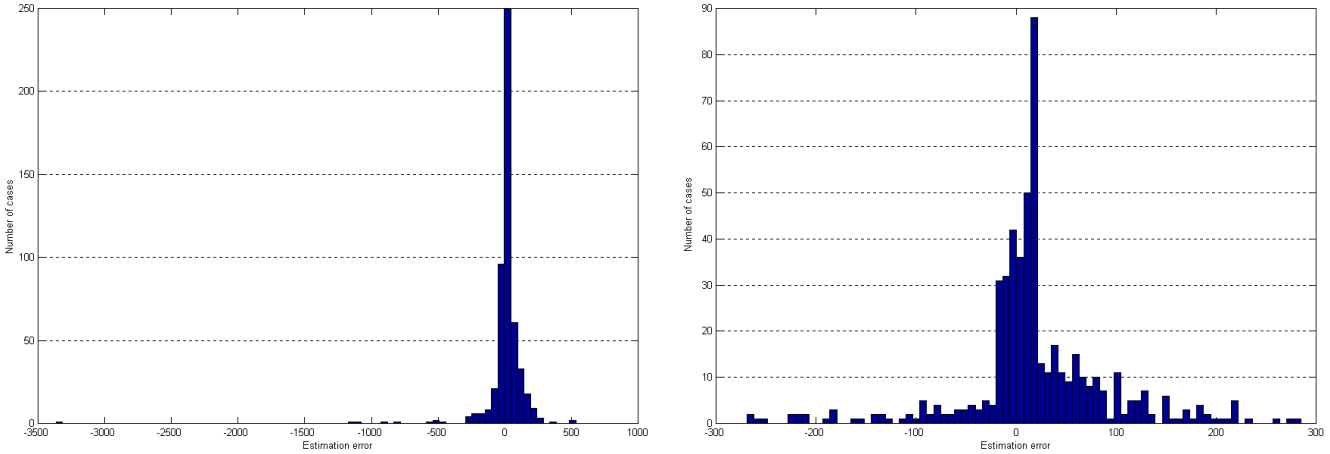


Figure 14.35: Model K1I: Error cumulative distribution.

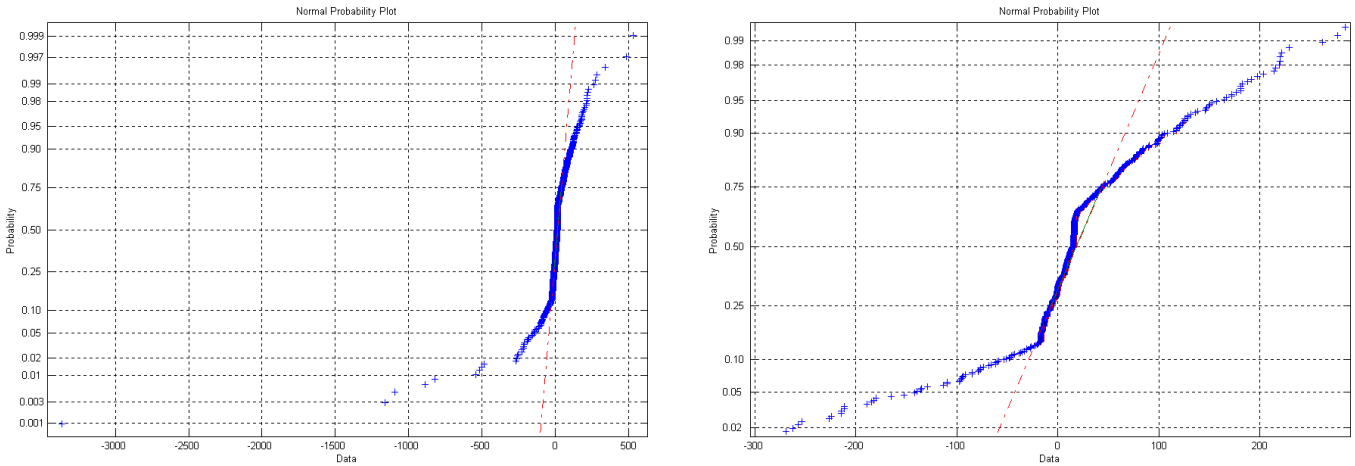
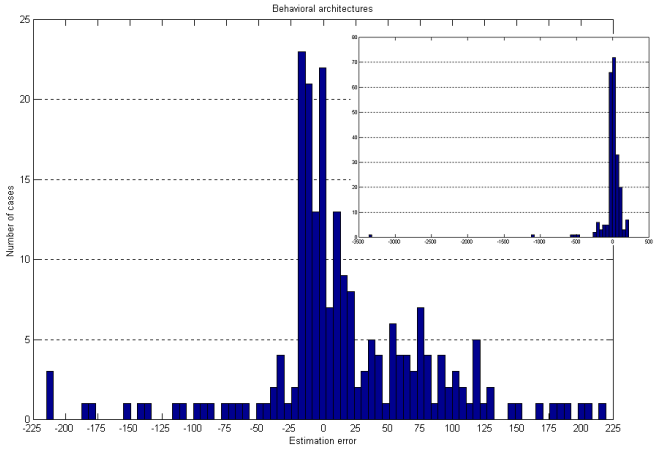
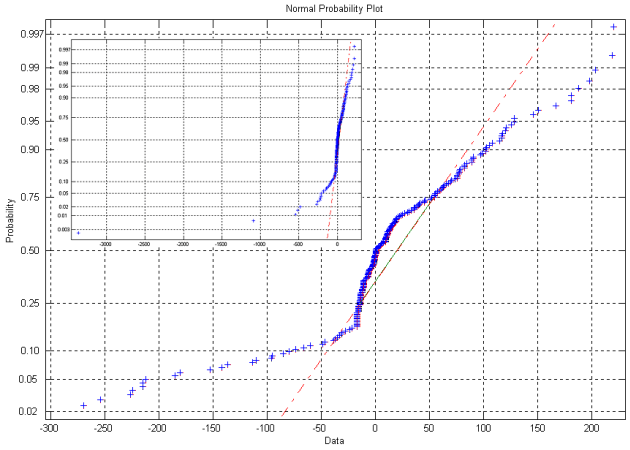
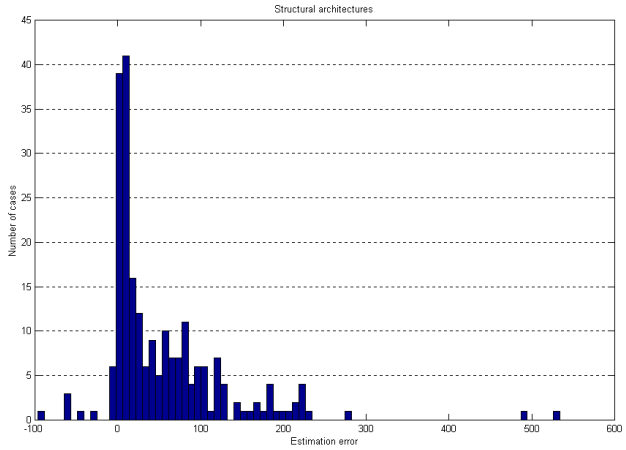
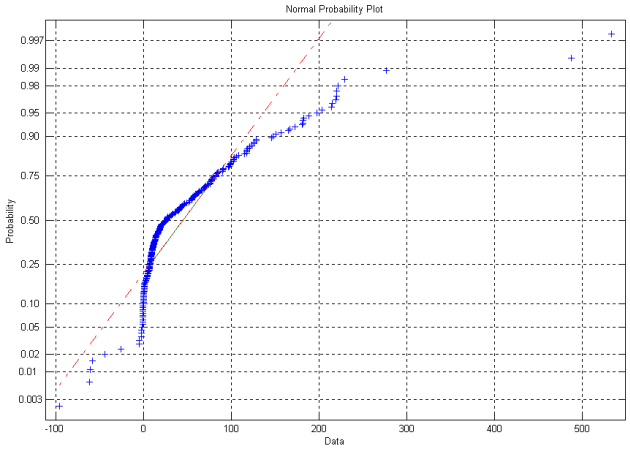


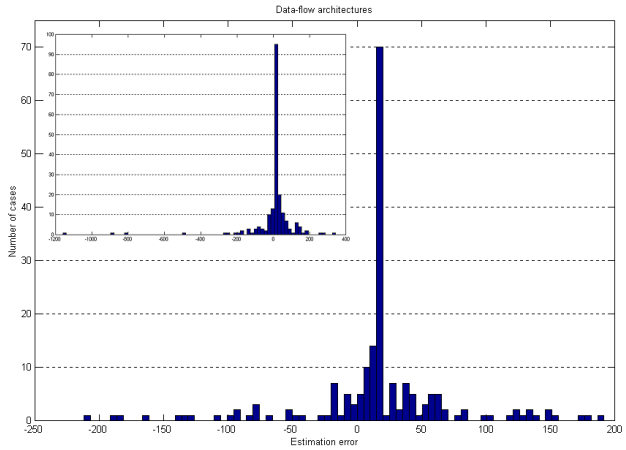
Figure 14.36: Models K1Ib, K1Is, K1Id: Error density and cumulative distributions.

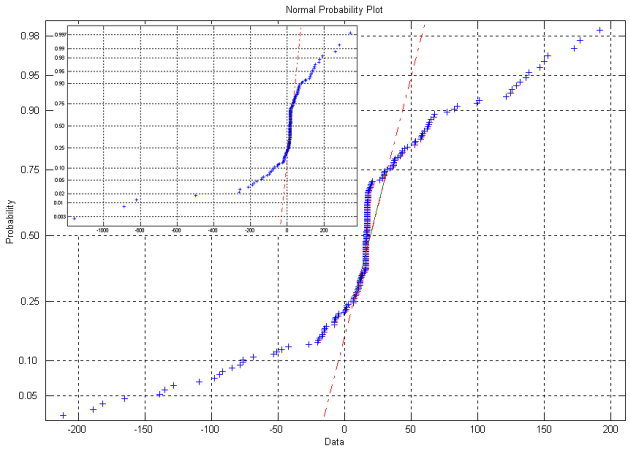












14.8.2 Detailed results

Real sizes versus estimated sizes for behavioral architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
ans_RISC8	reg_files	behave_reg_files	76	103.386	27.386
an-XC2S-USB	xc2sFPGA	BHV	190	219.758	29.758
an-XC2S-XR16	teBL	bhv	302	399.656	97.656
an-XC2S-XR16	Glue	BHV	109	145.654	36.654
an-XC2S-USB	xc2sFunc	BHV	16	1.001	-14.999
an-XC2S-XR16	Core	BHV	210	280.019	70.019
an-XC2S-USB	teBL	bhv	295	368.817	73.817
an-XC2S-USB	xc2sCore	BHV	59	55.204	-3.796
DLX	alu	behaviour	55	96.591	41.591
DLX	cache	behaviour	260	150.378	-109.622
DLX	clock_gen	behaviour	20	39.108	19.108
DLX	controller	behaviour	735	191.821	-543.179
DLX	dlx	behaviour	542	288.474	-253.526
DLX	dlx_bus_monitor	behaviour	129	69.098	-59.902
DLX	ir	behaviour	59	48.022	-10.978
DLX	latch	behaviour	17	33.282	16.282
DLX	memory	behaviour	167	87.828	-79.172
DLX	mux2	behaviour	14	1.866	-12.134
DLX	reg_1_out	behaviour	24	47.568	23.568
DLX	reg_2_1_out	behaviour	31	50.332	19.332
DLX	reg_2_out	behaviour	29	48.661	19.661
DLX	reg_3_out	behaviour	34	49.754	15.754
DLX	reg_file	behaviour	41	92.673	51.673
ERC32	AC245Generic	Behavior	67	30.232	-36.768
ERC32	AC377Generic	Behavior	46	35.364	-10.636
ERC32	FPURTGeneric	vhdl_behavioral	1127	1347.151	220.151
ERC32	IURTGeneric	vhdl_behavioral	1959	1822.904	-136.096
ERC32	RAM8	BEHAVIORAL	57	132.555	75.555
ERC32	TAPTest_iufpu	Behaviour	49	138.615	89.615
gl85	I8085	BEHAVIOR	1673	579.96	-1093.04
HC11	clock	behavoir	24	69.641	45.641
HC11	dev	behavior	48	61.989	13.989
HC11	hc11ram	behavior	41	63.318	22.318
i80386	i80386	behavior	858	716.129	-141.871
i8051	I8051_ALU	BHV	318	106.075	-211.925
i8051	I8051_CTR	BHV	3510	144.145	-3365.855
i8051	I8051_DBG	BHV	244	59.379	-184.621
i8051	I8051_RAM	BHV	195	110.346	-84.654
i8051	I8051_TSB	BHV	54	117.432	63.432
i8051	I8051_XRM	BHV	30	29.767	-0.233
Leon	RAM2P_168X32	behav	13	0.991	-12.009
Leon	RAM2P_136X32	behav	13	0.991	-12.009
Leon	RAM2P_16X32	behav	13	0.991	-12.009
Leon	RAM_2048x32	behavioral	14	0.991	-13.009
Leon	RAM_1024x32	behavioral	14	0.991	-13.009
Leon	RAM_512x30	behavioral	14	0.991	-13.009
Leon	RAM_512x28	behavioral	14	0.991	-13.009
Leon	RAM_256x30	behavioral	14	0.991	-13.009
Leon	RAM_256x28	behavioral	14	0.991	-13.009
Leon	RAM_256x26	behavioral	14	0.991	-13.009
Leon	atc25_syncram_sim	behavioral	29	43.177	14.177
Leon	atc25_2pram	behav	35	44.506	9.506
Leon	atc35_dpram_ss_dn	behav	38	55.634	17.634
Leon	ATC35_RAM_256x26	behavioral	17	6.637	-10.363
Leon	ATC35_RAM_1024x32	behavioral	17	6.637	-10.363
Leon	ATC35_RAM_2048x32	behavioral	17	6.637	-10.363
Leon	ATC35_RAM_256x28	behavioral	17	6.637	-10.363
Leon	ATC35_RAM_1024x34	behavioral	17	6.637	-10.363
Leon	ATC35_RAM_2048x34	behavioral	17	6.637	-10.363
Leon	DPRAMRWRW_16X32	behav	21	4.289	-16.711

(continues on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	DPRAMRWRW_136X32	behav	21	4.289	-16.711
Leon	DPRAMRWRW_168X32	behav	21	4.289	-16.711
Leon	SW204420	behavioral	32	39.414	7.414
Leon	SU004020	behavioral	29	38.609	9.609
Leon	SA108019	behavioral	27	32.111	5.111
Leon	generic_dpram_as	behav	45	97.226	52.226
Leon	R2048x34M8	behavioral	18	0.991	-17.009
Leon	RF68X32M1	behav	17	1.487	-15.513
Leon	RF68X33M1	behav	17	1.487	-15.513
Leon	RF136X32M1	behav	17	1.487	-15.513
Leon	RF136X33M1	behav	17	1.487	-15.513
Leon	R1024X34M4	behavioral	18	0.991	-17.009
Leon	R256X28M4	behavioral	18	0.991	-17.009
Leon	R1024X33M4	behavioral	18	0.991	-17.009
Leon	R2048X32M8	behavioral	18	0.991	-17.009
Leon	R1024X32M4	behavioral	18	0.991	-17.009
Leon	R256X26M4	behavioral	18	0.991	-17.009
Leon	R256X25M4	behavioral	18	0.991	-17.009
Leon	R256X24M4	behavioral	18	0.991	-17.009
Leon	umc18_syncram_ss	behavioral	50	125.921	75.921
Leon	umc18_dpram_ss	behav	58	140.820	82.820
Leon	virtex_regfile_cp	behav	35	45.926	10.926
Leon	virtex_regfile	behav	36	45.349	9.349
Leon	virtex_syncram	behav	62	56.523	-5.477
Leon	RAMB4_S16_S16	behav	38	38.725	0.725
Leon	RAMB4_S1	behav	14	-0.082	-14.082
Leon	RAMB4_S2	behav	14	-0.082	-14.082
Leon	RAMB4_S4	behav	14	-0.082	-14.082
Leon	RAMB4_S8	behav	14	-0.082	-14.082
SuperscalarDLX	Dlx	BehaviorPipelined	2383	1900.524	-482.476
SuperscalarDLX	Environment	Behavior	307	154.692	-152.308
T80	NoICE_TB	behaviour	53	63.612	10.612
TE51	te51mux	bhv	27	-6.179	-33.179
TE51	te51d	BHV	87	97.591	10.591
TE51	te51dec	bhv	267	-2.297	-269.297
TE51	te51mcode	BHV	522	7.983	-514.017
TE51	te51alu	BHV	78	56.300	-21.700
TE51	te51regs	BHV	127	60.674	-66.326
TE51	te51ctrl	bhv	394	299.191	-94.809
TE51	te51c	BHV	189	297.401	108.401
xapp146	MULTI_DVM	BEHAVE	590	375.134	-214.866
xapp146	MULTI_DVM_TB	BEHAVIOR	87	177.495	90.495
xapp146	SHIFT16	DEFINITION	29	28.437	-0.563
xapp146	SHIFT8b	DEFINITION2	31	30.108	-0.892
xapp146	TOP_LEVEL	BEHAVE	141	140.851	-0.149
xapp146	TOP_LEVEL_TB	BEHAVE	147	268.118	121.118
xapp146	upcnt5	DEFINITION2	28	21.336	-6.664
xapp328	CNT_25	BEHAVIOURAL	47	52.691	5.691
xapp328	cnt3	DEFINITION	24	30.169	6.169
xapp328	CNT_5	BEHAVIOURAL	29	29.592	0.592
xapp328	command_state_machine	BEHAVIOURAL	108	185.567	77.567
xapp328	DNLD_INTERFACE	BEHAVIOURAL	96	180.555	84.555
xapp328	FLASH_CNTR	BEHAVIOURAL	378	153.825	-224.175
xapp328	i2c_master	behave	343	493.693	150.693
xapp328	lcd_control	behave	129	80.512	-48.488
xapp328	main_ctrl_state_machine	behave	161	137.902	-23.098
xapp328	pxa_bufif2	behavioral	10	2.112	-7.888
xapp328	pxa_mux	behavioral	7	-3.452	-10.452
xapp328	pxa_dff_apar_p0	behavioral	18	29.592	11.592
xapp328	pxa_tff_apar_p0	behavioral	20	29.592	9.592
xapp328	mpeg_chip_ctrl	behave	139	174.741	35.741
xapp328	on_off_logic	behave	27	30.169	3.169

(continues on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
xapp328	PARALLEL_PORT	BEHAVIOURAL	63	177.773	114.773
xapp328	play_logic_state_machine	behave	80	206.085	126.085
xapp328	play_modes	behave	126	231.420	105.420
xapp328	power_ctrl	behave	59	137.793	78.793
xapp328	SHIFT8	DEFINITION	31	30.108	-0.892
xapp328	sound_control	behave	125	223.799	98.799
xapp328	upcnt2	DEFINITION	28	29.014	1.014
xapp328	upcnt3	DEFINITION	28	29.014	1.014
xapp328	upcnt4	DEFINITION	28	29.014	1.014
xapp328	updwncnt4	DEFINITION	32	29.592	-2.408
xapp333	i2c	behave	164	189.454	25.454
xapp333	i2c_control	behave	612	778.898	166.898
xapp333	SHIFT8	DEFINITION	31	30.108	-0.892
xapp333	uC_interface	BEHAVIOUR	243	215.698	-27.302
xapp333	upcnt4	DEFINITION	28	21.336	-6.664
xapp345	irda_uart	behavior	71	73.593	2.593
xapp345	irda_uart_tb	behavior	101	229.390	128.390
xapp345	jk_ff	behavior	35	44.455	9.455
xapp345	rxcver	behavior	125	195.796	70.796
xapp345	sirendec	behavior	135	354.330	219.330
xapp345	txmit	behavior	105	159.945	54.945
xapp345	uart	behavior	63	55.344	-7.656
xapp345	uart_tb	behavior	101	229.390	128.390
xapp348	sck_logic	DEFINITION	138	195.428	57.428
xapp348	spi_control_sm	DEFINITION	242	278.012	36.012
xapp348	spi_rcv_shift_reg	DEFINITION	76	165.642	89.642
xapp348	spi_xmit_shift_reg	DEFINITION	41	60.903	19.903
xapp348	uC_interface	BEHAVIOUR	294	258.830	-35.170
xapp348	upcnt4	DEFINITION	24	22.491	-1.509
xapp348	upcnt5	DEFINITION	24	22.491	-1.509
xapp349	uC_interface	BEHAVIOUR	257	360.018	103.018
xapp354	am30lv0064d	behavior	22	-9.110	-31.110
xapp354	AMD_FLASH_TB	BEHAVIOR	325	211.713	-113.287
xapp354	NAND_INTERFACE	BEHAVIOR	138	215.768	77.768
xapp354	k9f4008w0a	behavior	22	-9.110	-31.110
xapp354	NAND_FLASH_TB	BEHAVIOR	306	210.384	-95.616
xapp355	ADC_INTERFACE	BEHAVE	590	375.134	-214.866
xapp355	ADC_INTERFACE_TB	BEHAVIOR	87	177.495	90.495
xapp355	SHIFT16	DEFINITION	29	28.437	-0.563
xapp355	SHIFT8	DEFINITION	31	30.108	-0.892
xapp355	TOP_LEVEL	BEHAVE	114	154.537	40.537
xapp355	TOP_LEVEL_TB	BEHAVE	144	262.077	118.077
xapp355	upcnt5	DEFINITION	28	21.336	-6.664
xapp356	ADC_INTERFACE	BEHAVIOR	300	333.020	33.020
xapp356	SHIFT16	DEFINITION	28	28.437	0.437
xapp356	SHIFT8	DEFINITION	30	30.108	0.108
xapp356	TEMP_INTERFACE	BEHAVIOR	100	94.513	-5.487
xapp356	TOP_LEVEL	BEHAVIOR	240	386.215	146.215
xapp356	TOP_LEVEL_TB	BEHAVIOR	304	344.184	40.184
xapp356	UPCNT11	DEFINITION	27	21.336	-5.664
xapp356	UPCNT15	DEFINITION	27	21.336	-5.664
xapp356	UPCNT5	DEFINITION	27	21.336	-5.664
xapp356	XPATH	BEHAVIOR	429	248.621	-180.379
xapp357	CLK_DIVIDER	DEFINITION	16	31.324	15.324
xapp357	LED_TEST	BEHAVIOUR	116	173.668	57.668
xapp358	tx_rx_entity	behavioral	60	57.353	-2.647
xapp363	clk_gen	behavioral	17	33.572	16.572
xapp365	ISO_CLK_DIVIDER	DEFINITION	55	76.089	21.089
xapp367	AudioController	DEFINITION	21	34.088	13.088
xapp367	chatterbox	BEHAVIOUR	271	330.467	59.467
xapp367	DTMFController	DEFINITION	31	31.938	0.938
xapp367	FlipFlop	definition	15	27.965	12.965

(continues on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
xapp367	FlipFlopR	definition	17	27.965	10.965
xapp367	IrqController	BEHAVIOUR	83	44.400	-38.600
xapp367	MemoryManager	BEHAVE	98	51.759	-46.241
xapp367	PowerSupplyController	DEFINITION	20	31.840	11.840
xapp367	RFTransceiverController	DEFINITION	31	42.975	11.975
xapp369	DECODE_MAN	BEHAVE	283	279.380	-3.620
xapp369	TOP_LEVEL	BEHAVE	117	163.179	46.179
xapp370	CLK_DIVIDER	DEFINITION	19	32.995	13.995
xapp370	cooltrak	BEHAVIOUR	245	283.689	38.689
xapp370	MULTI_DVM	BEHAVE	606	379.586	-226.414
xapp370	SHIFT16	DEFINITION	29	28.437	-0.563
xapp370	SHIFT8	DEFINITION	26	30.108	4.108
xapp370	SPEED	DEFINITION	45	106.335	61.335
xapp370	upcnt5	DEFINITION	28	21.336	-6.664
xapp336	DEC_16B20B	BEHAVIOUR	80	98.602	18.602
xapp336	DEC_FUNC	BEHAVIOUR	191	273.356	82.356
xapp336	DIS_GEN_LOW	BEHAVIOUR	86	149.142	63.142
xapp336	DIS_GEN_UP	BEHAVIOUR	83	148.608	65.608
xapp336	ENC_16B20B	BEHAVIOUR	74	75.799	1.799
xapp336	ENC_FUNC	BEHAVIOUR	126	201.776	75.776
xapp336	ERR_CHECK	BEHAVIOUR	77	179.313	102.313
xapp336	ERR_DET	BEHAVIOUR	60	180.529	120.529
xapp336	DECODER	BEHAVIOUR	175	356.189	181.189
xapp336	ENCODER_LOW	BEHAVIOUR	196	384.209	188.209
xapp336	ENCODER_UP	BEHAVIOUR	197	394.803	197.803
xapp336	MAIN_TB	BEHAVIOUR	88	133.521	45.521
xapp336	S_GEN	BEHAVIOUR	57	174.327	117.327
xapp336_8	ERR_CHECK	BEHAVIOUR	77	179.313	102.313
xapp336_8	DEC_FUNC	BEHAVIOUR	191	273.356	82.356
xapp336_8	DIS_GEN	BEHAVIOUR	79	146.938	67.938
xapp336_8	ENC_FUNC	BEHAVIOUR	135	166.618	31.618
xapp336_8	DECODER	BEHAVIOUR	173	354.519	181.519
xapp336_8	ENCODER	BEHAVIOUR	202	405.693	203.693
xapp336_8	MAIN_TB	BEHAVIOUR	92	147.181	55.181
xapp336_8	S_GEN	BEHAVIOUR	57	174.327	117.327
Leon	fs90_dpram_ss	behav	40	95.632	55.632
Leon	fs90_syncram_sim	behavioral	34	96.035	62.035
Leon	generic_syncram	behavioral	41	118.152	77.152
Leon	generic_dpram_ss	behav	48	102.790	54.790
Leon	syncram	behav	44	28.664	-15.336
TE51	te51	BHV	54	44.680	-9.320
Leon	RAMB4_S16	behav	14	-0.082	-14.082
xapp354	am30lv0064d_top	behavior	64	-9.367	-73.367

Real sizes versus estimated sizes for structural architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
ans_RISC8	ans_risc8	STRUCT_ANS_RISC8	363	304.284	-58.716
ans_RISC8	alu	STRUCT_ALU	230	203.826	-26.174
ans_RISC8	control	STRUCT_CONTROL	246	258.499	12.499
gl85struct	acc_ctrl	structure	41	143.901	102.901
gl85struct	alulogic	structure	70	346.957	276.957
gl85struct	ALU_8BIT	structure	43	142.368	99.368
gl85struct	alu_ctrl	structure	61	226.063	165.063
gl85struct	bc_pc_sp	structure	38	80.937	42.937
gl85struct	buf8	structure	15	15.187	0.187
gl85struct	ctl_lgc1	structure	28	175.402	147.402
gl85struct	ctl_lgc2	structure	45	162.407	117.407
gl85struct	dataaddr	structure	54	126.214	72.214
gl85struct	decod2_4	structure	14	30.857	16.857
gl85struct	DECOD3_8	structure	24	49.267	25.267
gl85struct	flagunit	structure	97	318.371	221.371
gl85struct	g16bctr	structure	36	120.034	84.034
gl85struct	g4bctr	structure	69	194.788	125.788
gl85struct	gl85	structure	91	187.795	96.795
gl85struct	hldlogic	structure	14	27.486	13.486

(continues on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
gl85struct	hltlogic	structure	16	34.408	18.408
gl85struct	hl_de_wz	structure	53	111.852	58.852
gl85struct	inst_reg	structure	34	59.787	25.787
gl85struct	interupt	structure	57	239.673	182.673
gl85struct	intrupt1	structure	59	142.377	83.377
gl85struct	intrupt2	structure	54	122.929	68.929
gl85struct	intrupt3	structure	42	108.371	66.371
gl85struct	inv8	structure	15	15.187	0.187
gl85struct	M5	structure	22	46.395	24.395
gl85struct	mcdecode	structure	91	305.162	214.162
gl85struct	MUX2TO1	STR_MUX2TO1	15	14.263	-0.737
gl85struct	mux_4bit	structure	24	67.759	43.759
gl85struct	ocnand	structure	18	45.326	27.326
gl85struct	oprlogic	structure	54	177.202	123.202
gl85struct	parity1	structure	15	22.518	7.518
gl85struct	pc_cntrl	structure	92	321.364	229.364
gl85struct	prioenco	structure	25	66.024	41.024
gl85struct	rdwrgen	structure	76	232.946	156.946
gl85struct	reg8bits	structure	30	48.563	18.563
gl85struct	regctrl0	structure	73	288.362	215.362
gl85struct	regctrl1	structure	160	647.705	487.705
gl85struct	regctrl2	structure	55	132.923	77.923
gl85struct	regpad	structure	75	294.948	219.948
gl85struct	regpair	structure	20	46.774	26.774
gl85struct	regpairs	structure	20	46.774	26.774
gl85struct	reg_8bit	structure	24	42.855	18.855
gl85struct	reg_ctrl	structure	153	686.851	533.851
gl85struct	reg_ram	structure	20	46.774	26.774
gl85struct	shflogic	structure	44	100.48	56.48
gl85struct	sn54181	structure	86	267.385	181.385
gl85struct	SN85150	structure	31	72.950	41.950
gl85struct	sp_cntrl	structure	56	228.359	172.359
gl85struct	stater	structure	61	159.511	98.511
gl85struct	tempctrl	structure	40	139.570	99.570
gl85struct	vectrgen	structure	26	75.111	49.111
HC11	hc11core	structure	27	29.874	2.874
i8051	I8051_ALL	STR	196	228.107	32.107
Leon	ahbtest	struct	90	99.667	9.667
Leon	pci_esa	struct	31	38.569	7.569
Leon	pci_is	struct	43	45.504	2.504
Leon	atc25_pciiodpad	syn	9	20.529	11.529
Leon	atc25_pciiopad	syn	7	21.049	14.049
Leon	atc25_pcitoutpad	syn	6	12.968	6.968
Leon	atc25_pcioutpad	syn	4	11.939	7.939
Leon	atc25_odpad	syn	18	15.526	-2.474
Leon	atc25_iodpad	syn	18	24.426	6.426
Leon	atc25_iopadu	syn	16	24.946	8.946
Leon	atc25_iopad	syn	16	24.946	8.946
Leon	atc25_inpad	syn	4	11.939	7.939
Leon	atc25_smpad	syn	4	11.939	7.939
Leon	atc25_outpad	syn	15	14.197	-0.803
Leon	atc25_toutpadu	syn	16	16.045	0.045
Leon	umc18_smiopad	syn	10	22.477	12.477
Leon	atc35_odpad	syn	18	15.526	-2.474
Leon	atc35_iodpad	syn	18	24.426	6.426
Leon	atc35_iopad	syn	16	24.946	8.946
Leon	atc35_toutpadu	syn	16	16.045	0.045
Leon	atc35_outpad	syn	15	14.197	-0.803
Leon	atc35_smpad	syn	4	11.939	7.939
Leon	atc35_inpad	syn	4	11.939	7.939
Leon	fs90_odpad	syn	21	15.616	-5.384
Leon	fs90_iodpad	syn	21	24.926	3.926

(continues on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	fs90_smiopad	syn	21	25.545	4.545
Leon	fs90_iopad	syn	21	25.545	4.545
Leon	fs90_toutpadu	syn	21	18.072	-2.928
Leon	fs90_outpad	syn	21	15.517	-5.483
Leon	fs90_smpad	syn	7	13.367	6.367
Leon	fs90_inpad	syn	9	13.268	4.268
Leon	umc18_odpad	syn	16	14.197	-1.803
Leon	umc18_iopad	syn	18	25.045	7.045
Leon	umc18_toutpadu	syn	18	17.473	-0.527
Leon	umc18_outpad	syn	15	14.197	-0.803
Leon	umc18_smpad	syn	4	11.939	7.939
Leon	umc18_inpad	syn	4	11.939	7.939
PIC16C5X	pic_core	structural	419	356.963	-62.037
T80	NoICE	struct	152	55.853	-96.147
Leon	umc18_iodpad	syn	18	24.426	6.426
ans_RISC8	reg_top	STRUCT_REG_TOP	309	264.837	-44.163
ans_RISC8	inst_decoder	STRUCT	320	258.645	-61.355

Real sizes versus estimated sizes for data-flow architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
ans_RISC8	reg_w	rtl_reg_w	19	61.747	42.747
ans_RISC8	reg_status	rtl_reg_status	46	75.951	29.951
ans_RISC8	reg_ioport	rtl_reg_ioport	39	67.461	28.461
ans_RISC8	reg_fsr	rtl_reg_fsr	23	62.962	39.962
ans_RISC8	prog_count	rtl_prog_count	77	72.041	-4.959
ans_RISC8	mux_win	rtl_mux_win	52	31.959	-20.041
ans_RISC8	mux_fwe	rtl_mux_fwe	46	29.528	-16.472
ans_RISC8	mux_fin	rtl_mux_fin	48	30.743	-17.257
ans_RISC8	mux_cz_write	rtl_mux_cz_write	45	29.205	-15.795
ans_RISC8	mux_alub	rtl_mux_alub	46	30.136	-15.864
ans_RISC8	mux_alua	rtl_mux_alua	50	31.351	-18.649
ans_RISC8	ir_reg	rtl_ir_reg	40	71.855	31.855
ans_RISC8	ir_decode	rtl_ir_decode	74	66.579	-7.421
ans_RISC8	clock_div	rtl_clock_div	30	65.817	35.817
ans_RISC8	alu_dp	rtl_alu_dp	82	68.107	-13.893
ans_RISC8	aluop_gen	rtl_aluop_gen	61	33.782	-27.218
ax8	A90S1200	rtl	167	151.781	-15.219
ax8	A90S2313	rtl	286	176.206	-109.794
ax8	AX8	rtl	647	457.565	-189.435
ax8	AX_ALU	rtl	212	120.168	-91.832
ax8	AX_PCS	rtl	79	72.155	-6.845
ax8	AX_Port	rtl	51	110.822	59.822
ax8	AX_RAM	rtl	31	65.008	34.008
ax8	AX_Reg	rtl	202	123.458	-78.542
ax8	AX_Reg	rtl2	210	80.980	-129.020
ax8	AX_TC16	rtl	277	208.283	-68.717
ax8	AX_TC8	rtl	99	128.307	29.307
ax8	AX_UART	rtl	215	314.134	99.134
DLX	dlx	rtl	234	518.536	284.536
ERC32	uart	VHDL_RTL	536	685.724	149.724
HC11	hc11cpu	rtl	2073	1184.805	-888.195
Jane	Neuron	dataflow	292	464.532	172.532
Leon	acache	rtl	174	365.731	191.731
Leon	ahbarb	rtl	186	448.360	262.360
Leon	ahbstat	rtl	69	126.170	57.170
Leon	apbmst	rtl	78	202.216	124.216
Leon	cache	rtl	79	141.966	62.966
Leon	cachemem	rtl	75	82.825	7.825
Leon	clkgen	rtl	30	67.141	37.141
Leon	dcache	rtl	378	454.505	76.505
Leon	div	rtl	118	238.934	120.934
Leon	fpaux	rtl	42	167.862	125.862

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	fp	rtl	707	541.642	-165.358
Leon	fp1eu	rtl	564	479.017	-84.983
Leon	icache	rtl	230	406.857	176.857
Leon	ioport	rtl	106	142.808	36.808
Leon	irqctrl	rtl	84	136.783	52.783
Leon	irqctrl2	rtl	111	137.061	26.061
Leon	iu	rtl	2175	2306.732	131.732
Leon	lconf	rtl	33	69.631	36.631
Leon	leon	rtl	85	137.987	52.987
Leon	leon_pci	rtl	215	273.796	58.796
Leon	mcore	rtl	187	188.567	1.567
Leon	mctrl	rtl	553	476.356	-76.644
Leon	fpu	rtl	32	34.557	2.557
Leon	mul	rtl	265	213.439	-51.561
Leon	GEN_XOR2	rtl	2	19.198	17.198
Leon	GEN_OR2	rtl	2	19.198	17.198
Leon	GEN_AND2	rtl	2	19.198	17.198
Leon	pci_arb	rtl	150	212.636	62.636
Leon	proc	rtl	163	264.064	101.064
Leon	rstgen	rtl	20	63.037	43.037
Leon	atc25_regfile_iu	rtl	57	52.373	-4.627
Leon	atc25_regfile_cp	rtl	51	108.955	57.955
Leon	atc25_syncram	rtl	56	97.334	41.334
Leon	pp33t015vt	rtl	3	19.198	16.198
Leon	pp33b015vt	rtl	8	23.558	15.558
Leon	pp33o01	rtl	2	18.591	16.591
Leon	pt33b04u	rtl	8	23.558	15.558
Leon	pt33b03u	rtl	8	23.558	15.558
Leon	pt33b02u	rtl	8	23.558	15.558
Leon	pt33b01u	rtl	8	23.558	15.558
Leon	pt33b04	rtl	8	23.558	15.558
Leon	pt33b03	rtl	8	23.558	15.558
Leon	pt33b02	rtl	8	23.558	15.558
Leon	pt33b01	rtl	8	23.558	15.558
Leon	pt33t03u	rtl	3	19.198	16.198
Leon	pt33t02u	rtl	3	19.198	16.198
Leon	pt33t01u	rtl	3	19.198	16.198
Leon	pt33o04	rtl	2	18.591	16.591
Leon	atc35_regfile_cp	rtl	41	78.537	37.537
Leon	atc35_regfile	rtl	58	100.973	42.973
Leon	atc35_syncram	rtl	36	35.367	-0.633
Leon	pt3b03	rtl	8	23.558	15.558
Leon	pt3b02	rtl	8	23.558	15.558
Leon	pt3b01	rtl	8	23.558	15.558
Leon	pc3t03u	rtl	3	19.198	16.198
Leon	pc3t02u	rtl	3	19.198	16.198
Leon	pc3t01u	rtl	3	19.198	16.198
Leon	fs90_regfile	rtl	55	91.387	36.387
Leon	fs90_syncram	rtl	50	78.148	28.148
Leon	uyfaa	rtl	14	20.593	6.593
Leon	vyfa2gsa	rtl	12	21.629	9.629
Leon	genodpad	rtl	2	18.591	16.591
Leon	geniopad	rtl	5	23.558	18.558
Leon	geniodpad	rtl	5	22.950	17.950
Leon	gentoutpadu	rtl	3	19.198	16.198
Leon	genoutpad	rtl	2	18.591	16.591
Leon	gensmpad	rtl	2	18.591	16.591
Leon	generic_regfile_iu	rtl	89	107.087	18.087
Leon	generic_regfile_cp	rtl	36	54.979	18.979
Leon	generic_smult	rtl	21	73.570	52.570
Leon	geninpad	rtl	2	18.591	16.591
Leon	pciiodpad	rtl	14	28.824	14.824

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	pciiopad	rtl	19	30.152	11.152
Leon	pcitoutpad	rtl	12	25.072	13.072
Leon	pcioutpad	rtl	12	23.745	11.745
Leon	iodpad	rtl	24	38.965	14.965
Leon	odpad	rtl	24	32.805	8.805
Leon	smiopad	rtl	29	41.372	12.372
Leon	iopad	rtl	29	41.372	12.372
Leon	toutpadu	rtl	24	35.213	11.213
Leon	outpad	rtl	24	32.805	8.805
Leon	smpad	rtl	21	31.476	10.476
Leon	inpad	rtl	21	31.476	10.476
Leon	hw_smult	rtl	30	36.542	6.542
Leon	regfile_cp	rtl	37	37.080	0.080
Leon	umc18_regfile	rtl	33	34.690	1.690
Leon	umc18_syncram	rtl	50	42.083	-7.917
Leon	OR2DL	rtl	2	19.198	17.198
Leon	EXOR2DL	rtl	2	19.198	17.198
Leon	AND2DL	rtl	2	19.198	17.198
Leon	INVDL	rtl	2	18.591	16.591
Leon	C3B42	rtl	8	23.558	15.558
Leon	CD3O40T	rtl	7	18.591	11.591
Leon	CD3O20T	rtl	7	18.591	11.591
Leon	CD3O10T	rtl	7	18.591	11.591
Leon	CD3B40T	rtl	8	23.558	15.558
Leon	CD3B20T	rtl	8	23.558	15.558
Leon	CD3B10T	rtl	8	23.558	15.558
Leon	C3B40	rtl	8	23.558	15.558
Leon	C3B20	rtl	8	23.558	15.558
Leon	C3B10	rtl	8	23.558	15.558
Leon	C3B40U	rtl	8	23.558	15.558
Leon	timers	rtl	161	180.986	19.986
Leon	uart	rtl	267	219.334	-47.666
Leon	wprot	rtl	94	176.052	82.052
PIC16C5X	fadr_mux	dataflow	12	19.198	7.198
PIC16C5X	pic_alu	dataflow	78	24.308	-53.692
PIC16C5X	reg_cons	dataflow	12	19.198	7.198
ppx16	P16C55	rtl	149	166.093	17.093
ppx16	P16F84	rtl	186	199.720	13.720
ppx16	PPX16	rtl	235	363.766	128.766
ppx16	PPX_ALU	rtl	216	76.554	-139.446
ppx16	PPX_Ctrl	rtl	57	36.416	-20.584
ppx16	PPX_PCS	rtl	99	116.576	17.576
ppx16	PPX_Port	rtl	50	111.623	61.623
ppx16	PPX_RAM	rtl	39	68.453	29.453
ppx16	PPX_TMR	rtl	92	149.495	57.495
rd1007	sd_cnfg	RTL	118	110.118	-7.882
rd1007	sd_rfrsh	RTL	40	104.475	64.475
rd1007	sd_sig	RTL	197	541.206	344.206
rd1007	sd_state	RTL	60	106.761	46.761
rd1007	sd_top	RTL	124	153.379	29.379
T51	I8052	rtl	145	151.636	6.636
T51	T51	rtl	848	590.745	-257.255
T51	T51_ALU	rtl	404	192.163	-211.837
T51	T51_Port	rtl	43	109.962	66.962
T51	T51_RAM	rtl	55	70.830	15.830
T80	MonZ80	rtl	882	60.532	-821.468
T80	T80	rtl	865	366.825	-498.175
T80	T80a	rtl	156	302.703	146.703
T80	T80s	rtl	97	89.314	-7.686
T80	T80_ALU	rtl	265	82.814	-182.186
T80	T80_MCode	rtl	1333	172.970	-1160.030
xapp333	micro_master_tb	RTL	299	222.505	-76.495

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
xapp333	micro_slave_tb	RTL	202	222.505	20.505
xapp333	micro_tb	RTL	358	222.505	-135.495
xapp358	receive	receive_rtl	163	69.110	-93.89
xapp363	clk_top	rtl	41	104.475	63.475
xapp363	gpio_top	rtl	22	66.854	44.854
xapp363	sam_top	rtl	418	502.113	84.113
xapp363	smedia_state	rtl	298	200.002	-97.998
xapp363	smedia_top	rtl	113	249.078	136.078
xapp363	spi	rtl	83	235.657	152.657
xapp363	spi_switch	rtl	55	70.784	15.784
xapp363	ssp_icc	rtl	631	368.555	-262.445
xapp363	ssp_icc_switch	rtl	115	72.283	-42.717
xapp365	iso9141	rtl	213	351.191	138.191
Leon	GEN_NOT	rtl	2	18.591	16.591
Leon	pc3d01	rtl	2	18.591	16.591
Leon	pt33o03	rtl	2	18.591	16.591
Leon	pt33o02	rtl	2	18.591	16.591
Leon	pt33o01	rtl	2	18.591	16.591
Leon	pt33d20u	rtl	3	22.343	19.343
Leon	pt33d20	rtl	2	18.591	16.591
Leon	pt33d00u	rtl	3	22.343	19.343
Leon	pt33d00	rtl	2	18.591	16.591
Leon	pt3o03	rtl	2	18.591	16.591
Leon	pt3o02	rtl	2	18.591	16.591
Leon	pt3o01	rtl	2	18.591	16.591
Leon	pc3d21	rtl	2	18.591	16.591
Leon	wyfa2gsa	rtl	18	27.811	9.811
Leon	rfbypass	rtl	44	111.422	67.422
Leon	regfile_iu	rtl	47	47.771	0.771
Leon	C3B20U	rtl	8	23.558	15.558
Leon	C3B10U	rtl	8	23.558	15.558
Leon	C3O40	rtl	2	18.591	16.591
Leon	C3O20	rtl	2	18.591	16.591
Leon	C3O10	rtl	2	18.591	16.591
Leon	C3I42	rtl	2	18.591	16.591
Leon	C3I40	rtl	2	18.591	16.591

14.9 K2I

14.9.1 Result summary

Population statistical properties and model accuracy:

	All architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	121.865	126.340	4.475
Variance	82132.638	21505.667	60083.399
Standard deviation	286.588	146.648	245.119
	Behavioral architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	148.815	136.455	-12.360
Variance	121245.948	15239.741	106228.235
Standard deviation	348.204	123.449	325.927
	Structural architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	54.806	108.606	53.800
Variance	5960.385	16422.581	9398.082
Standard deviation	77.204	128.151	96.944
	Data-flow architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	124.114	123.577	-0.537
Variance	72713.624	30941.414	31725.508
Standard deviation	269.655	175.902	178.117

Correlation between estimated and real values:

	Correlation coefficient $(L, \hat{L})$
All architectures	0.5182
Behavioral architectures only	0.3519
Structural architectures only	0.6562
Data-flow architectures only	0.7582

Figure 14.37: Model K2I: Real vs. estimated lines of code.

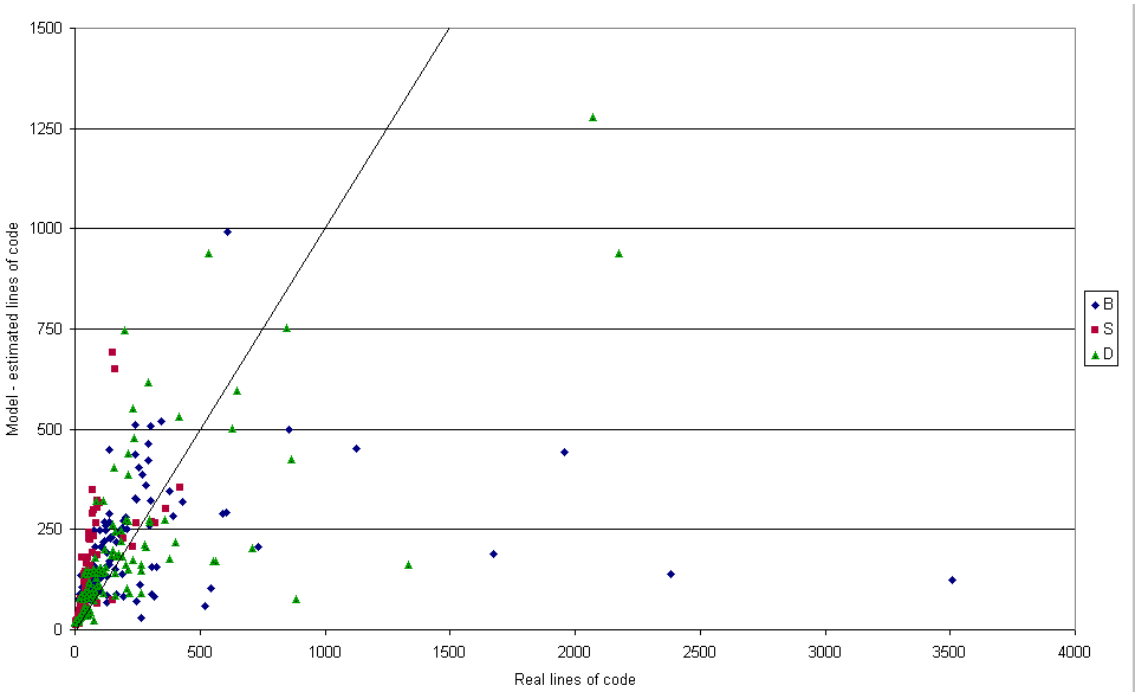


Figure 14.38: Models K2Ib, K2Is, K2Id: Real vs. estimated lines of code.

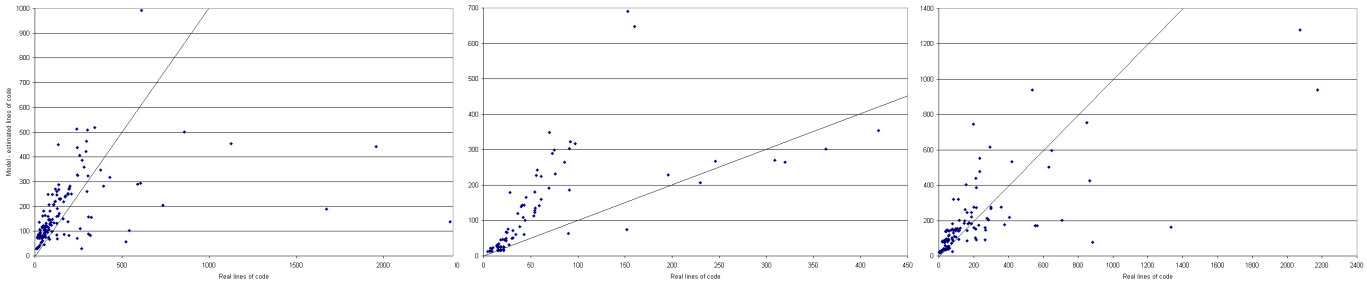


Figure 14.39: Model K2I: Error density distribution.

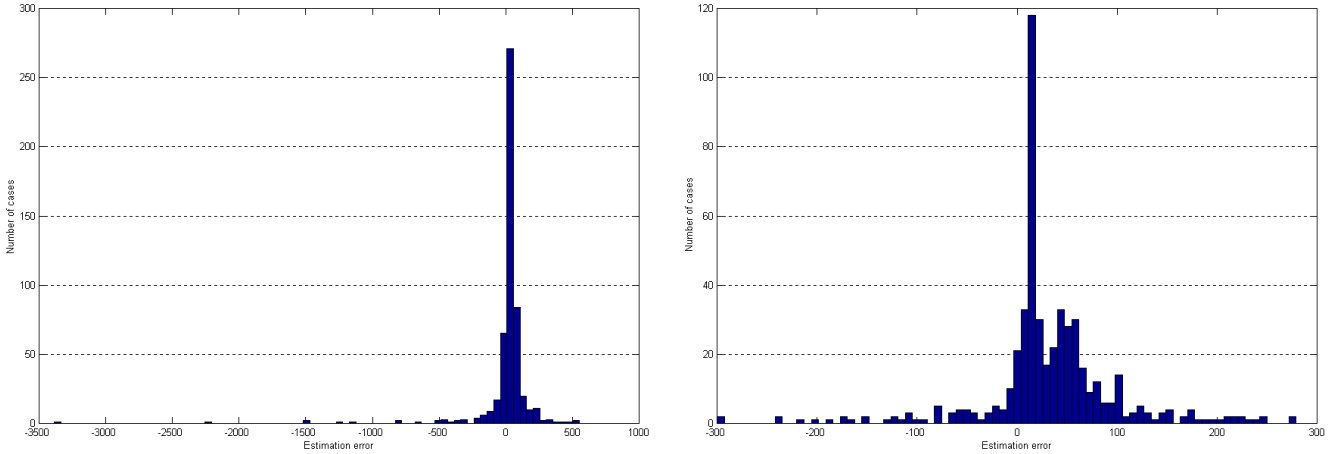


Figure 14.40: Model K2I: Error cumulative distribution.

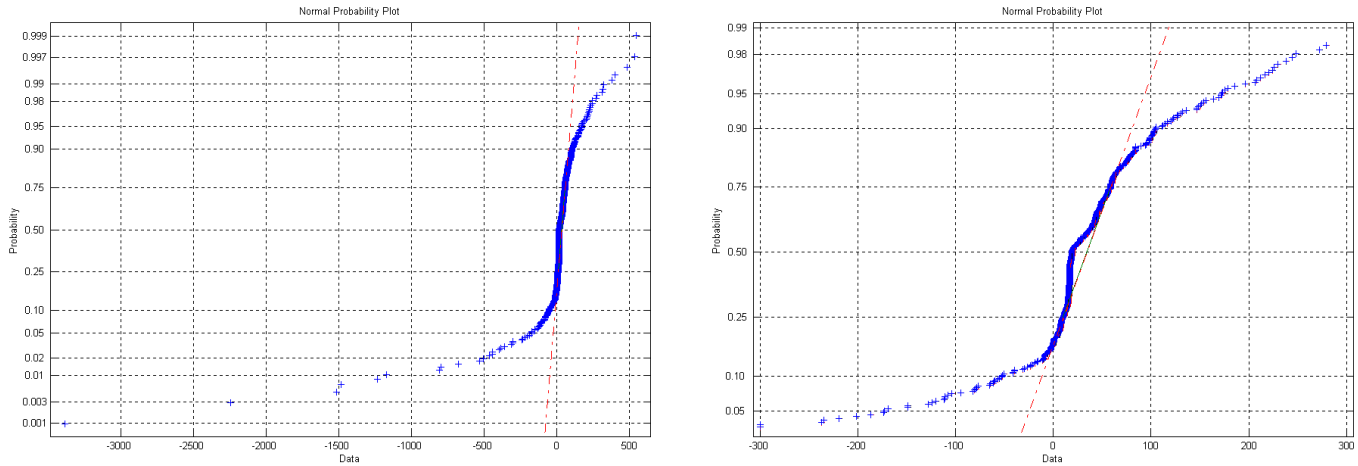
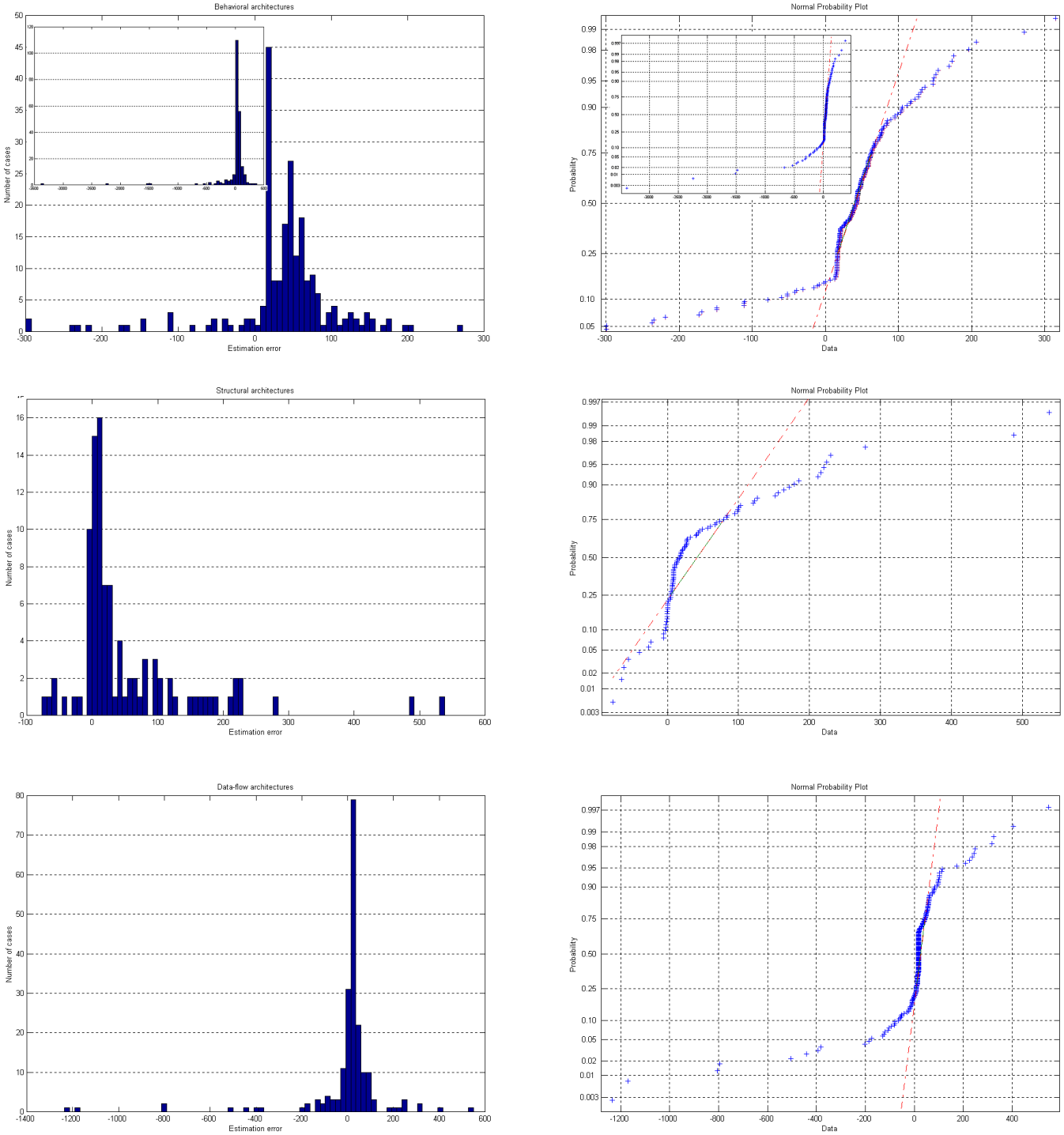


Figure 14.41: Models K2Ib, K2Is, K2Id: Error density and cumulative distributions.



14.9.2 Detailed results

Real sizes versus estimated sizes for behavioral architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
ans_RISC8	reg_files	behave_reg_files	76	160.522	84.522
an-XC2S-USB	xc2sFPGA	BHV	190	253.052	63.052
an-XC2S-XR16	teBL	bhv	302	508.174	206.174
an-XC2S-XR16	Glue	BHV	109	207.941	98.941
an-XC2S-USB	xc2sFunc	BHV	16	31.964	15.964
an-XC2S-XR16	Core	BHV	210	249.756	39.756
an-XC2S-USB	teBL	bhv	295	464.016	169.016
an-XC2S-USB	xc2sCore	BHV	59	86.129	27.129
DLX	alu	behaviour	55	86.947	31.947
DLX	cache	behaviour	260	111.270	-148.730
DLX	clock_gen	behaviour	20	87.480	67.480
DLX	controller	behaviour	735	205.680	-529.320
DLX	dlx	behaviour	542	101.931	-440.069
DLX	dlx_bus_monitor	behaviour	129	69.008	-59.992
DLX	ir	behaviour	59	81.676	22.676
DLX	latch	behaviour	17	78.965	61.965
DLX	memory	behaviour	167	87.744	-79.256
DLX	mux2	behaviour	14	34.248	20.248
DLX	reg_1_out	behaviour	24	78.406	54.406
DLX	reg_2_1_out	behaviour	31	83.913	52.913
DLX	reg_2_out	behaviour	29	80.880	51.880
DLX	reg_3_out	behaviour	34	83.354	49.354
DLX	reg_file	behaviour	41	80.320	39.320
ERC32	AC245Generic	Behavior	67	101.742	34.742
ERC32	AC377Generic	Behavior	46	81.064	35.064
ERC32	FPURTGeneric	vhdl_behavioral	1127	453.016	-673.984
ERC32	IURTGeneric	vhdl_behavioral	1959	442.602	-1516.398
ERC32	RAM8	BEHAVIORAL	57	121.241	64.241
ERC32	TAPTest_iufpu	Behaviour	49	180.764	131.764
gl85	I8085	BEHAVIOR	1673	190.299	-1482.701
HC11	clock	behavoir	24	135.422	111.422
HC11	dev	behavior	48	75.404	27.404
HC11	hc11ram	behavior	41	76.734	35.734
i80386	i80386	behavior	858	500.176	-357.824
i8051	I8051_ALU	BHV	318	82.985	-235.015
i8051	I8051_CTR	BHV	3510	122.659	-3387.341
i8051	I8051_DBG	BHV	244	71.173	-172.827
i8051	I8051_RAM	BHV	195	83.222	-111.778
i8051	I8051_TSB	BHV	54	106.667	52.667
i8051	I8051_XRM	BHV	30	72.739	42.739
Leon	RAM2P_168X32	behav	13	33.427	20.427
Leon	RAM2P_136X32	behav	13	33.427	20.427
Leon	RAM2P_16X32	behav	13	33.427	20.427
Leon	RAM_2048x32	behavioral	14	33.427	19.427
Leon	RAM_1024x32	behavioral	14	33.427	19.427
Leon	RAM_512x30	behavioral	14	33.427	19.427
Leon	RAM_512x28	behavioral	14	33.427	19.427
Leon	RAM_256x30	behavioral	14	33.427	19.427
Leon	RAM_256x28	behavioral	14	33.427	19.427
Leon	RAM_256x26	behavioral	14	33.427	19.427
Leon	atc25_syncram_sim	behavioral	29	74.069	45.069
Leon	atc25_2pram	behav	35	75.398	40.398
Leon	atc35_dpram_ss_dn	behav	38	75.398	37.398
Leon	ATC35_RAM_256x26	behavioral	17	33.473	16.473
Leon	ATC35_RAM_1024x32	behavioral	17	33.473	16.473
Leon	ATC35_RAM_2048x32	behavioral	17	33.473	16.473
Leon	ATC35_RAM_256x28	behavioral	17	33.473	16.473
Leon	ATC35_RAM_1024x34	behavioral	17	33.473	16.473
Leon	ATC35_RAM_2048x34	behavioral	17	33.473	16.473
Leon	DPRAMRWRW_16X32	behav	21	35.342	14.342

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	DPRAMRWRW_136X32	behav	21	35.342	14.342
Leon	DPRAMRWRW_168X32	behav	21	35.342	14.342
Leon	SW204420	behavioral	32	103.839	71.839
Leon	SU004020	behavioral	29	105.654	76.654
Leon	SA108019	behavioral	27	89.453	62.453
Leon	generic_dpram_as	behav	45	120.115	75.115
Leon	R2048x34M8	behavioral	18	33.427	15.427
Leon	RF68X32M1	behav	17	33.940	16.940
Leon	RF68X33M1	behav	17	33.940	16.940
Leon	RF136X32M1	behav	17	33.940	16.940
Leon	RF136X33M1	behav	17	33.940	16.940
Leon	R1024X34M4	behavioral	18	33.427	15.427
Leon	R256X28M4	behavioral	18	33.427	15.427
Leon	R1024X33M4	behavioral	18	33.427	15.427
Leon	R2048X32M8	behavioral	18	33.427	15.427
Leon	R1024X32M4	behavioral	18	33.427	15.427
Leon	R256X26M4	behavioral	18	33.427	15.427
Leon	R256X25M4	behavioral	18	33.427	15.427
Leon	R256X24M4	behavioral	18	33.427	15.427
Leon	umc18_syncram_ss	behavioral	50	118.227	68.227
Leon	umc18_dpram_ss	behav	58	163.155	105.155
Leon	virtex_regfile_cp	behav	35	54.741	19.741
Leon	virtex_regfile	behav	36	54.182	18.182
Leon	virtex_syncram	behav	62	75.031	13.031
Leon	RAMB4_S16_S16	behav	38	71.087	33.087
Leon	RAMB4_S1	behav	14	32.354	18.354
Leon	RAMB4_S2	behav	14	32.354	18.354
Leon	RAMB4_S4	behav	14	32.354	18.354
Leon	RAMB4_S8	behav	14	32.354	18.354
SuperscalarDLX	Dlx	BehaviorPipelined	2383	138.324	-2244.676
SuperscalarDLX	Environment	Behavior	307	88.098	-218.902
T80	NoICE_TB	behaviour	53	44.500	-8.500
TE51	te51mux	bhv	27	43.940	16.940
TE51	te51d	BHV	87	130.926	43.926
TE51	te51dec	bhv	267	30.049	-236.951
TE51	te51mcode	BHV	522	58.102	-463.898
TE51	te51alu	BHV	78	75.562	-2.438
TE51	te51regs	BHV	127	85.459	-41.541
TE51	te51ctrl	bhv	394	282.872	-111.128
TE51	te51c	BHV	189	248.899	59.899
xapp146	MULTI_DVM	BEHAVE	590	290.158	-299.842
xapp146	MULTI_DVM_TB	BEHAVIOR	87	182.240	95.240
xapp146	SHIFT16	DEFINITION	29	71.410	42.410
xapp146	SHIFT8b	DEFINITION2	31	74.443	43.443
xapp146	TOP_LEVEL	BEHAVE	141	230.697	89.697
xapp146	TOP_LEVEL_TB	BEHAVE	147	231.133	84.133
xapp146	upcnt5	DEFINITION2	28	76.523	48.523
xapp328	CNT_25	BEHAVIOURAL	47	122.596	75.596
xapp328	cnt3	DEFINITION	24	73.088	49.088
xapp328	CNT_5	BEHAVIOURAL	29	72.529	43.529
xapp328	command_state_machine	BEHAVIOURAL	108	133.210	25.210
xapp328	DNLD_INTERFACE	BEHAVIOURAL	96	124.109	28.109
xapp328	FLASH_CNTR	BEHAVIOURAL	378	346.825	-31.175
xapp328	i2c_master	behave	343	518.410	175.410
xapp328	lcd_control	behave	129	191.991	62.991
xapp328	main_ctrl_state_machine	behave	161	150.300	-10.700
xapp328	pxa_bufif2	behavioral	10	28.930	18.930
xapp328	pxa_mux	behavioral	7	28.930	21.930
xapp328	pxa_dff_apar_p0	behavioral	18	72.529	54.529
xapp328	pxa_tff_apar_p0	behavioral	20	72.529	52.529
xapp328	mpeg_chip_ctrl	behave	139	171.703	32.703
xapp328	on_off_logic	behave	27	73.088	46.088

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
xapp328	PARALLEL_PORT	BEHAVIOURAL	63	124.109	61.109
xapp328	play_logic_state_machine	behave	80	206.917	126.917
xapp328	play_modes	behave	126	159.166	33.166
xapp328	power_ctrl	behave	59	120.280	61.280
xapp328	SHIFT8	DEFINITION	31	74.443	43.443
xapp328	sound_control	behave	125	247.362	122.362
xapp328	upcnt2	DEFINITION	28	71.969	43.969
xapp328	upcnt3	DEFINITION	28	71.969	43.969
xapp328	upcnt4	DEFINITION	28	71.969	43.969
xapp328	updwncnt4	DEFINITION	32	72.529	40.529
xapp333	i2c	behave	164	219.210	55.210
xapp333	i2c_control	behave	612	992.618	380.618
xapp333	SHIFT8	DEFINITION	31	74.443	43.443
xapp333	uC_interface	BEHAVIOUR	243	327.234	84.234
xapp333	upcnt4	DEFINITION	28	76.523	48.523
xapp345	irda_uart	behavior	71	118.074	47.074
xapp345	irda_uart_tb	behavior	101	247.739	146.739
xapp345	jk_ff	behavior	35	72.529	37.529
xapp345	rxcver	behavior	125	261.295	136.295
xapp345	sirendec	behavior	135	449.542	314.542
xapp345	txmit	behavior	105	208.036	103.036
xapp345	uart	behavior	63	105.390	42.390
xapp345	uart_tb	behavior	101	247.739	146.739
xapp348	sck_logic	DEFINITION	138	288.655	150.655
xapp348	spi_control_sm	DEFINITION	242	438.456	196.456
xapp348	spi_rcv_shift_reg	DEFINITION	76	249.397	173.397
xapp348	spi_xmit_shift_reg	DEFINITION	41	114.449	73.449
xapp348	uC_interface	BEHAVIOUR	294	421.230	127.230
xapp348	upcnt4	DEFINITION	24	77.642	53.642
xapp348	upcnt5	DEFINITION	24	77.642	53.642
xapp349	uC_interface	BEHAVIOUR	257	405.423	148.423
xapp354	am30lv0064d	behavior	22	36.939	14.939
xapp354	AMD_FLASH_TB	BEHAVIOR	325	155.768	-169.232
xapp354	NAND_INTERFACE	BEHAVIOR	138	268.718	130.718
xapp354	k9f4008w0a	behavior	22	36.939	14.939
xapp354	NAND_FLASH_TB	BEHAVIOR	306	157.221	-148.779
xapp355	ADC_INTERFACE	BEHAVE	590	290.158	-299.842
xapp355	ADC_INTERFACE_TB	BEHAVIOR	87	182.240	95.240
xapp355	SHIFT16	DEFINITION	29	71.410	42.410
xapp355	SHIFT8	DEFINITION	31	74.443	43.443
xapp355	TOP_LEVEL	BEHAVE	114	218.556	104.556
xapp355	TOP_LEVEL_TB	BEHAVE	144	227.874	83.874
xapp355	upcnt5	DEFINITION	28	76.523	48.523
xapp356	ADC_INTERFACE	BEHAVIOR	300	260.210	-39.790
xapp356	SHIFT16	DEFINITION	28	71.410	43.410
xapp356	SHIFT8	DEFINITION	30	74.443	44.443
xapp356	TEMP_INTERFACE	BEHAVIOR	100	147.690	47.690
xapp356	TOP_LEVEL	BEHAVIOR	240	512.054	272.054
xapp356	TOP_LEVEL_TB	BEHAVIOR	304	322.709	18.709
xapp356	UPCNT11	DEFINITION	27	76.523	49.523
xapp356	UPCNT15	DEFINITION	27	76.523	49.523
xapp356	UPCNT5	DEFINITION	27	76.523	49.523
xapp356	XPATH	BEHAVIOR	429	318.508	-110.492
xapp357	CLK_DIVIDER	DEFINITION	16	74.207	58.207
xapp357	LED_TEST	BEHAVIOUR	116	269.985	153.985
xapp358	tx_rx_entity	behavioral	60	96.476	36.476
xapp363	clk_gen	behavioral	17	77.800	60.800
xapp365	ISO_CLK_DIVIDER	DEFINITION	55	102.726	47.726
xapp367	AudioController	DEFINITION	21	79.715	58.715
xapp367	chatterbox	BEHAVIOUR	271	387.346	116.346
xapp367	DTMFController	DEFINITION	31	95.284	64.284
xapp367	FlipFlop	definition	15	73.647	58.647

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
xapp367	FlipFlopR	definition	17	73.647	56.647
xapp367	IrqController	BEHAVIOUR	83	67.070	-15.930
xapp367	MemoryManager	BEHAVE	98	98.388	0.388
xapp367	PowerSupplyController	DEFINITION	20	76.122	56.122
xapp367	RFTransceiverController	DEFINITION	31	91.290	60.290
xapp369	DECODE_MAN	BEHAVE	283	359.513	76.513
xapp369	TOP_LEVEL	BEHAVE	117	221.708	104.708
xapp370	CLK_DIVIDER	DEFINITION	19	77.240	58.240
xapp370	cooltrak	BEHAVIOUR	245	324.562	79.562
xapp370	MULTI_DVM	BEHAVE	606	293.192	-312.808
xapp370	SHIFT16	DEFINITION	29	71.410	42.410
xapp370	SHIFT8	DEFINITION	26	74.443	48.443
xapp370	SPEED	DEFINITION	45	161.963	116.963
xapp370	upcnt5	DEFINITION	28	76.523	48.523
xapp336	DEC_16B20B	BEHAVIOUR	80	105.907	25.907
xapp336	DEC_FUNC	BEHAVIOUR	191	138.955	-52.045
xapp336	DIS_GEN_LOW	BEHAVIOUR	86	144.477	58.477
xapp336	DIS_GEN_UP	BEHAVIOUR	83	148.070	65.070
xapp336	ENC_16B20B	BEHAVIOUR	74	94.269	20.269
xapp336	ENC_FUNC	BEHAVIOUR	126	132.335	6.335
xapp336	ERR_CHECK	BEHAVIOUR	77	114.686	37.686
xapp336	ERR_DET	BEHAVIOUR	60	114.449	54.449
xapp336	DECODER	BEHAVIOUR	175	242.631	67.631
xapp336	ENCODER_LOW	BEHAVIOUR	196	270.761	74.761
xapp336	ENCODER_UP	BEHAVIOUR	197	274.354	77.354
xapp336	MAIN_TB	BEHAVIOUR	88	125.537	37.537
xapp336	S_GEN	BEHAVIOUR	57	115.245	58.245
xapp336_8	ERR_CHECK	BEHAVIOUR	77	114.686	37.686
xapp336_8	DEC_FUNC	BEHAVIOUR	191	138.955	-52.045
xapp336_8	DIS_GEN	BEHAVIOUR	79	145.036	66.036
xapp336_8	ENC_FUNC	BEHAVIOUR	135	162.686	27.686
xapp336_8	DECODER	BEHAVIOUR	173	239.597	66.597
xapp336_8	ENCODER	BEHAVIOUR	202	281.099	79.099
xapp336_8	MAIN_TB	BEHAVIOUR	92	136.416	44.416
xapp336_8	S_GEN	BEHAVIOUR	57	115.245	58.245
Leon	fs90_dpram_ss	behav	40	73.720	33.720
Leon	fs90_syncram_sim	behavioral	34	74.069	40.069
Leon	generic_syncram	behavioral	41	118.786	77.786
Leon	generic_dpram_ss	behav	48	120.115	72.115
Leon	syncram	behav	44	61.082	17.082
TE51	te51	BHV	54	93.326	39.326
Leon	RAMB4_S16	behav	14	32.354	18.354
xapp354	am30lv0064d_top	behavior	64	131.847	67.847

Real sizes versus estimated sizes for structural architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
ans_RISC8	ans_risc8	STRUCT_ANS_RISC8	363	301.513	-61.487
ans_RISC8	alu	STRUCT_ALU	230	206.800	-23.200
ans_RISC8	control	STRUCT_CONTROL	246	267.182	21.182
gl85struct	acc_ctrl	structure	41	143.829	102.829
gl85struct	alulogic	structure	70	348.970	278.970
gl85struct	ALU_8BIT	structure	43	143.874	100.874
gl85struct	alu_ctrl	structure	61	224.956	163.956
gl85struct	bc_pc_sp	structure	38	82.826	44.826
gl85struct	buf8	structure	15	15.489	0.489
gl85struct	ctl_lgc1	structure	28	179.492	151.492
gl85struct	ctl_lgc2	structure	45	165.557	120.557
gl85struct	dataaddr	structure	54	127.764	73.764
gl85struct	decod2_4	structure	14	30.961	16.961
gl85struct	DECOD3_8	structure	24	48.954	24.954
gl85struct	flagunit	structure	97	317.404	220.404
gl85struct	g16bctr	structure	36	119.898	83.898
gl85struct	g4bctr	structure	69	191.754	122.754
gl85struct	gl85	structure	91	186.011	95.011
gl85struct	hldlogic	structure	14	27.766	13.766

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
gl85struct	hltlogic	structure	16	34.668	18.668
gl85struct	hl_de_wz	structure	53	113.503	60.503
gl85struct	inst_reg	structure	34	60.293	26.293
gl85struct	interupt	structure	57	242.424	185.424
gl85struct	intrupt1	structure	59	142.141	83.141
gl85struct	intrupt2	structure	54	122.692	68.692
gl85struct	intrupt3	structure	42	108.903	66.903
gl85struct	inv8	structure	15	15.489	0.489
gl85struct	M5	structure	22	46.142	24.142
gl85struct	mcdecode	structure	91	303.060	212.060
gl85struct	MUX2TO1	STR_MUX2TO1	15	16.359	1.359
gl85struct	mux_4bit	structure	24	67.228	43.228
gl85struct	ocnand	structure	18	45.706	27.706
gl85struct	oprlogic	structure	54	180.324	126.324
gl85struct	parity1	structure	15	22.778	7.778
gl85struct	pc_cntrl	structure	92	321.884	229.884
gl85struct	prioenco	structure	25	65.631	40.631
gl85struct	rdwrgen	structure	76	231.953	155.953
gl85struct	reg8bits	structure	30	48.982	18.982
gl85struct	regctrl0	structure	73	289.134	216.134
gl85struct	regctrl1	structure	160	648.234	488.234
gl85struct	regctrl2	structure	55	134.614	79.614
gl85struct	regpad	structure	75	299.128	224.128
gl85struct	regpair	structure	20	47.351	27.351
gl85struct	regpairs	structure	20	47.351	27.351
gl85struct	reg_8bit	structure	24	43.274	19.274
gl85struct	reg_ctrl	structure	153	690.811	537.811
gl85struct	reg_ram	structure	20	47.351	27.351
gl85struct	shflogic	structure	44	100.233	56.233
gl85struct	sn54181	structure	86	264.355	178.355
gl85struct	SN85150	structure	31	72.259	41.259
gl85struct	sp_cntrl	structure	56	227.659	171.659
gl85struct	stater	structure	61	160.090	99.090
gl85struct	tempctrl	structure	40	138.946	98.946
gl85struct	vectrgen	structure	26	75.312	49.312
HC11	hc11core	structure	27	31.007	4.007
i8051	I8051_ALL	STR	196	228.088	32.088
Leon	ahbtest	struct	90	63.847	-26.153
Leon	pci_esa	struct	31	51.522	20.522
Leon	pci_is	struct	43	60.965	17.965
Leon	atc25_pciiodpad	syn	9	21.191	12.191
Leon	atc25_pciiopad	syn	7	21.889	14.889
Leon	atc25_pcitoutpad	syn	6	13.348	7.348
Leon	atc25_pcioutpad	syn	4	12.241	8.241
Leon	atc25_odpad	syn	18	15.728	-2.272
Leon	atc25_iodpad	syn	18	25.089	7.089
Leon	atc25_iopadu	syn	16	25.786	9.786
Leon	atc25_iopad	syn	16	25.786	9.786
Leon	atc25_inpad	syn	4	12.241	8.241
Leon	atc25_smpad	syn	4	12.241	8.241
Leon	atc25_outpad	syn	15	14.498	-0.502
Leon	atc25_toutpadu	syn	16	16.426	0.426
Leon	umc18_smiopad	syn	10	23.218	13.218
Leon	atc35_odpad	syn	18	15.728	-2.272
Leon	atc35_iodpad	syn	18	25.089	7.089
Leon	atc35_iopad	syn	16	25.786	9.786
Leon	atc35_toutpadu	syn	16	16.426	0.426
Leon	atc35_outpad	syn	15	14.498	-0.502
Leon	atc35_smpad	syn	4	12.241	8.241
Leon	atc35_inpad	syn	4	12.241	8.241
Leon	fs90_odpad	syn	21	15.62	-5.38
Leon	fs90_iodpad	syn	21	25.391	4.391

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	fs90_smiopad	syn	21	26.088	5.088
Leon	fs90_iopad	syn	21	26.088	5.088
Leon	fs90_toutpadu	syn	21	17.958	-3.042
Leon	fs90_outpad	syn	21	15.620	-5.380
Leon	fs90_smpad	syn	7	13.471	6.471
Leon	fs90_inpad	syn	9	13.471	4.471
Leon	umc18_odpad	syn	16	14.498	-1.502
Leon	umc18_iopad	syn	18	25.786	7.786
Leon	umc18_toutpadu	syn	18	17.656	-0.344
Leon	umc18_outpad	syn	15	14.498	-0.502
Leon	umc18_smpad	syn	4	12.241	8.241
Leon	umc18_inpad	syn	4	12.241	8.241
PIC16C5X	pic_core	structural	419	353.944	-65.056
T80	NoICE	struct	152	74.912	-77.088
Leon	umc18_iodpad	syn	18	25.089	7.089
ans_RISC8	reg_top	STRUCT_REG_TOP	309	269.636	-39.364
ans_RISC8	inst_decoder	STRUCT	320	264.838	-55.162

Real sizes versus estimated sizes for data-flow architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
ans_RISC8	reg_w	rtl_reg_w	19	79.164	60.164
ans_RISC8	reg_status	rtl_reg_status	46	86.476	40.476
ans_RISC8	reg_ioport	rtl_reg_ioport	39	86.364	47.364
ans_RISC8	reg_fsr	rtl_reg_fsr	23	81.253	58.253
ans_RISC8	prog_count	rtl_prog_count	77	94.448	17.448
ans_RISC8	mux_win	rtl_mux_win	52	41.473	-10.527
ans_RISC8	mux_fwe	rtl_mux_fwe	46	37.294	-8.706
ans_RISC8	mux_fin	rtl_mux_fin	48	39.383	-8.617
ans_RISC8	mux_cz_write	rtl_mux_cz_write	45	35.864	-9.136
ans_RISC8	mux_alub	rtl_mux_alub	46	38.339	-7.661
ans_RISC8	mux_alua	rtl_mux_alua	50	40.428	-9.572
ans_RISC8	ir_reg	rtl_ir_reg	40	91.861	51.861
ans_RISC8	ir_decode	rtl_ir_decode	74	73.007	-0.993
ans_RISC8	clock_div	rtl_clock_div	30	83.230	53.230
ans_RISC8	alu_dp	rtl_alu_dp	82	85.319	3.319
ans_RISC8	aluop_gen	rtl_aluop_gen	61	44.606	-16.394
ax8	A90S1200	rtl	167	182.170	15.170
ax8	A90S2313	rtl	286	205.350	-80.650
ax8	AX8	rtl	647	596.458	-50.542
ax8	AX_ALU	rtl	212	151.772	-60.228
ax8	AX_PCS	rtl	79	93.688	14.688
ax8	AX_Port	rtl	51	145.144	94.144
ax8	AX_RAM	rtl	31	82.583	51.583
ax8	AX_Reg	rtl	202	161.176	-40.824
ax8	AX_Reg	rtl2	210	102.592	-107.408
ax8	AX_TC16	rtl	277	212.548	-64.452
ax8	AX_TC8	rtl	99	144.675	45.675
ax8	AX_UART	rtl	215	387.255	172.255
DLX	dlx	rtl	234	552.528	318.528
ERC32	uart	VHDL_RTL	536	939.905	403.905
HC11	hc11cpu	rtl	2073	1277.454	-795.546
Jane	Neuron	dataflow	292	616.600	324.600
Leon	acache	rtl	174	188.951	14.951
Leon	ahbarb	rtl	186	220.925	34.925
Leon	ahbstat	rtl	69	147.203	78.203
Leon	apbmst	rtl	78	151.430	73.430
Leon	cache	rtl	79	181.350	102.350
Leon	cachemem	rtl	75	103.442	28.442
Leon	clkgen	rtl	30	86.098	56.098
Leon	dcache	rtl	378	176.348	-201.652
Leon	div	rtl	118	200.781	82.781
Leon	fpaux	rtl	42	142.109	100.109

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	fp	rtl	707	202.944	-504.056
Leon	fp1eu	rtl	564	171.025	-392.975
Leon	icache	rtl	230	173.673	-56.327
Leon	ioport	rtl	106	154.14	48.14
Leon	irqctrl	rtl	84	142.284	58.284
Leon	irqctrl2	rtl	111	141.688	30.688
Leon	iu	rtl	2175	939.901	-1235.099
Leon	lconf	rtl	33	75.665	42.665
Leon	leon	rtl	85	129.878	44.878
Leon	leon_pci	rtl	215	271.994	56.994
Leon	mcore	rtl	187	182.904	-4.096
Leon	mctrl	rtl	553	170.225	-382.775
Leon	fpu	rtl	32	39.818	7.818
Leon	mul	rtl	265	160.949	-104.051
Leon	GEN_XOR2	rtl	2	19.535	17.535
Leon	GEN_OR2	rtl	2	19.535	17.535
Leon	GEN_AND2	rtl	2	19.535	17.535
Leon	pci_arb	rtl	150	261.990	111.990
Leon	proc	rtl	163	244.636	81.636
Leon	rstgen	rtl	20	80.121	60.121
Leon	atc25_regfile_iu	rtl	57	54.620	-2.380
Leon	atc25_regfile_cp	rtl	51	94.651	43.651
Leon	atc25_syncram	rtl	56	114.551	58.551
Leon	pp33t015vt	rtl	3	19.535	16.535
Leon	pp33b015vt	rtl	8	24.270	16.270
Leon	pp33o01	rtl	2	18.491	16.491
Leon	pt33b04u	rtl	8	24.270	16.270
Leon	pt33b03u	rtl	8	24.270	16.270
Leon	pt33b02u	rtl	8	24.270	16.270
Leon	pt33b01u	rtl	8	24.270	16.270
Leon	pt33b04	rtl	8	24.270	16.270
Leon	pt33b03	rtl	8	24.270	16.270
Leon	pt33b02	rtl	8	24.270	16.270
Leon	pt33b01	rtl	8	24.270	16.270
Leon	pt33t03u	rtl	3	19.535	16.535
Leon	pt33t02u	rtl	3	19.535	16.535
Leon	pt33t01u	rtl	3	19.535	16.535
Leon	pt33o04	rtl	2	18.491	16.491
Leon	atc35_regfile_cp	rtl	41	96.811	55.811
Leon	atc35_regfile	rtl	58	119.684	61.684
Leon	atc35_syncram	rtl	36	36.299	0.299
Leon	pt3b03	rtl	8	24.270	16.270
Leon	pt3b02	rtl	8	24.270	16.270
Leon	pt3b01	rtl	8	24.270	16.270
Leon	pc3t03u	rtl	3	19.535	16.535
Leon	pc3t02u	rtl	3	19.535	16.535
Leon	pc3t01u	rtl	3	19.535	16.535
Leon	fs90_regfile	rtl	55	93.992	38.992
Leon	fs90_syncram	rtl	50	78.722	28.722
Leon	uyfaa	rtl	14	21.625	7.625
Leon	vyfa2gsa	rtl	12	23.714	11.714
Leon	genodpad	rtl	2	18.491	16.491
Leon	geniopad	rtl	5	24.27	19.27
Leon	geniodpad	rtl	5	23.225	18.225
Leon	gentoutpadu	rtl	3	19.535	16.535
Leon	genoutpad	rtl	2	18.491	16.491
Leon	gensmpad	rtl	2	18.491	16.491
Leon	generic_regfile_iu	rtl	89	110.766	21.766
Leon	generic_regfile_cp	rtl	36	58.579	22.579
Leon	generic_smult	rtl	21	80.778	59.778
Leon	geninpad	rtl	2	18.491	16.491
Leon	pciiodpad	rtl	14	29.099	15.099

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	pciiopad	rtl	19	30.864	11.864
Leon	pcitoutpad	rtl	12	25.409	13.409
Leon	pcioutpad	rtl	12	23.645	11.645
Leon	iodpad	rtl	24	39.240	15.240
Leon	odpad	rtl	24	32.705	8.705
Leon	smiopad	rtl	29	42.084	13.084
Leon	iopad	rtl	29	42.084	13.084
Leon	toutpadu	rtl	24	35.550	11.550
Leon	outpad	rtl	24	32.705	8.705
Leon	smpad	rtl	21	31.376	10.376
Leon	inpad	rtl	21	31.376	10.376
Leon	hw_smult	rtl	30	36.879	6.879
Leon	regfile_cp	rtl	37	40.501	3.501
Leon	umc18_regfile	rtl	33	37.832	4.832
Leon	umc18_syncram	rtl	50	42.836	-7.164
Leon	OR2DL	rtl	2	19.535	17.535
Leon	EXOR2DL	rtl	2	19.535	17.535
Leon	AND2DL	rtl	2	19.535	17.535
Leon	INVDL	rtl	2	18.491	16.491
Leon	C3B42	rtl	8	24.270	16.270
Leon	CD3O40T	rtl	7	18.491	11.491
Leon	CD3O20T	rtl	7	18.491	11.491
Leon	CD3O10T	rtl	7	18.491	11.491
Leon	CD3B40T	rtl	8	24.270	16.270
Leon	CD3B20T	rtl	8	24.270	16.270
Leon	CD3B10T	rtl	8	24.270	16.270
Leon	C3B40	rtl	8	24.270	16.270
Leon	C3B20	rtl	8	24.270	16.270
Leon	C3B10	rtl	8	24.270	16.270
Leon	C3B40U	rtl	8	24.270	16.270
Leon	timers	rtl	161	142.558	-18.442
Leon	uart	rtl	267	147.069	-119.931
Leon	wprot	rtl	94	144.966	50.966
PIC16C5X	fadr_mux	dataflow	12	19.535	7.535
PIC16C5X	pic_alu	dataflow	78	25.032	-52.968
PIC16C5X	reg_cons	dataflow	12	19.535	7.535
ppx16	P16C55	rtl	149	197.494	48.494
ppx16	P16F84	rtl	186	246.931	60.931
ppx16	PPX16	rtl	235	479.081	244.081
ppx16	PPX_ALU	rtl	216	91.769	-124.231
ppx16	PPX_Ctrl	rtl	57	38.560	-18.440
ppx16	PPX_PCS	rtl	99	154.931	55.931
ppx16	PPX_Port	rtl	50	147.706	97.706
ppx16	PPX_RAM	rtl	39	86.286	47.286
ppx16	PPX_TMR	rtl	92	147.809	55.809
rd1007	sd_cnfg	RTL	118	144.948	26.948
rd1007	sd_rfrsh	RTL	40	138.793	98.793
rd1007	sd_sig	RTL	197	746.233	549.233
rd1007	sd_state	RTL	60	141.541	81.541
rd1007	sd_top	RTL	124	157.474	33.474
T51	I8052	rtl	145	185.076	40.076
T51	T51	rtl	848	753.369	-94.631
T51	T51_ALU	rtl	404	217.273	-186.727
T51	T51_Port	rtl	43	144.762	101.762
T51	T51_RAM	rtl	55	88.738	33.738
T80	MonZ80	rtl	882	77.075	-804.925
T80	T80	rtl	865	424.490	-440.510
T80	T80a	rtl	156	403.968	247.968
T80	T80s	rtl	97	109.062	12.062
T80	T80_ALU	rtl	265	91.870	-173.130
T80	T80_MCode	rtl	1333	161.994	-1171.006
xapp333	micro_master_tb	RTL	299	275.934	-23.066

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
xapp333	micro_slave_tb	RTL	202	275.934	73.934
xapp333	micro_tb	RTL	358	275.934	-82.066
xapp358	receive	receive_rtl	163	84.275	-78.725
xapp363	clk_top	rtl	41	138.793	97.793
xapp363	gpio_top	rtl	22	85.319	63.319
xapp363	sam_top	rtl	418	532.217	114.217
xapp363	smedia_state	rtl	298	268.272	-29.728
xapp363	smedia_top	rtl	113	320.649	207.649
xapp363	spi	rtl	83	321.086	238.086
xapp363	spi_switch	rtl	55	91.202	36.202
xapp363	ssp_icc	rtl	631	502.993	-128.007
xapp363	ssp_icc_switch	rtl	115	92.905	-22.095
xapp365	iso9141	rtl	213	438.574	225.574
Leon	GEN_NOT	rtl	2	18.491	16.491
Leon	pc3d01	rtl	2	18.491	16.491
Leon	pt33o03	rtl	2	18.491	16.491
Leon	pt33o02	rtl	2	18.491	16.491
Leon	pt33o01	rtl	2	18.491	16.491
Leon	pt33d20u	rtl	3	22.181	19.181
Leon	pt33d20	rtl	2	18.491	16.491
Leon	pt33d00u	rtl	3	22.181	19.181
Leon	pt33d00	rtl	2	18.491	16.491
Leon	pt3o03	rtl	2	18.491	16.491
Leon	pt3o02	rtl	2	18.491	16.491
Leon	pt3o01	rtl	2	18.491	16.491
Leon	pc3d21	rtl	2	18.491	16.491
Leon	wyfa2gsa	rtl	18	31.582	13.582
Leon	rfbypass	rtl	44	147.334	103.334
Leon	regfile_iu	rtl	47	51.629	4.629
Leon	C3B20U	rtl	8	24.270	16.270
Leon	C3B10U	rtl	8	24.270	16.270
Leon	C3O40	rtl	2	18.491	16.491
Leon	C3O20	rtl	2	18.491	16.491
Leon	C3O10	rtl	2	18.491	16.491
Leon	C3I42	rtl	2	18.491	16.491
Leon	C3I40	rtl	2	18.491	16.491

14.10 K3I

14.10.1 Result summary

Population statistical properties and model accuracy:

	All architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	121.865	114.243	-7.622
Variance	82132.638	6905.120	66684.773
Standard deviation	286.588	83.097	258.234
	Behavioral architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	148.815	129.455	-19.360
Variance	121245.948	5973.264	103554.023
Standard deviation	348.204	77.287	321.798
	Structural architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	54.806	64.547	9.741
Variance	5960.385	3502.494	3481.460
Standard deviation	77.204	59.182	59.004
	Data-flow architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	124.114	121.259	-2.855
Variance	72713.624	8148.742	56068.995
Standard deviation	269.655	90.270	236.789

Correlation between estimated and real values:

	Correlation coefficient $(L, \hat{L})$
All architectures	0.4693
Behavioral architectures only	0.4397
Structural architectures only	0.6546
Data-flow architectures only	0.5093

Figure 14.42: Model K3I: Real vs. estimated lines of code.

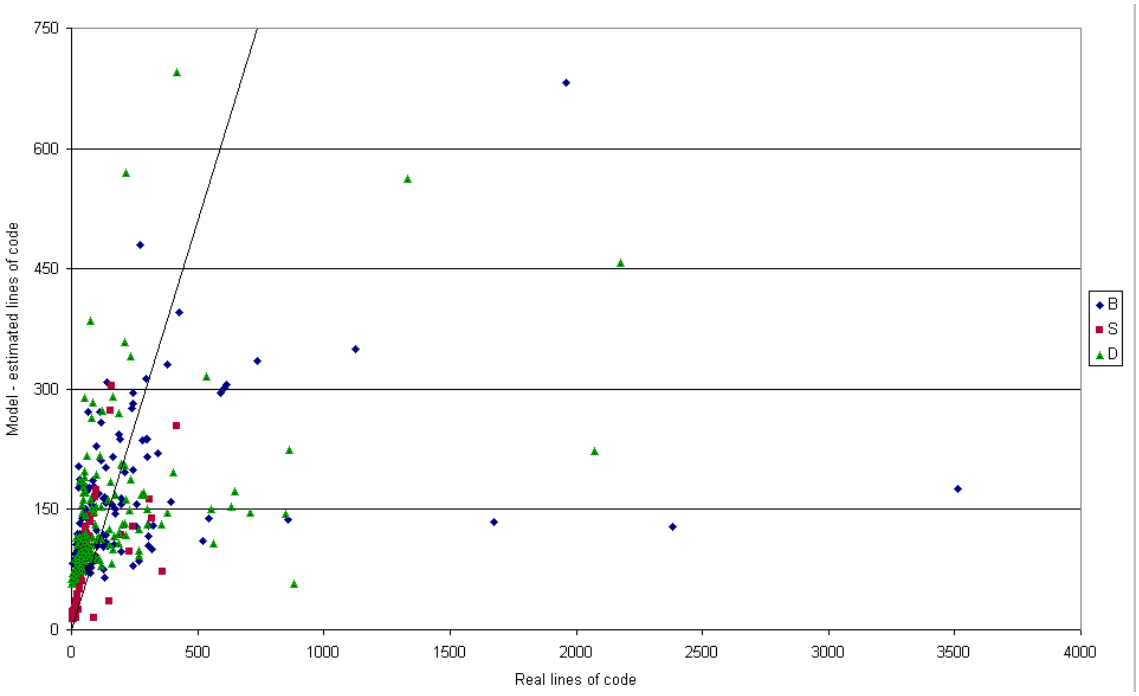


Figure 14.43: Models K3Ib, K3Is, K3Id: Real vs. estimated lines of code.

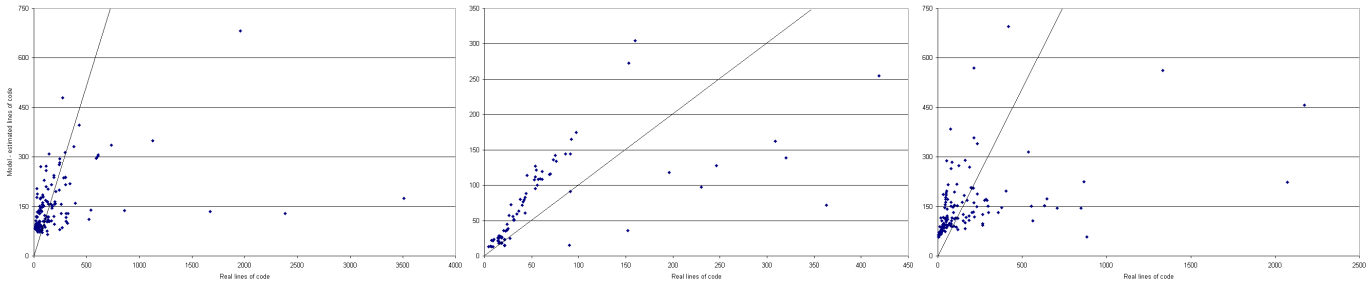


Figure 14.44: Model K3I: Error density distribution.

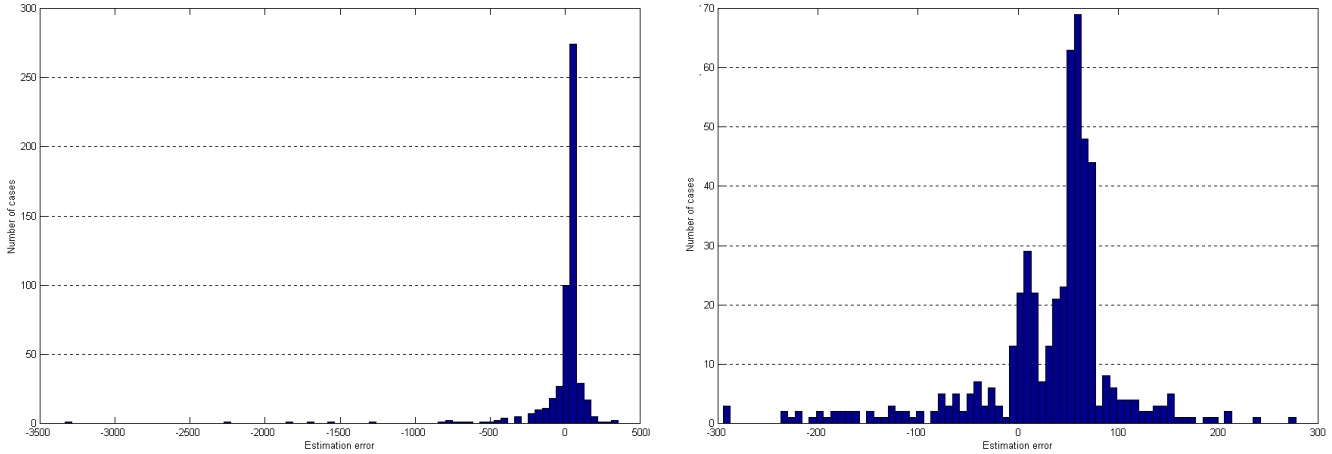


Figure 14.45: Model K3I: Error cumulative distribution.

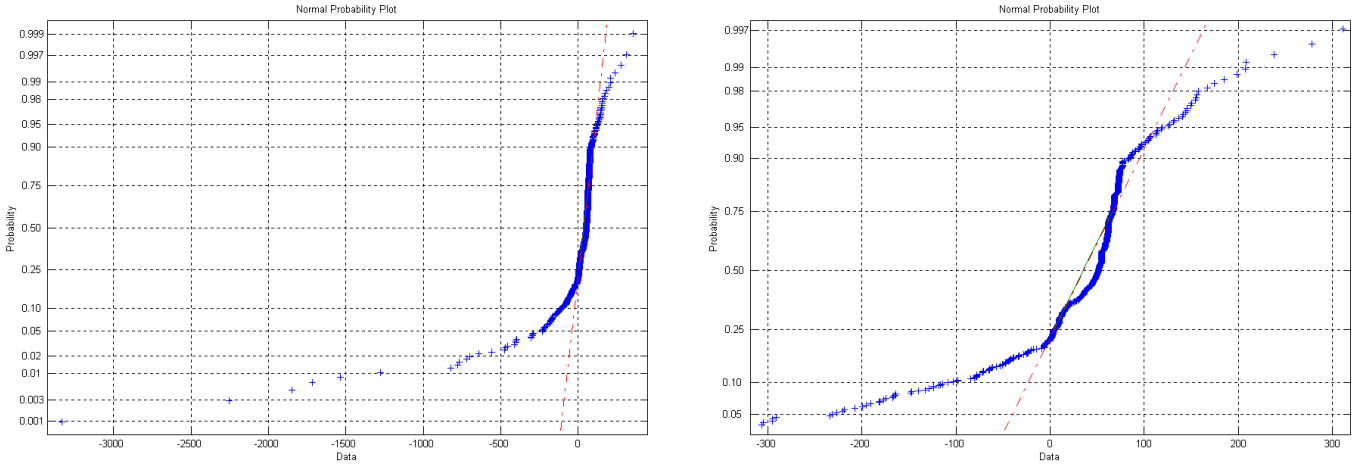
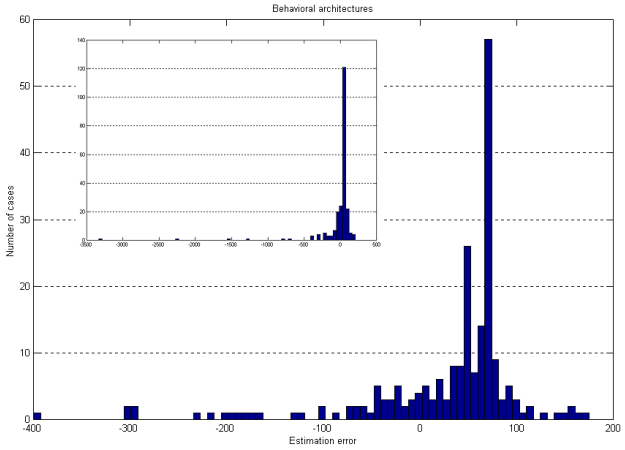
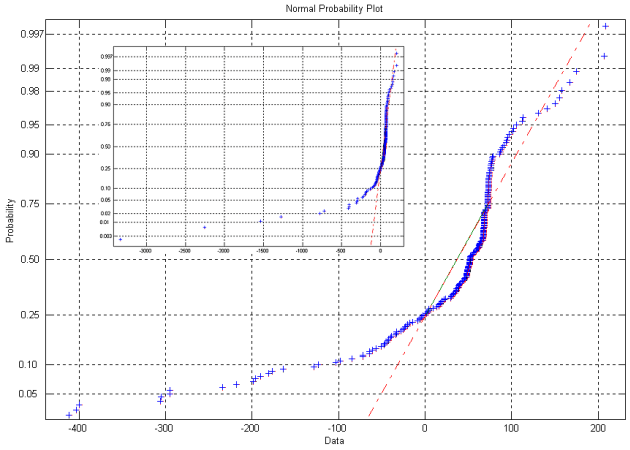
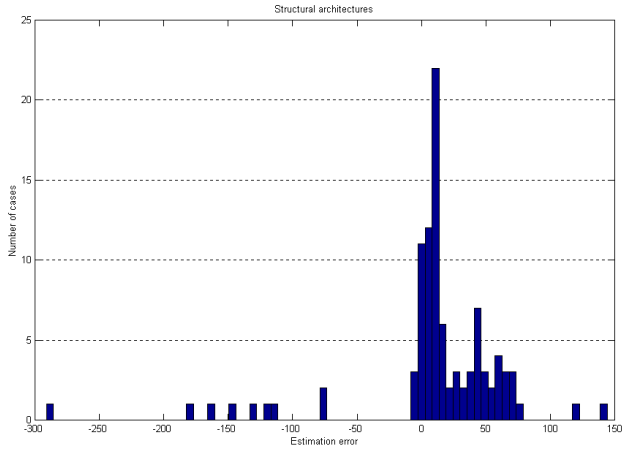
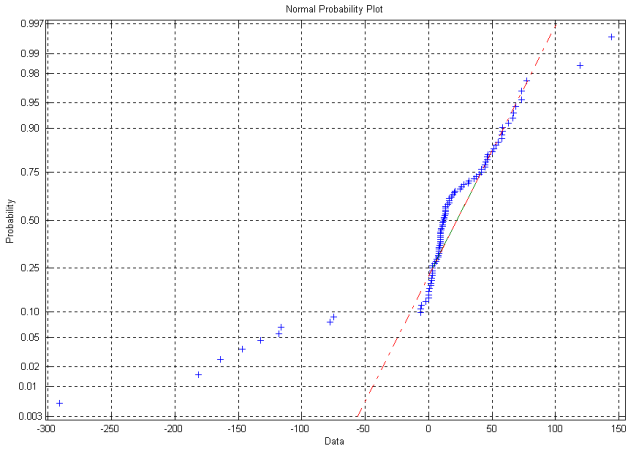


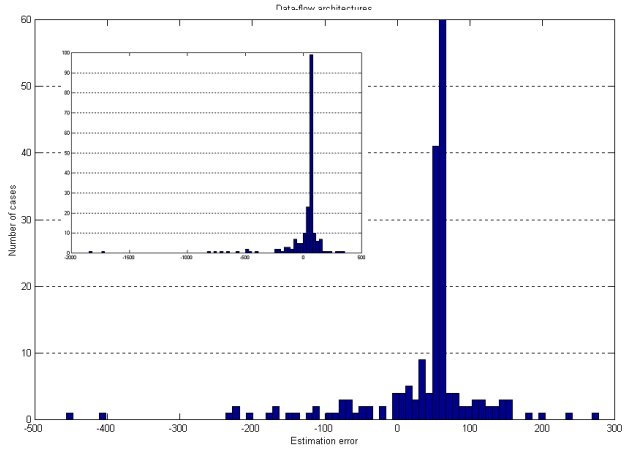
Figure 14.46: Models K3Ib, K3Is, K3Id: Error density and cumulative distributions.

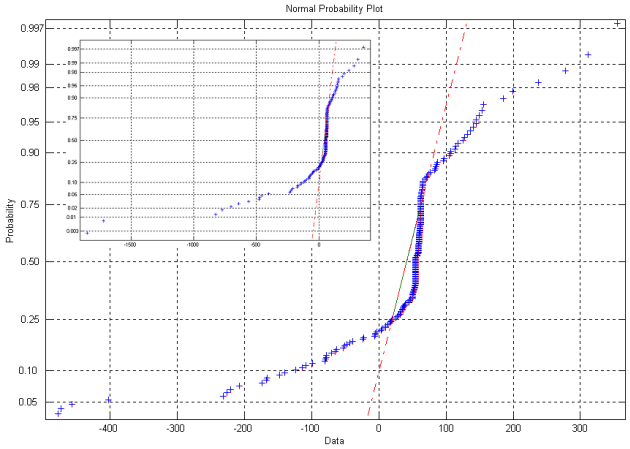












14.10.2 Detailed results

Real sizes versus estimated sizes for behavioral architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
ans_RISC8	reg_files	behave_reg_files	76	76.224	0.224
an-XC2S-USB	xc2sFPGA	BHV	190	237.932	47.932
an-XC2S-XR16	teBL	bhv	302	238.060	-63.940
an-XC2S-XR16	Glue	BHV	109	169.215	60.215
an-XC2S-USB	xc2sFunc	BHV	16	88.795	72.795
an-XC2S-XR16	Core	BHV	210	195.701	-14.299
an-XC2S-USB	teBL	bhv	295	238.060	-56.940
an-XC2S-USB	xc2sCore	BHV	59	144.942	85.942
DLX	alu	behaviour	55	104.390	49.390
DLX	cache	behaviour	260	157.150	-102.850
DLX	clock_gen	behaviour	20	106.866	86.866
DLX	controller	behaviour	735	335.284	-399.716
DLX	dlx	behaviour	542	139.068	-402.932
DLX	dlx_bus_monitor	behaviour	129	64.548	-64.452
DLX	ir	behaviour	59	91.170	32.170
DLX	latch	behaviour	17	89.872	72.872
DLX	memory	behaviour	167	106.158	-60.842
DLX	mux2	behaviour	14	88.076	74.076
DLX	reg_1_out	behaviour	24	88.076	64.076
DLX	reg_2_1_out	behaviour	31	98.353	67.353
DLX	reg_2_out	behaviour	29	92.317	63.317
DLX	reg_3_out	behaviour	34	96.557	62.557
DLX	reg_file	behaviour	41	90.521	49.521
ERC32	AC245Generic	Behavior	67	137.024	70.024
ERC32	AC377Generic	Behavior	46	90.735	44.735
ERC32	FPURTGeneric	vhdl_behavioral	1127	349.268	-777.732
ERC32	IURTGeneric	vhdl_behavioral	1959	682.227	-1276.773
ERC32	RAM8	BEHAVIORAL	57	96.203	39.203
ERC32	TAPTest_iufpu	Behaviour	49	105.045	56.045
gl85	I8085	BEHAVIOR	1673	134.792	-1538.208
HC11	clock	behavoir	24	119.657	95.657
HC11	dev	behavior	48	90.816	42.816
HC11	hc11ram	behavior	41	92.145	51.145
i80386	i80386	behavior	858	137.241	-720.759
i8051	I8051_ALU	BHV	318	99.721	-218.279
i8051	I8051_CTR	BHV	3510	175.251	-3334.749
i8051	I8051_DBG	BHV	244	80.314	-163.686
i8051	I8051_RAM	BHV	195	96.778	-98.222
i8051	I8051_TSB	BHV	54	127.980	73.980
i8051	I8051_XRM	BHV	30	78.701	48.701
Leon	RAM2P_168X32	behav	13	86.766	73.766
Leon	RAM2P_136X32	behav	13	86.766	73.766
Leon	RAM2P_16X32	behav	13	86.766	73.766
Leon	RAM_2048x32	behavioral	14	86.766	72.766
Leon	RAM_1024x32	behavioral	14	86.766	72.766
Leon	RAM_512x30	behavioral	14	86.766	72.766
Leon	RAM_512x28	behavioral	14	86.766	72.766
Leon	RAM_256x30	behavioral	14	86.766	72.766
Leon	RAM_256x28	behavioral	14	86.766	72.766
Leon	RAM_256x26	behavioral	14	86.766	72.766
Leon	atc25_syncram_sim	behavioral	29	80.03	51.03
Leon	atc25_2pram	behav	35	81.359	46.359
Leon	atc35_dpram_ss_dn	behav	38	81.359	43.359
Leon	ATC35_RAM_256x26	behavioral	17	90.358	73.358
Leon	ATC35_RAM_1024x32	behavioral	17	90.358	73.358
Leon	ATC35_RAM_2048x32	behavioral	17	90.358	73.358
Leon	ATC35_RAM_256x28	behavioral	17	90.358	73.358
Leon	ATC35_RAM_1024x34	behavioral	17	90.358	73.358
Leon	ATC35_RAM_2048x34	behavioral	17	90.358	73.358
Leon	DPRAMRWRW_16X32	behav	21	89.211	68.211

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	DPRAMRWRW_136X32	behav	21	89.211	68.211
Leon	DPRAMRWRW_168X32	behav	21	89.211	68.211
Leon	SW204420	behavioral	32	187.696	155.696
Leon	SU004020	behavioral	29	203.858	174.858
Leon	SA108019	behavioral	27	177.765	150.765
Leon	generic_dpram_as	behav	45	83.155	38.155
Leon	R2048x34M8	behavioral	18	86.766	68.766
Leon	RF68X32M1	behav	17	84.970	67.970
Leon	RF68X33M1	behav	17	84.970	67.970
Leon	RF136X32M1	behav	17	84.970	67.970
Leon	RF136X33M1	behav	17	84.970	67.970
Leon	R1024X34M4	behavioral	18	86.766	68.766
Leon	R256X28M4	behavioral	18	86.766	68.766
Leon	R1024X33M4	behavioral	18	86.766	68.766
Leon	R2048X32M8	behavioral	18	86.766	68.766
Leon	R1024X32M4	behavioral	18	86.766	68.766
Leon	R256X26M4	behavioral	18	86.766	68.766
Leon	R256X25M4	behavioral	18	86.766	68.766
Leon	R256X24M4	behavioral	18	86.766	68.766
Leon	umc18_syncram_ss	behavioral	50	80.030	30.030
Leon	umc18_dpram_ss	behav	58	79.564	21.564
Leon	virtex_regfile_cp	behav	35	99.988	64.988
Leon	virtex_regfile	behav	36	98.192	62.192
Leon	virtex_syncram	behav	62	138.196	76.196
Leon	RAMB4_S16_S16	behav	38	72.633	34.633
Leon	RAMB4_S1	behav	14	86.766	72.766
Leon	RAMB4_S2	behav	14	86.766	72.766
Leon	RAMB4_S4	behav	14	86.766	72.766
Leon	RAMB4_S8	behav	14	86.766	72.766
SuperscalarDLX	Dlx	BehaviorPipelined	2383	128.83	-2254.17
SuperscalarDLX	Environment	Behavior	307	116.757	-190.243
T80	NoICE_TB	behaviour	53	129.085	76.085
TE51	te51mux	bhv	27	116.757	89.757
TE51	te51d	BHV	87	178.291	91.291
TE51	te51dec	bhv	267	86.350	-180.650
TE51	te51mcode	BHV	522	110.694	-411.306
TE51	te51alu	BHV	78	86.999	8.999
TE51	te51regs	BHV	127	103.961	-23.039
TE51	te51ctrl	bhv	394	160.069	-233.931
TE51	te51c	BHV	189	243.929	54.929
xapp146	MULTI_DVM	BEHAVE	590	295.238	-294.762
xapp146	MULTI_DVM_TB	BEHAVIOR	87	105.045	18.045
xapp146	SHIFT16	DEFINITION	29	77.371	48.371
xapp146	SHIFT8b	DEFINITION2	31	83.408	52.408
xapp146	TOP_LEVEL	BEHAVE	141	308.585	167.585
xapp146	TOP_LEVEL_TB	BEHAVE	147	105.045	-41.955
xapp146	upcnt5	DEFINITION2	28	94.407	66.407
xapp328	CNT_25	BEHAVIOURAL	47	96.852	49.852
xapp328	cnt3	DEFINITION	24	82.759	58.759
xapp328	CNT_5	BEHAVIOURAL	29	80.963	51.963
xapp328	command_state_machine	BEHAVIOURAL	108	111.794	3.794
xapp328	DNLD_INTERFACE	BEHAVIOURAL	96	93.684	-2.316
xapp328	FLASH_CNTR	BEHAVIOURAL	378	330.972	-47.028
xapp328	i2c_master	behave	343	219.634	-123.366
xapp328	lcd_control	behave	129	164.922	35.922
xapp328	main_ctrl_state_machine	behave	161	156.792	-4.208
xapp328	pxa_bufif2	behavioral	10	82.759	72.759
xapp328	pxa_mux	behavioral	7	82.759	75.759
xapp328	pxa_dff_apar_p0	behavioral	18	80.963	62.963
xapp328	pxa_tff_apar_p0	behavioral	20	80.963	60.963
xapp328	mpeg_chip_ctrl	behave	139	105.333	-33.667
xapp328	on_off_logic	behave	27	82.759	55.759

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
xapp328	PARALLEL_PORT	BEHAVIOURAL	63	93.684	30.684
xapp328	play_logic_state_machine	behave	80	83.408	3.408
xapp328	play_modes	behave	126	75.575	-50.425
xapp328	power_ctrl	behave	59	88.795	29.795
xapp328	SHIFT8	DEFINITION	31	83.408	52.408
xapp328	sound_control	behave	125	164.136	39.136
xapp328	upcnt2	DEFINITION	28	79.167	51.167
xapp328	upcnt3	DEFINITION	28	79.167	51.167
xapp328	upcnt4	DEFINITION	28	79.167	51.167
xapp328	updwncnt4	DEFINITION	32	80.963	48.963
xapp333	i2c	behave	164	215.713	51.713
xapp333	i2c_control	behave	612	306.134	-305.866
xapp333	SHIFT8	DEFINITION	31	83.408	52.408
xapp333	uC_interface	BEHAVIOUR	243	282.414	39.414
xapp333	upcnt4	DEFINITION	28	94.407	66.407
xapp345	irda_uart	behavior	71	176.495	105.495
xapp345	irda_uart_tb	behavior	101	105.045	4.045
xapp345	jk_ff	behavior	35	80.963	45.963
xapp345	rxcver	behavior	125	105.108	-19.892
xapp345	sirendec	behavior	135	109.934	-25.066
xapp345	txmit	behavior	105	86.999	-18.001
xapp345	uart	behavior	63	176.495	113.495
xapp345	uart_tb	behavior	101	105.045	4.045
xapp348	sck_logic	DEFINITION	138	157.377	19.377
xapp348	spi_control_sm	DEFINITION	242	199.432	-42.568
xapp348	spi_rcv_shift_reg	DEFINITION	76	78.020	2.020
xapp348	spi_xmit_shift_reg	DEFINITION	41	73.780	32.780
xapp348	uC_interface	BEHAVIOUR	294	313.617	19.617
xapp348	upcnt4	DEFINITION	24	97.999	73.999
xapp348	upcnt5	DEFINITION	24	97.999	73.999
xapp349	uC_interface	BEHAVIOUR	257	128.332	-128.668
xapp354	am30lv0064d	behavior	22	99.511	77.511
xapp354	AMD_FLASH_TB	BEHAVIOR	325	129.309	-195.691
xapp354	NAND_INTERFACE	BEHAVIOR	138	117.830	-20.170
xapp354	k9f4008w0a	behavior	22	99.511	77.511
xapp354	NAND_FLASH_TB	BEHAVIOR	306	129.309	-176.691
xapp355	ADC_INTERFACE	BEHAVE	590	295.238	-294.762
xapp355	ADC_INTERFACE_TB	BEHAVIOR	87	105.045	18.045
xapp355	SHIFT16	DEFINITION	29	77.371	48.371
xapp355	SHIFT8	DEFINITION	31	83.408	52.408
xapp355	TOP_LEVEL	BEHAVE	114	272.068	158.068
xapp355	TOP_LEVEL_TB	BEHAVE	144	105.045	-38.955
xapp355	upcnt5	DEFINITION	28	94.407	66.407
xapp356	ADC_INTERFACE	BEHAVIOR	300	215.744	-84.256
xapp356	SHIFT16	DEFINITION	28	77.371	49.371
xapp356	SHIFT8	DEFINITION	30	83.408	53.408
xapp356	TEMP_INTERFACE	BEHAVIOR	100	123.379	23.379
xapp356	TOP_LEVEL	BEHAVIOR	240	276.309	36.309
xapp356	TOP_LEVEL_TB	BEHAVIOR	304	105.045	-198.955
xapp356	UPCNT11	DEFINITION	27	94.407	67.407
xapp356	UPCNT15	DEFINITION	27	94.407	67.407
xapp356	UPCNT5	DEFINITION	27	94.407	67.407
xapp356	XPATH	BEHAVIOR	429	396.234	-32.766
xapp357	CLK_DIVIDER	DEFINITION	16	86.35	70.35
xapp357	LED_TEST	BEHAVIOUR	116	210.757	94.757
xapp358	tx_rx_entity	behavioral	60	172.679	112.679
xapp363	clk_gen	behavioral	17	94.182	77.182
xapp365	ISO_CLK_DIVIDER	DEFINITION	55	102.814	47.814
xapp367	AudioController	DEFINITION	21	96.627	75.627
xapp367	chatterbox	BEHAVIOUR	271	479.534	208.534
xapp367	DTMFController	DEFINITION	31	132.421	101.421
xapp367	FlipFlop	definition	15	84.554	69.554

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
xapp367	FlipFlopR	definition	17	84.554	67.554
xapp367	IrqController	BEHAVIOUR	83	185.648	102.648
xapp367	MemoryManager	BEHAVE	98	229.228	131.228
xapp367	PowerSupplyController	DEFINITION	20	88.795	68.795
xapp367	RFTransceiverController	DEFINITION	31	118.977	87.977
xapp369	DECODE_MAN	BEHAVE	283	236.710	-46.290
xapp369	TOP_LEVEL	BEHAVE	117	258.199	141.199
xapp370	CLK_DIVIDER	DEFINITION	19	92.386	73.386
xapp370	cooltrak	BEHAVIOUR	245	294.625	49.625
xapp370	MULTI_DVM	BEHAVE	606	301.274	-304.726
xapp370	SHIFT16	DEFINITION	29	77.371	48.371
xapp370	SHIFT8	DEFINITION	26	83.408	57.408
xapp370	SPEED	DEFINITION	45	84.554	39.554
xapp370	upcnt5	DEFINITION	28	94.407	66.407
xapp336	DEC_16B20B	BEHAVIOUR	80	148.300	68.300
xapp336	DEC_FUNC	BEHAVIOUR	191	119.128	-71.872
xapp336	DIS_GEN_LOW	BEHAVIOUR	86	154.148	68.148
xapp336	DIS_GEN_UP	BEHAVIOUR	83	161.98	78.98
xapp336	ENC_16B20B	BEHAVIOUR	74	135.314	61.314
xapp336	ENC_FUNC	BEHAVIOUR	126	117.631	-8.369
xapp336	ERR_CHECK	BEHAVIOUR	77	70.837	-6.163
xapp336	ERR_DET	BEHAVIOUR	60	73.780	13.780
xapp336	DECODER	BEHAVIOUR	175	150.978	-24.022
xapp336	ENCODER_LOW	BEHAVIOUR	196	156.453	-39.547
xapp336	ENCODER_UP	BEHAVIOUR	197	164.285	-32.715
xapp336	MAIN_TB	BEHAVIOUR	88	150.915	62.915
xapp336	S_GEN	BEHAVIOUR	57	72.633	15.633
xapp336_8	ERR_CHECK	BEHAVIOUR	77	70.837	-6.163
xapp336_8	DEC_FUNC	BEHAVIOUR	191	119.128	-71.872
xapp336_8	DIS_GEN	BEHAVIOUR	79	155.943	76.943
xapp336_8	ENC_FUNC	BEHAVIOUR	135	202.737	67.737
xapp336_8	DECODER	BEHAVIOUR	173	144.942	-28.058
xapp336_8	ENCODER	BEHAVIOUR	202	158.249	-43.751
xapp336_8	MAIN_TB	BEHAVIOUR	92	150.915	58.915
xapp336_8	S_GEN	BEHAVIOUR	57	72.633	15.633
Leon	fs90_dpram_ss	behav	40	75.972	35.972
Leon	fs90_syncram_sim	behavioral	34	80.030	46.030
Leon	generic_syncram	behavioral	41	81.826	40.826
Leon	generic_dpram_ss	behav	48	83.155	35.155
Leon	syncram	behav	44	138.196	94.196
TE51	te51	BHV	54	151.116	97.116
Leon	RAMB4_S16	behav	14	86.766	72.766
xapp354	am30lv0064d_top	behavior	64	271.382	207.382

Real sizes versus estimated sizes for structural architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
ans_RISC8	ans_risc8	STRUCT_ANS_RISC8	363	71.886	-291.114
ans_RISC8	alu	STRUCT_ALU	230	97.499	-132.501
ans_RISC8	control	STRUCT_CONTROL	246	127.871	-118.129
gl85struct	acc_ctrl	structure	41	76.941	35.941
gl85struct	alulogic	structure	70	115.991	45.991
gl85struct	ALU_8BIT	structure	43	82.725	39.725
gl85struct	alu_ctrl	structure	61	108.040	47.040
gl85struct	bc_pc_sp	structure	38	79.807	41.807
gl85struct	buf8	structure	15	24.961	9.961
gl85struct	ctl_lgc1	structure	28	72.784	44.784
gl85struct	ctl_lgc2	structure	45	113.737	68.737
gl85struct	dataaddr	structure	54	111.694	57.694
gl85struct	decod2_4	structure	14	21.665	7.665
gl85struct	DECOD3_8	structure	24	37.194	13.194
gl85struct	flagunit	structure	97	174.435	77.435
gl85struct	g16bctr	structure	36	63.516	27.516
gl85struct	g4bctr	structure	69	115.014	46.014
gl85struct	gl85	structure	91	90.883	-0.117
gl85struct	hldlogic	structure	14	25.850	11.850

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
gl85struct	hltlogic	structure	16	28.196	12.196
gl85struct	hl_de_wz	structure	53	107.926	54.926
gl85struct	inst_reg	structure	34	59.098	25.098
gl85struct	interupt	structure	57	108.393	51.393
gl85struct	intrupt1	structure	59	109.094	50.094
gl85struct	intrupt2	structure	54	95.478	41.478
gl85struct	intrupt3	structure	42	79.382	37.382
gl85struct	inv8	structure	15	24.961	9.961
gl85struct	M5	structure	22	35.294	13.294
gl85struct	mcdecode	structure	91	143.994	52.994
gl85struct	MUX2TO1	STR_MUX2TO1	15	21.049	6.049
gl85struct	mux_4bit	structure	24	36.244	12.244
gl85struct	ocnand	structure	18	25.658	7.658
gl85struct	oprlogic	structure	54	127.258	73.258
gl85struct	parity1	structure	15	28.196	13.196
gl85struct	pc_cntrl	structure	92	165.052	73.052
gl85struct	prioenco	structure	25	38.337	13.337
gl85struct	rdwrgen	structure	76	134.211	58.211
gl85struct	reg8bits	structure	30	56.123	26.123
gl85struct	regctrl0	structure	73	135.77	62.77
gl85struct	regctrl1	structure	160	304.257	144.257
gl85struct	regctrl2	structure	55	121.389	66.389
gl85struct	regpad	structure	75	141.950	66.950
gl85struct	regpair	structure	20	36.094	16.094
gl85struct	regpairs	structure	20	36.094	16.094
gl85struct	reg_8bit	structure	24	44.587	20.587
gl85struct	reg_ctrl	structure	153	272.468	119.468
gl85struct	reg_ram	structure	20	36.094	16.094
gl85struct	shflogic	structure	44	88.271	44.271
gl85struct	sn54181	structure	86	143.967	57.967
gl85struct	SN85150	structure	31	50.065	19.065
gl85struct	sp_cntrl	structure	56	100.313	44.313
gl85struct	stater	structure	61	119.309	58.309
gl85struct	tempctrl	structure	40	72.058	32.058
gl85struct	vectrgen	structure	26	57.369	31.369
HC11	hc11core	structure	27	24.811	-2.189
i8051	I8051_ALL	STR	196	118.148	-77.852
Leon	ahbtest	struct	90	15.181	-74.819
Leon	pci_esa	struct	31	51.522	20.522
Leon	pci_is	struct	43	60.965	17.965
Leon	atc25_pciiodpad	syn	9	21.555	12.555
Leon	atc25_pciiopad	syn	7	22.253	15.253
Leon	atc25_pcitoutpad	syn	6	14.122	8.122
Leon	atc25_pcioutpad	syn	4	13.425	9.425
Leon	atc25_odpad	syn	18	18.050	0.050
Leon	atc25_iodpad	syn	18	26.181	8.181
Leon	atc25_iopadu	syn	16	26.878	10.878
Leon	atc25_iopad	syn	16	26.878	10.878
Leon	atc25_inpad	syn	4	13.425	9.425
Leon	atc25_smpad	syn	4	13.425	9.425
Leon	atc25_outpad	syn	15	18.050	3.050
Leon	atc25_toutpadu	syn	16	18.748	2.748
Leon	umc18_smiopad	syn	10	23.582	13.582
Leon	atc35_odpad	syn	18	18.050	0.050
Leon	atc35_iodpad	syn	18	26.181	8.181
Leon	atc35_iopad	syn	16	26.878	10.878
Leon	atc35_toutpadu	syn	16	18.748	2.748
Leon	atc35_outpad	syn	15	18.050	3.050
Leon	atc35_smpad	syn	4	13.425	9.425
Leon	atc35_inpad	syn	4	13.425	9.425
Leon	fs90_odpad	syn	21	14.754	-6.246
Leon	fs90_iodpad	syn	21	22.885	1.885

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	fs90_smiopad	syn	21	23.582	2.582
Leon	fs90_iopad	syn	21	23.582	2.582
Leon	fs90_toutpadu	syn	21	15.452	-5.548
Leon	fs90_outpad	syn	21	14.754	-6.246
Leon	fs90_smpad	syn	7	13.425	6.425
Leon	fs90_inpad	syn	9	13.425	4.425
Leon	umc18_odpad	syn	16	18.050	2.050
Leon	umc18_iopad	syn	18	26.878	8.878
Leon	umc18_toutpadu	syn	18	18.748	0.748
Leon	umc18_outpad	syn	15	18.050	3.050
Leon	umc18_smpad	syn	4	13.425	9.425
Leon	umc18_inpad	syn	4	13.425	9.425
PIC16C5X	pic_core	structural	419	254.511	-164.489
T80	NoICE	struct	152	35.589	-116.411
Leon	umc18_iodpad	syn	18	26.181	8.181
ans_RISC8	reg_top	STRUCT_REG_TOP	309	161.979	-147.021
ans_RISC8	inst_decoder	STRUCT	320	138.504	-181.496

Real sizes versus estimated sizes for data-flow architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
ans_RISC8	reg_w	rtl_reg_w	19	69.790	50.790
ans_RISC8	reg_status	rtl_reg_status	46	114.421	68.421
ans_RISC8	reg_ioport	rtl_reg_ioport	39	113.313	74.313
ans_RISC8	reg_fsr	rtl_reg_fsr	23	82.542	59.542
ans_RISC8	prog_count	rtl_prog_count	77	162.932	85.932
ans_RISC8	mux_win	rtl_mux_win	52	197.306	145.306
ans_RISC8	mux_fwe	rtl_mux_fwe	46	171.803	125.803
ans_RISC8	mux_fin	rtl_mux_fin	48	184.554	136.554
ans_RISC8	mux_cz_write	rtl_mux_cz_write	45	162.932	117.932
ans_RISC8	mux_alub	rtl_mux_alub	46	178.179	132.179
ans_RISC8	mux_alua	rtl_mux_alua	50	190.93	140.93
ans_RISC8	ir_reg	rtl_ir_reg	40	146.578	106.578
ans_RISC8	ir_decode	rtl_ir_decode	74	385.260	311.260
ans_RISC8	clock_div	rtl_clock_div	30	94.185	64.185
ans_RISC8	alu_dp	rtl_alu_dp	82	106.937	24.937
ans_RISC8	aluop_gen	rtl_aluop_gen	61	216.433	155.433
ax8	A90S1200	rtl	167	116.331	-50.669
ax8	A90S2313	rtl	286	171.532	-114.468
ax8	AX8	rtl	647	173.400	-473.600
ax8	AX_ALU	rtl	212	133.025	-78.975
ax8	AX_PCS	rtl	79	151.510	72.510
ax8	AX_Port	rtl	51	289.026	238.026
ax8	AX_RAM	rtl	31	83.871	52.871
ax8	AX_Reg	rtl	202	205.289	3.289
ax8	AX_Reg	rtl2	210	205.289	-4.711
ax8	AX_TC16	rtl	277	168.200	-108.800
ax8	AX_TC8	rtl	99	111.926	12.926
ax8	AX_UART	rtl	215	161.824	-53.176
DLX	dlx	rtl	234	340.747	106.747
ERC32	uart	VHDL_RTL	536	315.399	-220.601
HC11	hc11cpu	rtl	2073	223.587	-1849.413
Jane	Neuron	dataflow	292	167.905	-124.095
Leon	acache	rtl	174	167.978	-6.022
Leon	ahbarb	rtl	186	108.268	-77.732
Leon	ahbstat	rtl	69	98.461	29.461
Leon	apbmst	rtl	78	97.357	19.357
Leon	cache	rtl	79	264.271	185.271
Leon	cachemem	rtl	75	91.038	16.038
Leon	clkgen	rtl	30	76.724	46.724
Leon	dcache	rtl	378	146.821	-231.179
Leon	div	rtl	118	79.570	-38.430
Leon	fpaux	rtl	42	74.151	32.151

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	fp	rtl	707	145.523	-561.477
Leon	fp1eu	rtl	564	107.243	-456.757
Leon	icache	rtl	230	149.478	-80.522
Leon	ioport	rtl	106	116.060	10.060
Leon	irqctrl	rtl	84	88.211	4.211
Leon	irqctrl2	rtl	111	87.615	-23.385
Leon	iu	rtl	2175	456.987	-1718.013
Leon	lconf	rtl	33	116.615	83.615
Leon	leon	rtl	85	284.044	199.044
Leon	leon_pci	rtl	215	569.816	354.816
Leon	mcore	rtl	187	269.553	82.553
Leon	mctrl	rtl	553	150.547	-402.453
Leon	fpu	rtl	32	186.219	154.219
Leon	mul	rtl	265	98.298	-166.702
Leon	GEN_XOR2	rtl	2	63.415	61.415
Leon	GEN_OR2	rtl	2	63.415	61.415
Leon	GEN_AND2	rtl	2	63.415	61.415
Leon	pci_arb	rtl	150	106.742	-43.258
Leon	proc	rtl	163	290.203	127.203
Leon	rstgen	rtl	20	70.747	50.747
Leon	atc25_regfile_iu	rtl	57	110.452	53.452
Leon	atc25_regfile_cp	rtl	51	85.708	34.708
Leon	atc25_syncram	rtl	56	117.345	61.345
Leon	pp33t015vt	rtl	3	63.415	60.415
Leon	pp33b015vt	rtl	8	70.376	62.376
Leon	pp33o01	rtl	2	57.039	55.039
Leon	pt33b04u	rtl	8	70.376	62.376
Leon	pt33b03u	rtl	8	70.376	62.376
Leon	pt33b02u	rtl	8	70.376	62.376
Leon	pt33b01u	rtl	8	70.376	62.376
Leon	pt33b04	rtl	8	70.376	62.376
Leon	pt33b03	rtl	8	70.376	62.376
Leon	pt33b02	rtl	8	70.376	62.376
Leon	pt33b01	rtl	8	70.376	62.376
Leon	pt33t03u	rtl	3	63.415	60.415
Leon	pt33t02u	rtl	3	63.415	60.415
Leon	pt33t01u	rtl	3	63.415	60.415
Leon	pt33o04	rtl	2	57.039	55.039
Leon	atc35_regfile_cp	rtl	41	85.708	44.708
Leon	atc35_regfile	rtl	58	110.452	52.452
Leon	atc35_syncram	rtl	36	98.977	62.977
Leon	pt3b03	rtl	8	70.376	62.376
Leon	pt3b02	rtl	8	70.376	62.376
Leon	pt3b01	rtl	8	70.376	62.376
Leon	pc3t03u	rtl	3	63.415	60.415
Leon	pc3t02u	rtl	3	63.415	60.415
Leon	pc3t01u	rtl	3	63.415	60.415
Leon	fs90_regfile	rtl	55	92.084	37.084
Leon	fs90_syncram	rtl	50	94.385	44.385
Leon	uyfaa	rtl	14	76.166	62.166
Leon	vyfa2gsa	rtl	12	88.918	76.918
Leon	genodpad	rtl	2	57.039	55.039
Leon	geniopad	rtl	5	70.376	65.376
Leon	geniodpad	rtl	5	64.000	59.000
Leon	gentoutpadu	rtl	3	63.415	60.415
Leon	genoutpad	rtl	2	57.039	55.039
Leon	gensmpad	rtl	2	57.039	55.039
Leon	generic_regfile_iu	rtl	89	147.809	58.809
Leon	generic_regfile_cp	rtl	36	99.248	63.248
Leon	generic_smult	rtl	21	66.074	45.074
Leon	geninpad	rtl	2	57.039	55.039
Leon	pciiodpad	rtl	14	73.184	59.184

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	pciiopad	rtl	19	79.560	60.560
Leon	pcitoutpad	rtl	12	72.599	60.599
Leon	pcioutpad	rtl	12	66.223	54.223
Leon	iodpad	rtl	24	88.289	64.289
Leon	odpad	rtl	24	81.328	57.328
Leon	smiopad	rtl	29	94.665	65.665
Leon	iopad	rtl	29	94.665	65.665
Leon	toutpadu	rtl	24	87.704	63.704
Leon	outpad	rtl	24	81.328	57.328
Leon	smpad	rtl	21	79.999	58.999
Leon	inpad	rtl	21	79.999	58.999
Leon	hw_smult	rtl	30	89.034	59.034
Leon	regfile_cp	rtl	37	94.892	57.892
Leon	umc18_regfile	rtl	33	92.084	59.084
Leon	umc18_syncram	rtl	50	103.569	53.569
Leon	OR2DL	rtl	2	63.415	61.415
Leon	EXOR2DL	rtl	2	63.415	61.415
Leon	AND2DL	rtl	2	63.415	61.415
Leon	INVDL	rtl	2	57.039	55.039
Leon	C3B42	rtl	8	70.376	62.376
Leon	CD3O40T	rtl	7	57.039	50.039
Leon	CD3O20T	rtl	7	57.039	50.039
Leon	CD3O10T	rtl	7	57.039	50.039
Leon	CD3B40T	rtl	8	70.376	62.376
Leon	CD3B20T	rtl	8	70.376	62.376
Leon	CD3B10T	rtl	8	70.376	62.376
Leon	C3B40	rtl	8	70.376	62.376
Leon	C3B20	rtl	8	70.376	62.376
Leon	C3B10	rtl	8	70.376	62.376
Leon	C3B40U	rtl	8	70.376	62.376
Leon	timers	rtl	161	83.153	-77.847
Leon	uart	rtl	267	92.996	-174.004
Leon	wprot	rtl	94	90.893	-3.107
PIC16C5X	fadr_mux	dataflow	12	63.415	51.415
PIC16C5X	pic_alu	dataflow	78	96.680	18.680
PIC16C5X	reg_cons	dataflow	12	63.415	51.415
ppx16	P16C55	rtl	149	126.100	-22.900
ppx16	P16F84	rtl	186	120.923	-65.077
ppx16	PPX16	rtl	235	187.356	-47.644
ppx16	PPX_ALU	rtl	216	117.722	-98.278
ppx16	PPX_Ctrl	rtl	57	171.194	114.194
ppx16	PPX_PCS	rtl	99	154.169	55.169
ppx16	PPX_Port	rtl	50	108.630	58.630
ppx16	PPX_RAM	rtl	39	92.906	53.906
ppx16	PPX_TMR	rtl	92	131.054	39.054
rd1007	sd_cnfg	RTL	118	113.313	-4.687
rd1007	sd_rfrsh	RTL	40	76.166	36.166
rd1007	sd_sig	RTL	197	206.733	9.733
rd1007	sd_state	RTL	60	92.799	32.799
rd1007	sd_top	RTL	124	273.47	149.47
T51	I8052	rtl	145	162.348	17.348
T51	T51	rtl	848	144.925	-703.075
T51	T51_ALU	rtl	404	196.476	-207.524
T51	T51_Port	rtl	43	156.011	113.011
T51	T51_RAM	rtl	55	121.018	66.018
T80	MonZ80	rtl	882	57.039	-824.961
T80	T80	rtl	865	225.009	-639.991
T80	T80a	rtl	156	183.855	27.855
T80	T80s	rtl	97	193.527	96.527
T80	T80_ALU	rtl	265	125.263	-139.737
T80	T80_MCode	rtl	1333	562.064	-770.936
xapp333	micro_master_tb	RTL	299	131.701	-167.299

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
xapp333	micro_slave_tb	RTL	202	131.701	-70.299
xapp333	micro_tb	RTL	358	131.701	-226.299
xapp358	receive	receive_rtl	163	100.561	-62.439
xapp363	clk_top	rtl	41	76.166	35.166
xapp363	gpio_top	rtl	22	106.937	84.937
xapp363	sam_top	rtl	418	695.982	277.982
xapp363	smedia_state	rtl	298	150.459	-147.541
xapp363	smedia_top	rtl	113	217.688	104.688
xapp363	spi	rtl	83	115.807	32.807
xapp363	spi_switch	rtl	55	142.697	87.697
xapp363	ssp_icc	rtl	631	152.954	-478.046
xapp363	ssp_icc_switch	rtl	115	152.954	37.954
xapp365	iso9141	rtl	213	358.276	145.276
Leon	GEN_NOT	rtl	2	57.039	55.039
Leon	pc3d01	rtl	2	57.039	55.039
Leon	pt33o03	rtl	2	57.039	55.039
Leon	pt33o02	rtl	2	57.039	55.039
Leon	pt33o01	rtl	2	57.039	55.039
Leon	pt33d20u	rtl	3	57.624	54.624
Leon	pt33d20	rtl	2	57.039	55.039
Leon	pt33d00u	rtl	3	57.624	54.624
Leon	pt33d00	rtl	2	57.039	55.039
Leon	pt3o03	rtl	2	57.039	55.039
Leon	pt3o02	rtl	2	57.039	55.039
Leon	pt3o01	rtl	2	57.039	55.039
Leon	pc3d21	rtl	2	57.039	55.039
Leon	wyfa2gsa	rtl	18	115.006	97.006
Leon	rfbypass	rtl	44	114.585	70.585
Leon	regfile_iu	rtl	47	111.781	64.781
Leon	C3B20U	rtl	8	70.376	62.376
Leon	C3B10U	rtl	8	70.376	62.376
Leon	C3O40	rtl	2	57.039	55.039
Leon	C3O20	rtl	2	57.039	55.039
Leon	C3O10	rtl	2	57.039	55.039
Leon	C3I42	rtl	2	57.039	55.039
Leon	C3I40	rtl	2	57.039	55.039

14.11 K4I

14.11.1 Result summary

Population statistical properties and model accuracy:

	All architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	121.865	116.546	-5.319
Variance	82132.638	6743.752	65271.673
Standard deviation	286.588	82.120	255.483
	Behavioral architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	148.815	129.404	-19.411
Variance	121245.948	5502.347	102527.175
Standard deviation	348.204	74.178	320.199
	Structural architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	54.806	68.457	13.651
Variance	5960.385	3522.343	4806.282
Standard deviation	77.204	59.349	69.327
	Data-flow architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	124.114	125.428	1.314
Variance	72713.624	8367.777	52767.730
Standard deviation	269.655	91.476	229.712

Correlation between estimated and real values:

	Correlation coefficient $(L, \hat{L})$
All architectures	0.5015
Behavioral architectures only	0.4689
Structural architectures only	0.5103
Data-flow architectures only	0.5739

Figure 14.47: Model K4I: Real vs. estimated lines of code.

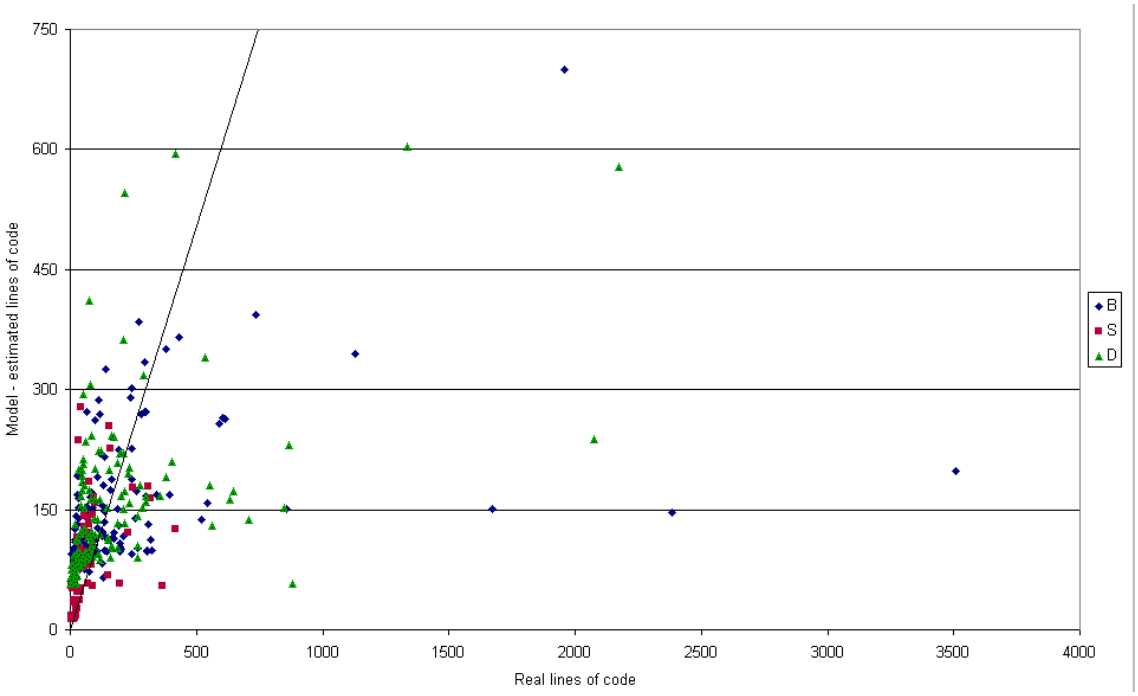


Figure 14.48: Models K4Ib, K4Is, K4Id: Real vs. estimated lines of code.

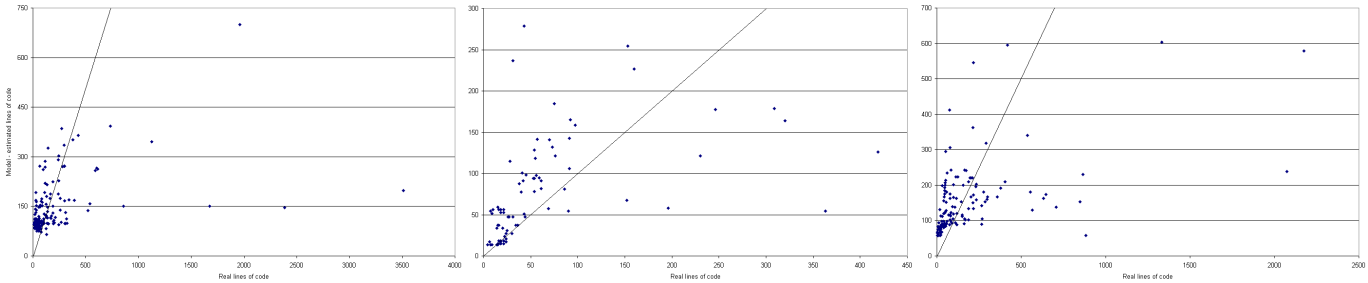


Figure 14.49: Model K4I: Error density distribution.

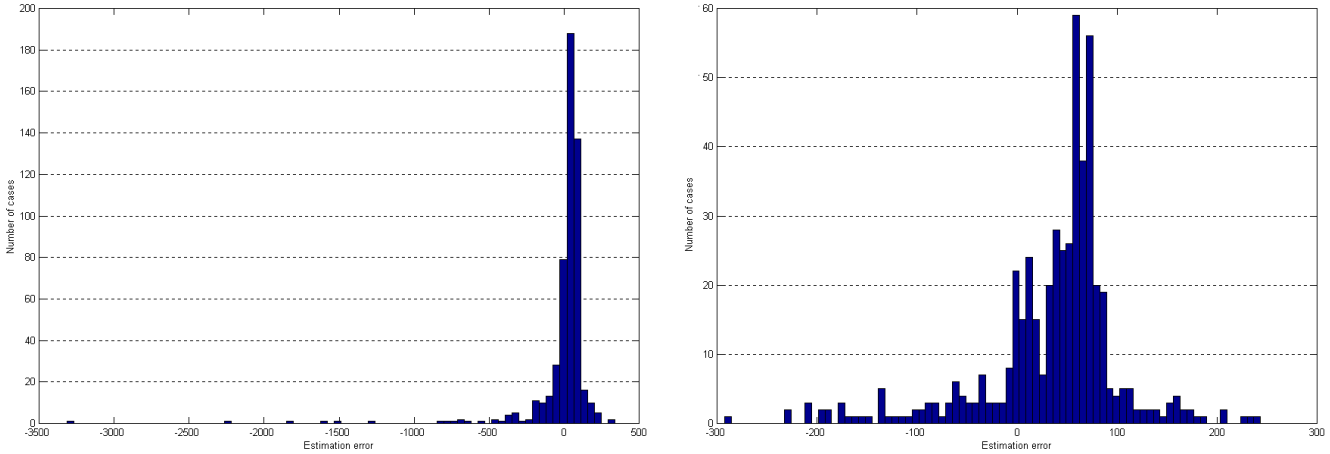


Figure 14.50: Model K4I: Error cumulative distribution.

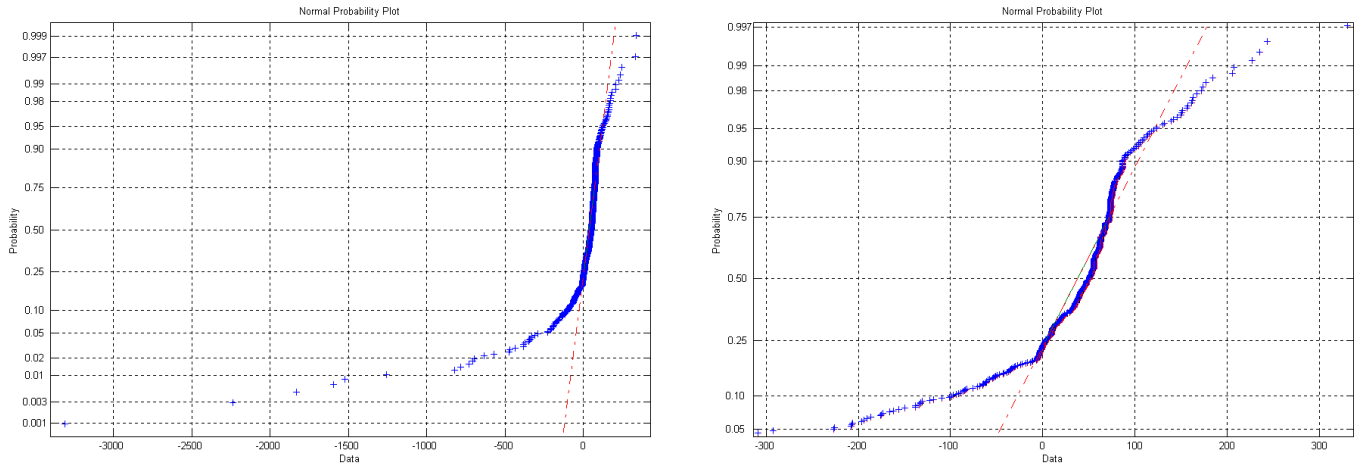
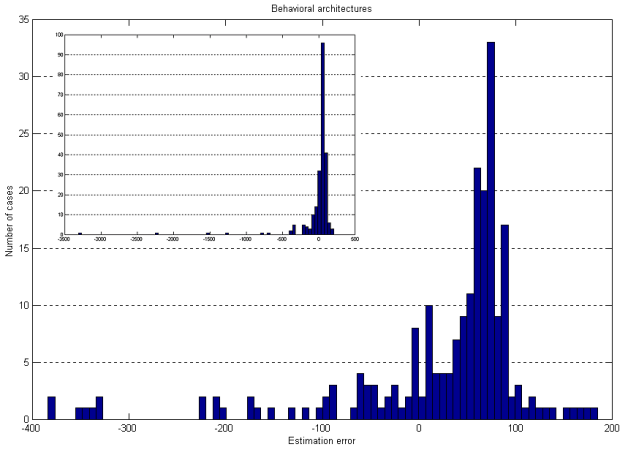
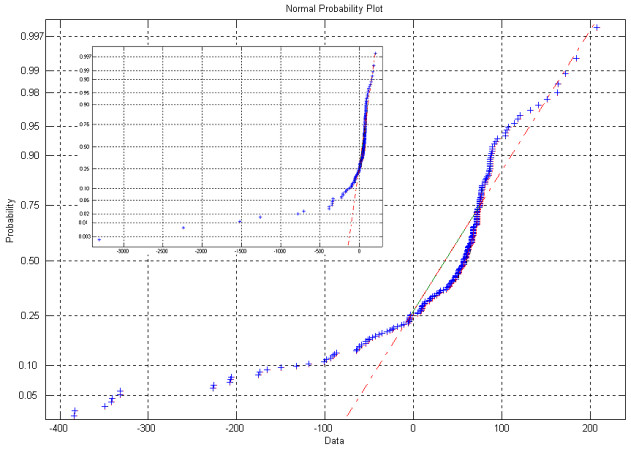
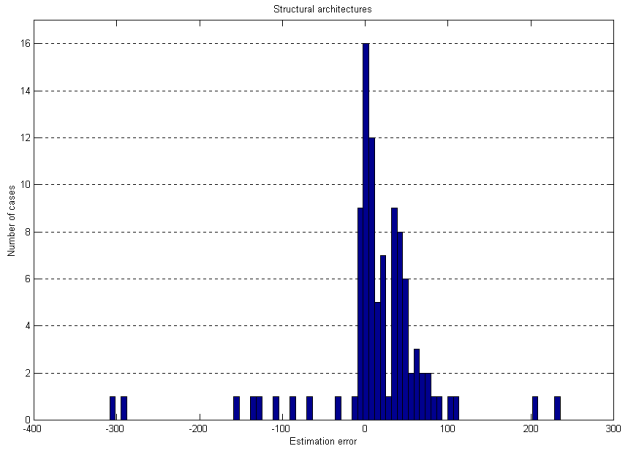
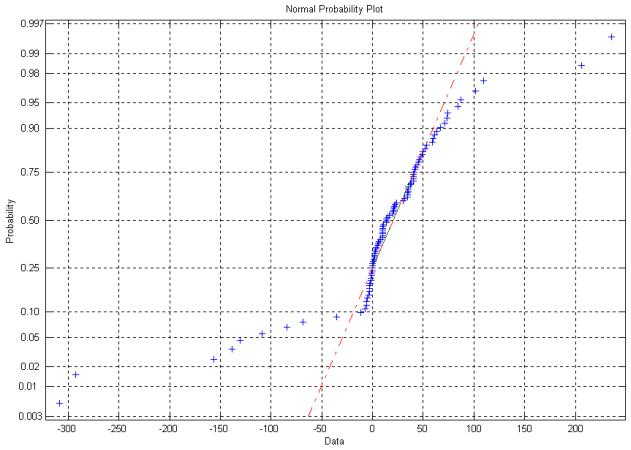


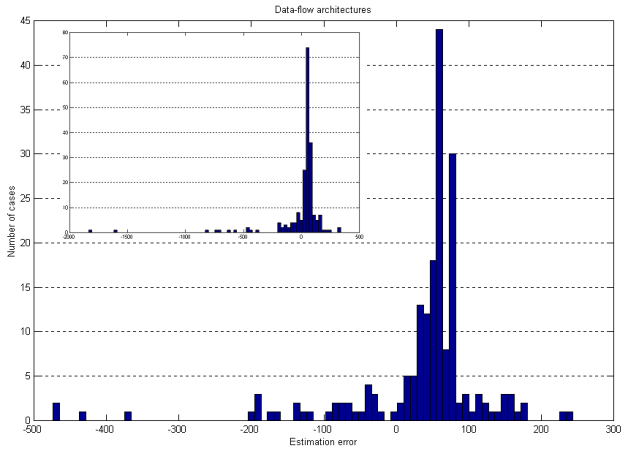
Figure 14.51: Models K4Ib, K4Is, K4Id: Error density and cumulative distributions.

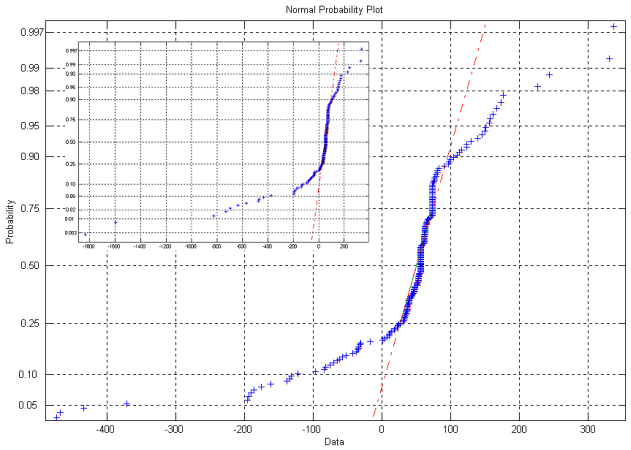












14.11.2 Detailed Results

Real sizes versus estimated sizes for behavioral architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
ans_RISC8	reg_files	behave_reg_files	76	81.123	5.123
an-XC2S-USB	xc2sFPGA	BHV	190	224.231	34.231
an-XC2S-XR16	teBL	bhv	302	272.382	-29.618
an-XC2S-XR16	Glue	BHV	109	191.290	82.290
an-XC2S-USB	xc2sFunc	BHV	16	102.594	86.594
an-XC2S-XR16	Core	BHV	210	116.333	-93.667
an-XC2S-USB	teBL	bhv	295	272.382	-22.618
an-XC2S-USB	xc2sCore	BHV	59	114.064	55.064
DLX	alu	behaviour	55	119.382	64.382
DLX	cache	behaviour	260	173.498	-86.502
DLX	clock_gen	behaviour	20	125.712	105.712
DLX	controller	behaviour	735	393.257	-341.743
DLX	dlx	behaviour	542	158.514	-383.486
DLX	dlx_bus_monitor	behaviour	129	64.438	-64.562
DLX	ir	behaviour	59	99.844	40.844
DLX	latch	behaviour	17	103.710	86.710
DLX	memory	behaviour	167	115.627	-51.373
DLX	mux2	behaviour	14	100.643	86.643
DLX	reg_1_out	behaviour	24	100.643	76.643
DLX	reg_2_1_out	behaviour	31	112.113	81.113
DLX	reg_2_out	behaviour	29	104.844	75.844
DLX	reg_3_out	behaviour	34	109.046	75.046
DLX	reg_file	behaviour	41	101.777	60.777
ERC32	AC245Generic	Behavior	67	153.607	86.607
ERC32	AC377Generic	Behavior	46	103.301	57.301
ERC32	FPURTGeneric	vhdl_behavioral	1127	344.828	-782.172
ERC32	IURTGeneric	vhdl_behavioral	1959	699.394	-1259.606
ERC32	RAM8	BEHAVIORAL	57	108.254	51.254
ERC32	TAPTest_iufpu	Behaviour	49	97.258	48.258
gl85	I8085	BEHAVIOR	1673	150.744	-1522.256
HC11	clock	behavoir	24	142.201	118.201
HC11	dev	behavior	48	99.052	51.052
HC11	hc11ram	behavior	41	100.381	59.381
i80386	i80386	behavior	858	150.251	-707.749
i8051	I8051_ALU	BHV	318	112.131	-205.869
i8051	I8051_CTR	BHV	3510	198.559	-3311.441
i8051	I8051_DBG	BHV	244	94.191	-149.809
i8051	I8051_RAM	BHV	195	104.063	-90.937
i8051	I8051_TSB	BHV	54	97.258	43.258
i8051	I8051_XRM	BHV	30	87.452	57.452
Leon	RAM2P_168X32	behav	13	86.123	73.123
Leon	RAM2P_136X32	behav	13	86.123	73.123
Leon	RAM2P_16X32	behav	13	86.123	73.123
Leon	RAM_2048x32	behavioral	14	86.123	72.123
Leon	RAM_1024x32	behavioral	14	86.123	72.123
Leon	RAM_512x30	behavioral	14	86.123	72.123
Leon	RAM_512x28	behavioral	14	86.123	72.123
Leon	RAM_256x30	behavioral	14	86.123	72.123
Leon	RAM_256x28	behavioral	14	86.123	72.123
Leon	RAM_256x26	behavioral	14	86.123	72.123
Leon	atc25_syncram_sim	behavioral	29	88.782	59.782
Leon	atc25_2pram	behav	35	90.111	55.111
Leon	atc35_dpram_ss_dn	behav	38	90.111	52.111
Leon	ATC35_RAM_256x26	behavioral	17	92.258	75.258
Leon	ATC35_RAM_1024x32	behavioral	17	92.258	75.258
Leon	ATC35_RAM_2048x32	behavioral	17	92.258	75.258
Leon	ATC35_RAM_256x28	behavioral	17	92.258	75.258
Leon	ATC35_RAM_1024x34	behavioral	17	92.258	75.258
Leon	ATC35_RAM_2048x34	behavioral	17	92.258	75.258
Leon	DPRAMRWRW_16X32	behav	21	87.257	66.257

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	DPRAMRWRW_136X32	behav	21	87.257	66.257
Leon	DPRAMRWRW_168X32	behav	21	87.257	66.257
Leon	SW204420	behavioral	32	164.227	132.227
Leon	SU004020	behavioral	29	191.833	162.833
Leon	SA108019	behavioral	27	168.557	141.557
Leon	generic_dpram_as	behav	45	93.179	48.179
Leon	R2048x34M8	behavioral	18	86.123	68.123
Leon	RF68X32M1	behav	17	83.056	66.056
Leon	RF68X33M1	behav	17	83.056	66.056
Leon	RF136X32M1	behav	17	83.056	66.056
Leon	RF136X33M1	behav	17	83.056	66.056
Leon	R1024X34M4	behavioral	18	86.123	68.123
Leon	R256X28M4	behavioral	18	86.123	68.123
Leon	R1024X33M4	behavioral	18	86.123	68.123
Leon	R2048X32M8	behavioral	18	86.123	68.123
Leon	R1024X32M4	behavioral	18	86.123	68.123
Leon	R256X26M4	behavioral	18	86.123	68.123
Leon	R256X25M4	behavioral	18	86.123	68.123
Leon	R256X24M4	behavioral	18	86.123	68.123
Leon	umc18_syncram_ss	behavioral	50	88.782	38.782
Leon	umc18_dpram_ss	behav	58	87.044	29.044
Leon	virtex_regfile_cp	behav	35	87.368	52.368
Leon	virtex_regfile	behav	36	84.3	48.3
Leon	virtex_syncram	behav	62	91.849	29.849
Leon	RAMB4_S16_S16	behav	38	74.988	36.988
Leon	RAMB4_S1	behav	14	86.123	72.123
Leon	RAMB4_S2	behav	14	86.123	72.123
Leon	RAMB4_S4	behav	14	86.123	72.123
Leon	RAMB4_S8	behav	14	86.123	72.123
SuperscalarDLX	Dlx	BehaviorPipelined	2383	145.732	-2237.268
SuperscalarDLX	Environment	Behavior	307	131.195	-175.805
T80	NoICE_TB	behaviour	53	97.258	44.258
TE51	te51mux	bhv	27	131.195	104.195
TE51	te51d	BHV	87	151.867	64.867
TE51	te51dec	bhv	267	101.459	-165.541
TE51	te51mcode	BHV	522	138.011	-383.989
TE51	te51alu	BHV	78	99.526	21.526
TE51	te51regs	BHV	127	116.333	-10.667
TE51	te51ctrl	bhv	394	168.118	-225.882
TE51	te51c	BHV	189	150.409	-38.591
xapp146	MULTI_DVM	BEHAVE	590	258.007	-331.993
xapp146	MULTI_DVM_TB	BEHAVIOR	87	97.258	10.258
xapp146	SHIFT16	DEFINITION	29	86.123	57.123
xapp146	SHIFT8b	DEFINITION2	31	93.392	62.392
xapp146	TOP_LEVEL	BEHAVE	141	325.695	184.695
xapp146	TOP_LEVEL_TB	BEHAVE	147	97.258	-49.742
xapp146	upcnt5	DEFINITION2	28	105.187	77.187
xapp328	CNT_25	BEHAVIOURAL	47	106.321	59.321
xapp328	cnt3	DEFINITION	24	95.325	71.325
xapp328	CNT_5	BEHAVIOURAL	29	92.258	63.258
xapp328	command_state_machine	BEHAVIOURAL	108	126.669	18.669
xapp328	DNLD_INTERFACE	BEHAVIOURAL	96	104.862	8.862
xapp328	FLASH_CNTR	BEHAVIOURAL	378	351.114	-26.886
xapp328	i2c_master	behave	343	169.322	-173.678
xapp328	lcd_control	behave	129	180.318	51.318
xapp328	main_ctrl_state_machine	behave	161	174.334	13.334
xapp328	pxa_bufif2	behavioral	10	95.325	85.325
xapp328	pxa_mux	behavioral	7	95.325	88.325
xapp328	pxa_dff_apar_p0	behavioral	18	92.258	74.258
xapp328	pxa_tff_apar_p0	behavioral	20	92.258	72.258
xapp328	mpeg_chip_ctrl	behave	139	114.724	-24.276
xapp328	on_off_logic	behave	27	95.325	68.325

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
xapp328	PARALLEL_PORT	BEHAVIOURAL	63	104.862	41.862
xapp328	play_logic_state_machine	behave	80	93.392	13.392
xapp328	play_modes	behave	126	83.056	-42.944
xapp328	power_ctrl	behave	59	102.594	43.594
xapp328	SHIFT8	DEFINITION	31	93.392	62.392
xapp328	sound_control	behave	125	155.919	30.919
xapp328	upcnt2	DEFINITION	28	89.190	61.190
xapp328	upcnt3	DEFINITION	28	89.190	61.190
xapp328	upcnt4	DEFINITION	28	89.190	61.190
xapp328	updwncnt4	DEFINITION	32	92.258	60.258
xapp333	i2c	behave	164	187.782	23.782
xapp333	i2c_control	behave	612	262.718	-349.282
xapp333	SHIFT8	DEFINITION	31	93.392	62.392
xapp333	uC_interface	BEHAVIOUR	243	302.511	59.511
xapp333	upcnt4	DEFINITION	28	105.187	77.187
xapp345	irda_uart	behavior	71	148.800	77.800
xapp345	irda_uart_tb	behavior	101	97.258	-3.742
xapp345	jk_ff	behavior	35	92.258	57.258
xapp345	rxcver	behavior	125	121.333	-3.667
xapp345	sirendec	behavior	135	99.526	-35.474
xapp345	txmit	behavior	105	99.526	-5.474
xapp345	uart	behavior	63	148.800	85.800
xapp345	uart_tb	behavior	101	97.258	-3.742
xapp348	sck_logic	DEFINITION	138	148.325	10.325
xapp348	spi_control_sm	DEFINITION	242	187.922	-54.078
xapp348	spi_rcv_shift_reg	DEFINITION	76	84.190	8.190
xapp348	spi_xmit_shift_reg	DEFINITION	41	79.988	38.988
xapp348	uC_interface	BEHAVIOUR	294	334.828	40.828
xapp348	upcnt4	DEFINITION	24	111.321	87.321
xapp348	upcnt5	DEFINITION	24	111.321	87.321
xapp349	uC_interface	BEHAVIOUR	257	138.799	-118.201
xapp354	am30lv0064d	behavior	22	108.980	86.980
xapp354	AMD_FLASH_TB	BEHAVIOR	325	98.587	-226.413
xapp354	NAND_INTERFACE	BEHAVIOR	138	133.938	-4.062
xapp354	k9f4008w0a	behavior	22	108.980	86.980
xapp354	NAND_FLASH_TB	BEHAVIOR	306	98.587	-207.413
xapp355	ADC_INTERFACE	BEHAVE	590	258.007	-331.993
xapp355	ADC_INTERFACE_TB	BEHAVIOR	87	97.258	10.258
xapp355	SHIFT16	DEFINITION	29	86.123	57.123
xapp355	SHIFT8	DEFINITION	31	93.392	62.392
xapp355	TOP_LEVEL	BEHAVE	114	286.434	172.434
xapp355	TOP_LEVEL_TB	BEHAVE	144	97.258	-46.742
xapp355	upcnt5	DEFINITION	28	105.187	77.187
xapp356	ADC_INTERFACE	BEHAVIOR	300	167.539	-132.461
xapp356	SHIFT16	DEFINITION	28	86.123	58.123
xapp356	SHIFT8	DEFINITION	30	93.392	63.392
xapp356	TEMP_INTERFACE	BEHAVIOR	100	112.455	12.455
xapp356	TOP_LEVEL	BEHAVIOR	240	290.635	50.635
xapp356	TOP_LEVEL_TB	BEHAVIOR	304	97.258	-206.742
xapp356	UPCNT11	DEFINITION	27	105.187	78.187
xapp356	UPCNT15	DEFINITION	27	105.187	78.187
xapp356	UPCNT5	DEFINITION	27	105.187	78.187
xapp356	XPATH	BEHAVIOR	429	365.131	-63.869
xapp357	CLK_DIVIDER	DEFINITION	16	101.459	85.459
xapp357	LED_TEST	BEHAVIOUR	116	220.030	104.030
xapp358	tx_rx_entity	behavioral	60	149.274	89.274
xapp363	clk_gen	behavioral	17	111.796	94.796
xapp365	ISO_CLK_DIVIDER	DEFINITION	55	102.817	47.817
xapp367	AudioController	DEFINITION	21	112.930	91.930
xapp367	chatterbox	BEHAVIOUR	271	385.316	114.316
xapp367	DTMFController	DEFINITION	31	151.867	120.867
xapp367	FlipFlop	definition	15	98.392	83.392

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
xapp367	FlipFlopR	definition	17	98.392	81.392
xapp367	IrqController	BEHAVIOUR	83	93.392	10.392
xapp367	MemoryManager	BEHAVE	98	261.560	163.560
xapp367	PowerSupplyController	DEFINITION	20	102.594	82.594
xapp367	RFTransceiverController	DEFINITION	31	138.938	107.938
xapp369	DECODE_MAN	BEHAVE	283	269.478	-13.522
xapp369	TOP_LEVEL	BEHAVE	117	268.829	151.829
xapp370	CLK_DIVIDER	DEFINITION	19	108.728	89.728
xapp370	cooltrak	BEHAVIOUR	245	226.685	-18.315
xapp370	MULTI_DVM	BEHAVE	606	265.276	-340.724
xapp370	SHIFT16	DEFINITION	29	86.123	57.123
xapp370	SHIFT8	DEFINITION	26	93.392	67.392
xapp370	SPEED	DEFINITION	45	98.392	53.392
xapp370	upcnt5	DEFINITION	28	105.187	77.187
xapp336	DEC_16B20B	BEHAVIOUR	80	106.795	26.795
xapp336	DEC_FUNC	BEHAVIOUR	191	130.072	-60.928
xapp336	DIS_GEN_LOW	BEHAVIOUR	86	161.914	75.914
xapp336	DIS_GEN_UP	BEHAVIOUR	83	172.250	89.250
xapp336	ENC_16B20B	BEHAVIOUR	74	100.661	26.661
xapp336	ENC_FUNC	BEHAVIOUR	126	122.653	-3.347
xapp336	ERR_CHECK	BEHAVIOUR	77	71.921	-5.079
xapp336	ERR_DET	BEHAVIOUR	60	79.988	19.988
xapp336	DECODER	BEHAVIOUR	175	121.333	-53.667
xapp336	ENCODER_LOW	BEHAVIOUR	196	97.593	-98.407
xapp336	ENCODER_UP	BEHAVIOUR	197	107.93	-89.07
xapp336	MAIN_TB	BEHAVIOUR	88	97.258	9.258
xapp336	S_GEN	BEHAVIOUR	57	74.988	17.988
xapp336_8	ERR_CHECK	BEHAVIOUR	77	71.921	-5.079
xapp336_8	DEC_FUNC	BEHAVIOUR	191	130.072	-60.928
xapp336_8	DIS_GEN	BEHAVIOUR	79	164.982	85.982
xapp336_8	ENC_FUNC	BEHAVIOUR	135	215.714	80.714
xapp336_8	DECODER	BEHAVIOUR	173	114.064	-58.936
xapp336_8	ENCODER	BEHAVIOUR	202	100.661	-101.339
xapp336_8	MAIN_TB	BEHAVIOUR	92	97.258	5.258
xapp336_8	S_GEN	BEHAVIOUR	57	74.988	17.988
Leon	fs90_dpram_ss	behav	40	80.909	40.909
Leon	fs90_syncram_sim	behavioral	34	88.782	54.782
Leon	generic_syncram	behavioral	41	91.849	50.849
Leon	generic_dpram_ss	behav	48	93.179	45.179
Leon	syncram	behav	44	91.849	47.849
TE51	te51	BHV	54	147.666	93.666
Leon	RAMB4_S16	behav	14	86.123	72.123
xapp354	am30lv0064d_top	behavior	64	271.836	207.836

Real sizes versus estimated sizes for structural architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
ans_RISC8	ans_risc8	STRUCT_ANS_RISC8	363	54.416	-308.584
ans_RISC8	alu	STRUCT_ALU	230	121.101	-108.899
ans_RISC8	control	STRUCT_CONTROL	246	177.431	-68.569
gl85struct	acc_ctrl	structure	41	100.870	59.870
gl85struct	alulogic	structure	70	141.121	71.121
gl85struct	ALU_8BIT	structure	43	51.067	8.067
gl85struct	alu_ctrl	structure	61	91.107	30.107
gl85struct	bc_pc_sp	structure	38	87.476	49.476
gl85struct	buf8	structure	15	13.740	-1.260
gl85struct	ctl_lgc1	structure	28	114.970	86.970
gl85struct	ctl_lgc2	structure	45	98.298	53.298
gl85struct	dataaddr	structure	54	94.314	40.314
gl85struct	decod2_4	structure	14	13.740	-0.260
gl85struct	DECOD3_8	structure	24	17.089	-6.911
gl85struct	flagunit	structure	97	158.287	61.287
gl85struct	g16bctr	structure	36	37.179	1.179
gl85struct	g4bctr	structure	69	57.552	-11.448
gl85struct	gl85	structure	91	142.817	51.817
gl85struct	hldlogic	structure	14	33.831	19.831

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
gl85struct	hltlogic	structure	16	37.179	21.179
gl85struct	hl_de_wz	structure	53	94.173	41.173
gl85struct	inst_reg	structure	34	37.250	3.250
gl85struct	interupt	structure	57	141.333	84.333
gl85struct	intrupt1	structure	59	94.455	35.455
gl85struct	intrupt2	structure	54	77.855	23.855
gl85struct	intrupt3	structure	42	90.895	48.895
gl85struct	inv8	structure	15	13.740	-1.260
gl85struct	M5	structure	22	23.786	1.786
gl85struct	mcdecode	structure	91	106.140	15.140
gl85struct	MUX2TO1	STR_MUX2TO1	15	58.909	43.909
gl85struct	mux_4bit	structure	24	20.437	-3.563
gl85struct	ocnand	structure	18	17.089	-0.911
gl85struct	oprlogic	structure	54	128.222	74.222
gl85struct	parity1	structure	15	37.179	22.179
gl85struct	pc_cntrl	structure	92	165.337	73.337
gl85struct	prioenco	structure	25	30.482	5.482
gl85struct	rdwrgen	structure	76	121.384	45.384
gl85struct	reg8bits	structure	30	27.134	-2.866
gl85struct	regctrl0	structure	73	131.782	58.782
gl85struct	regctrl1	structure	160	226.880	66.880
gl85struct	regctrl2	structure	55	118.459	63.459
gl85struct	regpad	structure	75	184.580	109.580
gl85struct	regpair	structure	20	33.831	13.831
gl85struct	regpairs	structure	20	33.831	13.831
gl85struct	reg_8bit	structure	24	27.134	3.134
gl85struct	reg_ctrl	structure	153	254.727	101.727
gl85struct	reg_ram	structure	20	33.831	13.831
gl85struct	shflogic	structure	44	47.295	3.295
gl85struct	sn54181	structure	86	81.203	-4.797
gl85struct	SN85150	structure	31	47.224	16.224
gl85struct	sp_cntrl	structure	56	97.733	41.733
gl85struct	stater	structure	61	81.415	20.415
gl85struct	tempctrl	structure	40	77.431	37.431
gl85struct	vectrgen	structure	26	47.224	21.224
HC11	hc11core	structure	27	47.578	20.578
i8051	I8051_ALL	STR	196	57.764	-138.236
Leon	ahbtest	struct	90	54.472	-35.528
Leon	pci_esa	struct	31	236.773	205.773
Leon	pci_is	struct	43	278.675	235.675
Leon	atc25_pciiodpad	syn	9	51.285	42.285
Leon	atc25_pciiopad	syn	7	54.634	47.634
Leon	atc25_pcitoutpad	syn	6	17.089	11.089
Leon	atc25_pcioutpad	syn	4	13.740	9.740
Leon	atc25_odpad	syn	18	15.070	-2.930
Leon	atc25_iodpad	syn	18	52.615	34.615
Leon	atc25_iopadu	syn	16	55.963	39.963
Leon	atc25_iopad	syn	16	55.963	39.963
Leon	atc25_inpad	syn	4	13.740	9.740
Leon	atc25_smpad	syn	4	13.740	9.740
Leon	atc25_outpad	syn	15	15.070	0.070
Leon	atc25_toutpadu	syn	16	18.418	2.418
Leon	umc18_smiopad	syn	10	55.963	45.963
Leon	atc35_odpad	syn	18	15.070	-2.930
Leon	atc35_iodpad	syn	18	52.615	34.615
Leon	atc35_iopad	syn	16	55.963	39.963
Leon	atc35_toutpadu	syn	16	18.418	2.418
Leon	atc35_outpad	syn	15	15.070	0.070
Leon	atc35_smpad	syn	4	13.740	9.740
Leon	atc35_inpad	syn	4	13.740	9.740
Leon	fs90_odpad	syn	21	15.070	-5.930
Leon	fs90_iodpad	syn	21	52.615	31.615

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	fs90_smiopad	syn	21	55.963	34.963
Leon	fs90_iopad	syn	21	55.963	34.963
Leon	fs90_toutpadu	syn	21	18.418	-2.582
Leon	fs90_outpad	syn	21	15.070	-5.930
Leon	fs90_smpad	syn	7	13.740	6.740
Leon	fs90_inpad	syn	9	13.740	4.740
Leon	umc18_odpad	syn	16	15.070	-0.930
Leon	umc18_iopad	syn	18	55.963	37.963
Leon	umc18_toutpadu	syn	18	18.418	0.418
Leon	umc18_outpad	syn	15	15.07	0.070
Leon	umc18_smpad	syn	4	13.740	9.740
Leon	umc18_inpad	syn	4	13.740	9.740
PIC16C5X	pic_core	structural	419	126.305	-292.695
T80	NoICE	struct	152	67.668	-84.332
Leon	umc18_iodpad	syn	18	52.615	34.615
ans_RISC8	reg_top	STRUCT_REG_TOP	309	178.589	-130.411
ans_RISC8	inst_decoder	STRUCT	320	163.896	-156.104

Real sizes versus estimated sizes for data-flow architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
ans_RISC8	reg_w	rtl_reg_w	19	72.174	53.174
ans_RISC8	reg_status	rtl_reg_status	46	121.629	75.629
ans_RISC8	reg_ioport	rtl_reg_ioport	39	119.464	80.464
ans_RISC8	reg_fsr	rtl_reg_fsr	23	86.304	63.304
ans_RISC8	prog_count	rtl_prog_count	77	175.072	98.072
ans_RISC8	mux_win	rtl_mux_win	52	213.473	161.473
ans_RISC8	mux_fwe	rtl_mux_fwe	46	185.214	139.214
ans_RISC8	mux_fin	rtl_mux_fin	48	199.343	151.343
ans_RISC8	mux_cz_write	rtl_mux_cz_write	45	175.072	130.072
ans_RISC8	mux_alub	rtl_mux_alub	46	192.278	146.278
ans_RISC8	mux_alua	rtl_mux_alua	50	206.408	156.408
ans_RISC8	ir_reg	rtl_ir_reg	40	155.700	115.700
ans_RISC8	ir_decode	rtl_ir_decode	74	411.744	337.744
ans_RISC8	clock_div	rtl_clock_div	30	98.269	68.269
ans_RISC8	alu_dp	rtl_alu_dp	82	112.399	30.399
ans_RISC8	aluop_gen	rtl_aluop_gen	61	234.668	173.668
ax8	A90S1200	rtl	167	101.996	-65.004
ax8	A90S2313	rtl	286	152.362	-133.638
ax8	AX8	rtl	647	173.516	-473.484
ax8	AX_ALU	rtl	212	150.498	-61.502
ax8	AX_PCS	rtl	79	162.271	83.271
ax8	AX_Port	rtl	51	294.501	243.501
ax8	AX_RAM	rtl	31	87.634	56.634
ax8	AX_Reg	rtl	202	220.614	18.614
ax8	AX_Reg	rtl2	210	220.614	10.614
ax8	AX_TC16	rtl	277	179.971	-97.029
ax8	AX_TC8	rtl	99	118.552	19.552
ax8	AX_UART	rtl	215	172.906	-42.094
DLX	dlx	rtl	234	158.057	-75.943
ERC32	uart	VHDL_RTL	536	339.957	-196.043
HC11	hc11cpu	rtl	2073	238.389	-1834.611
Jane	Neuron	dataflow	292	318.202	26.202
Leon	acache	rtl	174	241.266	67.266
Leon	ahbarb	rtl	186	133.836	-52.164
Leon	ahbstat	rtl	69	116.114	47.114
Leon	apbmst	rtl	78	115.177	37.177
Leon	cache	rtl	79	306.346	227.346
Leon	cachemem	rtl	75	88.683	13.683
Leon	clkgen	rtl	30	86.642	56.642
Leon	dcache	rtl	378	191.271	-186.729
Leon	div	rtl	118	87.682	-30.318
Leon	fpaux	rtl	42	82.370	40.370

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	fp	rtl	707	137.673	-569.327
Leon	fp1eu	rtl	564	129.466	-434.534
Leon	icache	rtl	230	195.935	-34.065
Leon	ioport	rtl	106	137.801	31.801
Leon	irqctrl	rtl	84	96.215	12.215
Leon	irqctrl2	rtl	111	94.241	-16.759
Leon	iu	rtl	2175	578.983	-1596.017
Leon	lconf	rtl	33	111.444	78.444
Leon	leon	rtl	85	242.348	157.348
Leon	leon_pci	rtl	215	545.937	330.937
Leon	mcore	rtl	187	209.075	22.075
Leon	mctrl	rtl	553	181.064	-371.936
Leon	fpu	rtl	32	199.001	167.001
Leon	mul	rtl	265	89.656	-175.344
Leon	GEN_XOR2	rtl	2	65.110	63.110
Leon	GEN_OR2	rtl	2	65.110	63.110
Leon	GEN_AND2	rtl	2	65.110	63.110
Leon	pci_arb	rtl	150	115.436	-34.564
Leon	proc	rtl	163	241.983	78.983
Leon	rstgen	rtl	20	73.928	53.928
Leon	atc25_regfile_iu	rtl	57	90.904	33.904
Leon	atc25_regfile_cp	rtl	51	83.839	32.839
Leon	atc25_syncram	rtl	56	88.963	32.963
Leon	pp33t015vt	rtl	3	65.110	62.110
Leon	pp33b015vt	rtl	8	82.014	74.014
Leon	pp33o01	rtl	2	58.045	56.045
Leon	pt33b04u	rtl	8	82.014	74.014
Leon	pt33b03u	rtl	8	82.014	74.014
Leon	pt33b02u	rtl	8	82.014	74.014
Leon	pt33b01u	rtl	8	82.014	74.014
Leon	pt33b04	rtl	8	82.014	74.014
Leon	pt33b03	rtl	8	82.014	74.014
Leon	pt33b02	rtl	8	82.014	74.014
Leon	pt33b01	rtl	8	82.014	74.014
Leon	pt33t03u	rtl	3	65.110	62.110
Leon	pt33t02u	rtl	3	65.110	62.110
Leon	pt33t01u	rtl	3	65.110	62.110
Leon	pt33o04	rtl	2	58.045	56.045
Leon	atc35_regfile_cp	rtl	41	83.839	42.839
Leon	atc35_regfile	rtl	58	90.904	32.904
Leon	atc35_syncram	rtl	36	88.963	52.963
Leon	pt3b03	rtl	8	82.014	74.014
Leon	pt3b02	rtl	8	82.014	74.014
Leon	pt3b01	rtl	8	82.014	74.014
Leon	pc3t03u	rtl	3	65.110	62.110
Leon	pc3t02u	rtl	3	65.110	62.110
Leon	pc3t01u	rtl	3	65.110	62.110
Leon	fs90_regfile	rtl	55	90.904	35.904
Leon	fs90_syncram	rtl	50	88.963	38.963
Leon	uyfaa	rtl	14	79.239	65.239
Leon	vyfa2gsa	rtl	12	93.369	81.369
Leon	genodpad	rtl	2	58.045	56.045
Leon	geniopad	rtl	5	82.014	77.014
Leon	geniodpad	rtl	5	74.949	69.949
Leon	gentoutpadu	rtl	3	65.11	62.11
Leon	genoutpad	rtl	2	58.045	56.045
Leon	gensmpad	rtl	2	58.045	56.045
Leon	generic_regfile_iu	rtl	89	92.233	3.233
Leon	generic_regfile_cp	rtl	36	83.839	47.839
Leon	generic_smult	rtl	21	67.768	46.768
Leon	geninpad	rtl	2	58.045	56.045
Leon	pciiodpad	rtl	14	74.949	60.949

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon	pciiopad	rtl	19	82.014	63.014
Leon	pcitoutpad	rtl	12	65.110	53.110
Leon	pcioutpad	rtl	12	58.045	46.045
Leon	iodpad	rtl	24	76.279	52.279
Leon	odpad	rtl	24	59.374	35.374
Leon	smiopad	rtl	29	83.344	54.344
Leon	iopad	rtl	29	83.344	54.344
Leon	toutpadu	rtl	24	66.439	42.439
Leon	outpad	rtl	24	59.374	35.374
Leon	smpad	rtl	21	58.045	37.045
Leon	inpad	rtl	21	58.045	37.045
Leon	hw_smult	rtl	30	67.768	37.768
Leon	regfile_cp	rtl	37	83.839	46.839
Leon	umc18_regfile	rtl	33	90.904	57.904
Leon	umc18_syncram	rtl	50	88.963	38.963
Leon	OR2DL	rtl	2	65.110	63.110
Leon	EXOR2DL	rtl	2	65.110	63.110
Leon	AND2DL	rtl	2	65.11	63.11
Leon	INVDL	rtl	2	58.045	56.045
Leon	C3B42	rtl	8	82.014	74.014
Leon	CD3O40T	rtl	7	58.045	51.045
Leon	CD3O20T	rtl	7	58.045	51.045
Leon	CD3O10T	rtl	7	58.045	51.045
Leon	CD3B40T	rtl	8	82.014	74.014
Leon	CD3B20T	rtl	8	82.014	74.014
Leon	CD3B10T	rtl	8	82.014	74.014
Leon	C3B40	rtl	8	82.014	74.014
Leon	C3B20	rtl	8	82.014	74.014
Leon	C3B10	rtl	8	82.014	74.014
Leon	C3B40U	rtl	8	82.014	74.014
Leon	timers	rtl	161	90.683	-70.317
Leon	uart	rtl	267	104.982	-162.018
Leon	wprot	rtl	94	105.100	11.100
PIC16C5X	fadr_mux	dataflow	12	65.110	53.110
PIC16C5X	pic_alu	dataflow	78	101.346	23.346
PIC16C5X	reg_cons	dataflow	12	65.110	53.110
ppx16	P16C55	rtl	149	111.836	-37.164
ppx16	P16F84	rtl	186	101.996	-84.004
ppx16	PPX16	rtl	235	202.803	-32.197
ppx16	PPX_ALU	rtl	216	133.710	-82.290
ppx16	PPX_Ctrl	rtl	57	180.958	123.958
ppx16	PPX_PCS	rtl	99	164.930	65.930
ppx16	PPX_Port	rtl	50	124.404	74.404
ppx16	PPX_RAM	rtl	39	97.358	58.358
ppx16	PPX_TMR	rtl	92	139.747	47.747
rd1007	sd_cnfg	RTL	118	119.464	1.464
rd1007	sd_rfrsh	RTL	40	79.239	39.239
rd1007	sd_sig	RTL	197	221.107	24.107
rd1007	sd_state	RTL	60	97.357	37.357
rd1007	sd_top	RTL	124	223.882	99.882
T51	I8052	rtl	145	152.362	7.362
T51	T51	rtl	848	152.739	-695.261
T51	T51_ALU	rtl	404	210.054	-193.946
T51	T51_Port	rtl	43	166.973	123.973
T51	T51_RAM	rtl	55	127.858	72.858
T80	MonZ80	rtl	882	58.045	-823.955
T80	T80	rtl	865	230.413	-634.587
T80	T80a	rtl	156	200.028	44.028
T80	T80s	rtl	97	201.242	104.242
T80	T80_ALU	rtl	265	142.522	-122.478
T80	T80_MCode	rtl	1333	603.141	-729.859
xapp333	micro_master_tb	RTL	299	167.101	-131.899

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
xapp333	micro_slave_tb	RTL	202	167.101	-34.899
xapp333	micro_tb	RTL	358	167.101	-190.899
xapp358	receive	receive_rtl	163	105.334	-57.666
xapp363	clk_top	rtl	41	79.239	38.239
xapp363	gpio_top	rtl	22	112.399	90.399
xapp363	sam_top	rtl	418	595.171	177.171
xapp363	smedia_state	rtl	298	159.688	-138.312
xapp363	smedia_top	rtl	113	222.970	109.970
xapp363	spi	rtl	83	122.540	39.540
xapp363	spi_switch	rtl	55	151.712	96.712
xapp363	ssp_icc	rtl	631	162.765	-468.235
xapp363	ssp_icc_switch	rtl	115	162.765	47.765
xapp365	iso9141	rtl	213	363.021	150.021
Leon	GEN_NOT	rtl	2	58.045	56.045
Leon	pc3d01	rtl	2	58.045	56.045
Leon	pt33o03	rtl	2	58.045	56.045
Leon	pt33o02	rtl	2	58.045	56.045
Leon	pt33o01	rtl	2	58.045	56.045
Leon	pt33d20u	rtl	3	67.884	64.884
Leon	pt33d20	rtl	2	58.045	56.045
Leon	pt33d00u	rtl	3	67.884	64.884
Leon	pt33d00	rtl	2	58.045	56.045
Leon	pt3o03	rtl	2	58.045	56.045
Leon	pt3o02	rtl	2	58.045	56.045
Leon	pt3o01	rtl	2	58.045	56.045
Leon	pc3d21	rtl	2	58.045	56.045
Leon	wyfa2gsa	rtl	18	131.469	113.469
Leon	rfbypass	rtl	44	121.211	77.211
Leon	regfile_iu	rtl	47	92.233	45.233
Leon	C3B20U	rtl	8	82.014	74.014
Leon	C3B10U	rtl	8	82.014	74.014
Leon	C3O40	rtl	2	58.045	56.045
Leon	C3O20	rtl	2	58.045	56.045
Leon	C3O10	rtl	2	58.045	56.045
Leon	C3I42	rtl	2	58.045	56.045
Leon	C3I40	rtl	2	58.045	56.045

14.12 K1E

14.12.1 Result summary

Population statistical properties and model accuracy:

	All architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	152.359	147.103	-5.256
Variance	632469.174	314085.433	81746.477
Standard deviation	795.279	560.433	285.913
	Behavioral architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	274.560	228.312	-46.247
Variance	1505949.100	721036.609	197401.984
Standard deviation	1227.171	849.139	444.299
	Structural architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	68.632	74.340	5.707
Variance	19345.818	16671.824	2161.553
Standard deviation	139.089	129.119	46.493
	Data-flow architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	71.242	98.821	27.579
Variance	39803.835	42370.684	3058.138
Standard deviation	199.509	205.841	55.300

Correlation between estimated and real values:

	Correlation coefficient $(L, \hat{L})$
All architectures	0.9702
Behavioral architectures only	0.9739
Structural architectures only	0.9426
Data-flow architectures only	0.9633

Figure 14.52: Model K1E: Real vs. estimated lines of code.

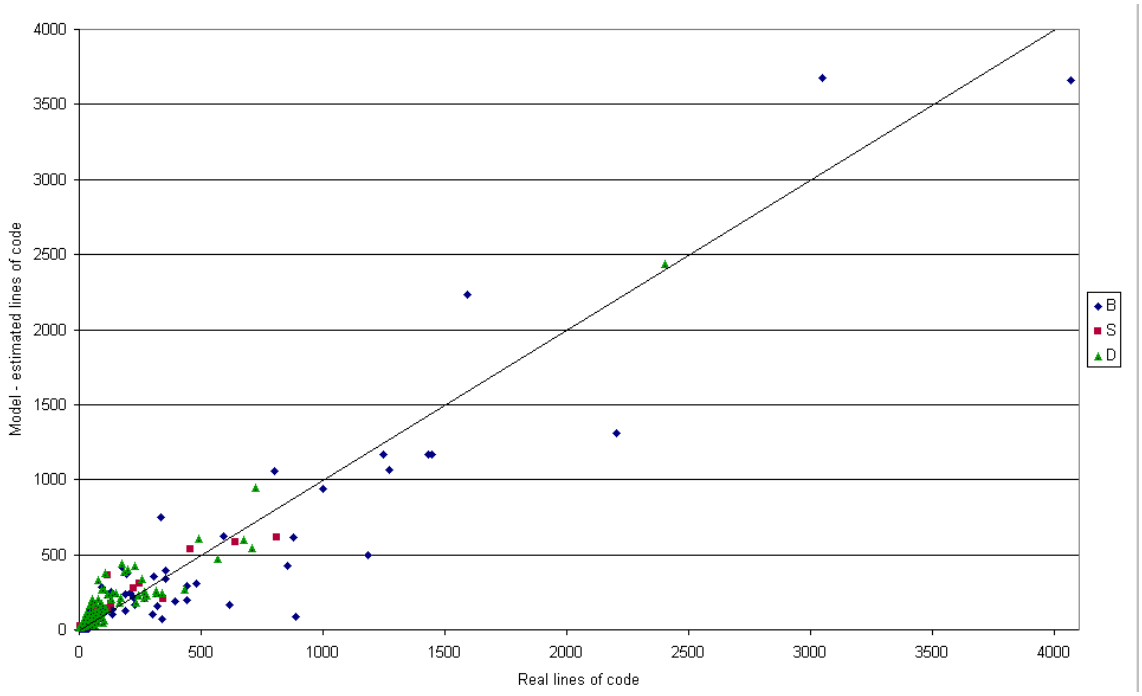


Figure 14.53: Models K1Eb, K1Es, K1Ed: Real vs. estimated lines of code.

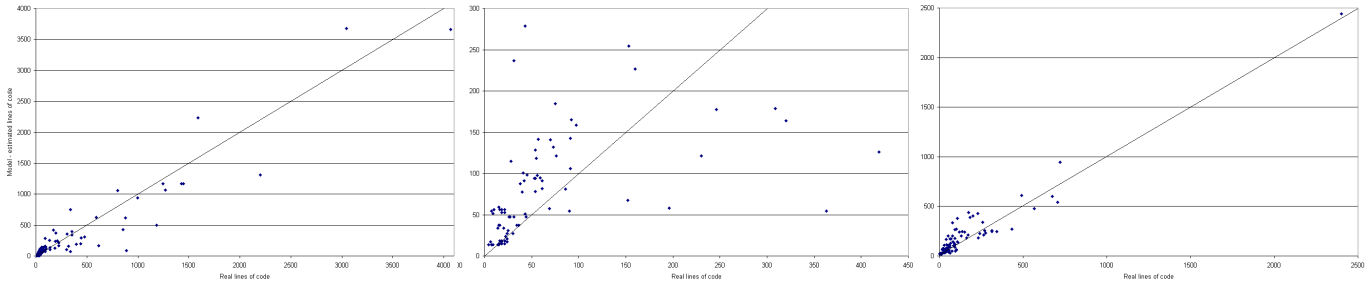


Figure 14.54: Model K1E: Error density distribution.

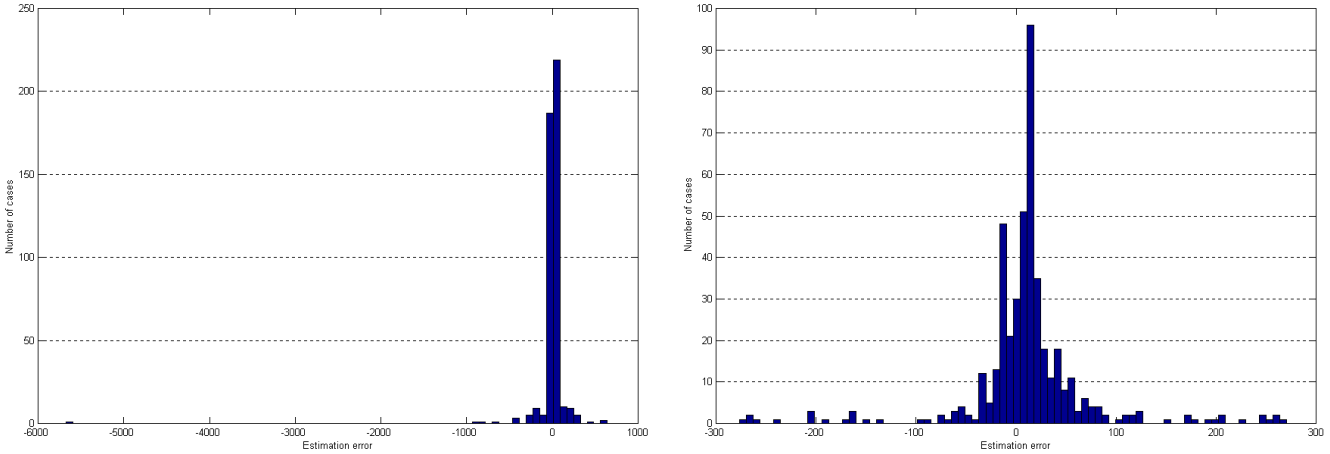


Figure 14.55: Model K1E: Error cumulative distribution.

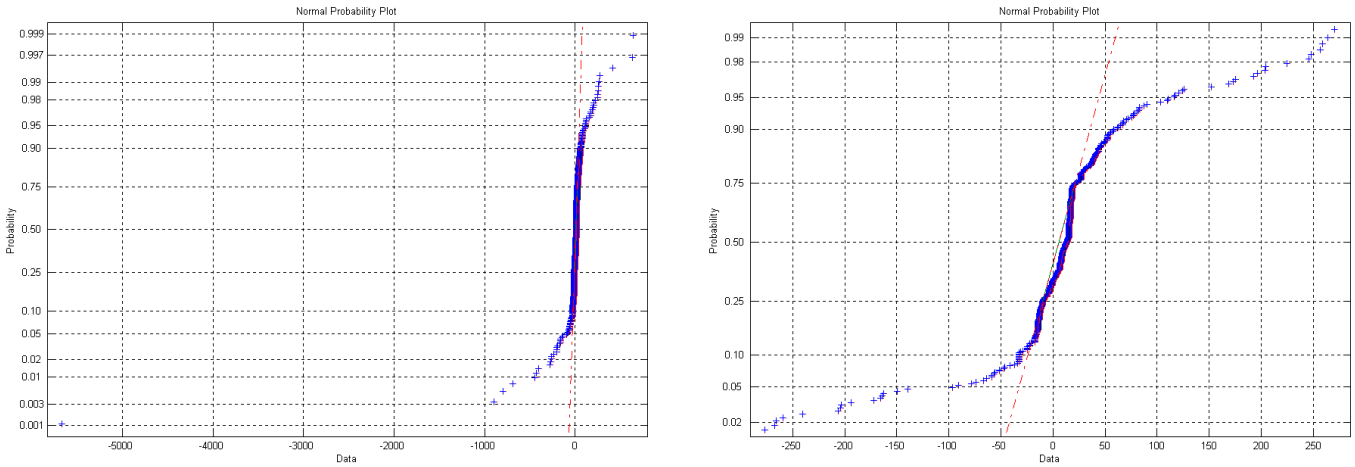
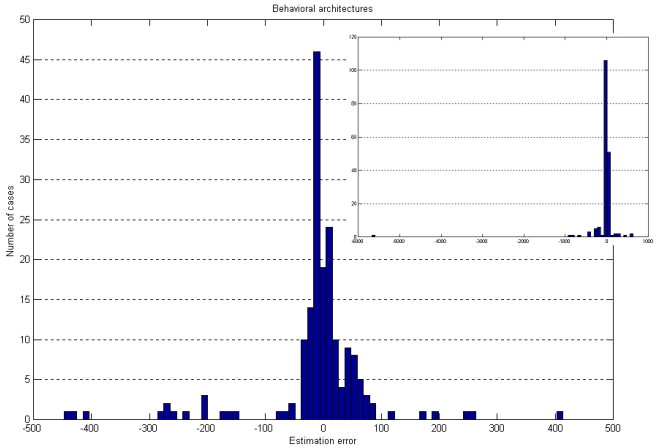
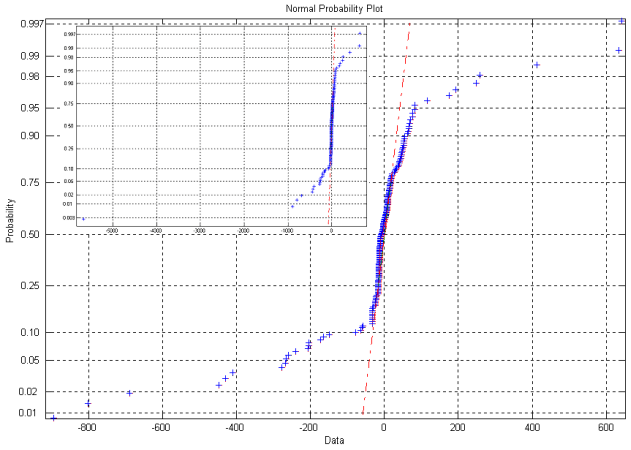
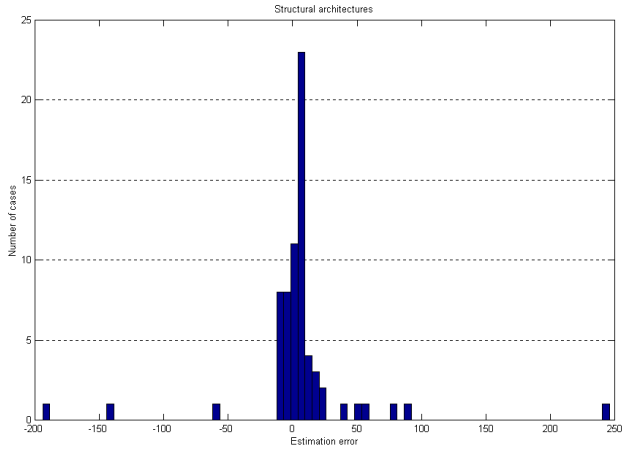
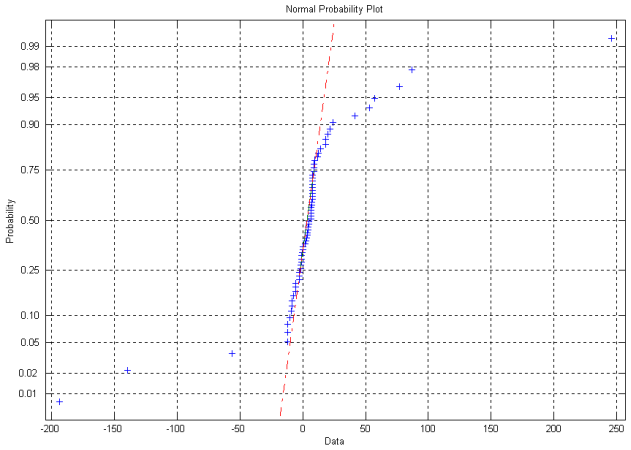


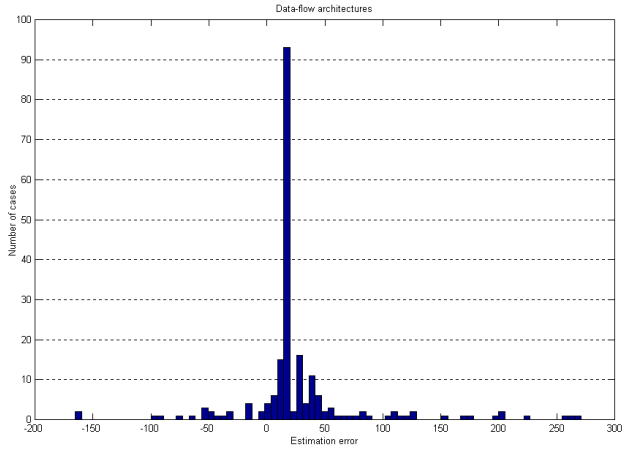
Figure 14.56: Models K1Eb, K1Es, K1Ed: Error density and cumulative distributions.

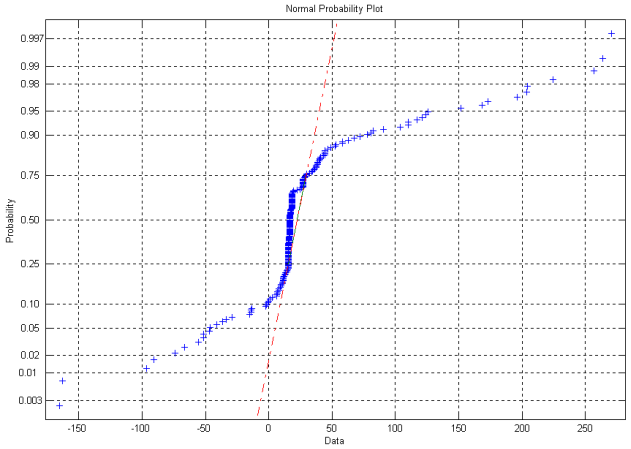












14.12.2 Detailed Results

Real sizes versus estimated sizes for behavioral architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
AMCC5933	amcc5933	behavior	593	624.350	31.350
DLX2	Gen_Immediate	behavioral	33	27.387	-5.613
DLX2	stats	behavioral	24	30.860	6.860
DLX2	dlx_control	behavioral	337	750.236	413.236
DLX2	dlx_data_path	behavioral	173	420.816	247.816
DLX2	Data_Reg	behavioral	25	28.139	3.139
DLX2	Bypass_Logic	behavioral	43	52.539	9.539
DLX2	Decode_Ird	behavioral	102	112.305	10.305
DLX2	Decode_PC	behavioral	93	285.763	192.763
DLX2	dlx_stats	behavioral	24	29.968	5.968
DLX2	Dmemory	behavioral	89	155.655	66.655
DLX2	Imemory	behavioral	59	107.938	48.938
DLX2	Ireg_Fetch	behavioral	32	58.443	26.443
DLX2	Ireg_Decode	behavioral	37	77.520	40.520
DLX2	Ireg_Execute	behavioral	41	41.130	0.130
DLX2	Ireg_Memory	behavioral	34	42.783	8.783
DLX2	Ireg_Writeback	behavioral	28	42.943	14.943
DLX2	Register_File	behavioral	41	108.897	67.897
DSP320VC33	dsp320vc33	vhdl_behavioral	3045	3678.916	633.916
DSP320VC33	sram16k32	vhdl_behavioral	479	306.898	-172.102
DSP320VC33	sram1k32	vhdl_behavioral	442	292.762	-149.238
DSP6211	dsp6211	vhdl_behavioral	4068	3659.314	-408.686
DSP6211	km416s4030	vhdl_behavioral	1446	1169.487	-276.513
DSP6211	MT58L32L32P	behave	224	212.326	-11.674
DSP6211	sram1k16	vhdl_behavioral	353	337.960	-15.040
DSP6211	sram1k8	vhdl_behavioral	307	353.128	46.128
DSP6415	at9366	vhdl_behavioral	352	397.777	45.777
DSP6415	cy7c453	vhdl_behavioral	801	1059.881	258.881
DSP6415	dsp6415	vhdl_behavioral	15508	9836.380	-5671.620
DSP6415	idt71v546	vhdl_behavioral	1271	1066.695	-204.305
DSP6415	km416s4030	vhdl_behavioral	1429	1169.487	-259.513
DSP6415	MS32PCI	Behavior	615	167.973	-447.027
DSP6415	MT58L32L32P	behave	224	212.326	-11.674
DSP6415	sram1k16	vhdl_behavioral	353	337.960	-15.040
DSP6415	sram1k8	vhdl_behavioral	307	353.128	46.128
DSP6415	TG32PCI	Behavior	1185	495.997	-689.003
fw09	addressing_decode	behave	137	106.198	-30.802
fw09	alu	behave	191	235.985	44.985
fw09	busstatus	behave	39	59.939	20.939
fw09	instruction_decode	behave	890	89.087	-800.913
fw09	mainstate	behave	21	42.250	21.250
fw09	mux16alu_left	behave	67	86.693	19.693
fw09	mux16alu_right	behave	63	54.804	-8.196
fw09	muxaddressbus	behave	32	41.095	9.095
fw09	muxdatabus	behave	54	53.072	-0.928
fw09	pb_decode	behave	95	135.924	40.924
fw09	reg8	behave	24	41.673	17.673
fw09	registercc	behave	52	135.218	83.218
fw09	registerd	behave	85	101.451	16.451
fw09	registerdp	behave	42	91.784	49.784
fw09	registerindexstack	behave	97	149.357	52.357
fw09	registerpc	behave	85	149.935	64.935
fw09	registertemp	behave	53	91.207	38.207
fw09	regpage	behave	30	41.673	11.673
fw09	statedecode	behave	2201	1306.161	-894.839
fw09	transfer_decode	behave	81	89.665	8.665
fw09	vectortable	behave	32	43.405	11.405
HDLLib	core_mac	behv	133	249.755	116.755
HDLLib	ram128x8	behv	338	70.821	-267.179
IEEE1149	br_cell	behavioral	13	30.169	17.169

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
IEEE1149	dr_cell	behavioral	26	63.729	37.729
IEEE1149	ir_cell	behavioral	28	60.369	32.369
IEEE1149	mux_2_1	behavioral	12	27.387	15.387
IEEE1149	mux_4_1	behavioral	17	26.232	9.232
IEEE1149	tap_controller	behavioral	138	133.373	-4.627
Manticore	demo_xform	behavioural	226	162.139	-63.861
Manticore	rasterizer	behavioural	880	613.996	-266.004
Manticore	raster_ctrl	behav	302	98.737	-203.263
Manticore	raster_vars_reg	behavioural	854	425.334	-428.666
Manticore	raster_var_type_reg	behavioural	17	27.387	10.387
Manticore	sdram_control	behav	395	188.753	-206.247
Manticore	sdram_control_param	behav	440	199.881	-240.119
Manticore	vgafifo_ctrl	behavioural	319	154.690	-164.310
Manticore	vgaout	behavior	188	129.034	-58.966
RLS	Fp_Divide	bhv	76	76.636	0.636
RLS	MainCtl	Bhv	194	369.483	175.483
RLS	RLSTop	bhv	211	240.874	29.874
RLS	rls_mult	bhv	92	111.332	19.332
RLS	SubCtl	bhv	999	940.556	-58.444
SPIM-Pipe	control	behavioral	40	82.577	42.577
SPIM-Pipe	decode	behavioral	66	117.739	51.739
SPIM-Pipe	execute	behavioral	43	31.158	-11.842
SPIM-Pipe	fetch	behavioral	48	44.498	-3.502
SPIM-Pipe	memory	behavioral	36	36.783	0.783
SPIM-Pipe	pipe_reg1	behavioral	21	28.480	7.480
SPIM-Pipe	pipe_reg2	behavioral	50	40.506	-9.494
SPIM-Pipe	pipe_reg3	behavioral	33	38.446	5.446
SPIM-Pipe	pipe_reg4	behavioral	29	31.760	2.760
SPIM	control	behavioral	47	113.974	66.974
SPIM	decode	behavioral	70	140.389	70.389
SPIM	execute	behavioral	49	49.423	0.423
SPIM	fetch	behavioral	52	42.766	-9.234
SPIM	memory	behavioral	34	36.783	2.783
STD8980	std8980	vhdl_behavioral	1246	1167.900	-78.100
ZR36060	zr36060	vhdl_behavioral	1590	2231.259	641.259
DLX2	Main_Alu	behavioral	54	106.710	52.710
DLX2	Inverter	behavioral	13	28.542	15.542
DLX2	Data_Mux2	behavioral	20	27.387	7.387
DLX2	Data_Mux4	behavioral	28	26.232	-1.768
Leon2	atc25_syncram_sim	behavioral	29	43.177	14.177
Leon2	atc25_2pram	behav	38	52.852	14.852
Leon2	atc25_dpram_sim	behav	53	66.124	13.124
Leon2	RAM_256x26	behavioral	14	0.991	-13.009
Leon2	RAM_256x28	behavioral	14	0.991	-13.009
Leon2	RAM_256x30	behavioral	14	0.991	-13.009
Leon2	RAM_512x28	behavioral	14	0.991	-13.009
Leon2	RAM_512x30	behavioral	14	0.991	-13.009
Leon2	RAM_512x32	behavioral	14	0.991	-13.009
Leon2	RAM_1024x32	behavioral	14	0.991	-13.009
Leon2	RAM_2048x32	behavioral	14	0.991	-13.009
Leon2	RAM2P_16X32	behav	13	0.991	-12.009
Leon2	RAM2P_136X32	behav	13	0.991	-12.009
Leon2	RAM2P_168X32	behav	13	0.991	-12.009
Leon2	DPRAM_256x26	behav	20	5.717	-14.283
Leon2	DPRAM_256x28	behav	20	5.717	-14.283
Leon2	DPRAM_256x30	behav	20	5.717	-14.283
Leon2	DPRAM_256x32	behav	20	5.717	-14.283
Leon2	DPRAM_512x28	behav	20	5.717	-14.283
Leon2	DPRAM_512x30	behav	20	5.717	-14.283
Leon2	DPRAM_512x32	behav	20	5.717	-14.283
Leon2	atc35_dpram_ss_dn	behav	38	55.634	17.634
Leon2	ATC35_RAM_256x26	behavioral	17	6.637	-10.363

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	ATC35_RAM_1024x32	behavioral	17	6.637	-10.363
Leon2	ATC35_RAM_2048x32	behavioral	17	6.637	-10.363
Leon2	ATC35_RAM_256x28	behavioral	17	6.637	-10.363
Leon2	ATC35_RAM_1024x34	behavioral	17	6.637	-10.363
Leon2	ATC35_RAM_2048x34	behavioral	17	6.637	-10.363
Leon2	DPRAMRWRW_16X32	behav	21	4.289	-16.711
Leon2	DPRAMRWRW_136X32	behav	21	4.289	-16.711
Leon2	DPRAMRWRW_168X32	behav	21	4.289	-16.711
Leon2	fs90_syncram_sim	behavioral	34	96.035	62.035
Leon2	fs90_dpram_ss	behav	40	95.632	55.632
Leon2	SA108019	behavioral	27	32.111	5.111
Leon2	SU004020	behavioral	29	38.609	9.609
Leon2	SW204420	behavioral	32	39.414	7.414
Leon2	generic_syncram	behavioral	41	118.152	77.152
Leon2	generic_dpram_ss	behav	48	102.790	54.790
Leon2	generic_dpram_as	behav	45	97.226	52.226
Leon2	syncram	behav	56	43.970	-12.030
Leon2	dpsyncram	behav	53	59.653	6.653
Leon2	tsmc25_syncram_ss	behavioral	57	43.754	-13.246
Leon2	tsmc25_dpram_ss	behav	91	69.859	-21.141
Leon2	ram4096x32	behavioral	23	0.495	-22.505
Leon2	ram1024x32	behavioral	23	0.495	-22.505
Leon2	ram2400x32	behavioral	28	3.277	-24.723
Leon2	ram2048x32	behavioral	23	0.495	-22.505
Leon2	ram256x24	behavioral	28	3.277	-24.723
Leon2	ram256x27	behavioral	28	3.277	-24.723
Leon2	ram512x23	behavioral	28	3.277	-24.723
Leon2	dpram16x32	behavioral	38	5.717	-32.283
Leon2	dpram136x32	behavioral	38	5.717	-32.283
Leon2	dpram168x32	behavioral	38	5.717	-32.283
Leon2	dpram256x26	behavioral	38	5.717	-32.283
Leon2	dpram256x28	behavioral	38	5.717	-32.283
Leon2	dpram256x30	behavioral	38	5.717	-32.283
Leon2	dpram256x32	behavioral	38	5.717	-32.283
Leon2	dpram512x28	behavioral	38	5.717	-32.283
Leon2	dpram512x30	behavioral	38	5.717	-32.283
Leon2	TIEHI	behavioral	9	-1.720	-10.720
Leon2	TIELO	behavioral	9	-1.720	-10.720
Leon2	umc18_dpram_ss	behav	58	140.82	82.82
Leon2	umc18_syncram_ss	behavioral	50	125.921	75.921
Leon2	R256X24M4	behavioral	18	0.991	-17.009
Leon2	R256X25M4	behavioral	18	0.991	-17.009
Leon2	R256X26M4	behavioral	18	0.991	-17.009
Leon2	R1024X32M4	behavioral	18	0.991	-17.009
Leon2	R2048X32M8	behavioral	18	0.991	-17.009
Leon2	R256X28M4	behavioral	18	0.991	-17.009
Leon2	RF136X32M1	behav	17	1.487	-15.513
Leon2	RF168X32M1	behav	17	1.487	-15.513
Leon2	RAMB4_S16	behav	14	-0.082	-14.082
Leon2	RAMB4_S8	behav	14	-0.082	-14.082
Leon2	RAMB4_S4	behav	14	-0.082	-14.082
Leon2	RAMB4_S2	behav	14	-0.082	-14.082
Leon2	RAMB4_S1	behav	14	-0.082	-14.082
Leon2	RAMB4_S1_S1	behav	42	38.725	-3.275
Leon2	RAMB4_S2_S2	behav	42	38.725	-3.275
Leon2	RAMB4_S8_S8	behav	42	38.725	-3.275
Leon2	RAMB4_S4_S4	behav	42	38.725	-3.275
Leon2	RAMB4_S16_S16	behav	42	38.725	-3.275
Leon2	virtex_syncram	behav	62	56.523	-5.477
Leon2	virtex_regfile	behav	54	49.461	-4.539
Leon2	virtex_regfile_cp	behav	36	48.708	12.708
Leon2	virtex_dpram	behav	74	88.099	14.099

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$

Real sizes versus estimated sizes for structural architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
fw09	core_6809e	structure	808	614.133	-193.867
IEEE1149	dr	structural	33	50.974	17.974
IEEE1149	ir	structural	40	57.922	17.922
IEEE1149	testable_nibble_comparator	structural	63	150.019	87.019
LFSR	prpt	structural	223	276.391	53.391
LFSR	prpt	struct_generate	74	76.665	2.665
LFSR	prpt_stage	structural	54	78.305	24.305
LFSR	prpt_stage1	structural	47	67.149	20.149
Manticore	calc_test	structural	457	533.935	76.935
Manticore	frame_buffer_test	structural	641	584.407	-56.593
Manticore	manticore_fifo	struct	68	74.648	6.648
Manticore	pll2x	SYN	56	43.650	-12.350
Manticore	shiftreg_8x8	structural	54	58.242	4.242
Manticore	slope_calc	structural	344	204.309	-139.691
Manticore	vgafifo	SYN	65	56.201	-8.799
Manticore	write_fifo_address	SYN	62	52.986	-9.014
Manticore	write_fifo_data	SYN	66	55.891	-10.109
Manticore	write_fifo_mask	SYN	57	49.346	-7.654
Manticore	zfifo	SYN	65	56.201	-8.799
SPIM-Pipe	spim_pipe	structural	248	305.356	57.356
SPIM	ss_spim	structural	118	139.491	21.491
Leon2	ahbtest	struct	90	99.667	9.667
Leon2	dcom	struct	130	171.435	41.435
Leon2	dma	struct	115	361.004	246.004
Leon2	pci_is	struct	44	46.123	2.123
Leon2	atc25_inpad	syn	4	11.939	7.939
Leon2	atc25_smpad	syn	4	11.939	7.939
Leon2	atc25_outpad	syn	15	14.197	-0.803
Leon2	atc25_toutpadu	syn	16	16.045	0.045
Leon2	atc25_iopad	syn	16	24.946	8.946
Leon2	atc25_iopadu	syn	16	24.946	8.946
Leon2	atc25_iodpad	syn	18	24.426	6.426
Leon2	atc25_odpad	syn	18	15.526	-2.474
Leon2	atc25_pcioutpad	syn	4	11.939	7.939
Leon2	atc25_pcitoutpad	syn	6	12.968	6.968
Leon2	atc25_pciiopad	syn	7	21.049	14.049
Leon2	atc25_pciiodpad	syn	9	20.529	11.529
Leon2	atc35_inpad	syn	4	11.939	7.939
Leon2	atc35_smpad	syn	4	11.939	7.939
Leon2	atc35_outpad	syn	15	14.197	-0.803
Leon2	atc35_toutpadu	syn	16	16.045	0.045
Leon2	atc35_iopad	syn	16	24.946	8.946
Leon2	atc35_iodpad	syn	18	24.426	6.426
Leon2	atc35_odpad	syn	18	15.526	-2.474
Leon2	fs90_inpad	syn	9	13.268	4.268
Leon2	fs90_smpad	syn	7	13.367	6.367
Leon2	fs90_outpad	syn	21	15.517	-5.483
Leon2	fs90_toutpadu	syn	21	18.072	-2.928
Leon2	fs90_iopad	syn	21	25.545	4.545
Leon2	fs90_smiopad	syn	21	25.545	4.545
Leon2	fs90_iodpad	syn	21	24.926	3.926
Leon2	fs90_odpad	syn	21	15.616	-5.384
Leon2	tsmc25_inpad	syn	4	11.939	7.939
Leon2	tsmc25_smpad	syn	4	11.939	7.939
Leon2	tsmc25_outpad	syn	30	18.202	-11.798
Leon2	tsmc25_toutpadu	syn	27	21.325	-5.675
Leon2	tsmc25_iopad	syn	25	28.798	3.798

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	tsmc25_iodpad	syn	30	28.332	-1.668
Leon2	tsmc25_odpad	syn	30	18.202	-11.798
Leon2	tsmc25_smiopad	syn	26	28.897	2.897
Leon2	umc18_inpad	syn	4	11.939	7.939
Leon2	umc18_smpad	syn	4	11.939	7.939
Leon2	umc18_outpad	syn	15	14.197	-0.803
Leon2	umc18_toutpadu	syn	18	17.473	-0.527
Leon2	umc18_iopad	syn	18	25.045	7.045
Leon2	umc18_iodpad	syn	18	24.426	6.426
Leon2	umc18_odpad	syn	16	14.197	-1.803
Leon2	umc18_smiopad	syn	10	22.477	12.477

Real sizes versus estimated sizes for data-flow architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
RTC	ALU	ALU_rtl	315	248.336	-66.664
RTC	BCD2BIN	BCD2BIN_rtl	18	62.111	44.111
RTC	BIN2BCD	BIN2BCD_rtl	18	62.111	44.111
RTC	cal_modul	cal_modul_rtl	241	227.416	-13.584
RTC	CLOCK_DIV	CLOCK_DIV_rtl	26	72.420	46.420
RTC	CMDINTERF	CMDINTERF_rtl	49	77.716	28.716
RTC	JAHR_COUNTER	JAHR_COUNTER_rtl	36	65.613	29.613
RTC	monat_counter	monat_counter_rtl	38	64.820	26.820
RTC	MUX2TO1	MUX2TO1_rtl	11	25.031	14.031
RTC	PC	PC_rtl	24	62.793	38.793
RTC	secmin_counter	secmin_counter_rtl	38	64.820	26.820
RTC	tag_counter	tag_counter_rtl	58	95.212	37.212
RTC	uhr_counter	uhr_counter_rtl	38	64.820	26.820
RTC	weekday_counter	weekday_counter_rtl	38	64.820	26.820
RTC	weeknr_counter	weeknr_counter_rtl	38	64.820	26.820
RTC-alt	ALU	ALU_rtl	313	257.331	-55.669
RTC-alt	BCD2BIN	BCD2BIN_rtl	18	62.111	44.111
RTC-alt	BIN2BCD	BIN2BCD_rtl	18	62.111	44.111
RTC-alt	cal_modul	cal_modul_rtl	269	253.663	-15.337
RTC-alt	CLOCK_DIV	CLOCK_DIV_rtl	23	71.812	48.812
RTC-alt	CMDINTERF	CMDINTERF_rtl	49	77.716	28.716
RTC-alt	JAHR_COUNTER	JAHR_COUNTER_rtl	36	65.613	29.613
RTC-alt	monat_counter	monat_counter_rtl	38	64.820	26.820
RTC-alt	MUX2TO1	MUX2TO1_rtl	11	25.031	14.031
RTC-alt	PC	PC_rtl	24	62.793	38.793
RTC-alt	secmin_counter	secmin_counter_rtl	38	64.820	26.820
RTC-alt	tag_counter	tag_counter_rtl	58	95.212	37.212
RTC-alt	uhr_counter	uhr_counter_rtl	38	64.820	26.820
RTC-alt	weekday_counter	weekday_counter_rtl	38	64.820	26.820
RTC-alt	weeknr_counter	weeknr_counter_rtl	38	64.820	26.820
Leon2	acache	rtl	201	404.819	203.819
Leon2	ahbarb	rtl	175	438.588	263.588
Leon2	ahbmst	rtl	94	266.974	172.974
Leon2	ahbstat	rtl	68	125.951	57.951
Leon2	apbmst	rtl	78	202.216	124.216
Leon2	cache	rtl	81	143.730	62.730
Leon2	cachemem	rtl	92	89.537	-2.463
Leon2	dcache	rtl	491	607.798	116.798
Leon2	dcom_uart	rtl	232	180.721	-51.279
Leon2	div	rtl	118	238.934	120.934
Leon2	dsu	rtl	432	269.240	-162.760
Leon2	dsu_mem	rtl	64	30.142	-33.858
Leon2	fpaux	rtl	42	167.862	125.862
Leon2	fp	rtl	707	541.642	-165.358
Leon2	fp1eu	rtl	566	475.416	-90.584
Leon2	fpu_core	rtl	54	39.892	-14.108
Leon2	fpu_lth	rtl	722	946.640	224.640

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	icache	rtl	230	426.401	196.401
Leon2	ioport	rtl	107	142.947	35.947
Leon2	irqctrl	rtl	84	136.783	52.783
Leon2	irqctrl2	rtl	111	137.061	26.061
Leon2	iu	rtl	2401	2439.549	38.549
Leon2	lconf	rtl	38	70.037	32.037
Leon2	leon	rtl	131	203.240	72.240
Leon2	leon_pci	rtl	258	338.870	80.870
Leon2	mcore	rtl	172	212.253	40.253
Leon2	mctrl	rtl	675	600.913	-74.087
Leon2	fpu	rtl	32	34.557	2.557
Leon2	mul	rtl	265	213.439	-51.561
Leon2	GEN_NOT	rtl	2	18.591	16.591
Leon2	GEN_AND2	rtl	2	19.198	17.198
Leon2	GEN_OR2	rtl	2	19.198	17.198
Leon2	GEN_XOR2	rtl	2	19.198	17.198
Leon2	pci	rtl	28	67.603	39.603
Leon2	pci_arb	rtl	152	242.310	90.310
Leon2	proc	rtl	138	248.270	110.270
Leon2	rstgen	rtl	29	106.690	77.690
Leon2	sdmctrl	rtl	342	245.393	-96.607
Leon2	pt33d00	rtl	2	18.591	16.591
Leon2	pt33d00u	rtl	3	22.343	19.343
Leon2	pt33d20	rtl	2	18.591	16.591
Leon2	pt33d20u	rtl	3	22.343	19.343
Leon2	pt33o01	rtl	2	18.591	16.591
Leon2	pt33o02	rtl	2	18.591	16.591
Leon2	pt33o03	rtl	2	18.591	16.591
Leon2	pt33o04	rtl	2	18.591	16.591
Leon2	pt33t01u	rtl	3	19.198	16.198
Leon2	pt33t02u	rtl	3	19.198	16.198
Leon2	pt33t03u	rtl	3	19.198	16.198
Leon2	pt33b01	rtl	8	23.558	15.558
Leon2	pt33b02	rtl	8	23.558	15.558
Leon2	pt33b03	rtl	8	23.558	15.558
Leon2	pt33b04	rtl	8	23.558	15.558
Leon2	pt33b01u	rtl	8	23.558	15.558
Leon2	pt33b02u	rtl	8	23.558	15.558
Leon2	pt33b03u	rtl	8	23.558	15.558
Leon2	pt33b04u	rtl	8	23.558	15.558
Leon2	pp33o01	rtl	2	18.591	16.591
Leon2	pp33b015vt	rtl	8	23.558	15.558
Leon2	pp33t015vt	rtl	3	19.198	16.198
Leon2	atc25_syncram	rtl	61	101.711	40.711
Leon2	atc25_dpram	rtl	94	116.549	22.549
Leon2	atc25_regfile_iu	rtl	63	87.426	24.426
Leon2	atc25_regfile_cp	rtl	51	108.955	57.955
Leon2	pc3d01	rtl	2	18.591	16.591
Leon2	pc3d21	rtl	2	18.591	16.591
Leon2	pt3o01	rtl	2	18.591	16.591
Leon2	pt3o02	rtl	2	18.591	16.591
Leon2	pt3o03	rtl	2	18.591	16.591
Leon2	pc3t01u	rtl	3	19.198	16.198
Leon2	pc3t02u	rtl	3	19.198	16.198
Leon2	pc3t03u	rtl	3	19.198	16.198
Leon2	pt3b01	rtl	8	23.558	15.558
Leon2	pt3b02	rtl	8	23.558	15.558
Leon2	pt3b03	rtl	8	23.558	15.558
Leon2	atc35_syncram	rtl	36	35.367	-0.633
Leon2	atc35_regfile	rtl	58	100.973	42.973
Leon2	atc35_regfile_cp	rtl	41	78.537	37.537
Leon2	RAM64K36	rtl	51	202.833	151.833

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	axcel_regfile_iu	rtl	183	387.508	204.508
Leon2	axcel_regfile_cp	rtl	109	379.756	270.756
Leon2	axcel_syncram	rtl	78	334.670	256.670
Leon2	uyfaa	rtl	14	20.593	6.593
Leon2	vyfa2gsa	rtl	12	21.629	9.629
Leon2	wyfa2gsa	rtl	18	27.811	9.811
Leon2	fs90_syncram	rtl	50	78.148	28.148
Leon2	fs90_regfile	rtl	55	91.387	36.387
Leon2	rfbypass	rtl	44	111.422	67.422
Leon2	generic_regfile_iu	rtl	85	101.630	16.630
Leon2	generic_regfile_cp	rtl	36	54.979	18.979
Leon2	generic_smult	rtl	21	73.570	52.570
Leon2	generic_clkgen	rtl	27	67.033	40.033
Leon2	geninpad	rtl	2	18.591	16.591
Leon2	gensmpad	rtl	2	18.591	16.591
Leon2	genoutpad	rtl	2	18.591	16.591
Leon2	gentoutpadu	rtl	3	19.198	16.198
Leon2	geniopad	rtl	5	23.558	18.558
Leon2	geniodpad	rtl	5	22.950	17.950
Leon2	genodpad	rtl	2	18.591	16.591
Leon2	regfile_iu	rtl	59	58.742	-0.258
Leon2	regfile_cp	rtl	49	46.971	-2.029
Leon2	hw_smult	rtl	30	36.542	6.542
Leon2	inpad	rtl	26	34.053	8.053
Leon2	smpad	rtl	26	34.053	8.053
Leon2	outpad	rtl	29	35.382	6.382
Leon2	toutpadu	rtl	29	38.150	9.150
Leon2	iopad	rtl	34	44.669	10.669
Leon2	smiopad	rtl	34	44.669	10.669
Leon2	odpad	rtl	29	35.382	6.382
Leon2	iodpad	rtl	29	41.902	12.902
Leon2	pcioutpad	rtl	12	23.745	11.745
Leon2	pcitoutpad	rtl	12	25.072	13.072
Leon2	pciiopad	rtl	19	30.152	11.152
Leon2	pciiodpad	rtl	14	28.824	14.824
Leon2	clkgen	rtl	17	29.650	12.650
Leon2	RAM256x9SST	rtl	25	59.443	34.443
Leon2	RAM256x9SA	rtl	25	58.835	33.835
Leon2	proasic_regfile_iu	rtl	69	172.705	103.705
Leon2	proasic_regfile_cp	rtl	63	173.137	110.137
Leon2	proasic_syncram	rtl	103	271.607	168.607
Leon2	PDIDGZ	rtl	2	18.591	16.591
Leon2	PDISDGZ	rtl	2	18.591	16.591
Leon2	PDT02DGZ	rtl	4	19.198	15.198
Leon2	PDT04DGZ	rtl	4	19.198	15.198
Leon2	PDT08DGZ	rtl	4	19.198	15.198
Leon2	PDT12DGZ	rtl	4	19.198	15.198
Leon2	PDT16DGZ	rtl	4	19.198	15.198
Leon2	PDT24DGZ	rtl	4	19.198	15.198
Leon2	PDU02DGZ	rtl	5	23.558	18.558
Leon2	PDU04DGZ	rtl	5	23.558	18.558
Leon2	PDU08DGZ	rtl	5	23.558	18.558
Leon2	PDU12DGZ	rtl	5	23.558	18.558
Leon2	PDU16DGZ	rtl	5	23.558	18.558
Leon2	PDU24DGZ	rtl	5	23.558	18.558
Leon2	PDB02DGZ	rtl	5	23.558	18.558
Leon2	PDB04DGZ	rtl	5	23.558	18.558
Leon2	PDB08DGZ	rtl	5	23.558	18.558
Leon2	PDB12DGZ	rtl	5	23.558	18.558
Leon2	PDB16DGZ	rtl	5	23.558	18.558
Leon2	PDB24DGZ	rtl	5	23.558	18.558
Leon2	PDB02SDGZ	rtl	5	23.558	18.558

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	PDB04SDGZ	rtl	5	23.558	18.558
Leon2	PDB08SDGZ	rtl	5	23.558	18.558
Leon2	PDB12SDGZ	rtl	5	23.558	18.558
Leon2	PDB16SDGZ	rtl	5	23.558	18.558
Leon2	PDB24SDGZ	rtl	5	23.558	18.558
Leon2	tsmc25_syncram	rtl	95	48.677	-46.323
Leon2	tsmc25_dpram	rtl	102	65.388	-36.612
Leon2	tsmc25_regfile_iu	rtl	102	60.761	-41.239
Leon2	tsmc25_regfile_cp	rtl	77	47.799	-29.201
Leon2	C3I40	rtl	2	18.591	16.591
Leon2	C3I42	rtl	2	18.591	16.591
Leon2	C3O10	rtl	2	18.591	16.591
Leon2	C3O20	rtl	2	18.591	16.591
Leon2	C3O40	rtl	2	18.591	16.591
Leon2	C3B10U	rtl	8	23.558	15.558
Leon2	C3B20U	rtl	8	23.558	15.558
Leon2	C3B40U	rtl	8	23.558	15.558
Leon2	C3B10	rtl	8	23.558	15.558
Leon2	C3B20	rtl	8	23.558	15.558
Leon2	C3B40	rtl	8	23.558	15.558
Leon2	CD3B10T	rtl	8	23.558	15.558
Leon2	CD3B20T	rtl	8	23.558	15.558
Leon2	CD3B40T	rtl	8	23.558	15.558
Leon2	CD3O10T	rtl	7	18.591	11.591
Leon2	CD3O20T	rtl	7	18.591	11.591
Leon2	CD3O40T	rtl	7	18.591	11.591
Leon2	C3B42	rtl	8	23.558	15.558
Leon2	INVDL	rtl	2	18.591	16.591
Leon2	AND2DL	rtl	2	19.198	17.198
Leon2	OR2DL	rtl	2	19.198	17.198
Leon2	EXOR2DL	rtl	2	19.198	17.198
Leon2	umc18_syncram	rtl	56	42.083	-13.917
Leon2	umc18_regfile	rtl	43	44.164	1.164
Leon2	timers	rtl	164	180.759	16.759
Leon2	uart	rtl	276	229.106	-46.894
Leon2	wprot	rtl	94	176.191	82.191

14.13 K2E

14.13.1 Result summary

Population statistical properties and model accuracy:

	All architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	152.359	105.360	-46.999
Variance	632469.174	41053.279	390765.354
Standard deviation	795.279	202.616	625.112
	Behavioral architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	274.560	145.447	-129.113
Variance	1505949.100	80532.063	948969.389
Standard deviation	1227.171	283.782	974.151
	Structural architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	68.632	69.808	1.176
Variance	19345.818	16044.495	1261.185
Standard deviation	139.089	126.667	35.513
	Data-flow architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	71.242	81.406	10.165
Variance	39803.835	12085.676	15290.088
Standard deviation	199.509	109.935	123.653

Correlation between estimated and real values:

	Correlation coefficient $(L, \hat{L})$
All architectures	0.8774
Behavioral architectures only	0.9153
Structural architectures only	0.9686
Data-flow architectures only	0.8343

Figure 14.57: Model K2E: Real vs. estimated lines of code.

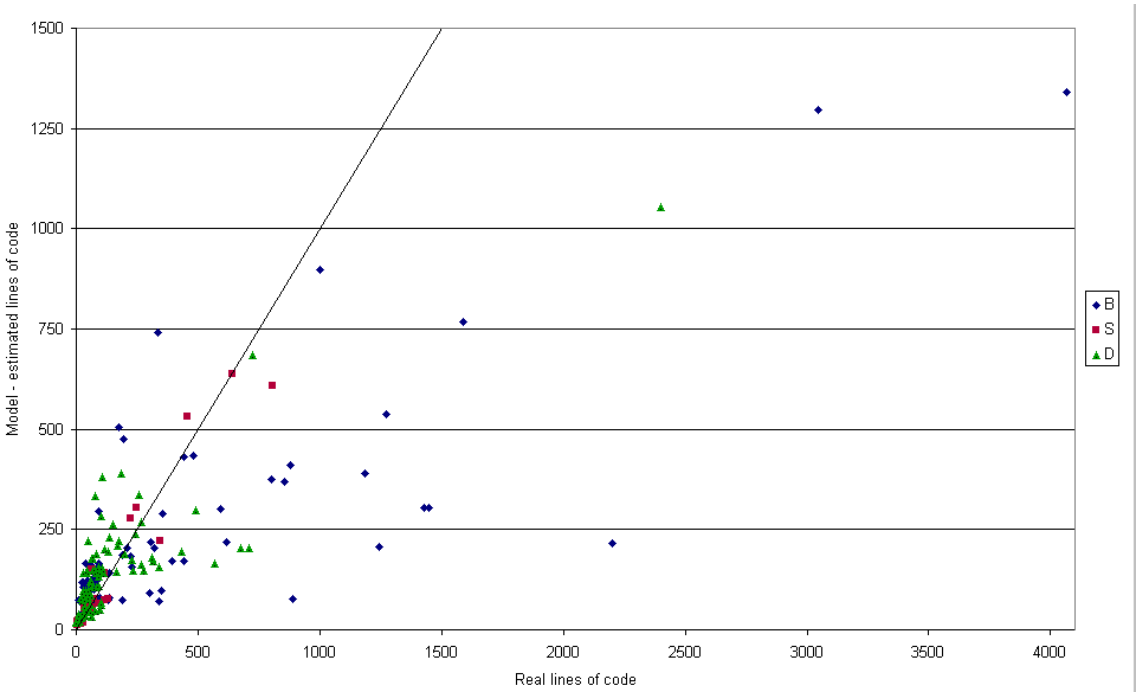


Figure 14.58: Models K2Eb, K2Es, K2Ed: Real vs. estimated lines of code.

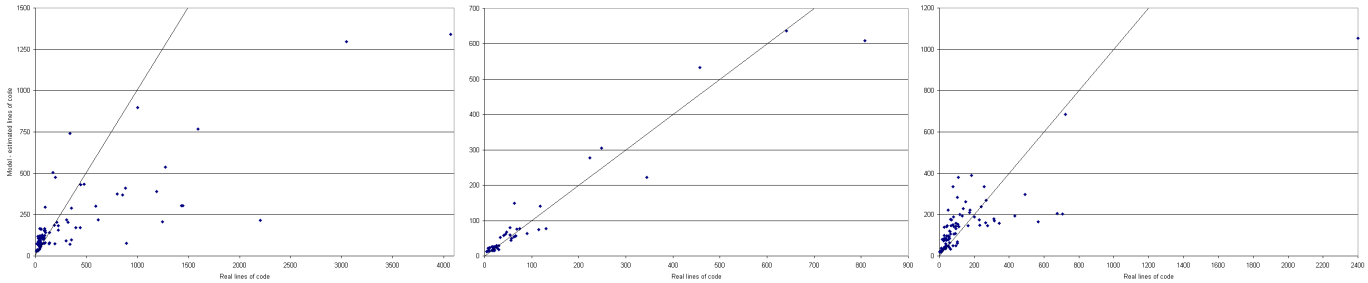


Figure 14.59: Model K2E: Error density distribution.

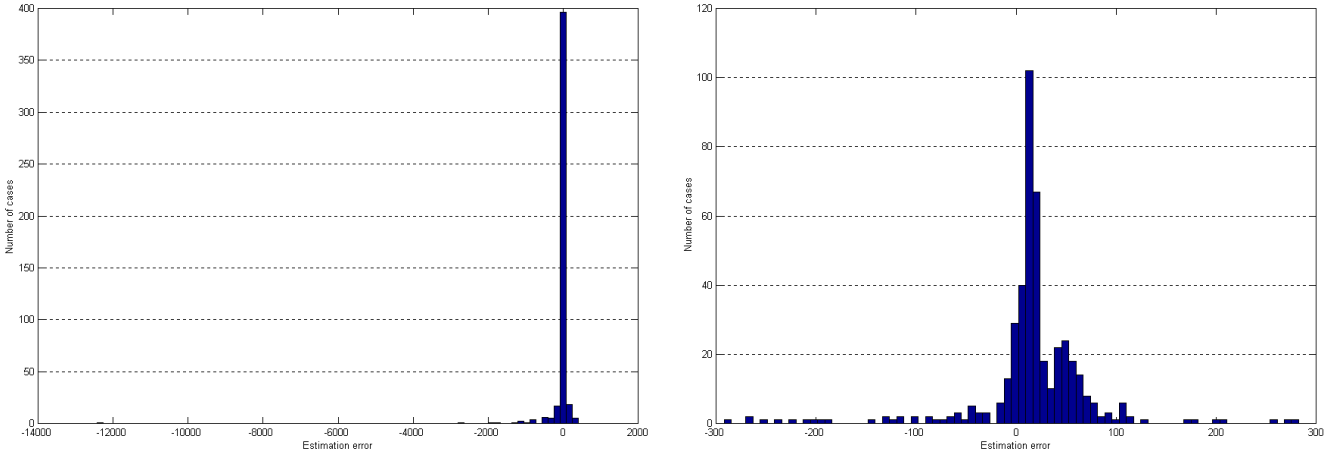


Figure 14.60: Model K2E: Error cumulative distribution.

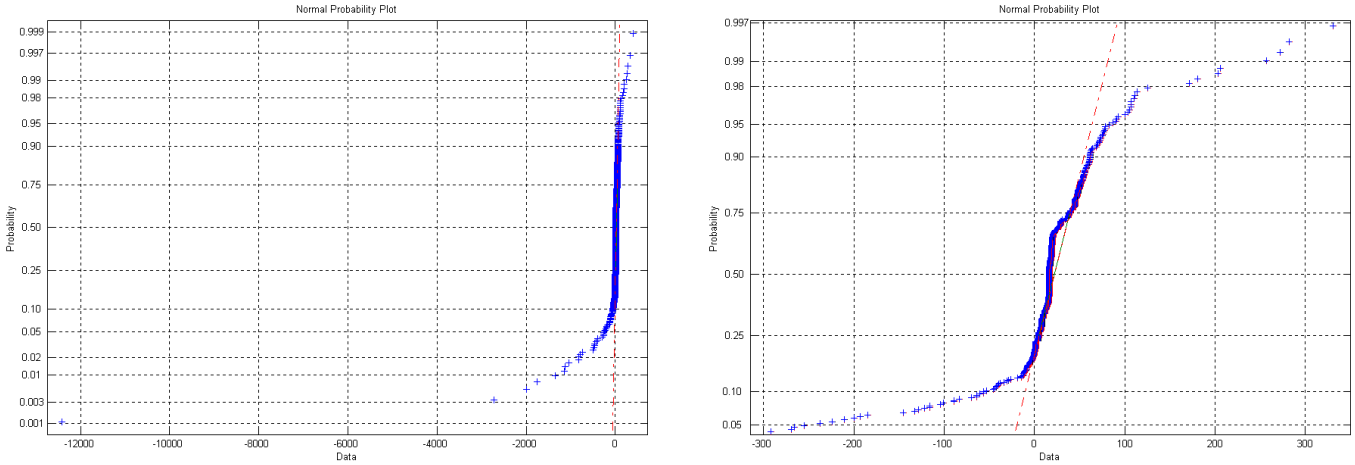
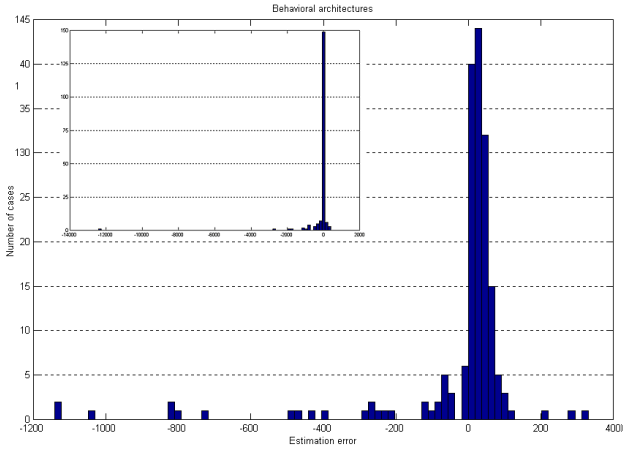
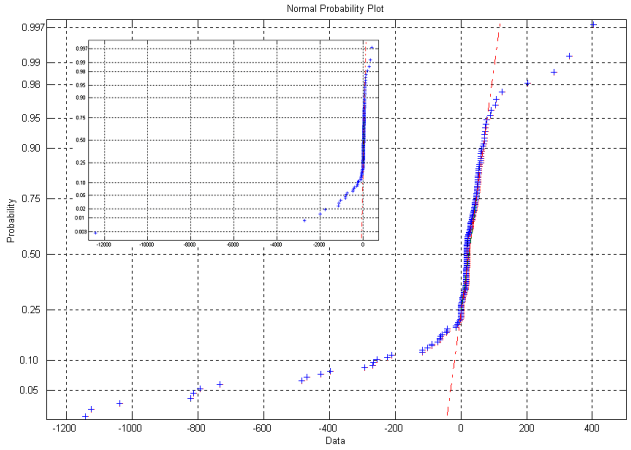
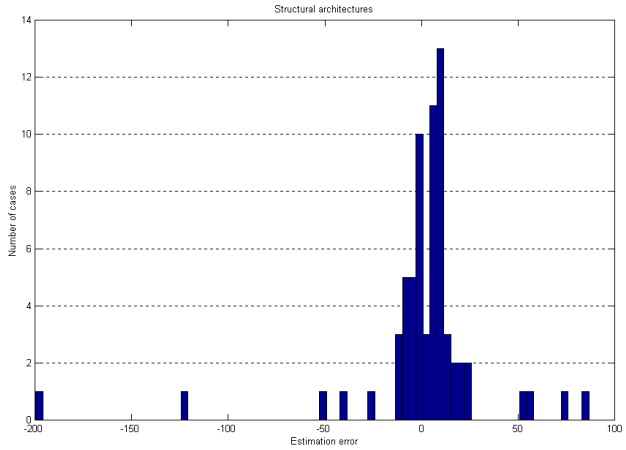
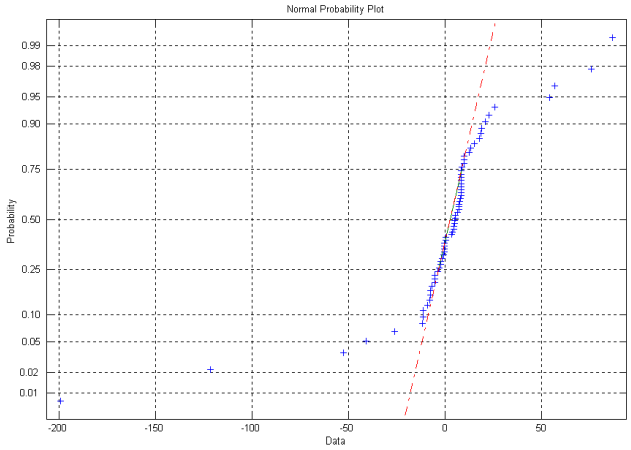


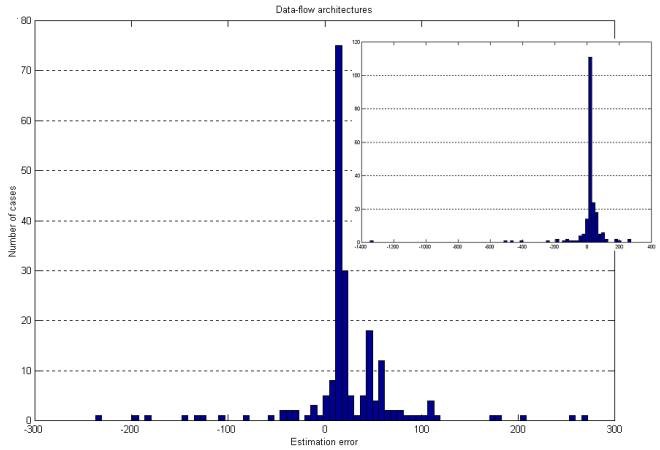
Figure 14.61: Models K2Eb, K2Es, K2Ed: Error density and cumulative distributions.

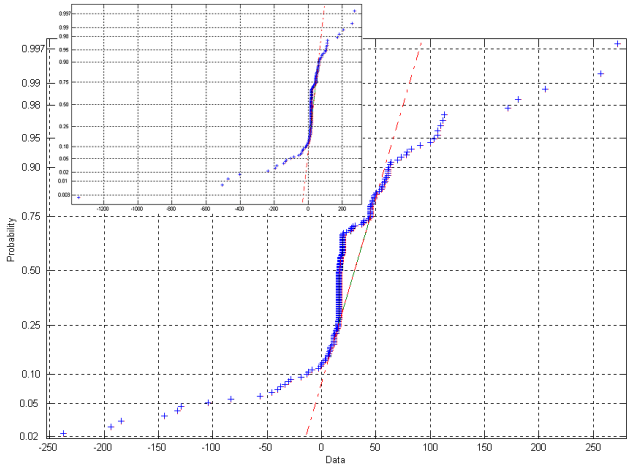












14.13.2 Detailed Results

Real sizes versus estimated sizes for behavioral architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
AMCC5933	amcc5933	behavior	593	301.236	-291.764
DLX2	Gen_Immediate	behavioral	33	73.088	40.088
DLX2	stats	behavioral	24	75.161	51.161
DLX2	dlx_control	behavioral	337	740.433	403.433
DLX2	dlx_data_path	behavioral	173	504.014	331.014
DLX2	Data_Reg	behavioral	25	73.858	48.858
DLX2	Bypass_Logic	behavioral	43	43.381	0.381
DLX2	Decode_Ird	behavioral	102	140.400	38.400
DLX2	Decode_PC	behavioral	93	296.352	203.352
DLX2	dlx_stats	behavioral	24	71.469	47.469
DLX2	Dmemory	behavioral	89	123.776	34.776
DLX2	Imemory	behavioral	59	80.669	21.669
DLX2	Ireg_Fetch	behavioral	32	74.417	42.417
DLX2	Ireg_Decode	behavioral	37	86.810	49.810
DLX2	Ireg_Execute	behavioral	41	91.302	50.302
DLX2	Ireg_Memory	behavioral	34	84.669	50.669
DLX2	Ireg_Writeback	behavioral	28	76.543	48.543
DLX2	Register_File	behavioral	41	166.352	125.352
DSP320VC33	dsp320vc33	vhdl_behavioral	3045	1295.226	-1749.774
DSP320VC33	sram16k32	vhdl_behavioral	479	435.038	-43.962
DSP320VC33	sram1k32	vhdl_behavioral	442	431.958	-10.042
DSP6211	dsp6211	vhdl_behavioral	4068	1341.541	-2726.459
DSP6211	km416s4030	vhdl_behavioral	1446	304.752	-1141.248
DSP6211	MT58L32L32P	behave	224	182.056	-41.944
DSP6211	sram1k16	vhdl_behavioral	353	289.285	-63.715
DSP6211	sram1k8	vhdl_behavioral	307	217.949	-89.051
DSP6415	at9366	vhdl_behavioral	352	97.017	-254.983
DSP6415	cy7c453	vhdl_behavioral	801	373.923	-427.077
DSP6415	dsp6415	vhdl_behavioral	15508	3088.234	-12419.766
DSP6415	idt71v546	vhdl_behavioral	1271	537.399	-733.601
DSP6415	km416s4030	vhdl_behavioral	1429	304.752	-1124.248
DSP6415	MS32PCI	Behavior	615	217.598	-397.402
DSP6415	MT58L32L32P	behave	224	182.056	-41.944
DSP6415	sram1k16	vhdl_behavioral	353	289.285	-63.715
DSP6415	sram1k8	vhdl_behavioral	307	217.949	-89.051
DSP6415	TG32PCI	Behavior	1185	391.190	-793.810
fw09	addressing_decode	behave	137	80.274	-56.726
fw09	alu	behave	191	75.003	-115.997
fw09	busstatus	behave	39	77.240	38.240
fw09	instruction_decode	behave	890	76.681	-813.319
fw09	mainstate	behave	21	73.088	52.088
fw09	mux16alu_left	behave	67	113.330	46.330
fw09	mux16alu_right	behave	63	70.850	7.850
fw09	muxaddressbus	behave	32	71.969	39.969
fw09	muxdatabus	behave	54	69.172	15.172
fw09	pb_decode	behave	95	80.274	-14.726
fw09	reg8	behave	24	72.529	48.529
fw09	registercc	behave	52	114.449	62.449
fw09	registerd	behave	85	110.533	25.533
fw09	registerdp	behave	42	115.568	73.568
fw09	registerindexstack	behave	97	154.132	57.132
fw09	registerpc	behave	85	154.691	69.691
fw09	registertemp	behave	53	115.008	62.008
fw09	regpage	behave	30	72.529	42.529
fw09	statedecode	behave	2201	216.379	-1984.621
fw09	transfer_decode	behave	81	77.240	-3.760
fw09	vectortable	behave	32	74.207	42.207
HDLLib	core_mac	behv	133	72.529	-60.471
HDLLib	ram128x8	behv	338	71.969	-266.031
IEEE1149	br_cell	behavioral	13	73.088	60.088

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
IEEE1149	dr_cell	behavioral	26	118.601	92.601
IEEE1149	ir_cell	behavioral	28	118.042	90.042
IEEE1149	mux_2_1	behavioral	12	73.088	61.088
IEEE1149	mux_4_1	behavioral	17	71.969	54.969
IEEE1149	tap_controller	behavioral	138	142.074	4.074
Manticore	demo_xform	behavioural	226	156.361	-69.639
Manticore	rasterizer	behavioural	880	411.756	-468.244
Manticore	raster_ctrl	behav	302	91.677	-210.323
Manticore	raster_vars_reg	behavioural	854	369.031	-484.969
Manticore	raster_var_type_reg	behavioural	17	73.088	56.088
Manticore	sdram_control	behav	395	170.958	-224.042
Manticore	sdram_control_param	behav	440	170.958	-269.042
Manticore	vgafifo_ctrl	behavioural	319	202.677	-116.323
Manticore	vgaout	behavior	188	185.910	-2.090
RLS	Fp_Divide	bhv	76	107.008	31.008
RLS	MainCtl	Bhv	194	476.577	282.577
RLS	RLSTop	bhv	211	202.305	-8.695
RLS	rls_mult	bhv	92	164.268	72.268
RLS	SubCtl	bhv	999	898.543	-100.457
SPIM-Pipe	control	behavioral	40	91.849	51.849
SPIM-Pipe	decode	behavioral	66	126.584	60.584
SPIM-Pipe	execute	behavioral	43	72.765	29.765
SPIM-Pipe	fetch	behavioral	48	76.681	28.681
SPIM-Pipe	memory	behavioral	36	71.410	35.410
SPIM-Pipe	pipe_reg1	behavioral	21	75.562	54.562
SPIM-Pipe	pipe_reg2	behavioral	50	102.779	52.779
SPIM-Pipe	pipe_reg3	behavioral	33	84.850	51.850
SPIM-Pipe	pipe_reg4	behavioral	29	82.985	53.985
SPIM	control	behavioral	47	94.883	47.883
SPIM	decode	behavioral	70	120.516	50.516
SPIM	execute	behavioral	49	77.477	28.477
SPIM	fetch	behavioral	52	75.003	23.003
SPIM	memory	behavioral	34	71.410	37.410
STD8980	std8980	vhdl_behavioral	1246	207.709	-1038.291
ZR36060	zr36060	vhdl_behavioral	1590	767.921	-822.079
DLX2	Main_Alu	behavioral	54	161.404	107.404
DLX2	Inverter	behavioral	13	74.207	61.207
DLX2	Data_Mux2	behavioral	20	73.088	53.088
DLX2	Data_Mux4	behavioral	28	71.969	43.969
Leon2	atc25_syncram_sim	behavioral	29	74.069	45.069
Leon2	atc25_2pram	behav	38	75.398	37.398
Leon2	atc25_dpram_sim	behav	53	76.194	23.194
Leon2	RAM_256x26	behavioral	14	33.427	19.427
Leon2	RAM_256x28	behavioral	14	33.427	19.427
Leon2	RAM_256x30	behavioral	14	33.427	19.427
Leon2	RAM_512x28	behavioral	14	33.427	19.427
Leon2	RAM_512x30	behavioral	14	33.427	19.427
Leon2	RAM_512x32	behavioral	14	33.427	19.427
Leon2	RAM_1024x32	behavioral	14	33.427	19.427
Leon2	RAM_2048x32	behavioral	14	33.427	19.427
Leon2	RAM2P_16X32	behav	13	33.427	20.427
Leon2	RAM2P_136X32	behav	13	33.427	20.427
Leon2	RAM2P_168X32	behav	13	33.427	20.427
Leon2	DPRAM_256x26	behav	20	39.588	19.588
Leon2	DPRAM_256x28	behav	20	39.588	19.588
Leon2	DPRAM_256x30	behav	20	39.588	19.588
Leon2	DPRAM_256x32	behav	20	39.588	19.588
Leon2	DPRAM_512x28	behav	20	39.588	19.588
Leon2	DPRAM_512x30	behav	20	39.588	19.588
Leon2	DPRAM_512x32	behav	20	39.588	19.588
Leon2	atc35_dpram_ss_dn	behav	38	75.398	37.398
Leon2	ATC35_RAM_256x26	behavioral	17	33.473	16.473

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	ATC35_RAM_1024x32	behavioral	17	33.473	16.473
Leon2	ATC35_RAM_2048x32	behavioral	17	33.473	16.473
Leon2	ATC35_RAM_256x28	behavioral	17	33.473	16.473
Leon2	ATC35_RAM_1024x34	behavioral	17	33.473	16.473
Leon2	ATC35_RAM_2048x34	behavioral	17	33.473	16.473
Leon2	DPRAMRWRW_16X32	behav	21	35.342	14.342
Leon2	DPRAMRWRW_136X32	behav	21	35.342	14.342
Leon2	DPRAMRWRW_168X32	behav	21	35.342	14.342
Leon2	fs90_syncram_sim	behavioral	34	74.069	40.069
Leon2	fs90_dpram_ss	behav	40	73.720	33.720
Leon2	SA108019	behavioral	27	89.453	62.453
Leon2	SU004020	behavioral	29	105.654	76.654
Leon2	SW204420	behavioral	32	103.839	71.839
Leon2	generic_syncram	behavioral	41	118.786	77.786
Leon2	generic_dpram_ss	behav	48	120.115	72.115
Leon2	generic_dpram_as	behav	45	120.115	75.115
Leon2	syncram	behav	56	76.388	20.388
Leon2	dpsyncram	behav	53	106.825	53.825
Leon2	tsmc25_syncram_ss	behavioral	57	74.628	17.628
Leon2	tsmc25_dpram_ss	behav	91	76.194	-14.806
Leon2	ram4096x32	behavioral	23	32.913	9.913
Leon2	ram1024x32	behavioral	23	32.913	9.913
Leon2	ram2400x32	behavioral	28	32.913	4.913
Leon2	ram2048x32	behavioral	23	32.913	9.913
Leon2	ram256x24	behavioral	28	32.913	4.913
Leon2	ram256x27	behavioral	28	32.913	4.913
Leon2	ram512x23	behavioral	28	32.913	4.913
Leon2	dpram16x32	behavioral	38	39.588	1.588
Leon2	dpram136x32	behavioral	38	39.588	1.588
Leon2	dpram168x32	behavioral	38	39.588	1.588
Leon2	dpram256x26	behavioral	38	39.588	1.588
Leon2	dpram256x28	behavioral	38	39.588	1.588
Leon2	dpram256x30	behavioral	38	39.588	1.588
Leon2	dpram256x32	behavioral	38	39.588	1.588
Leon2	dpram512x28	behavioral	38	39.588	1.588
Leon2	dpram512x30	behavioral	38	39.588	1.588
Leon2	TIEHI	behavioral	9	30.608	21.608
Leon2	TIELO	behavioral	9	30.608	21.608
Leon2	umc18_dpram_ss	behav	58	163.155	105.155
Leon2	umc18_syncram_ss	behavioral	50	118.227	68.227
Leon2	R256X24M4	behavioral	18	33.427	15.427
Leon2	R256X25M4	behavioral	18	33.427	15.427
Leon2	R256X26M4	behavioral	18	33.427	15.427
Leon2	R1024X32M4	behavioral	18	33.427	15.427
Leon2	R2048X32M8	behavioral	18	33.427	15.427
Leon2	R256X28M4	behavioral	18	33.427	15.427
Leon2	RF136X32M1	behav	17	33.940	16.940
Leon2	RF168X32M1	behav	17	33.940	16.940
Leon2	RAMB4_S16	behav	14	32.354	18.354
Leon2	RAMB4_S8	behav	14	32.354	18.354
Leon2	RAMB4_S4	behav	14	32.354	18.354
Leon2	RAMB4_S2	behav	14	32.354	18.354
Leon2	RAMB4_S1	behav	14	32.354	18.354
Leon2	RAMB4_S1_S1	behav	42	71.087	29.087
Leon2	RAMB4_S2_S2	behav	42	71.087	29.087
Leon2	RAMB4_S8_S8	behav	42	71.087	29.087
Leon2	RAMB4_S4_S4	behav	42	71.087	29.087
Leon2	RAMB4_S16_S16	behav	42	71.087	29.087
Leon2	virtex_syncram	behav	62	75.031	13.031
Leon2	virtex_regfile	behav	54	55.511	1.511
Leon2	virtex_regfile_cp	behav	36	54.741	18.741
Leon2	virtex_dpram	behav	74	99.696	25.696

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$

Real sizes versus estimated sizes for structural architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
fw09	core_6809e	structure	808	608.551	-199.449
IEEE1149	dr	structural	33	51.797	18.797
IEEE1149	ir	structural	40	58.547	18.547
IEEE1149	testable_nibble_comparator	structural	63	149.67	86.67
LFSR	prpt	structural	223	277.027	54.027
LFSR	prpt	struct_generate	74	77.301	3.301
LFSR	prpt_stage	structural	54	79.524	25.524
LFSR	prpt_stage1	structural	47	67.587	20.587
Manticore	calc_test	structural	457	532.772	75.772
Manticore	frame_buffer_test	structural	641	636.917	-4.083
Manticore	manticore_fifo	struct	68	76.602	8.602
Manticore	pll2x	SYN	56	43.979	-12.021
Manticore	shiftreg_8x8	structural	54	58.808	4.808
Manticore	slope_calc	structural	344	222.164	-121.836
Manticore	vgafifo	SYN	65	57.179	-7.821
Manticore	write_fifo_address	SYN	62	53.885	-8.115
Manticore	write_fifo_data	SYN	66	56.791	-9.209
Manticore	write_fifo_mask	SYN	57	50.119	-6.881
Manticore	zfifo	SYN	65	57.179	-7.821
SPIM-Pipe	spim_pipe	structural	248	304.77	56.77
SPIM	ss_spim	structural	118	140.448	22.448
Leon2	ahbtest	struct	90	63.847	-26.153
Leon2	dcom	struct	130	77.276	-52.724
Leon2	dma	struct	115	74.071	-40.929
Leon2	pci_is	struct	44	61.663	17.663
Leon2	atc25_inpad	syn	4	12.241	8.241
Leon2	atc25_smpad	syn	4	12.241	8.241
Leon2	atc25_outpad	syn	15	14.498	-0.502
Leon2	atc25_toutpadu	syn	16	16.426	0.426
Leon2	atc25_iopad	syn	16	25.786	9.786
Leon2	atc25_iopadu	syn	16	25.786	9.786
Leon2	atc25_iodpad	syn	18	25.089	7.089
Leon2	atc25_odpad	syn	18	15.728	-2.272
Leon2	atc25_pcioutpad	syn	4	12.241	8.241
Leon2	atc25_pcitoutpad	syn	6	13.348	7.348
Leon2	atc25_pciiopad	syn	7	21.889	14.889
Leon2	atc25_pciiodpad	syn	9	21.191	12.191
Leon2	atc35_inpad	syn	4	12.241	8.241
Leon2	atc35_smpad	syn	4	12.241	8.241
Leon2	atc35_outpad	syn	15	14.498	-0.502
Leon2	atc35_toutpadu	syn	16	16.426	0.426
Leon2	atc35_iopad	syn	16	25.786	9.786
Leon2	atc35_iodpad	syn	18	25.089	7.089
Leon2	atc35_odpad	syn	18	15.728	-2.272
Leon2	fs90_inpad	syn	9	13.471	4.471
Leon2	fs90_smpad	syn	7	13.471	6.471
Leon2	fs90_outpad	syn	21	15.62	-5.38
Leon2	fs90_toutpadu	syn	21	17.958	-3.042
Leon2	fs90_iopad	syn	21	26.088	5.088
Leon2	fs90_smiopad	syn	21	26.088	5.088
Leon2	fs90_iodpad	syn	21	25.391	4.391
Leon2	fs90_odpad	syn	21	15.62	-5.38
Leon2	tsmc25_inpad	syn	4	12.241	8.241
Leon2	tsmc25_smpad	syn	4	12.241	8.241
Leon2	tsmc25_outpad	syn	30	18.404	-11.596
Leon2	tsmc25_toutpadu	syn	27	21.508	-5.492
Leon2	tsmc25_iopad	syn	25	29.638	4.638

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	tsmc25_iodpad	syn	30	28.995	-1.005
Leon2	tsmc25_odpad	syn	30	18.404	-11.596
Leon2	tsmc25_smiopad	syn	26	29.638	3.638
Leon2	umc18_inpad	syn	4	12.241	8.241
Leon2	umc18_smpad	syn	4	12.241	8.241
Leon2	umc18_outpad	syn	15	14.498	-0.502
Leon2	umc18_toutpadu	syn	18	17.656	-0.344
Leon2	umc18_iopad	syn	18	25.786	7.786
Leon2	umc18_iodpad	syn	18	25.089	7.089
Leon2	umc18_odpad	syn	16	14.498	-1.502
Leon2	umc18_smiopad	syn	10	23.218	13.218

Real sizes versus estimated sizes for data-flow architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
RTC	ALU	ALU_rtl	315	170.041	-144.959
RTC	BCD2BIN	BCD2BIN_rtl	18	79.374	61.374
RTC	BIN2BCD	BIN2BCD_rtl	18	79.374	61.374
RTC	cal_modul	cal_modul_rtl	241	238.555	-2.445
RTC	CLOCK_DIV	CLOCK_DIV_rtl	26	79.449	53.449
RTC	CMDINTERF	CMDINTERF_rtl	49	96.232	47.232
RTC	JAHR_COUNTER	JAHR_COUNTER_rtl	36	84.212	48.212
RTC	monat_counter	monat_counter_rtl	38	82.957	44.957
RTC	MUX2TO1	MUX2TO1_rtl	11	37.663	26.663
RTC	PC	PC_rtl	24	79.756	55.756
RTC	secmin_counter	secmin_counter_rtl	38	82.957	44.957
RTC	tag_counter	tag_counter_rtl	58	85.344	27.344
RTC	uhr_counter	uhr_counter_rtl	38	82.957	44.957
RTC	weekday_counter	weekday_counter_rtl	38	82.957	44.957
RTC	weeknr_counter	weeknr_counter_rtl	38	82.957	44.957
RTC-alt	ALU	ALU_rtl	313	180.389	-132.611
RTC-alt	BCD2BIN	BCD2BIN_rtl	18	79.374	61.374
RTC-alt	BIN2BCD	BIN2BCD_rtl	18	79.374	61.374
RTC-alt	cal_modul	cal_modul_rtl	269	268.474	-0.526
RTC-alt	CLOCK_DIV	CLOCK_DIV_rtl	23	78.404	55.404
RTC-alt	CMDINTERF	CMDINTERF_rtl	49	96.232	47.232
RTC-alt	JAHR_COUNTER	JAHR_COUNTER_rtl	36	84.212	48.212
RTC-alt	monat_counter	monat_counter_rtl	38	82.957	44.957
RTC-alt	MUX2TO1	MUX2TO1_rtl	11	37.663	26.663
RTC-alt	PC	PC_rtl	24	79.756	55.756
RTC-alt	secmin_counter	secmin_counter_rtl	38	82.957	44.957
RTC-alt	tag_counter	tag_counter_rtl	58	85.344	27.344
RTC-alt	uhr_counter	uhr_counter_rtl	38	82.957	44.957
RTC-alt	weekday_counter	weekday_counter_rtl	38	82.957	44.957
RTC-alt	weeknr_counter	weeknr_counter_rtl	38	82.957	44.957
Leon2	acache	rtl	201	188.951	-12.049
Leon2	ahbarb	rtl	175	220.925	45.925
Leon2	ahbmst	rtl	94	157.906	63.906
Leon2	ahbstat	rtl	68	146.905	78.905
Leon2	apbmst	rtl	78	151.430	73.430
Leon2	cache	rtl	81	187.922	106.922
Leon2	cachemem	rtl	92	109.259	17.259
Leon2	dcache	rtl	491	297.540	-193.460
Leon2	dcom_uart	rtl	232	148.622	-83.378
Leon2	div	rtl	118	200.781	82.781
Leon2	dsu	rtl	432	194.683	-237.317
Leon2	dsu_mem	rtl	64	33.742	-30.258
Leon2	fpaux	rtl	42	142.109	100.109
Leon2	fp	rtl	707	202.944	-504.056
Leon2	fp1eu	rtl	566	164.881	-401.119
Leon2	fpu_core	rtl	54	45.590	-8.410
Leon2	fpu_lth	rtl	722	684.242	-37.758

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	icache	rtl	230	173.673	-56.327
Leon2	ioport	rtl	107	153.842	46.842
Leon2	irqctrl	rtl	84	142.284	58.284
Leon2	irqctrl2	rtl	111	141.688	30.688
Leon2	iu	rtl	2401	1052.890	-1348.110
Leon2	lconf	rtl	38	76.221	38.221
Leon2	leon	rtl	131	194.162	63.162
Leon2	leon_pci	rtl	258	336.277	78.277
Leon2	mcore	rtl	172	209.125	37.125
Leon2	mctrl	rtl	675	204.399	-470.601
Leon2	fpu	rtl	32	39.818	7.818
Leon2	mul	rtl	265	160.949	-104.051
Leon2	GEN_NOT	rtl	2	18.491	16.491
Leon2	GEN_AND2	rtl	2	19.535	17.535
Leon2	GEN_OR2	rtl	2	19.535	17.535
Leon2	GEN_XOR2	rtl	2	19.535	17.535
Leon2	pci	rtl	28	98.267	70.267
Leon2	pci_arb	rtl	152	261.990	109.990
Leon2	proc	rtl	138	229.091	91.091
Leon2	rstgen	rtl	29	140.496	111.496
Leon2	sdmctrl	rtl	342	157.645	-184.355
Leon2	pt33d00	rtl	2	18.491	16.491
Leon2	pt33d00u	rtl	3	22.181	19.181
Leon2	pt33d20	rtl	2	18.491	16.491
Leon2	pt33d20u	rtl	3	22.181	19.181
Leon2	pt33o01	rtl	2	18.491	16.491
Leon2	pt33o02	rtl	2	18.491	16.491
Leon2	pt33o03	rtl	2	18.491	16.491
Leon2	pt33o04	rtl	2	18.491	16.491
Leon2	pt33t01u	rtl	3	19.535	16.535
Leon2	pt33t02u	rtl	3	19.535	16.535
Leon2	pt33t03u	rtl	3	19.535	16.535
Leon2	pt33b01	rtl	8	24.270	16.270
Leon2	pt33b02	rtl	8	24.270	16.270
Leon2	pt33b03	rtl	8	24.270	16.270
Leon2	pt33b04	rtl	8	24.270	16.270
Leon2	pt33b01u	rtl	8	24.270	16.270
Leon2	pt33b02u	rtl	8	24.270	16.270
Leon2	pt33b03u	rtl	8	24.270	16.270
Leon2	pt33b04u	rtl	8	24.270	16.270
Leon2	pp33o01	rtl	2	18.491	16.491
Leon2	pp33b015vt	rtl	8	24.270	16.270
Leon2	pp33t015vt	rtl	3	19.535	16.535
Leon2	atc25_syncram	rtl	61	118.928	57.928
Leon2	atc25_dpram	rtl	94	133.033	39.033
Leon2	atc25_regfile_iu	rtl	63	105.779	42.779
Leon2	atc25_regfile_cp	rtl	51	94.651	43.651
Leon2	pc3d01	rtl	2	18.491	16.491
Leon2	pc3d21	rtl	2	18.491	16.491
Leon2	pt3o01	rtl	2	18.491	16.491
Leon2	pt3o02	rtl	2	18.491	16.491
Leon2	pt3o03	rtl	2	18.491	16.491
Leon2	pc3t01u	rtl	3	19.535	16.535
Leon2	pc3t02u	rtl	3	19.535	16.535
Leon2	pc3t03u	rtl	3	19.535	16.535
Leon2	pt3b01	rtl	8	24.270	16.270
Leon2	pt3b02	rtl	8	24.270	16.270
Leon2	pt3b03	rtl	8	24.270	16.270
Leon2	atc35_syncram	rtl	36	36.299	0.299
Leon2	atc35_regfile	rtl	58	119.684	61.684
Leon2	atc35_regfile_cp	rtl	41	96.811	55.811
Leon2	RAM64K36	rtl	51	222.646	171.646

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	axcel_regfile_iu	rtl	183	389.039	206.039
Leon2	axcel_regfile_cp	rtl	109	381.208	272.208
Leon2	axcel_syncram	rtl	78	334.886	256.886
Leon2	uyfaa	rtl	14	21.625	7.625
Leon2	vyfa2gsa	rtl	12	23.714	11.714
Leon2	wyfa2gsa	rtl	18	31.582	13.582
Leon2	fs90_syncram	rtl	50	78.722	28.722
Leon2	fs90_regfile	rtl	55	93.992	38.992
Leon2	rfbypass	rtl	44	147.334	103.334
Leon2	generic_regfile_iu	rtl	85	105.309	20.309
Leon2	generic_regfile_cp	rtl	36	58.579	22.579
Leon2	generic_smult	rtl	21	80.778	59.778
Leon2	generic_clkgen	rtl	27	85.319	58.319
Leon2	geninpad	rtl	2	18.491	16.491
Leon2	gensmpad	rtl	2	18.491	16.491
Leon2	genoutpad	rtl	2	18.491	16.491
Leon2	gentoutpadu	rtl	3	19.535	16.535
Leon2	geniopad	rtl	5	24.270	19.270
Leon2	geniodpad	rtl	5	23.225	18.225
Leon2	genodpad	rtl	2	18.491	16.491
Leon2	regfile_iu	rtl	59	62.600	3.600
Leon2	regfile_cp	rtl	49	50.392	1.392
Leon2	hw_smult	rtl	30	36.879	6.879
Leon2	inpad	rtl	26	33.953	7.953
Leon2	smpad	rtl	26	33.953	7.953
Leon2	outpad	rtl	29	35.282	6.282
Leon2	toutpadu	rtl	29	38.487	9.487
Leon2	iopad	rtl	34	45.381	11.381
Leon2	smiopad	rtl	34	45.381	11.381
Leon2	odpad	rtl	29	35.282	6.282
Leon2	iodpad	rtl	29	42.177	13.177
Leon2	pcioutpad	rtl	12	23.645	11.645
Leon2	pcitoutpad	rtl	12	25.409	13.409
Leon2	pciiopad	rtl	19	30.864	11.864
Leon2	pciiodpad	rtl	14	29.099	15.099
Leon2	clkgen	rtl	17	31.472	14.472
Leon2	RAM256x9SST	rtl	25	74.676	49.676
Leon2	RAM256x9SA	rtl	25	73.631	48.631
Leon2	proasic_regfile_iu	rtl	69	175.847	106.847
Leon2	proasic_regfile_cp	rtl	63	176.379	113.379
Leon2	proasic_syncram	rtl	103	283.775	180.775
Leon2	PDIDGZ	rtl	2	18.491	16.491
Leon2	PDISDGZ	rtl	2	18.491	16.491
Leon2	PDT02DGZ	rtl	4	19.535	15.535
Leon2	PDT04DGZ	rtl	4	19.535	15.535
Leon2	PDT08DGZ	rtl	4	19.535	15.535
Leon2	PDT12DGZ	rtl	4	19.535	15.535
Leon2	PDT16DGZ	rtl	4	19.535	15.535
Leon2	PDT24DGZ	rtl	4	19.535	15.535
Leon2	PDU02DGZ	rtl	5	24.270	19.270
Leon2	PDU04DGZ	rtl	5	24.270	19.270
Leon2	PDU08DGZ	rtl	5	24.270	19.270
Leon2	PDU12DGZ	rtl	5	24.270	19.270
Leon2	PDU16DGZ	rtl	5	24.270	19.270
Leon2	PDU24DGZ	rtl	5	24.270	19.270
Leon2	PDB02DGZ	rtl	5	24.270	19.270
Leon2	PDB04DGZ	rtl	5	24.270	19.270
Leon2	PDB08DGZ	rtl	5	24.270	19.270
Leon2	PDB12DGZ	rtl	5	24.270	19.270
Leon2	PDB16DGZ	rtl	5	24.270	19.270
Leon2	PDB24DGZ	rtl	5	24.270	19.270
Leon2	PDB02SDGZ	rtl	5	24.270	19.270

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	PDB04SDGZ	rtl	5	24.270	19.270
Leon2	PDB08SDGZ	rtl	5	24.270	19.270
Leon2	PDB12SDGZ	rtl	5	24.270	19.270
Leon2	PDB16SDGZ	rtl	5	24.270	19.270
Leon2	PDB24SDGZ	rtl	5	24.270	19.270
Leon2	tsmc25_syncram	rtl	95	49.430	-45.570
Leon2	tsmc25_dpram	rtl	102	68.272	-33.728
Leon2	tsmc25_regfile_iu	rtl	102	61.934	-40.066
Leon2	tsmc25_regfile_cp	rtl	77	48.535	-28.465
Leon2	C3I40	rtl	2	18.491	16.491
Leon2	C3I42	rtl	2	18.491	16.491
Leon2	C3O10	rtl	2	18.491	16.491
Leon2	C3O20	rtl	2	18.491	16.491
Leon2	C3O40	rtl	2	18.491	16.491
Leon2	C3B10U	rtl	8	24.270	16.270
Leon2	C3B20U	rtl	8	24.270	16.270
Leon2	C3B40U	rtl	8	24.270	16.270
Leon2	C3B10	rtl	8	24.270	16.270
Leon2	C3B20	rtl	8	24.270	16.270
Leon2	C3B40	rtl	8	24.270	16.270
Leon2	CD3B10T	rtl	8	24.270	16.270
Leon2	CD3B20T	rtl	8	24.270	16.270
Leon2	CD3B40T	rtl	8	24.270	16.270
Leon2	CD3O10T	rtl	7	18.491	11.491
Leon2	CD3O20T	rtl	7	18.491	11.491
Leon2	CD3O40T	rtl	7	18.491	11.491
Leon2	C3B42	rtl	8	24.270	16.270
Leon2	INVDL	rtl	2	18.491	16.491
Leon2	AND2DL	rtl	2	19.535	17.535
Leon2	OR2DL	rtl	2	19.535	17.535
Leon2	EXOR2DL	rtl	2	19.535	17.535
Leon2	umc18_syncram	rtl	56	42.836	-13.164
Leon2	umc18_regfile	rtl	43	47.306	4.306
Leon2	timers	rtl	164	145.390	-18.610
Leon2	uart	rtl	276	147.069	-128.931
Leon2	wprot	rtl	94	144.668	50.668

14.14 K3E

14.14.1 Result summary

Population statistical properties and model accuracy:

	All architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	152.359	124.985	-27.374
Variance	632469.174	71712.408	308337.868
Standard deviation	795.279	267.792	555.282
	Behavioral architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	274.560	175.419	-99.141
Variance	1505949.100	161704.142	720122.031
Standard deviation	1227.171	402.125	848.600
	Structural architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	68.632	45.667	-22.966
Variance	19345.818	3813.837	7373.657
Standard deviation	139.089	61.756	85.870
	Data-flow architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	71.242	106.212	34.971
Variance	39803.835	9845.238	34896.987
Standard deviation	199.509	99.223	186.807

Correlation between estimated and real values:

	Correlation coefficient $(L, \hat{L})$
All architectures	0.9293
Behavioral architectures only	0.9601
Structural architectures only	0.9189
Data-flow architectures only	0.3726

Figure 14.62: Model K3E: Real vs. estimated lines of code.

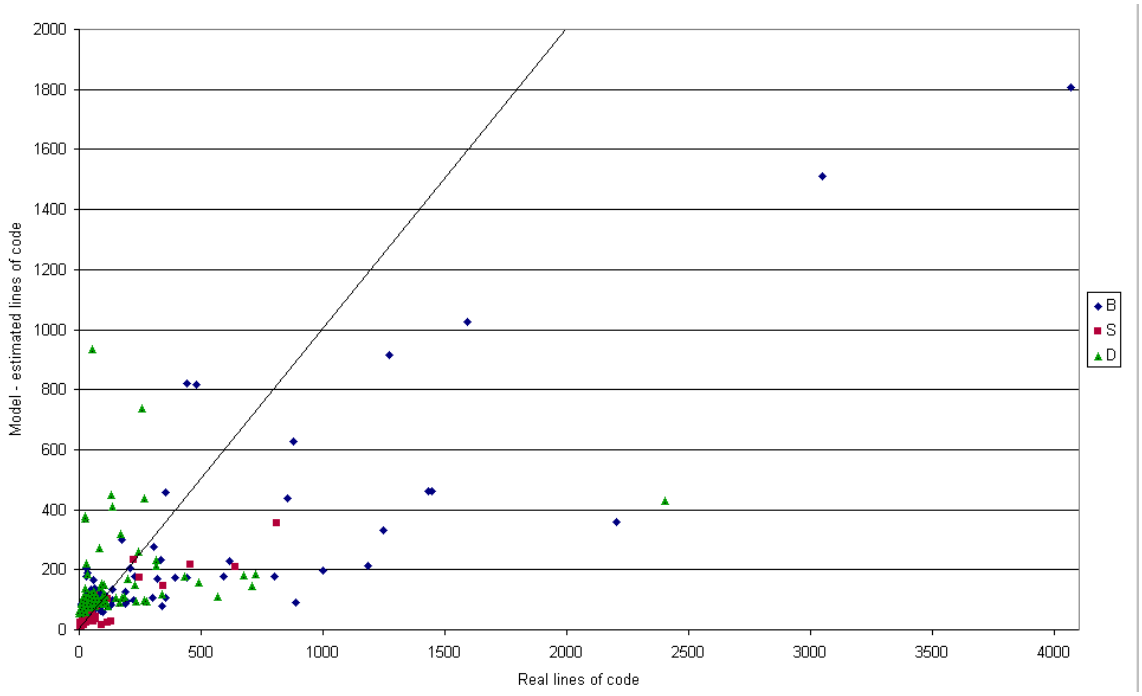


Figure 14.63: Models K3Eb, K3Es, K3Ed: Real vs. estimated lines of code.

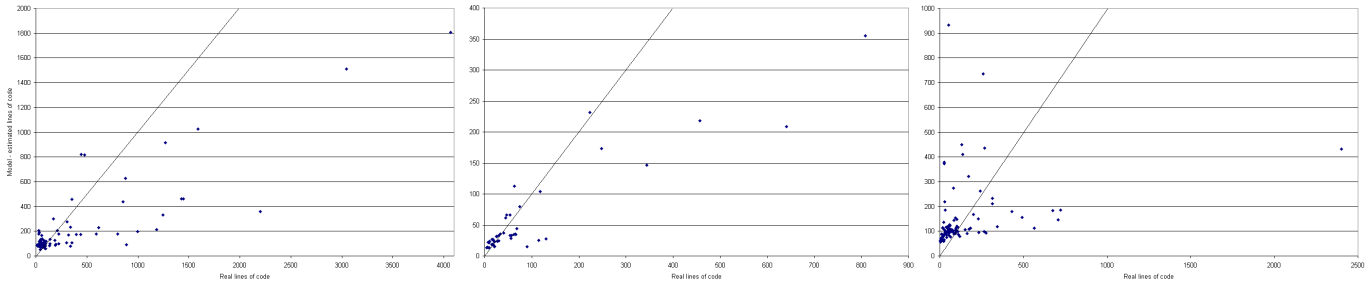


Figure 14.64: Model K3E: Error density distribution.

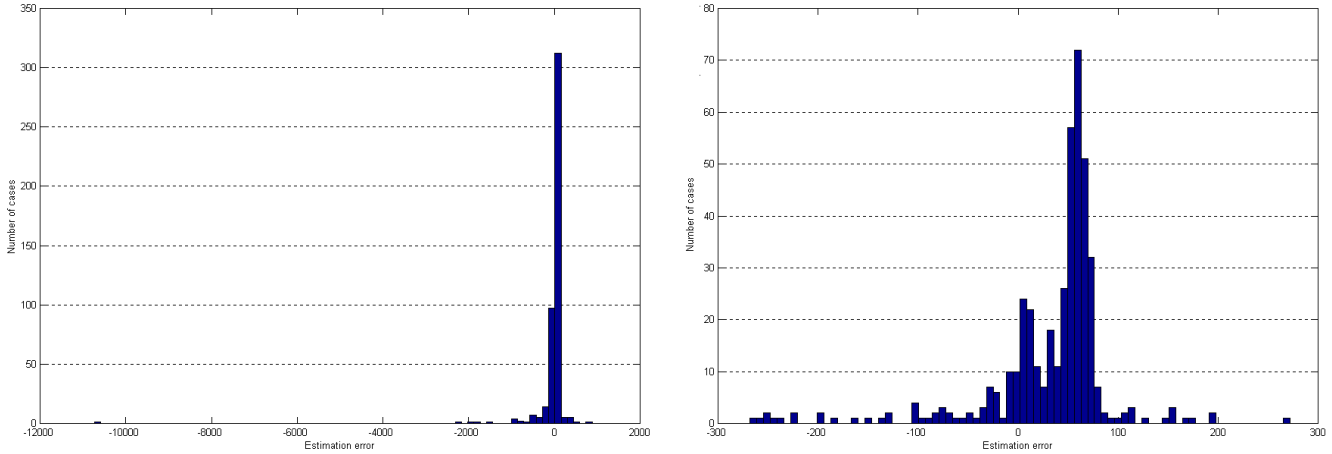


Figure 14.65: Model K3E: Error cumulative distribution.

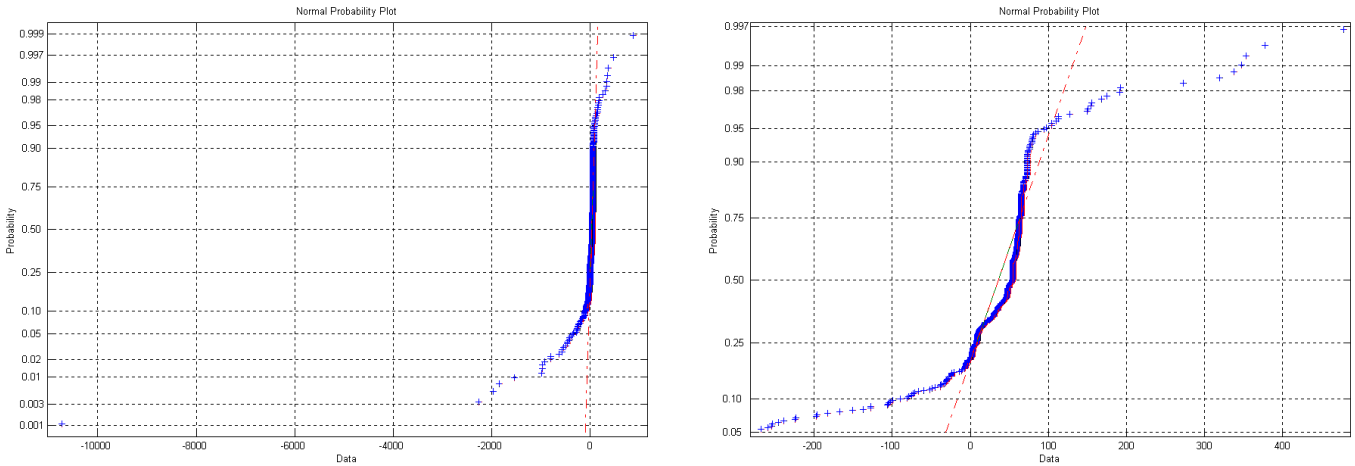
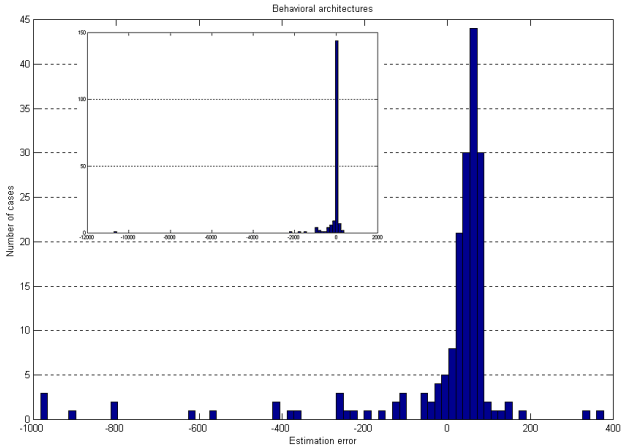
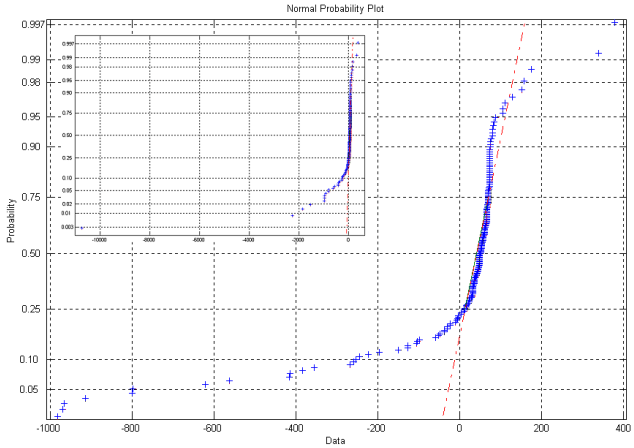
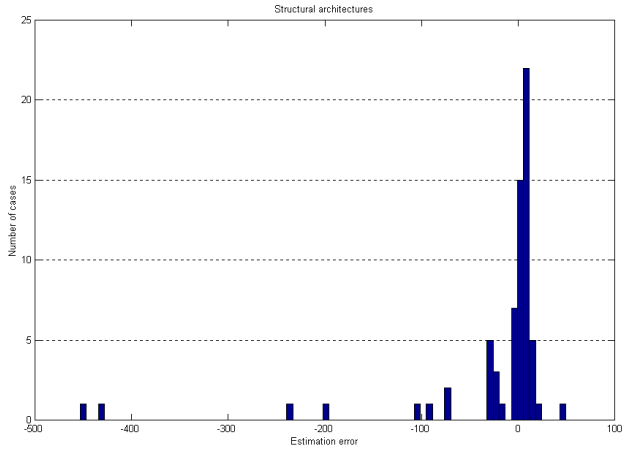
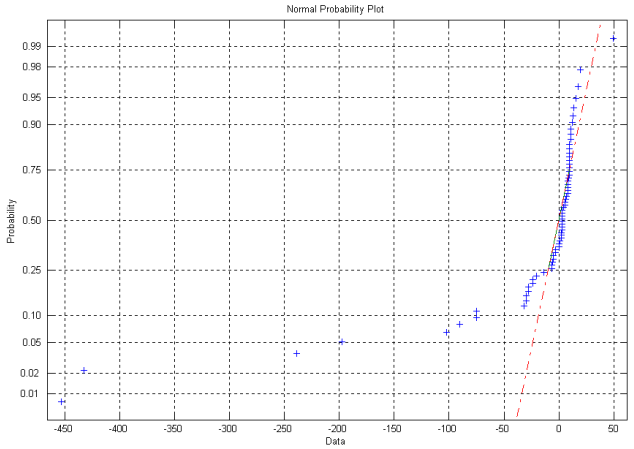


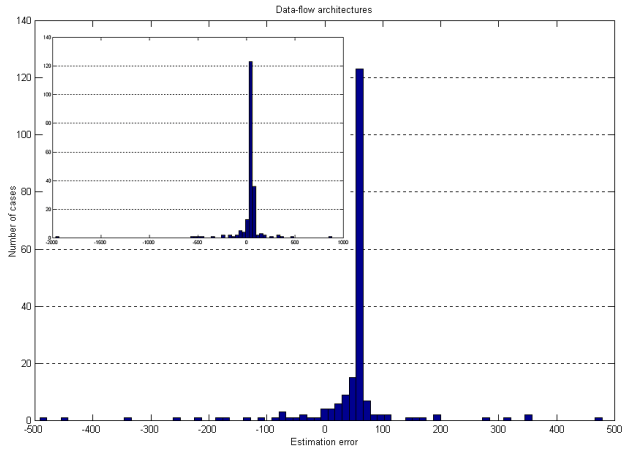
Figure 14.66: Models K3Eb, K3Es, K3Ed: Error density and cumulative distributions.

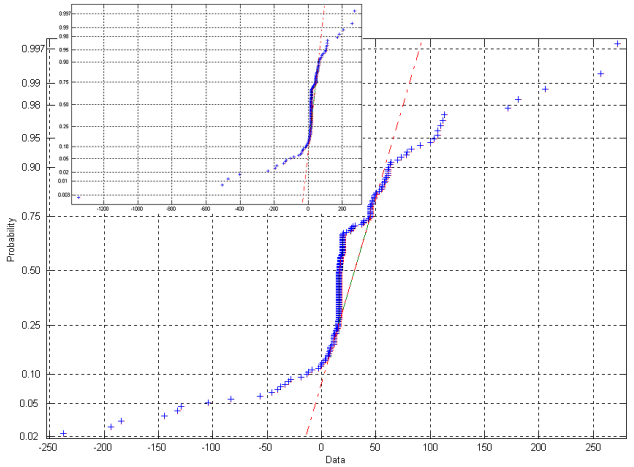












14.14.2 Detailed Results

Real sizes versus estimated sizes for behavioral architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
AMCC5933	amcc5933	behavior	593	178.080	-414.920
DLX2	Gen_Immediate	behavioral	33	82.759	49.759
DLX2	stats	behavioral	24	84.302	60.302
DLX2	dlx_control	behavioral	337	231.997	-105.003
DLX2	dlx_data_path	behavioral	173	300.144	127.144
DLX2	Data_Reg	behavioral	25	82.292	57.292
DLX2	Bypass_Logic	behavioral	43	53.052	10.052
DLX2	Decode_Ird	behavioral	102	119.160	17.160
DLX2	Decode_PC	behavioral	93	86.999	-6.001
DLX2	dlx_stats	behavioral	24	91.505	67.505
DLX2	Dmemory	behavioral	89	120.155	31.155
DLX2	Imemory	behavioral	59	94.579	35.579
DLX2	Ireg_Fetch	behavioral	32	84.088	52.088
DLX2	Ireg_Decode	behavioral	37	96.481	59.481
DLX2	Ireg_Execute	behavioral	41	93.554	52.554
DLX2	Ireg_Memory	behavioral	34	88.158	54.158
DLX2	Ireg_Writeback	behavioral	28	83.741	55.741
DLX2	Register_File	behavioral	41	91.240	50.240
DSP320VC33	dsp320vc33	vhdl_behavioral	3045	1510.761	-1534.239
DSP320VC33	sram16k32	vhdl_behavioral	479	817.049	338.049
DSP320VC33	sram1k32	vhdl_behavioral	442	818.915	376.915
DSP6211	dsp6211	vhdl_behavioral	4068	1808.158	-2259.842
DSP6211	km416s4030	vhdl_behavioral	1446	462.793	-983.207
DSP6211	MT58L32L32P	behave	224	96.787	-127.213
DSP6211	sram1k16	vhdl_behavioral	353	457.218	104.218
DSP6211	sram1k8	vhdl_behavioral	307	276.370	-30.630
DSP6415	at9366	vhdl_behavioral	352	106.688	-245.312
DSP6415	cy7c453	vhdl_behavioral	801	178.524	-622.476
DSP6415	dsp6415	vhdl_behavioral	15508	4776.262	-10731.738
DSP6415	idt71v546	vhdl_behavioral	1271	915.171	-355.829
DSP6415	km416s4030	vhdl_behavioral	1429	462.793	-966.207
DSP6415	MS32PCI	Behavior	615	229.829	-385.171
DSP6415	MT58L32L32P	behave	224	96.787	-127.213
DSP6415	sram1k16	vhdl_behavioral	353	457.218	104.218
DSP6415	sram1k8	vhdl_behavioral	307	276.370	-30.630
DSP6415	TG32PCI	Behavior	1185	214.337	-970.663
fw09	addressing_decode	behave	137	98.423	-38.577
fw09	alu	behave	191	85.203	-105.797
fw09	busstatus	behave	39	92.386	53.386
fw09	instruction_decode	behave	890	90.591	-799.409
fw09	mainstate	behave	21	82.759	61.759
fw09	mux16alu_left	behave	67	70.188	3.188
fw09	mux16alu_right	behave	63	75.575	12.575
fw09	muxaddressbus	behave	32	79.167	47.167
fw09	muxdatabus	behave	54	70.188	16.188
fw09	pb_decode	behave	95	98.423	3.423
fw09	reg8	behave	24	80.963	56.963
fw09	registercc	behave	52	73.780	21.780
fw09	registerd	behave	85	61.209	-23.791
fw09	registerdp	behave	42	77.371	35.371
fw09	registerindexstack	behave	97	59.413	-37.587
fw09	registerpc	behave	85	61.209	-23.791
fw09	registertemp	behave	53	75.575	22.575
fw09	regpage	behave	30	80.963	50.963
fw09	statedecode	behave	2201	357.640	-1843.360
fw09	transfer_decode	behave	81	92.386	11.386
fw09	vectortable	behave	32	86.350	54.350
HDLLib	core_mac	behv	133	80.963	-52.037
HDLLib	ram128x8	behv	338	79.167	-258.833
IEEE1149	br_cell	behavioral	13	82.759	69.759

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
IEEE1149	dr_cell	behavioral	26	83.408	57.408
IEEE1149	ir_cell	behavioral	28	81.612	53.612
IEEE1149	mux_2_1	behavioral	12	82.759	70.759
IEEE1149	mux_4_1	behavioral	17	79.167	62.167
IEEE1149	tap_controller	behavioral	138	132.845	-5.155
Manticore	demo_xform	behavioural	226	177.134	-48.866
Manticore	rasterizer	behavioural	880	626.123	-253.877
Manticore	raster_ctrl	behav	302	105.411	-196.589
Manticore	raster_vars_reg	behavioural	854	437.576	-416.424
Manticore	raster_var_type_reg	behavioural	17	82.759	65.759
Manticore	sdram_control	behav	395	171.797	-223.203
Manticore	sdram_control_param	behav	440	171.797	-268.203
Manticore	vgafifo_ctrl	behavioural	319	168.367	-150.633
Manticore	vgaout	behavior	188	128.107	-59.893
RLS	Fp_Divide	bhv	76	111.730	35.730
RLS	MainCtl	Bhv	194	94.831	-99.169
RLS	RLSTop	bhv	211	206.924	-4.076
RLS	rls_mult	bhv	92	84.970	-7.030
RLS	SubCtl	bhv	999	198.551	-800.449
SPIM-Pipe	control	behavioral	40	120.773	80.773
SPIM-Pipe	decode	behavioral	66	97.925	31.925
SPIM-Pipe	execute	behavioral	43	78.020	35.020
SPIM-Pipe	fetch	behavioral	48	90.591	42.591
SPIM-Pipe	memory	behavioral	36	77.371	41.371
SPIM-Pipe	pipe_reg1	behavioral	21	86.999	65.999
SPIM-Pipe	pipe_reg2	behavioral	50	133.645	83.645
SPIM-Pipe	pipe_reg3	behavioral	33	119.294	86.294
SPIM-Pipe	pipe_reg4	behavioral	29	99.721	70.721
SPIM	control	behavioral	47	126.809	79.809
SPIM	decode	behavioral	70	85.852	15.852
SPIM	execute	behavioral	49	89.444	40.444
SPIM	fetch	behavioral	52	85.203	33.203
SPIM	memory	behavioral	34	77.371	43.371
STD8980	std8980	vhdl_behavioral	1246	331.484	-914.516
ZR36060	zr36060	vhdl_behavioral	1590	1026.112	-563.888
DLX2	Main_Alu	behavioral	54	82.759	28.759
DLX2	Inverter	behavioral	13	86.350	73.350
DLX2	Data_Mux2	behavioral	20	82.759	62.759
DLX2	Data_Mux4	behavioral	28	79.167	51.167
Leon2	atc25_syncram_sim	behavioral	29	80.030	51.030
Leon2	atc25_2pram	behav	38	81.359	43.359
Leon2	atc25_dpram_sim	behav	53	80.213	27.213
Leon2	RAM_256x26	behavioral	14	86.766	72.766
Leon2	RAM_256x28	behavioral	14	86.766	72.766
Leon2	RAM_256x30	behavioral	14	86.766	72.766
Leon2	RAM_512x28	behavioral	14	86.766	72.766
Leon2	RAM_512x30	behavioral	14	86.766	72.766
Leon2	RAM_512x32	behavioral	14	86.766	72.766
Leon2	RAM_1024x32	behavioral	14	86.766	72.766
Leon2	RAM_2048x32	behavioral	14	86.766	72.766
Leon2	RAM2P_16X32	behav	13	86.766	73.766
Leon2	RAM2P_136X32	behav	13	86.766	73.766
Leon2	RAM2P_168X32	behav	13	86.766	73.766
Leon2	DPRAM_256x26	behav	20	85.619	65.619
Leon2	DPRAM_256x28	behav	20	85.619	65.619
Leon2	DPRAM_256x30	behav	20	85.619	65.619
Leon2	DPRAM_256x32	behav	20	85.619	65.619
Leon2	DPRAM_512x28	behav	20	85.619	65.619
Leon2	DPRAM_512x30	behav	20	85.619	65.619
Leon2	DPRAM_512x32	behav	20	85.619	65.619
Leon2	atc35_dpram_ss_dn	behav	38	81.359	43.359
Leon2	ATC35_RAM_256x26	behavioral	17	90.358	73.358

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	ATC35_RAM_1024x32	behavioral	17	90.358	73.358
Leon2	ATC35_RAM_2048x32	behavioral	17	90.358	73.358
Leon2	ATC35_RAM_256x28	behavioral	17	90.358	73.358
Leon2	ATC35_RAM_1024x34	behavioral	17	90.358	73.358
Leon2	ATC35_RAM_2048x34	behavioral	17	90.358	73.358
Leon2	DPRAMRWRW_16X32	behav	21	89.211	68.211
Leon2	DPRAMRWRW_136X32	behav	21	89.211	68.211
Leon2	DPRAMRWRW_168X32	behav	21	89.211	68.211
Leon2	fs90_syncram_sim	behavioral	34	80.030	46.030
Leon2	fs90_dpram_ss	behav	40	75.972	35.972
Leon2	SA108019	behavioral	27	177.765	150.765
Leon2	SU004020	behavioral	29	203.858	174.858
Leon2	SW204420	behavioral	32	187.696	155.696
Leon2	generic_syncram	behavioral	41	81.826	40.826
Leon2	generic_dpram_ss	behav	48	83.155	35.155
Leon2	generic_dpram_as	behav	45	83.155	38.155
Leon2	syncram	behav	56	166.381	110.381
Leon2	dpsyncram	behav	53	108.864	55.864
Leon2	tsmc25_syncram_ss	behavioral	57	81.826	24.826
Leon2	tsmc25_dpram_ss	behav	91	80.213	-10.787
Leon2	ram4096x32	behavioral	23	88.562	65.562
Leon2	ram1024x32	behavioral	23	88.562	65.562
Leon2	ram2400x32	behavioral	28	88.562	60.562
Leon2	ram2048x32	behavioral	23	88.562	65.562
Leon2	ram256x24	behavioral	28	88.562	60.562
Leon2	ram256x27	behavioral	28	88.562	60.562
Leon2	ram512x23	behavioral	28	88.562	60.562
Leon2	dpram16x32	behavioral	38	85.619	47.619
Leon2	dpram136x32	behavioral	38	85.619	47.619
Leon2	dpram168x32	behavioral	38	85.619	47.619
Leon2	dpram256x26	behavioral	38	85.619	47.619
Leon2	dpram256x28	behavioral	38	85.619	47.619
Leon2	dpram256x30	behavioral	38	85.619	47.619
Leon2	dpram256x32	behavioral	38	85.619	47.619
Leon2	dpram512x28	behavioral	38	85.619	47.619
Leon2	dpram512x30	behavioral	38	85.619	47.619
Leon2	TIEHI	behavioral	9	88.146	79.146
Leon2	TIELO	behavioral	9	88.146	79.146
Leon2	umc18_dpram_ss	behav	58	79.564	21.564
Leon2	umc18_syncram_ss	behavioral	50	80.03	30.03
Leon2	R256X24M4	behavioral	18	86.766	68.766
Leon2	R256X25M4	behavioral	18	86.766	68.766
Leon2	R256X26M4	behavioral	18	86.766	68.766
Leon2	R1024X32M4	behavioral	18	86.766	68.766
Leon2	R2048X32M8	behavioral	18	86.766	68.766
Leon2	R256X28M4	behavioral	18	86.766	68.766
Leon2	RF136X32M1	behav	17	84.97	67.97
Leon2	RF168X32M1	behav	17	84.97	67.97
Leon2	RAMB4_S16	behav	14	86.766	72.766
Leon2	RAMB4_S8	behav	14	86.766	72.766
Leon2	RAMB4_S4	behav	14	86.766	72.766
Leon2	RAMB4_S2	behav	14	86.766	72.766
Leon2	RAMB4_S1	behav	14	86.766	72.766
Leon2	RAMB4_S1_S1	behav	42	72.633	30.633
Leon2	RAMB4_S2_S2	behav	42	72.633	30.633
Leon2	RAMB4_S8_S8	behav	42	72.633	30.633
Leon2	RAMB4_S4_S4	behav	42	72.633	30.633
Leon2	RAMB4_S16_S16	behav	42	72.633	30.633
Leon2	virtex_syncram	behav	62	138.196	76.196
Leon2	virtex_regfile	behav	54	99.521	45.521
Leon2	virtex_regfile_cp	behav	36	99.988	63.988
Leon2	virtex_dpram	behav	74	127.654	53.654

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$

Real sizes versus estimated sizes for structural architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
fw09	core_6809e	structure	808	354.905	-453.095
IEEE1149	dr	structural	33	35.328	2.328
IEEE1149	ir	structural	40	36.976	-3.024
IEEE1149	testable_nibble_comparator	structural	63	112.527	49.527
LFSR	prpt	structural	223	231.374	8.374
LFSR	prpt	struct_generate	74	79.758	5.758
LFSR	prpt_stage	structural	54	65.897	11.897
LFSR	prpt_stage1	structural	47	66.218	19.218
Manticore	calc_test	structural	457	218.330	-238.670
Manticore	frame_buffer_test	structural	641	208.653	-432.347
Manticore	manticore_fifo	struct	68	43.919	-24.081
Manticore	pll2x	SYN	56	28.544	-27.456
Manticore	shiftreg_8x8	structural	54	33.296	-20.704
Manticore	slope_calc	structural	344	146.564	-197.436
Manticore	vgafifo	SYN	65	35.192	-29.808
Manticore	write_fifo_address	SYN	62	34.494	-27.506
Manticore	write_fifo_data	SYN	66	34.494	-31.506
Manticore	write_fifo_mask	SYN	57	32.914	-24.086
Manticore	zfifo	SYN	65	35.192	-29.808
SPIM-Pipe	spim_pipe	structural	248	172.861	-75.139
SPIM	ss_spim	structural	118	104.211	-13.789
Leon2	ahbtest	struct	90	15.181	-74.819
Leon2	dcom	struct	130	27.698	-102.302
Leon2	dma	struct	115	24.949	-90.051
Leon2	pci_is	struct	44	61.663	17.663
Leon2	atc25_inpad	syn	4	13.425	9.425
Leon2	atc25_smpad	syn	4	13.425	9.425
Leon2	atc25_outpad	syn	15	18.05	3.05
Leon2	atc25_toutpadu	syn	16	18.748	2.748
Leon2	atc25_iopad	syn	16	26.878	10.878
Leon2	atc25_iopadu	syn	16	26.878	10.878
Leon2	atc25_iodpad	syn	18	26.181	8.181
Leon2	atc25_odpad	syn	18	18.050	0.050
Leon2	atc25_pcioutpad	syn	4	13.425	9.425
Leon2	atc25_pcitoutpad	syn	6	14.122	8.122
Leon2	atc25_pciiopad	syn	7	22.253	15.253
Leon2	atc25_pciiodpad	syn	9	21.555	12.555
Leon2	atc35_inpad	syn	4	13.425	9.425
Leon2	atc35_smpad	syn	4	13.425	9.425
Leon2	atc35_outpad	syn	15	18.050	3.050
Leon2	atc35_toutpadu	syn	16	18.748	2.748
Leon2	atc35_iopad	syn	16	26.878	10.878
Leon2	atc35_iodpad	syn	18	26.181	8.181
Leon2	atc35_odpad	syn	18	18.050	0.050
Leon2	fs90_inpad	syn	9	13.425	4.425
Leon2	fs90_smpad	syn	7	13.425	6.425
Leon2	fs90_outpad	syn	21	14.754	-6.246
Leon2	fs90_toutpadu	syn	21	15.452	-5.548
Leon2	fs90_iopad	syn	21	23.582	2.582
Leon2	fs90_smiopad	syn	21	23.582	2.582
Leon2	fs90_iodpad	syn	21	22.885	1.885
Leon2	fs90_odpad	syn	21	14.754	-6.246
Leon2	tsmc25_inpad	syn	4	13.425	9.425
Leon2	tsmc25_smpad	syn	4	13.425	9.425
Leon2	tsmc25_outpad	syn	30	24.642	-5.358
Leon2	tsmc25_toutpadu	syn	27	23.692	-3.308
Leon2	tsmc25_iopad	syn	25	31.822	6.822

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	tsmc25_iodpad	syn	30	32.773	2.773
Leon2	tsmc25_odpad	syn	30	24.642	-5.358
Leon2	tsmc25_smiopad	syn	26	31.822	5.822
Leon2	umc18_inpad	syn	4	13.425	9.425
Leon2	umc18_smpad	syn	4	13.425	9.425
Leon2	umc18_outpad	syn	15	18.050	3.050
Leon2	umc18_toutpadu	syn	18	18.748	0.748
Leon2	umc18_iopad	syn	18	26.878	8.878
Leon2	umc18_iodpad	syn	18	26.181	8.181
Leon2	umc18_odpad	syn	16	18.050	2.050
Leon2	umc18_smiopad	syn	10	23.582	13.582

Real sizes versus estimated sizes for data-flow architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
RTC	ALU	ALU_rtl	315	211.281	-103.719
RTC	BCD2BIN	BCD2BIN_rtl	18	64.670	46.670
RTC	BIN2BCD	BIN2BCD_rtl	18	64.670	46.670
RTC	cal_modul	cal_modul_rtl	241	261.343	20.343
RTC	CLOCK_DIV	CLOCK_DIV_rtl	26	64.744	38.744
RTC	CMDINTERF	CMDINTERF_rtl	49	119.959	70.959
RTC	JAHR_COUNTER	JAHR_COUNTER_rtl	36	94.054	58.054
RTC	monat_counter	monat_counter_rtl	38	92.799	54.799
RTC	MUX2TO1	MUX2TO1_rtl	11	86.873	75.873
RTC	PC	PC_rtl	24	136.699	112.699
RTC	secmin_counter	secmin_counter_rtl	38	92.799	54.799
RTC	tag_counter	tag_counter_rtl	58	105.848	47.848
RTC	uhr_counter	uhr_counter_rtl	38	92.799	54.799
RTC	weekday_counter	weekday_counter_rtl	38	92.799	54.799
RTC	weeknr_counter	weeknr_counter_rtl	38	92.799	54.799
RTC-alt	ALU	ALU_rtl	313	233.709	-79.291
RTC-alt	BCD2BIN	BCD2BIN_rtl	18	64.670	46.670
RTC-alt	BIN2BCD	BIN2BCD_rtl	18	64.670	46.670
RTC-alt	cal_modul	cal_modul_rtl	269	436.668	167.668
RTC-alt	CLOCK_DIV	CLOCK_DIV_rtl	23	58.368	35.368
RTC-alt	CMDINTERF	CMDINTERF_rtl	49	119.959	70.959
RTC-alt	JAHR_COUNTER	JAHR_COUNTER_rtl	36	94.054	58.054
RTC-alt	monat_counter	monat_counter_rtl	38	92.799	54.799
RTC-alt	MUX2TO1	MUX2TO1_rtl	11	86.873	75.873
RTC-alt	PC	PC_rtl	24	136.699	112.699
RTC-alt	secmin_counter	secmin_counter_rtl	38	92.799	54.799
RTC-alt	tag_counter	tag_counter_rtl	58	105.848	47.848
RTC-alt	uhr_counter	uhr_counter_rtl	38	92.799	54.799
RTC-alt	weekday_counter	weekday_counter_rtl	38	92.799	54.799
RTC-alt	weeknr_counter	weeknr_counter_rtl	38	92.799	54.799
Leon2	acache	rtl	201	167.978	-33.022
Leon2	ahbarb	rtl	175	108.268	-66.732
Leon2	ahbmst	rtl	94	103.833	9.833
Leon2	ahbstat	rtl	68	98.163	30.163
Leon2	apbmst	rtl	78	97.357	19.357
Leon2	cache	rtl	81	273.626	192.626
Leon2	cachemem	rtl	92	95.63	3.63
Leon2	dcache	rtl	491	156.177	-334.823
Leon2	dcom_uart	rtl	232	94.549	-137.451
Leon2	div	rtl	118	79.57	-38.43
Leon2	dsu	rtl	432	179.041	-252.959
Leon2	dsu_mem	rtl	64	76.971	12.971
Leon2	fpaux	rtl	42	74.151	32.151
Leon2	fp	rtl	707	145.523	-561.477
Leon2	fp1eu	rtl	566	111.831	-454.169
Leon2	fpu_core	rtl	54	81.98	27.98
Leon2	fpu_lth	rtl	722	186.219	-535.781

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	icache	rtl	230	149.478	-80.522
Leon2	ioport	rtl	107	115.762	8.762
Leon2	irqctrl	rtl	84	88.211	4.211
Leon2	irqctrl2	rtl	111	87.615	-23.385
Leon2	iu	rtl	2401	431.484	-1969.516
Leon2	lconf	rtl	38	117.171	79.171
Leon2	leon	rtl	131	450.588	319.588
Leon2	leon_pci	rtl	258	736.359	478.359
Leon2	mcore	rtl	172	321.274	149.274
Leon2	mctrl	rtl	675	183.423	-491.577
Leon2	fpu	rtl	32	186.219	154.219
Leon2	mul	rtl	265	98.298	-166.702
Leon2	GEN_NOT	rtl	2	57.039	55.039
Leon2	GEN_AND2	rtl	2	63.415	61.415
Leon2	GEN_OR2	rtl	2	63.415	61.415
Leon2	GEN_XOR2	rtl	2	63.415	61.415
Leon2	pci	rtl	28	219.566	191.566
Leon2	pci_arb	rtl	152	106.742	-45.258
Leon2	proc	rtl	138	410.325	272.325
Leon2	rstgen	rtl	29	86.423	57.423
Leon2	sdmctrl	rtl	342	117.457	-224.543
Leon2	pt33d00	rtl	2	57.039	55.039
Leon2	pt33d00u	rtl	3	57.624	54.624
Leon2	pt33d20	rtl	2	57.039	55.039
Leon2	pt33d20u	rtl	3	57.624	54.624
Leon2	pt33o01	rtl	2	57.039	55.039
Leon2	pt33o02	rtl	2	57.039	55.039
Leon2	pt33o03	rtl	2	57.039	55.039
Leon2	pt33o04	rtl	2	57.039	55.039
Leon2	pt33t01u	rtl	3	63.415	60.415
Leon2	pt33t02u	rtl	3	63.415	60.415
Leon2	pt33t03u	rtl	3	63.415	60.415
Leon2	pt33b01	rtl	8	70.376	62.376
Leon2	pt33b02	rtl	8	70.376	62.376
Leon2	pt33b03	rtl	8	70.376	62.376
Leon2	pt33b04	rtl	8	70.376	62.376
Leon2	pt33b01u	rtl	8	70.376	62.376
Leon2	pt33b02u	rtl	8	70.376	62.376
Leon2	pt33b03u	rtl	8	70.376	62.376
Leon2	pt33b04u	rtl	8	70.376	62.376
Leon2	pp33o01	rtl	2	57.039	55.039
Leon2	pp33b015vt	rtl	8	70.376	62.376
Leon2	pp33t015vt	rtl	3	63.415	60.415
Leon2	atc25_syncram	rtl	61	121.937	60.937
Leon2	atc25_dpram	rtl	94	153.105	59.105
Leon2	atc25_regfile_iu	rtl	63	102.597	39.597
Leon2	atc25_regfile_cp	rtl	51	85.708	34.708
Leon2	pc3d01	rtl	2	57.039	55.039
Leon2	pc3d21	rtl	2	57.039	55.039
Leon2	pt3o01	rtl	2	57.039	55.039
Leon2	pt3o02	rtl	2	57.039	55.039
Leon2	pt3o03	rtl	2	57.039	55.039
Leon2	pc3t01u	rtl	3	63.415	60.415
Leon2	pc3t02u	rtl	3	63.415	60.415
Leon2	pc3t03u	rtl	3	63.415	60.415
Leon2	pt3b01	rtl	8	70.376	62.376
Leon2	pt3b02	rtl	8	70.376	62.376
Leon2	pt3b03	rtl	8	70.376	62.376
Leon2	atc35_syncram	rtl	36	98.977	62.977
Leon2	atc35_regfile	rtl	58	110.452	52.452
Leon2	atc35_regfile_cp	rtl	41	85.708	44.708
Leon2	RAM64K36	rtl	51	933.797	882.797

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	axcel_regfile_iu	rtl	183	111.545	-71.455
Leon2	axcel_regfile_cp	rtl	109	99.248	-9.752
Leon2	axcel_syncram	rtl	78	103.333	25.333
Leon2	uyfaa	rtl	14	76.166	62.166
Leon2	vyfa2gsa	rtl	12	88.918	76.918
Leon2	wyfa2gsa	rtl	18	115.006	97.006
Leon2	fs90_syncram	rtl	50	94.385	44.385
Leon2	fs90_regfile	rtl	55	92.084	37.084
Leon2	rfbypass	rtl	44	114.585	70.585
Leon2	generic_regfile_iu	rtl	85	143.217	58.217
Leon2	generic_regfile_cp	rtl	36	99.248	63.248
Leon2	generic_smult	rtl	21	66.074	45.074
Leon2	generic_clkgen	rtl	27	106.937	79.937
Leon2	geninpad	rtl	2	57.039	55.039
Leon2	gensmpad	rtl	2	57.039	55.039
Leon2	genoutpad	rtl	2	57.039	55.039
Leon2	gentoutpadu	rtl	3	63.415	60.415
Leon2	geniopad	rtl	5	70.376	65.376
Leon2	geniodpad	rtl	5	64.000	59.000
Leon2	genodpad	rtl	2	57.039	55.039
Leon2	regfile_iu	rtl	59	125.557	66.557
Leon2	regfile_cp	rtl	49	108.668	59.668
Leon2	hw_smult	rtl	30	89.034	59.034
Leon2	inpad	rtl	26	84.591	58.591
Leon2	smpad	rtl	26	84.591	58.591
Leon2	outpad	rtl	29	85.920	56.920
Leon2	toutpadu	rtl	29	92.296	63.296
Leon2	iopad	rtl	34	99.257	65.257
Leon2	smiopad	rtl	34	99.257	65.257
Leon2	odpad	rtl	29	85.920	56.920
Leon2	iodpad	rtl	29	92.881	63.881
Leon2	pcioutpad	rtl	12	66.223	54.223
Leon2	pcitoutpad	rtl	12	72.599	60.599
Leon2	pciiopad	rtl	19	79.560	60.560
Leon2	pciiodpad	rtl	14	73.184	59.184
Leon2	clkgen	rtl	17	111.529	94.529
Leon2	RAM256x9SST	rtl	25	378.482	353.482
Leon2	RAM256x9SA	rtl	25	372.106	347.106
Leon2	proasic_regfile_iu	rtl	69	106.953	37.953
Leon2	proasic_regfile_cp	rtl	63	99.248	36.248
Leon2	proasic_syncram	rtl	103	103.333	0.333
Leon2	PDIDGZ	rtl	2	57.039	55.039
Leon2	PDISDGZ	rtl	2	57.039	55.039
Leon2	PDT02DGZ	rtl	4	63.415	59.415
Leon2	PDT04DGZ	rtl	4	63.415	59.415
Leon2	PDT08DGZ	rtl	4	63.415	59.415
Leon2	PDT12DGZ	rtl	4	63.415	59.415
Leon2	PDT16DGZ	rtl	4	63.415	59.415
Leon2	PDT24DGZ	rtl	4	63.415	59.415
Leon2	PDU02DGZ	rtl	5	70.376	65.376
Leon2	PDU04DGZ	rtl	5	70.376	65.376
Leon2	PDU08DGZ	rtl	5	70.376	65.376
Leon2	PDU12DGZ	rtl	5	70.376	65.376
Leon2	PDU16DGZ	rtl	5	70.376	65.376
Leon2	PDU24DGZ	rtl	5	70.376	65.376
Leon2	PDB02DGZ	rtl	5	70.376	65.376
Leon2	PDB04DGZ	rtl	5	70.376	65.376
Leon2	PDB08DGZ	rtl	5	70.376	65.376
Leon2	PDB12DGZ	rtl	5	70.376	65.376
Leon2	PDB16DGZ	rtl	5	70.376	65.376
Leon2	PDB24DGZ	rtl	5	70.376	65.376
Leon2	PDB02SDGZ	rtl	5	70.376	65.376

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	PDB04SDGZ	rtl	5	70.376	65.376
Leon2	PDB08SDGZ	rtl	5	70.376	65.376
Leon2	PDB12SDGZ	rtl	5	70.376	65.376
Leon2	PDB16SDGZ	rtl	5	70.376	65.376
Leon2	PDB24SDGZ	rtl	5	70.376	65.376
Leon2	tsmc25_syncram	rtl	95	112.753	17.753
Leon2	tsmc25_dpram	rtl	102	148.513	46.513
Leon2	tsmc25_regfile_iu	rtl	102	119.636	17.636
Leon2	tsmc25_regfile_cp	rtl	77	104.076	27.076
Leon2	C3I40	rtl	2	57.039	55.039
Leon2	C3I42	rtl	2	57.039	55.039
Leon2	C3O10	rtl	2	57.039	55.039
Leon2	C3O20	rtl	2	57.039	55.039
Leon2	C3O40	rtl	2	57.039	55.039
Leon2	C3B10U	rtl	8	70.376	62.376
Leon2	C3B20U	rtl	8	70.376	62.376
Leon2	C3B40U	rtl	8	70.376	62.376
Leon2	C3B10	rtl	8	70.376	62.376
Leon2	C3B20	rtl	8	70.376	62.376
Leon2	C3B40	rtl	8	70.376	62.376
Leon2	CD3B10T	rtl	8	70.376	62.376
Leon2	CD3B20T	rtl	8	70.376	62.376
Leon2	CD3B40T	rtl	8	70.376	62.376
Leon2	CD3O10T	rtl	7	57.039	50.039
Leon2	CD3O20T	rtl	7	57.039	50.039
Leon2	CD3O40T	rtl	7	57.039	50.039
Leon2	C3B42	rtl	8	70.376	62.376
Leon2	INVDL	rtl	2	57.039	55.039
Leon2	AND2DL	rtl	2	63.415	61.415
Leon2	OR2DL	rtl	2	63.415	61.415
Leon2	EXOR2DL	rtl	2	63.415	61.415
Leon2	umc18_syncram	rtl	56	103.569	47.569
Leon2	umc18_regfile	rtl	43	101.268	58.268
Leon2	timers	rtl	164	91.317	-72.683
Leon2	uart	rtl	276	92.996	-183.004
Leon2	wprot	rtl	94	90.595	-3.405

14.15 K4E

14.15.1 Result summary

Population statistical properties and model accuracy:

	All architectures		
	L	$\hat{L}$	$\hat{L} - L$
Average value	152.359	126.539	-25.820
Variance	632469.174	82791.094	288247.088
Standard deviation	795.279	287.734	536.886
	Behavioral architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	274.560	178.519	-96.041
Variance	1505949.100	189567.584	668898.227
Standard deviation	1227.171	435.394	817.862
	Structural architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	68.632	42.050	-26.582
Variance	19345.818	1515.307	16921.783
Standard deviation	139.089	38.927	130.084
	Data-flow architectures only		
	L	$\hat{L}$	$\hat{L} - L$
Average value	71.242	108.090	36.849
Variance	39803.835	10062.727	32789.889
Standard deviation	199.509	100.313	181.080

Correlation between estimated and real values:

	Correlation coefficient $(L, \hat{L})$
All architectures	0.9330
Behavioral architectures only	0.9607
Structural architectures only	0.3638
Data-flow architectures only	0.4266

Figure 14.67: Model K4E: Real vs. estimated lines of code.

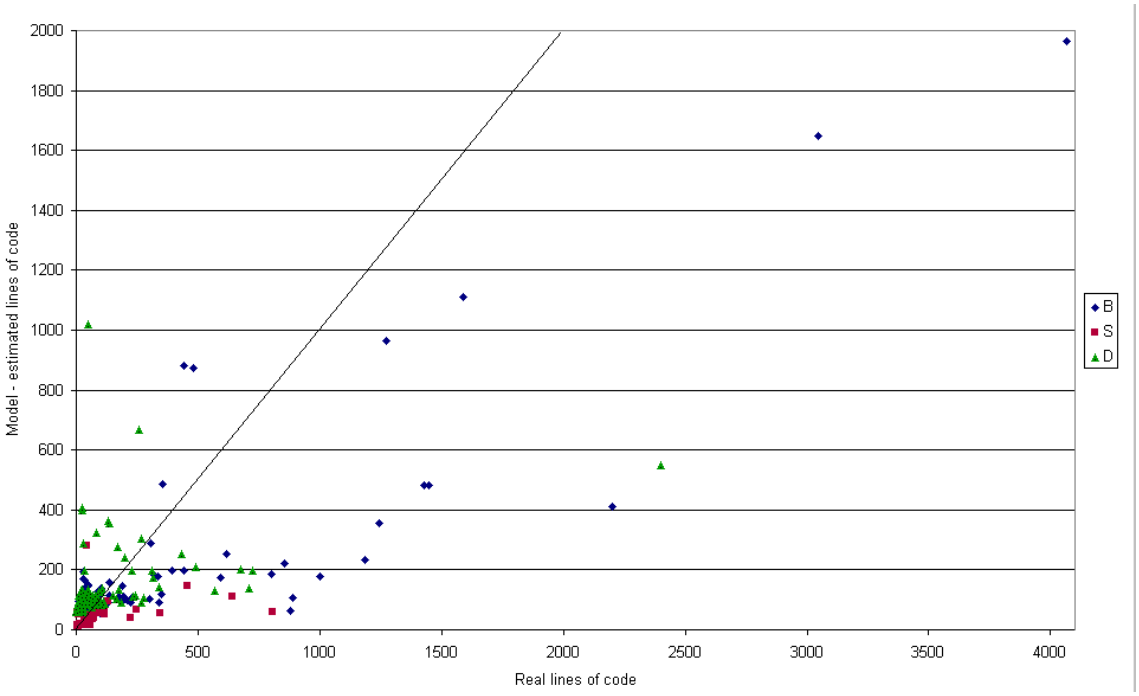


Figure 14.68: Models K4Eb, K4Es, K4Ed: Real vs. estimated lines of code.

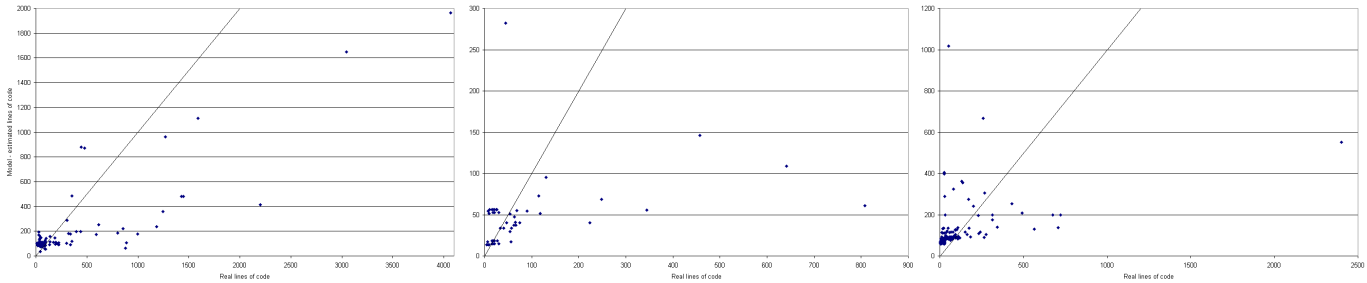


Figure 14.69: Model K4E: Error density distribution.

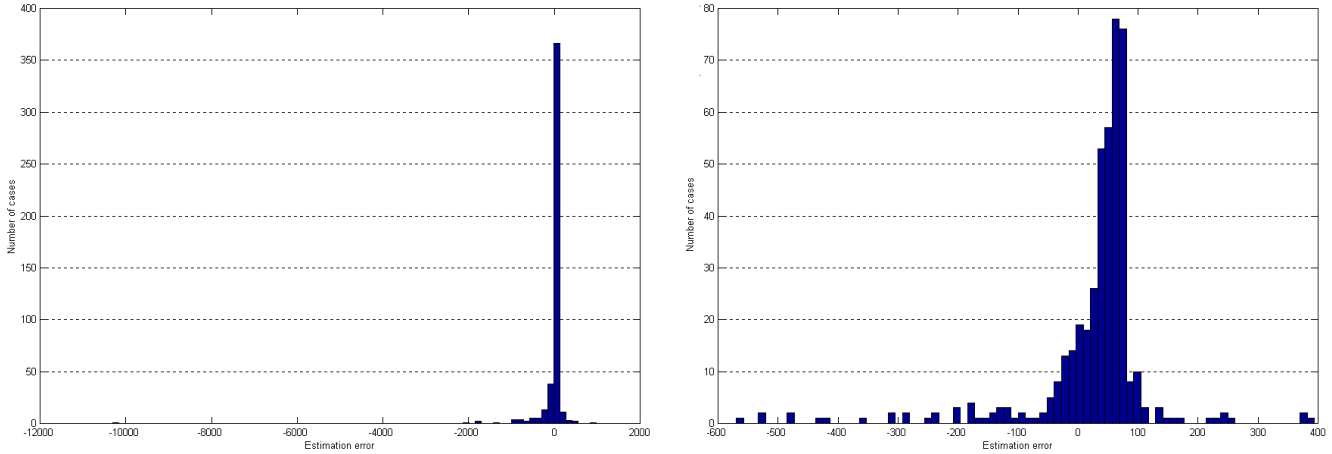


Figure 14.70: Model K4E: Error cumulative distribution.

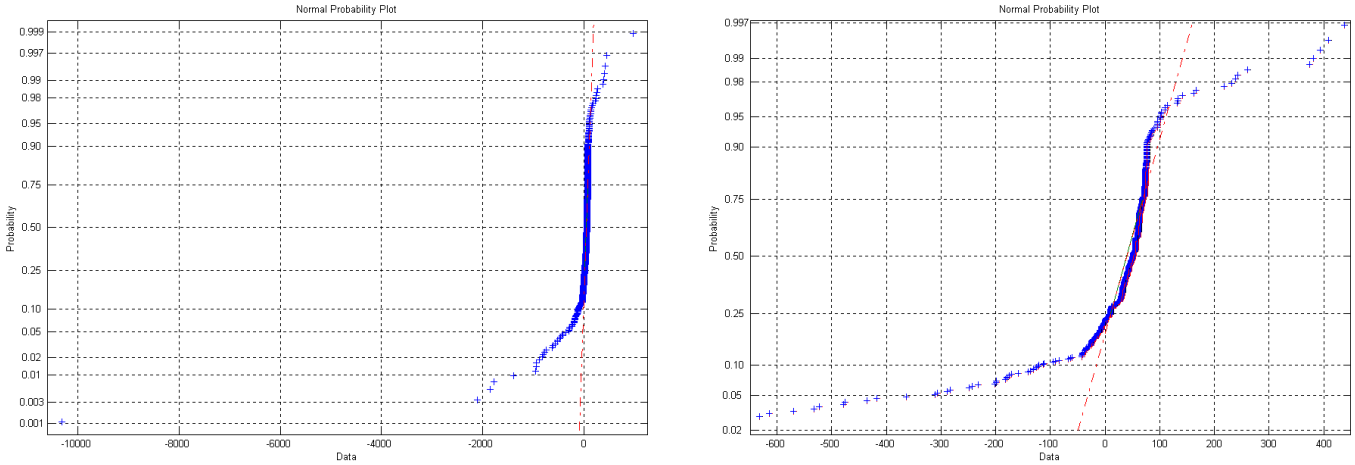
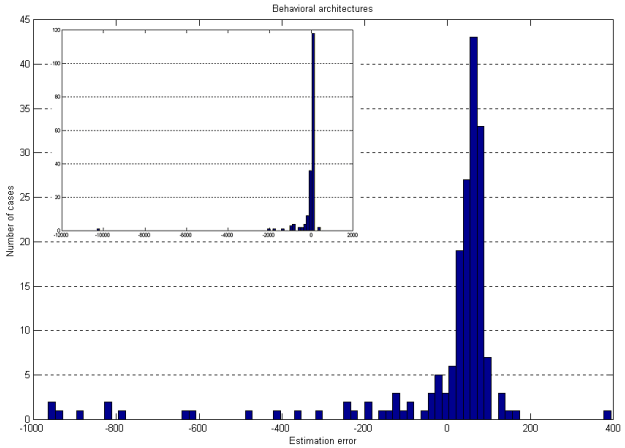
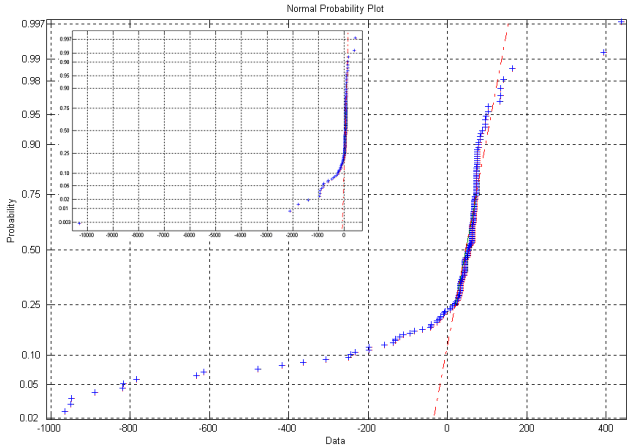
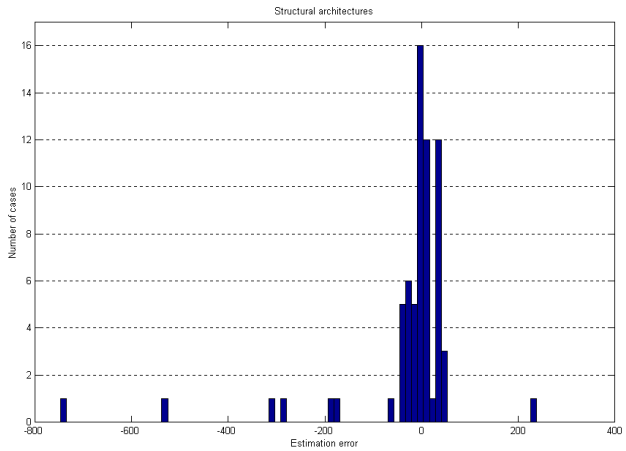
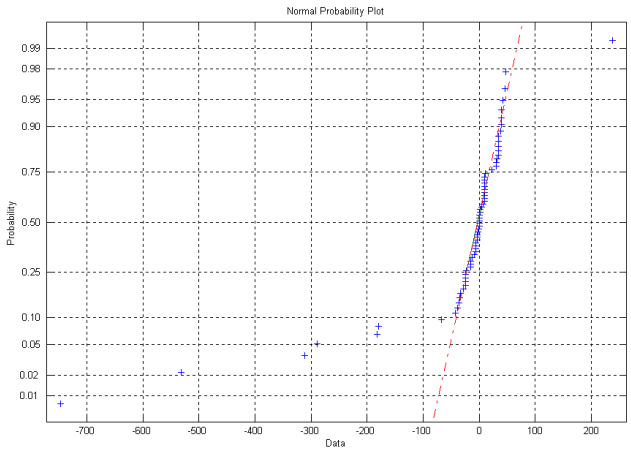


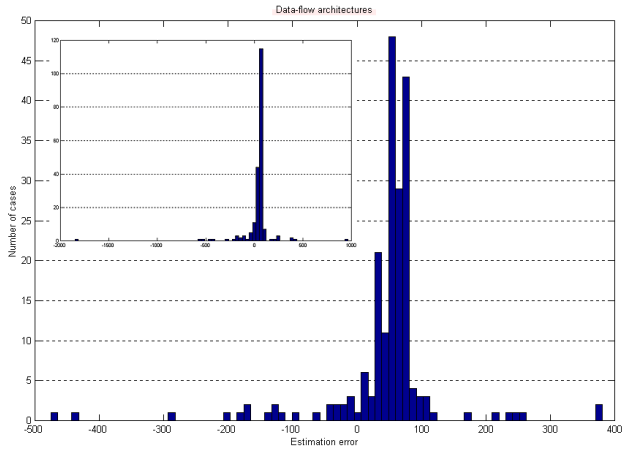
Figure 14.71: Models K4Eb, K4Es, K4Ed: Error density and cumulative distributions.

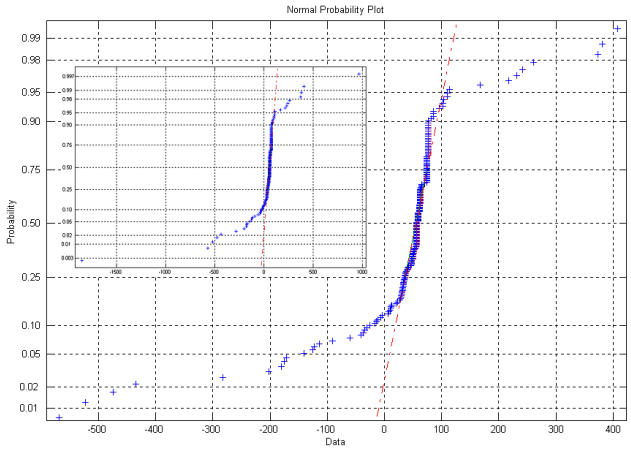












14.15.2 Detailed Results

Real sizes versus estimated sizes for behavioral architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
AMCC5933	amcc5933	behavior	593	175.444	-417.556
DLX2	Gen_Immediate	behavioral	33	95.325	62.325
DLX2	stats	behavioral	24	98.179	74.179
DLX2	dlx_control	behavioral	337	178.378	-158.622
DLX2	dlx_data_path	behavioral	173	109.637	-63.363
DLX2	Data_Reg	behavioral	25	93.587	68.587
DLX2	Bypass_Logic	behavioral	43	33.713	-9.287
DLX2	Decode_Ird	behavioral	102	140.159	38.159
DLX2	Decode_PC	behavioral	93	99.526	6.526
DLX2	dlx_stats	behavioral	24	97.258	73.258
DLX2	Dmemory	behavioral	89	125.790	36.790
DLX2	Imemory	behavioral	59	109.649	50.649
DLX2	Ireg_Fetch	behavioral	32	96.654	64.654
DLX2	Ireg_Decode	behavioral	37	116.442	79.442
DLX2	Ireg_Execute	behavioral	41	105.574	64.574
DLX2	Ireg_Memory	behavioral	34	100.412	66.412
DLX2	Ireg_Writeback	behavioral	28	96.229	68.229
DLX2	Register_File	behavioral	41	103.728	62.728
DSP320VC33	dsp320vc33	vhdl_behavioral	3045	1647.618	-1397.382
DSP320VC33	sram16k32	vhdl_behavioral	479	872.946	393.946
DSP320VC33	sram1k32	vhdl_behavioral	442	879.897	437.897
DSP6211	dsp6211	vhdl_behavioral	4068	1962.724	-2105.276
DSP6211	km416s4030	vhdl_behavioral	1446	481.786	-964.214
DSP6211	MT58L32L32P	behave	224	92.308	-131.692
DSP6211	sram1k16	vhdl_behavioral	353	486.383	133.383
DSP6211	sram1k8	vhdl_behavioral	307	289.626	-17.374
DSP6415	at9366	vhdl_behavioral	352	119.254	-232.746
DSP6415	cy7c453	vhdl_behavioral	801	186.846	-614.154
DSP6415	dsp6415	vhdl_behavioral	15508	5197.618	-10310.382
DSP6415	idt71v546	vhdl_behavioral	1271	963.764	-307.236
DSP6415	km416s4030	vhdl_behavioral	1429	481.786	-947.214
DSP6415	MS32PCI	Behavior	615	251.230	-363.770
DSP6415	MT58L32L32P	behave	224	92.308	-131.692
DSP6415	sram1k16	vhdl_behavioral	353	486.383	133.383
DSP6415	sram1k8	vhdl_behavioral	307	289.626	-17.374
DSP6415	TG32PCI	Behavior	1185	235.020	-949.980
fw09	addressing_decode	behave	137	115.997	-21.003
fw09	alu	behave	191	96.459	-94.541
fw09	busstatus	behave	39	108.728	69.728
fw09	instruction_decode	behave	890	105.661	-784.339
fw09	mainstate	behave	21	95.325	74.325
fw09	mux16alu_left	behave	67	73.854	6.854
fw09	mux16alu_right	behave	63	83.056	20.056
fw09	muxaddressbus	behave	32	89.190	57.190
fw09	muxdatabus	behave	54	73.854	19.854
fw09	pb_decode	behave	95	115.997	20.997
fw09	reg8	behave	24	92.258	68.258
fw09	registercc	behave	52	79.988	27.988
fw09	registerd	behave	85	58.517	-26.483
fw09	registerdp	behave	42	86.123	44.123
fw09	registerindexstack	behave	97	55.450	-41.550
fw09	registerpc	behave	85	58.517	-26.483
fw09	registertemp	behave	53	83.056	30.056
fw09	regpage	behave	30	92.258	62.258
fw09	statedecode	behave	2201	412.759	-1788.241
fw09	transfer_decode	behave	81	108.728	27.728
fw09	vectortable	behave	32	101.459	69.459
HDLLib	core_mac	behv	133	92.258	-40.742
HDLLib	ram128x8	behv	338	89.19	-248.81
IEEE1149	br_cell	behavioral	13	95.325	82.325

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
IEEE1149	dr_cell	behavioral	26	93.392	67.392
IEEE1149	ir_cell	behavioral	28	90.324	62.324
IEEE1149	mux_2_1	behavioral	12	95.325	83.325
IEEE1149	mux_4_1	behavioral	17	89.19	72.19
IEEE1149	tap_controller	behavioral	138	156.543	18.543
Manticore	demo_xform	behavioural	226	104.862	-121.138
Manticore	rasterizer	behavioural	880	63.054	-816.946
Manticore	raster_ctrl	behav	302	103.600	-198.400
Manticore	raster_vars_reg	behavioural	854	221.164	-632.836
Manticore	raster_var_type_reg	behavioural	17	95.325	78.325
Manticore	sdram_control	behav	395	196.571	-198.429
Manticore	sdram_control_param	behav	440	196.571	-243.429
Manticore	vgafifo_ctrl	behavioural	319	181.938	-137.062
Manticore	vgaout	behavior	188	145.408	-42.592
RLS	Fp_Divide	bhv	76	102.594	26.594
RLS	MainCtl	Bhv	194	109.863	-84.137
RLS	RLSTop	bhv	211	99.526	-111.474
RLS	rls_mult	bhv	92	83.056	-8.944
RLS	SubCtl	bhv	999	179.355	-819.645
SPIM-Pipe	control	behavioral	40	142.005	102.005
SPIM-Pipe	decode	behavioral	66	109.064	43.064
SPIM-Pipe	execute	behavioral	43	84.190	41.190
SPIM-Pipe	fetch	behavioral	48	105.661	57.661
SPIM-Pipe	memory	behavioral	36	86.123	50.123
SPIM-Pipe	pipe_reg1	behavioral	21	99.526	78.526
SPIM-Pipe	pipe_reg2	behavioral	50	145.744	95.744
SPIM-Pipe	pipe_reg3	behavioral	33	129.544	96.544
SPIM-Pipe	pipe_reg4	behavioral	29	112.131	83.131
SPIM	control	behavioral	47	149.274	102.274
SPIM	decode	behavioral	70	94.526	24.526
SPIM	execute	behavioral	49	100.661	51.661
SPIM	fetch	behavioral	52	96.459	44.459
SPIM	memory	behavioral	34	86.123	52.123
STD8980	std8980	vhdl_behavioral	1246	357.26	-888.74
ZR36060	zr36060	vhdl_behavioral	1590	1111.111	-478.889
DLX2	Main_Alu	behavioral	54	95.325	41.325
DLX2	Inverter	behavioral	13	101.459	88.459
DLX2	Data_Mux2	behavioral	20	95.325	75.325
DLX2	Data_Mux4	behavioral	28	89.19	61.19
Leon2	atc25_syncram_sim	behavioral	29	88.782	59.782
Leon2	atc25_2pram	behav	38	90.111	52.111
Leon2	atc25_dpram_sim	behav	53	85.111	32.111
Leon2	RAM_256x26	behavioral	14	86.123	72.123
Leon2	RAM_256x28	behavioral	14	86.123	72.123
Leon2	RAM_256x30	behavioral	14	86.123	72.123
Leon2	RAM_512x28	behavioral	14	86.123	72.123
Leon2	RAM_512x30	behavioral	14	86.123	72.123
Leon2	RAM_512x32	behavioral	14	86.123	72.123
Leon2	RAM_1024x32	behavioral	14	86.123	72.123
Leon2	RAM_2048x32	behavioral	14	86.123	72.123
Leon2	RAM2P_16X32	behav	13	86.123	73.123
Leon2	RAM2P_136X32	behav	13	86.123	73.123
Leon2	RAM2P_168X32	behav	13	86.123	73.123
Leon2	DPRAM_256x26	behav	20	81.123	61.123
Leon2	DPRAM_256x28	behav	20	81.123	61.123
Leon2	DPRAM_256x30	behav	20	81.123	61.123
Leon2	DPRAM_256x32	behav	20	81.123	61.123
Leon2	DPRAM_512x28	behav	20	81.123	61.123
Leon2	DPRAM_512x30	behav	20	81.123	61.123
Leon2	DPRAM_512x32	behav	20	81.123	61.123
Leon2	atc35_dpram_ss_dn	behav	38	90.111	52.111
Leon2	ATC35_RAM_256x26	behavioral	17	92.258	75.258

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	ATC35_RAM_1024x32	behavioral	17	92.258	75.258
Leon2	ATC35_RAM_2048x32	behavioral	17	92.258	75.258
Leon2	ATC35_RAM_256x28	behavioral	17	92.258	75.258
Leon2	ATC35_RAM_1024x34	behavioral	17	92.258	75.258
Leon2	ATC35_RAM_2048x34	behavioral	17	92.258	75.258
Leon2	DPRAMRWRW_16X32	behav	21	87.257	66.257
Leon2	DPRAMRWRW_136X32	behav	21	87.257	66.257
Leon2	DPRAMRWRW_168X32	behav	21	87.257	66.257
Leon2	fs90_syncram_sim	behavioral	34	88.782	54.782
Leon2	fs90_dpram_ss	behav	40	80.909	40.909
Leon2	SA108019	behavioral	27	168.557	141.557
Leon2	SU004020	behavioral	29	191.833	162.833
Leon2	SW204420	behavioral	32	164.227	132.227
Leon2	generic_syncram	behavioral	41	91.849	50.849
Leon2	generic_dpram_ss	behav	48	93.179	45.179
Leon2	generic_dpram_as	behav	45	93.179	48.179
Leon2	syncram	behav	56	91.849	35.849
Leon2	dpsyncram	behav	53	86.849	33.849
Leon2	tsmc25_syncram_ss	behavioral	57	91.849	34.849
Leon2	tsmc25_dpram_ss	behav	91	85.111	-5.889
Leon2	ram4096x32	behavioral	23	89.190	66.190
Leon2	ram1024x32	behavioral	23	89.190	66.190
Leon2	ram2400x32	behavioral	28	89.190	61.190
Leon2	ram2048x32	behavioral	23	89.190	66.190
Leon2	ram256x24	behavioral	28	89.190	61.190
Leon2	ram256x27	behavioral	28	89.190	61.190
Leon2	ram512x23	behavioral	28	89.190	61.190
Leon2	dpram16x32	behavioral	38	81.123	43.123
Leon2	dpram136x32	behavioral	38	81.123	43.123
Leon2	dpram168x32	behavioral	38	81.123	43.123
Leon2	dpram256x26	behavioral	38	81.123	43.123
Leon2	dpram256x28	behavioral	38	81.123	43.123
Leon2	dpram256x30	behavioral	38	81.123	43.123
Leon2	dpram256x32	behavioral	38	81.123	43.123
Leon2	dpram512x28	behavioral	38	81.123	43.123
Leon2	dpram512x30	behavioral	38	81.123	43.123
Leon2	TIEHI	behavioral	9	104.527	95.527
Leon2	TIELO	behavioral	9	104.527	95.527
Leon2	umc18_dpram_ss	behav	58	87.044	29.044
Leon2	umc18_syncram_ss	behavioral	50	88.782	38.782
Leon2	R256X24M4	behavioral	18	86.123	68.123
Leon2	R256X25M4	behavioral	18	86.123	68.123
Leon2	R256X26M4	behavioral	18	86.123	68.123
Leon2	R1024X32M4	behavioral	18	86.123	68.123
Leon2	R2048X32M8	behavioral	18	86.123	68.123
Leon2	R256X28M4	behavioral	18	86.123	68.123
Leon2	RF136X32M1	behav	17	83.056	66.056
Leon2	RF168X32M1	behav	17	83.056	66.056
Leon2	RAMB4_S16	behav	14	86.123	72.123
Leon2	RAMB4_S8	behav	14	86.123	72.123
Leon2	RAMB4_S4	behav	14	86.123	72.123
Leon2	RAMB4_S2	behav	14	86.123	72.123
Leon2	RAMB4_S1	behav	14	86.123	72.123
Leon2	RAMB4_S1_S1	behav	42	74.988	32.988
Leon2	RAMB4_S2_S2	behav	42	74.988	32.988
Leon2	RAMB4_S8_S8	behav	42	74.988	32.988
Leon2	RAMB4_S4_S4	behav	42	74.988	32.988
Leon2	RAMB4_S16_S16	behav	42	74.988	32.988
Leon2	virtex_syncram	behav	62	91.849	29.849
Leon2	virtex_regfile	behav	54	85.630	31.630
Leon2	virtex_regfile_cp	behav	36	87.368	51.368
Leon2	virtex_dpram	behav	74	86.849	12.849

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$

Real sizes versus estimated sizes for structural architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
fw09	core_6809e	structure	808	61.113	-746.887
IEEE1149	dr	structural	33	33.901	0.901
IEEE1149	ir	structural	40	33.901	-6.099
IEEE1149	testable_nibble_comparator	structural	63	47.436	-15.564
LFSR	prpt	structural	223	40.528	-182.472
LFSR	prpt	struct_generate	74	40.528	-33.472
LFSR	prpt_stage	structural	54	50.643	-3.357
LFSR	prpt_stage1	structural	47	40.528	-6.472
Manticore	calc_test	structural	457	146.236	-310.764
Manticore	frame_buffer_test	structural	641	109.127	-531.873
Manticore	manticore_fifo	struct	68	54.985	-13.015
Manticore	pll2x	SYN	56	17.159	-38.841
Manticore	shiftreg_8x8	structural	54	29.863	-24.137
Manticore	slope_calc	structural	344	55.463	-288.537
Manticore	vgafifo	SYN	65	40.740	-24.260
Manticore	write_fifo_address	SYN	62	37.391	-24.609
Manticore	write_fifo_data	SYN	66	37.391	-28.609
Manticore	write_fifo_mask	SYN	57	33.972	-23.028
Manticore	zfifo	SYN	65	40.740	-24.260
SPIM-Pipe	spim_pipe	structural	248	68.375	-179.625
SPIM	ss_spim	structural	118	51.279	-66.721
Leon2	ahbtest	struct	90	54.472	-35.528
Leon2	dcom	struct	130	95.125	-34.875
Leon2	dma	struct	115	72.817	-42.183
Leon2	pci_is	struct	44	282.024	238.024
Leon2	atc25_inpad	syn	4	13.740	9.740
Leon2	atc25_smpad	syn	4	13.740	9.740
Leon2	atc25_outpad	syn	15	15.070	0.070
Leon2	atc25_toutpadu	syn	16	18.418	2.418
Leon2	atc25_iopad	syn	16	55.963	39.963
Leon2	atc25_iopadu	syn	16	55.963	39.963
Leon2	atc25_iodpad	syn	18	52.615	34.615
Leon2	atc25_odpad	syn	18	15.070	-2.930
Leon2	atc25_pcioutpad	syn	4	13.740	9.740
Leon2	atc25_pcitoutpad	syn	6	17.089	11.089
Leon2	atc25_pciiopad	syn	7	54.634	47.634
Leon2	atc25_pciiodpad	syn	9	51.285	42.285
Leon2	atc35_inpad	syn	4	13.740	9.740
Leon2	atc35_smpad	syn	4	13.740	9.740
Leon2	atc35_outpad	syn	15	15.070	0.070
Leon2	atc35_toutpadu	syn	16	18.418	2.418
Leon2	atc35_iopad	syn	16	55.963	39.963
Leon2	atc35_iodpad	syn	18	52.615	34.615
Leon2	atc35_odpad	syn	18	15.070	-2.930
Leon2	fs90_inpad	syn	9	13.740	4.740
Leon2	fs90_smpad	syn	7	13.740	6.740
Leon2	fs90_outpad	syn	21	15.070	-5.930
Leon2	fs90_toutpadu	syn	21	18.418	-2.582
Leon2	fs90_iopad	syn	21	55.963	34.963
Leon2	fs90_smiopad	syn	21	55.963	34.963
Leon2	fs90_iodpad	syn	21	52.615	31.615
Leon2	fs90_odpad	syn	21	15.070	-5.930
Leon2	tsmc25_inpad	syn	4	13.740	9.740
Leon2	tsmc25_smpad	syn	4	13.740	9.740
Leon2	tsmc25_outpad	syn	30	15.07	-14.93
Leon2	tsmc25_toutpadu	syn	27	18.418	-8.582
Leon2	tsmc25_iopad	syn	25	55.963	30.963

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	tsmc25_iodpad	syn	30	52.615	22.615
Leon2	tsmc25_odpad	syn	30	15.070	-14.930
Leon2	tsmc25_smiopad	syn	26	55.963	29.963
Leon2	umc18_inpad	syn	4	13.740	9.740
Leon2	umc18_smpad	syn	4	13.740	9.740
Leon2	umc18_outpad	syn	15	15.070	0.070
Leon2	umc18_toutpadu	syn	18	18.418	0.418
Leon2	umc18_iopad	syn	18	55.963	37.963
Leon2	umc18_iodpad	syn	18	52.615	34.615
Leon2	umc18_odpad	syn	16	15.070	-0.930
Leon2	umc18_smiopad	syn	10	55.963	45.963

Real sizes versus estimated sizes for data-flow architectures:

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
RTC	ALU	ALU_rtl	315	173.965	-141.035
RTC	BCD2BIN	BCD2BIN_rtl	18	67.850	49.850
RTC	BIN2BCD	BIN2BCD_rtl	18	67.850	49.850
RTC	cal_modul	cal_modul_rtl	241	115.215	-125.785
RTC	CLOCK_DIV	CLOCK_DIV_rtl	26	66.439	40.439
RTC	CMDINTERF	CMDINTERF_rtl	49	134.761	85.761
RTC	JAHR_COUNTER	JAHR_COUNTER_rtl	36	100.098	64.098
RTC	monat_counter	monat_counter_rtl	38	97.357	59.357
RTC	MUX2TO1	MUX2TO1_rtl	11	113.177	102.177
RTC	PC	PC_rtl	24	134.392	110.392
RTC	secmin_counter	secmin_counter_rtl	38	97.357	59.357
RTC	tag_counter	tag_counter_rtl	58	112.474	54.474
RTC	uhr_counter	uhr_counter_rtl	38	97.357	59.357
RTC	weekday_counter	weekday_counter_rtl	38	97.357	59.357
RTC	weeknr_counter	weeknr_counter_rtl	38	97.357	59.357
RTC-alt	ALU	ALU_rtl	313	199.578	-113.422
RTC-alt	BCD2BIN	BCD2BIN_rtl	18	67.850	49.850
RTC-alt	BIN2BCD	BIN2BCD_rtl	18	67.850	49.850
RTC-alt	cal_modul	cal_modul_rtl	269	304.871	35.871
RTC-alt	CLOCK_DIV	CLOCK_DIV_rtl	23	59.374	36.374
RTC-alt	CMDINTERF	CMDINTERF_rtl	49	134.761	85.761
RTC-alt	JAHR_COUNTER	JAHR_COUNTER_rtl	36	100.098	64.098
RTC-alt	monat_counter	monat_counter_rtl	38	97.357	59.357
RTC-alt	MUX2TO1	MUX2TO1_rtl	11	113.177	102.177
RTC-alt	PC	PC_rtl	24	134.392	110.392
RTC-alt	secmin_counter	secmin_counter_rtl	38	97.357	59.357
RTC-alt	tag_counter	tag_counter_rtl	58	112.474	54.474
RTC-alt	uhr_counter	uhr_counter_rtl	38	97.357	59.357
RTC-alt	weekday_counter	weekday_counter_rtl	38	97.357	59.357
RTC-alt	weeknr_counter	weeknr_counter_rtl	38	97.357	59.357
Leon2	acache	rtl	201	241.266	40.266
Leon2	ahbarb	rtl	175	133.836	-41.164
Leon2	ahbmst	rtl	94	125.051	31.051
Leon2	ahbstat	rtl	68	115.126	47.126
Leon2	apbmst	rtl	78	115.177	37.177
Leon2	cache	rtl	81	323.283	242.283
Leon2	cachemem	rtl	92	88.683	-3.317
Leon2	dcache	rtl	491	208.207	-282.793
Leon2	dcom_uart	rtl	232	108.709	-123.291
Leon2	div	rtl	118	87.682	-30.318
Leon2	dsu	rtl	432	252.282	-179.718
Leon2	dsu_mem	rtl	64	75.102	11.102
Leon2	fpaux	rtl	42	82.370	40.370
Leon2	fp	rtl	707	137.673	-569.327
Leon2	fp1eu	rtl	566	130.608	-435.392
Leon2	fpu_core	rtl	54	86.266	32.266
Leon2	fpu_lth	rtl	722	199.001	-522.999

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	icache	rtl	230	195.935	-34.065
Leon2	ioport	rtl	107	136.814	29.814
Leon2	irqctrl	rtl	84	96.215	12.215
Leon2	irqctrl2	rtl	111	94.241	-16.759
Leon2	iu	rtl	2401	550.723	-1850.277
Leon2	lconf	rtl	38	114.744	76.744
Leon2	leon	rtl	131	363.021	232.021
Leon2	leon_pci	rtl	258	666.61	408.61
Leon2	mcore	rtl	172	275.388	103.388
Leon2	mctrl	rtl	675	199.896	-475.104
Leon2	fpu	rtl	32	199.001	167.001
Leon2	mul	rtl	265	89.656	-175.344
Leon2	GEN_NOT	rtl	2	58.045	56.045
Leon2	GEN_AND2	rtl	2	65.110	63.110
Leon2	GEN_OR2	rtl	2	65.110	63.110
Leon2	GEN_XOR2	rtl	2	65.110	63.110
Leon2	pci	rtl	28	288.977	260.977
Leon2	pci_arb	rtl	152	115.436	-36.564
Leon2	proc	rtl	138	355.393	217.393
Leon2	rstgen	rtl	29	90.292	61.292
Leon2	sdmctrl	rtl	342	140.529	-201.471
Leon2	pt33d00	rtl	2	58.045	56.045
Leon2	pt33d00u	rtl	3	67.884	64.884
Leon2	pt33d20	rtl	2	58.045	56.045
Leon2	pt33d20u	rtl	3	67.884	64.884
Leon2	pt33o01	rtl	2	58.045	56.045
Leon2	pt33o02	rtl	2	58.045	56.045
Leon2	pt33o03	rtl	2	58.045	56.045
Leon2	pt33o04	rtl	2	58.045	56.045
Leon2	pt33t01u	rtl	3	65.110	62.110
Leon2	pt33t02u	rtl	3	65.110	62.110
Leon2	pt33t03u	rtl	3	65.110	62.110
Leon2	pt33b01	rtl	8	82.014	74.014
Leon2	pt33b02	rtl	8	82.014	74.014
Leon2	pt33b03	rtl	8	82.014	74.014
Leon2	pt33b04	rtl	8	82.014	74.014
Leon2	pt33b01u	rtl	8	82.014	74.014
Leon2	pt33b02u	rtl	8	82.014	74.014
Leon2	pt33b03u	rtl	8	82.014	74.014
Leon2	pt33b04u	rtl	8	82.014	74.014
Leon2	pp33o01	rtl	2	58.045	56.045
Leon2	pp33b015vt	rtl	8	82.014	74.014
Leon2	pp33t015vt	rtl	3	65.110	62.110
Leon2	atc25_syncram	rtl	61	88.963	27.963
Leon2	atc25_dpram	rtl	94	128.276	34.276
Leon2	atc25_regfile_iu	rtl	63	92.233	29.233
Leon2	atc25_regfile_cp	rtl	51	83.839	32.839
Leon2	pc3d01	rtl	2	58.045	56.045
Leon2	pc3d21	rtl	2	58.045	56.045
Leon2	pt3o01	rtl	2	58.045	56.045
Leon2	pt3o02	rtl	2	58.045	56.045
Leon2	pt3o03	rtl	2	58.045	56.045
Leon2	pc3t01u	rtl	3	65.110	62.110
Leon2	pc3t02u	rtl	3	65.110	62.110
Leon2	pc3t03u	rtl	3	65.110	62.110
Leon2	pt3b01	rtl	8	82.014	74.014
Leon2	pt3b02	rtl	8	82.014	74.014
Leon2	pt3b03	rtl	8	82.014	74.014
Leon2	atc35_syncram	rtl	36	88.963	52.963
Leon2	atc35_regfile	rtl	58	90.904	32.904
Leon2	atc35_regfile_cp	rtl	41	83.839	42.839
Leon2	RAM64K36	rtl	51	1018.493	967.493

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	axcel_regfile_iu	rtl	183	92.233	-90.767
Leon2	axcel_regfile_cp	rtl	109	83.839	-25.161
Leon2	axcel_syncram	rtl	78	88.963	10.963
Leon2	uyfaa	rtl	14	79.239	65.239
Leon2	vyfa2gsa	rtl	12	93.369	81.369
Leon2	wyfa2gsa	rtl	18	131.469	113.469
Leon2	fs90_syncram	rtl	50	88.963	38.963
Leon2	fs90_regfile	rtl	55	90.904	35.904
Leon2	rfbypass	rtl	44	121.211	77.211
Leon2	generic_regfile_iu	rtl	85	92.233	7.233
Leon2	generic_regfile_cp	rtl	36	83.839	47.839
Leon2	generic_smult	rtl	21	67.768	46.768
Leon2	generic_clkgen	rtl	27	112.399	85.399
Leon2	geninpad	rtl	2	58.045	56.045
Leon2	gensmpad	rtl	2	58.045	56.045
Leon2	genoutpad	rtl	2	58.045	56.045
Leon2	gentoutpadu	rtl	3	65.110	62.110
Leon2	geniopad	rtl	5	82.014	77.014
Leon2	geniodpad	rtl	5	74.949	69.949
Leon2	genodpad	rtl	2	58.045	56.045
Leon2	regfile_iu	rtl	59	92.233	33.233
Leon2	regfile_cp	rtl	49	83.839	34.839
Leon2	hw_smult	rtl	30	67.768	37.768
Leon2	inpad	rtl	26	58.045	32.045
Leon2	smpad	rtl	26	58.045	32.045
Leon2	outpad	rtl	29	59.374	30.374
Leon2	toutpadu	rtl	29	66.439	37.439
Leon2	iopad	rtl	34	83.344	49.344
Leon2	smiopad	rtl	34	83.344	49.344
Leon2	odpad	rtl	29	59.374	30.374
Leon2	iodpad	rtl	29	76.279	47.279
Leon2	pcioutpad	rtl	12	58.045	46.045
Leon2	pcitoutpad	rtl	12	65.11	53.110
Leon2	pciiopad	rtl	19	82.014	63.014
Leon2	pciiodpad	rtl	14	74.949	60.949
Leon2	clkgen	rtl	17	112.399	95.399
Leon2	RAM256x9SST	rtl	25	405.707	380.707
Leon2	RAM256x9SA	rtl	25	398.642	373.642
Leon2	proasic_regfile_iu	rtl	69	92.233	23.233
Leon2	proasic_regfile_cp	rtl	63	83.839	20.839
Leon2	proasic_syncram	rtl	103	88.963	-14.037
Leon2	PDIDGZ	rtl	2	58.045	56.045
Leon2	PDISDGZ	rtl	2	58.045	56.045
Leon2	PDT02DGZ	rtl	4	65.110	61.110
Leon2	PDT04DGZ	rtl	4	65.110	61.110
Leon2	PDT08DGZ	rtl	4	65.110	61.110
Leon2	PDT12DGZ	rtl	4	65.110	61.110
Leon2	PDT16DGZ	rtl	4	65.110	61.110
Leon2	PDT24DGZ	rtl	4	65.110	61.110
Leon2	PDU02DGZ	rtl	5	82.014	77.014
Leon2	PDU04DGZ	rtl	5	82.014	77.014
Leon2	PDU08DGZ	rtl	5	82.014	77.014
Leon2	PDU12DGZ	rtl	5	82.014	77.014
Leon2	PDU16DGZ	rtl	5	82.014	77.014
Leon2	PDU24DGZ	rtl	5	82.014	77.014
Leon2	PDB02DGZ	rtl	5	82.014	77.014
Leon2	PDB04DGZ	rtl	5	82.014	77.014
Leon2	PDB08DGZ	rtl	5	82.014	77.014
Leon2	PDB12DGZ	rtl	5	82.014	77.014
Leon2	PDB16DGZ	rtl	5	82.014	77.014
Leon2	PDB24DGZ	rtl	5	82.014	77.014
Leon2	PDB02SDGZ	rtl	5	82.014	77.014

(continued on next page)

(continued from previous page)

Project	Entity	Architecture	L	$\hat{L}$	$\hat{L} - L$
Leon2	PDB04SDGZ	rtl	5	82.014	77.014
Leon2	PDB08SDGZ	rtl	5	82.014	77.014
Leon2	PDB12SDGZ	rtl	5	82.014	77.014
Leon2	PDB16SDGZ	rtl	5	82.014	77.014
Leon2	PDB24SDGZ	rtl	5	82.014	77.014
Leon2	tsmc25_syncram	rtl	95	88.963	-6.037
Leon2	tsmc25_dpram	rtl	102	128.276	26.276
Leon2	tsmc25_regfile_iu	rtl	102	90.904	-11.096
Leon2	tsmc25_regfile_cp	rtl	77	83.839	6.839
Leon2	C3I40	rtl	2	58.045	56.045
Leon2	C3I42	rtl	2	58.045	56.045
Leon2	C3O10	rtl	2	58.045	56.045
Leon2	C3O20	rtl	2	58.045	56.045
Leon2	C3O40	rtl	2	58.045	56.045
Leon2	C3B10U	rtl	8	82.014	74.014
Leon2	C3B20U	rtl	8	82.014	74.014
Leon2	C3B40U	rtl	8	82.014	74.014
Leon2	C3B10	rtl	8	82.014	74.014
Leon2	C3B20	rtl	8	82.014	74.014
Leon2	C3B40	rtl	8	82.014	74.014
Leon2	CD3B10T	rtl	8	82.014	74.014
Leon2	CD3B20T	rtl	8	82.014	74.014
Leon2	CD3B40T	rtl	8	82.014	74.014
Leon2	CD3O10T	rtl	7	58.045	51.045
Leon2	CD3O20T	rtl	7	58.045	51.045
Leon2	CD3O40T	rtl	7	58.045	51.045
Leon2	C3B42	rtl	8	82.014	74.014
Leon2	INVDL	rtl	2	58.045	56.045
Leon2	AND2DL	rtl	2	65.110	63.110
Leon2	OR2DL	rtl	2	65.110	63.110
Leon2	EXOR2DL	rtl	2	65.110	63.110
Leon2	umc18_syncram	rtl	56	88.963	32.963
Leon2	umc18_regfile	rtl	43	90.904	47.904
Leon2	timers	rtl	164	103.671	-60.329
Leon2	uart	rtl	276	104.982	-171.018
Leon2	wprot	rtl	94	104.113	10.113

14.16 Conclusions

Model aggregates described in this chapter exhibit good estimation accuracy. Resulting findings regarding coefficients of correlation between actual and estimated data are remarkably good, especially when the knowledge conditions of the bunch under estimation are high in information content. In addition, correlation degrades smoothly when the quantity of available information decreases.

We hold the experimentally verified belief that our models exhibit sufficiently robustness to be used in real operating conditions, since the above desirable behaviors appear both in external and internal validation (see figures 14.74 and 14.75).

As far as estimation error is concerned, we can say that such error smoothly increases when less information is available, and this happens on internal and external validation, though external validation errors are a little higher with respect to internal ones. As could be already guessed in many cases in previous chapters, bunches based mainly on the behavioral design paradigm (that is, bunches in which behavioral architectures and processes prevail) are harder to estimate than structural-based and data-flow-based ones.

Figure 14.72: Estimation error standard deviation in internal validation.

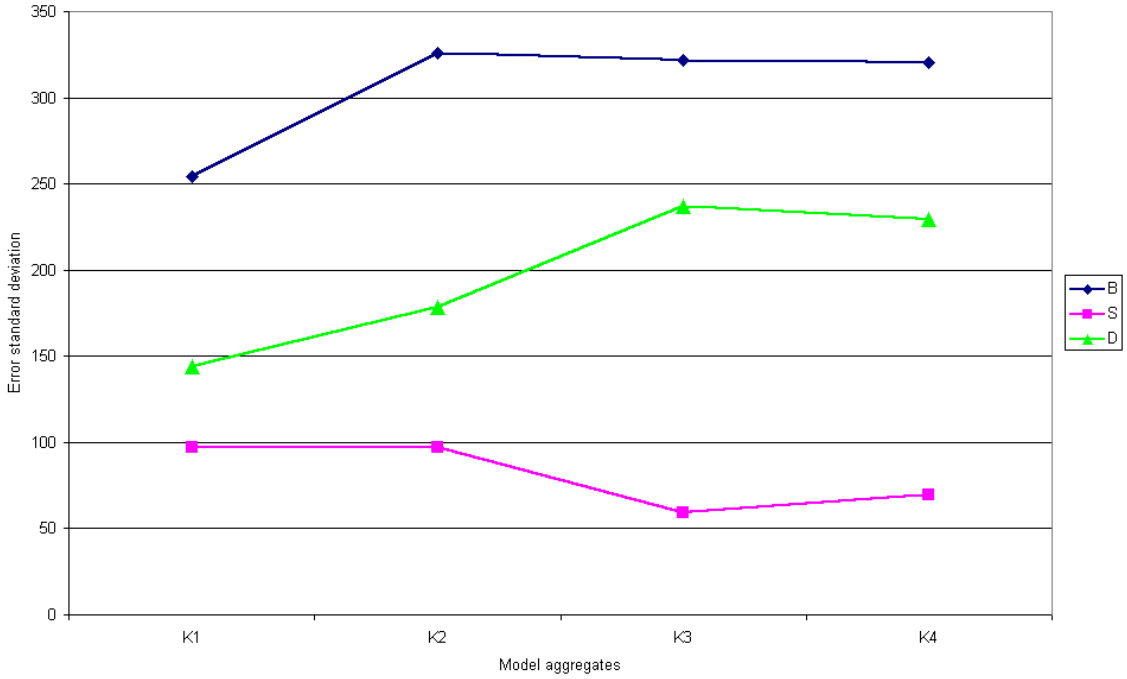


Figure 14.73: Estimation error standard deviation in external validation.

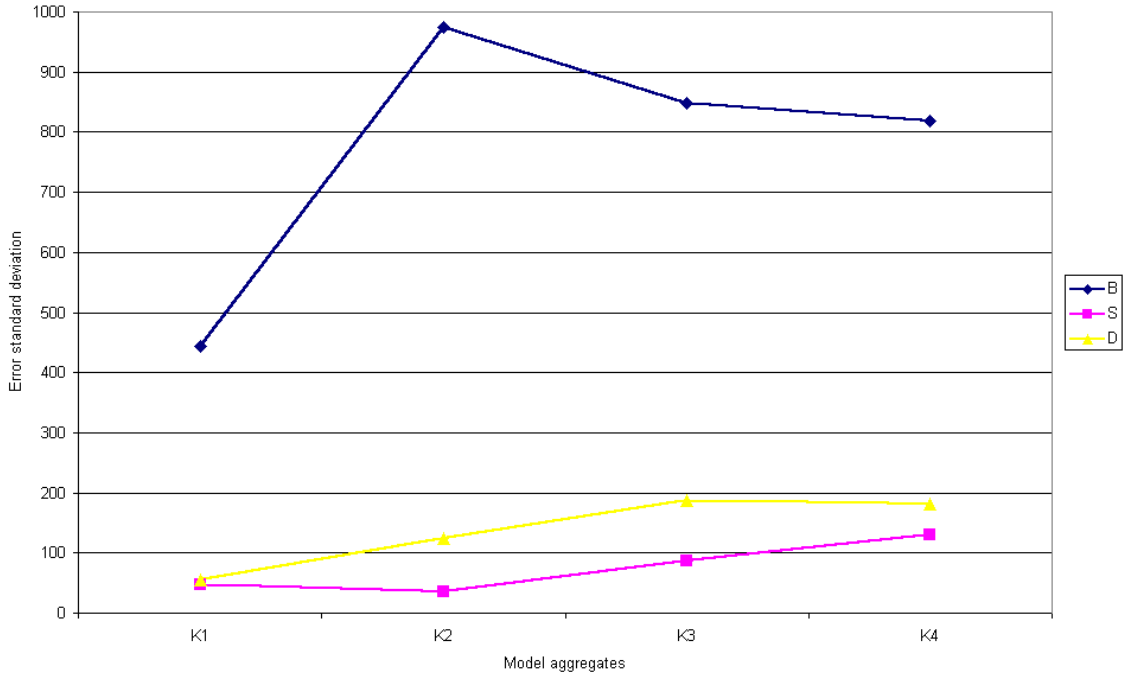


Figure 14.74: Coefficient of correlation between (L and $\hat{L}$) in internal validation.

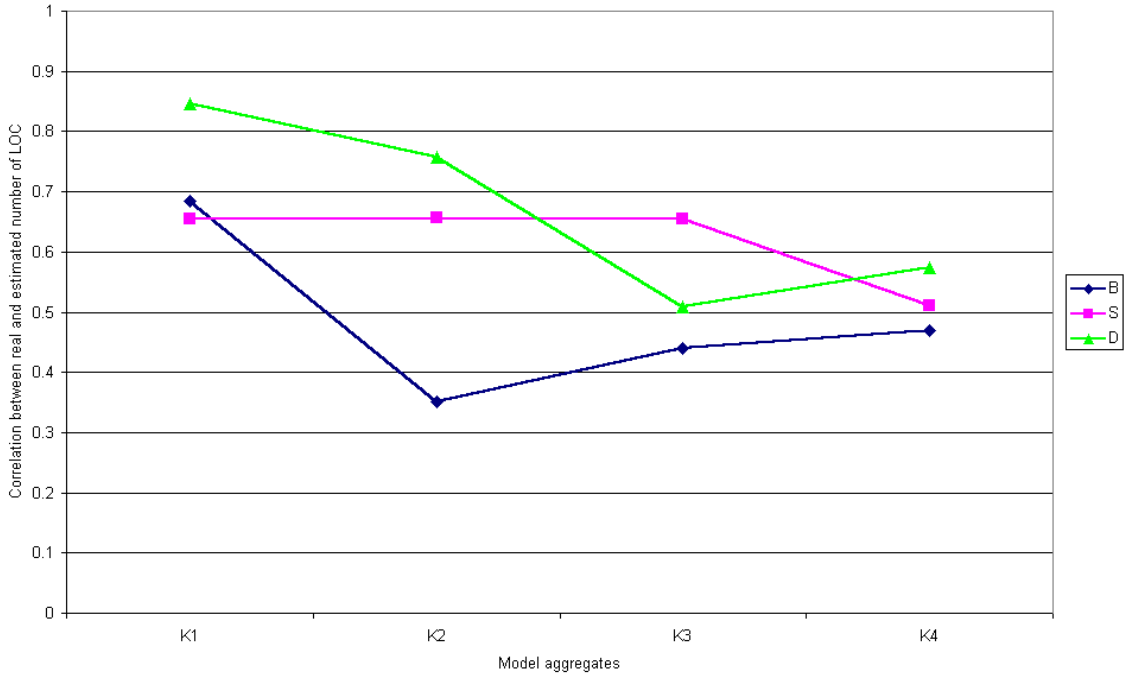
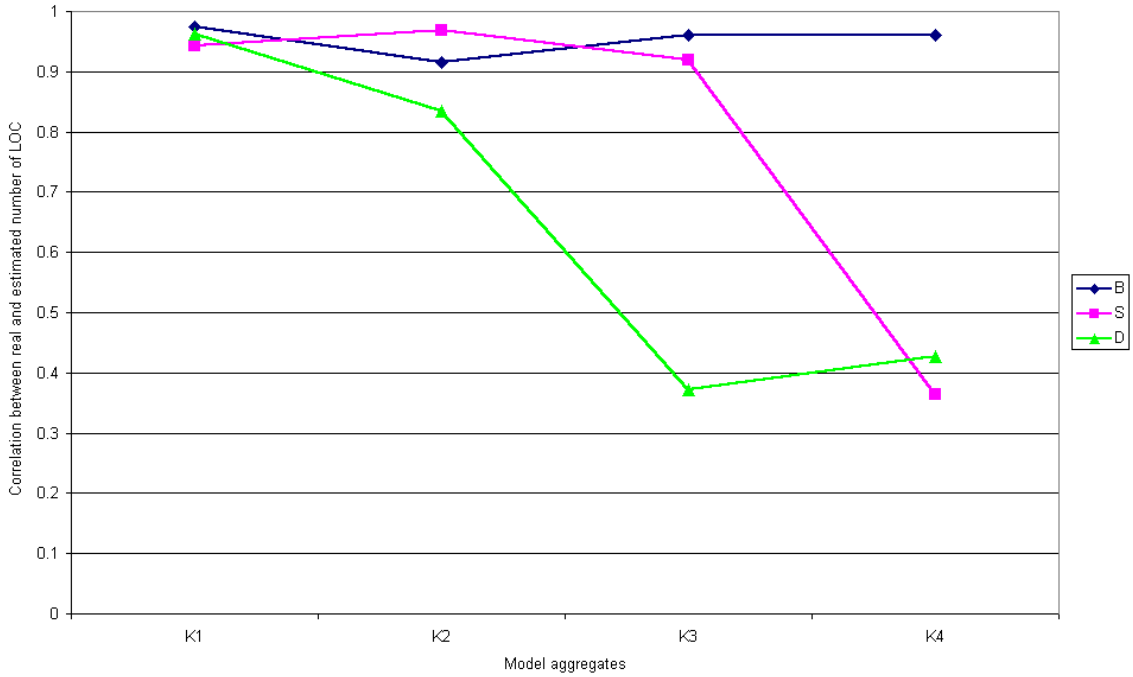


Figure 14.75: Coefficient of correlation between (L and $\hat{L}$) in external validation.



Chapter 15

SOG models

In the previous chapter we examined methods suitable to estimate the length of syntax objects and bunches: the aim of this chapter is the extension of those methods to the largest possible element: the SOG (or, to be more precisely, the KSOG).

In § 4.2 we showed how KSOGs are good at giving a formal representation of partially specified VHDL projects; in part III we gave a rich number of SOGs, each automatically generated for given real projects, thus proving that our formalism is both complete enough to cope with industrial-strength examples and rigorous enough to be implemented as an machine algorithm. Now it comes the time to show that aggregates of models introduced in the previous chapters, possibly supported by new models here created, are able to estimate, with a good degree of accuracy, the size of a complete SOG, starting from an incomplete KSOG[1], arrested at an arbitrary level of knowledge.

We will here tentatively formulate hypothesis on how the size of a SOG relates to simple indicators, then, as usual, those hypotheses will undergo the test of correlation coefficients. Once found at least one such hypothesis good enough to found a model on it, we will complete our methodology and validate it among internal and external cases.

Please note that SOG models are tuned with a higher degree of uncertainty with respect to syntax object models bunch models, since the number of cases in the tuning sets for the syntax object models is always high (usually some hundreds, in the worst cases many tens, in other cases some thousands), whereas SOGs in the tuning set are only 41 (one for each project). The situation is even worse for validation, where only 19 projects are available. This constraint is felt as particularly restrictive for those specialized models where there are only 1 or 2 cases suitable for model tuning purposes.

15.1 Levels

Most hypotheses we will introduce work on the basis of *level of a bunch*, which is an intuitive concept; yet we think that a formal definition would avoid confusion and immediately give an idea on how the implementation of the concept we describe are done.

[1] specifications appear in the form of incomplete KSOGs: in fact, a complete KSOG –that is a KSOG in which all nodes are completely known– is equivalent to a SOG representing a finished project

Given a KSOG $K(W, E, p)$ and a set B containing all the bunches in which K can be partitioned according to the definitions previously indicated, for each bunch $B_e \in B$ we define the function *level*, denoted as $L(\cdot)$ as follows:

- $e \in W : (E(e) \wedge \nexists (m, a) \in W : (E(m) \wedge m \neq e \wedge A(a) \wedge m \supset a \wedge a \rightarrow e)) \Rightarrow L(B_e) = 1$

  the intuitive meaning of this rule is as follows: bunches in which appears a top entity (that is, an entity not referred by any architecture of a different entity, thus including top level entities with structural recursion) have level 1;

- $R_e = B_i \in B : a \in B_i \wedge A(a) \wedge a \rightarrow e \Rightarrow L(B_e) = (max_{B_i \in R_e} L(B_i)) + 1$

  the intuitive meaning of this rule is as follows: given a bunch B_e rooted on entity e, let's identify all the bunches containing architectures referencing e: the set of all such bunches will be R_e. The level of B_e is given by the maximum level found among the bunches in R_e plus one.

Thanks to the this definition we will divide all the SOGs in a set of levels. As a consequence of the above rules, the level to which all the unreferenced bunches belong will be level 1; in addition any bunch will be assigned to a given level in such a way that it never happens that a bunch references another bunch belonging to the same level or a level above (except for self-referencing bunches, of course). For example, if a bunch is referenced only by one bunch belonging to level 1 and by one bunch in level 3, it should be assigned to level 4.

A simple algorithm implementation that achieves the bunch level tagging (in compliance with definitions) in an iterative fashion is the following:

```
tag all nodes as belonging to level 1;
repeat
      for each node i
            get all nodes referenced by node i;
            for each such node j
                if level(j) <= level(i)
                then set level(j):= level(i)+1;
            end for
      end for
until (fixed_point_is_reached);
```

Experiments, in accordance with theoretically derivable evidence, show that at every iteration of the external `for` loop one level is added, and that fixed point is reached in exactly $L + 1$ iterations, where L is the maximum level among all bunches, and the last iteration does not do anything and serves only to detect the fixed point.

15.2 SOG depths

In these following sections, we will use the term *node* or *bunch* indifferently, since the SOG is regarded as a DAG composed of bunches. As far as taxonomy issues are concerned, by projects and SOGs we mean the same thing; we also indicate the level of a bunch as *depth*, and by *depth of a given SOG* we mean the maximum level of all the bunches belonging to that SOG.

In this section we list all our tuning projects, and the distribution of their code (expressed in number of lines of code per each level.

SOGs with depth 1		
	Absolute values	Relative values
	L1	L1
ATL18	10813	1.000
Am2901	286	1.000
gl85	1638	1.000
i80386	757	1.000
SuperscalarDLX	1968	1.000
xapp349	232	1.000
Total	15694	1.000

SOGs with depth 2				
	Absolute values		Relative values	
	L1	L2	L1	L2
DLX	1112	2141	0.519	1.000
HC11	128	2166	0.059	1.000
Jane	28	278	0.101	1.000
PIC16C5X	204	877	0.233	1.000
rd1007	68	441	0.154	1.000
xapp328	1743	1849	0.943	1.000
xapp348	712	754	0.944	1.000
xapp354	539	693	0.778	1.000
xapp357	101	115	0.878	1.000
xapp365	155	374	0.414	1.000
xapp369	74	327	0.226	1.000
Total	4864	10015	0.486	1.000

SOGs with depth 3						
	Absolute values			Relative values		
	L1	L2	L3	L1	L2	L3
ADC0808	113	145	177	0.638	0.819	1.000
an-XC2S-USB	119	417	431	0.276	0.968	1.000
an-XC2S-XR16	167	539	801	0.208	0.673	1.000
ax8	362	1534	1999	0.181	0.767	1.000
Free6502	70	181	803	0.087	0.225	1.000
i8051	11	127	4208	0.003	0.030	1.000
ppx16	268	643	960	0.279	0.670	1.000
T51	115	944	1336	0.086	0.707	1.000
xapp358	36	187	312	0.115	0.599	1.000
xapp363	242	1248	1518	0.159	0.822	1.000
xapp367	149	392	422	0.353	0.929	1.000
xapp370	171	699	773	0.221	0.904	1.000
xapp336_8	42	284	783	0.054	0.363	1.000
Total	1865	7340	14523	0.128	0.505	1.000

SOGs with depth 4								
	Absolute values				Relative values			
	L1	L2	L3	L4	L1	L2	L3	L4
ans_RISC8	164	624	1406	1606	0.102	0.389	0.875	1.000
TE51	36	83	200	1060	0.034	0.078	0.189	1.000
xapp146	145	247	705	756	0.192	0.327	0.933	1.000
xapp333	394	701	1455	1508	0.261	0.465	0.965	1.000
xapp345	139	180	324	557	0.250	0.323	0.582	1.000
xapp355	116	191	649	728	0.159	0.262	0.891	1.000
xapp336	40	128	547	1121	0.036	0.114	0.488	1.000
Total	1034	2154	5286	7336	0.141	0.294	0.721	1.000

SOGs with depth 5										
	Absolute values					Relative values				
	L1	L2	L3	L4	L5	L1	L2	L3	L4	L5
T80	174	299	1269	2040	3577	0.049	0.084	0.355	0.570	1.000
xapp356	226	415	772	1116	1220	0.185	0.340	0.633	0.915	1.000
Total	400	714	2041	3156	4797	0.083	0.149	0.425	0.658	1.000

SOGs with depth 6, 7 and 8 have only one case;

SOGs with depth 6						
	L1	L2	L3	L4	L5	L6
gl85struct	97	394	1490	1793	1817	1844
Total	0.053	0.214	0.808	0.972	0.985	1.000

SOGs with depth 7							
	L1	L2	L3	L4	L5	L6	L7
ERC32	347	1072	4740	6385	7770	14238	15393
Total	0.023	0.070	0.308	0.415	0.505	0.925	1.000

SOGs with depth 8								
	L1	L2	L3	L4	L5	L6	L7	L8
Leon	1041	1547	3771	6526	7944	8510	9042	9238
Total	0.113	0.167	0.408	0.706	0.860	0.921	0.979	1.000

15.3 Hypotheses

In the following paragraphs we will examine a few hypotheses that could become the basis of a model able to estimate the size of a whole SOG, given a few information, e.g. the size of a limited number of bunches located in particular positions plus some other indicator like the number of levels.

15.3.1 Hypothesis 1

Hypothesis: is there any relationship between the level at which a node is located and its size?

If this hypothesis should be proven true, it would be possible to decompose the problem of estimating the size of a level in two simpler subproblems: the first is to estimate the number of nodes in each level, and the second is to estimate the average size of each node in that level that –according to the hypotheses here explained– would be simple, by knowing the level number.

Experimental results: the correlation coefficient between the node sizes and the level numbers is very low (0.1240) thus showing that there is poor relationship between the two quantities. The stated hypothesis is therefore to be considered false.

15.3.2 Hypothesis 2

Hypothesis: is there any relationship between the level number and the cumulative size[2] of that level?

If this hypothesis should be proven true, it would be possible to create a good estimator of a level size depending on its level number, the completely focus on the second half of the problem, that is, trying to estimate the number of levels in the SOG.

Experimental results: the correlation coefficient between the level size and the level numbers is low (0.2898) thus showing that there is poor relationship between the two quantities. The stated hypothesis is therefore to be considered false.

[2]By cumulative size of a level we intend the sum of the sizes of all the bunches belonging to that level.

15.3.3 Hypothesis 3

Hypothesis: is there any relationship between the level number and the number of nodes belonging to that level?

This hypothesis can be regarded as the *dual* hypothesis with respect the the previous one. Should it be proven true, it would be possible to create a good estimator of the number of nodes contained in each level, thus decomposing the problem of estimating the cumulative size of a level in two subproblems, of which the first is solved by the above estimator, and the second would be to find a reasonable estimator of the expected size of a node.

Experimental results: the correlation coefficient between the level numbers and the numbers of nodes in the levels is low (0.1278), thus showing that there is poor relationship between the two quantities. The stated hypothesis is therefore to be considered false.

15.3.4 Hypothesis 4

Hypothesis: is there any relationship between the level number and the ratio between the number of nodes in that level and the number of nodes in all the project?

This hypothesis is an improvement over hypothesis 3, with the addition that the number of nodes in each levels are normalized by dividing them by the overall size of the project expressed as a node count.

If this hypothesis should be proven true, same consequences as above apply, except for a scaling parameter which should be possible to estimate by way of an alternate method.

Experimental results: the coefficient of correlation between the level numbers and the ratios between node count in those levels and overall node count is low (-0.1085), thus showing that there is poor correlation between the two variable. The stated hypothesis is therefore to be considered false.

15.3.5 Hypothesis 5

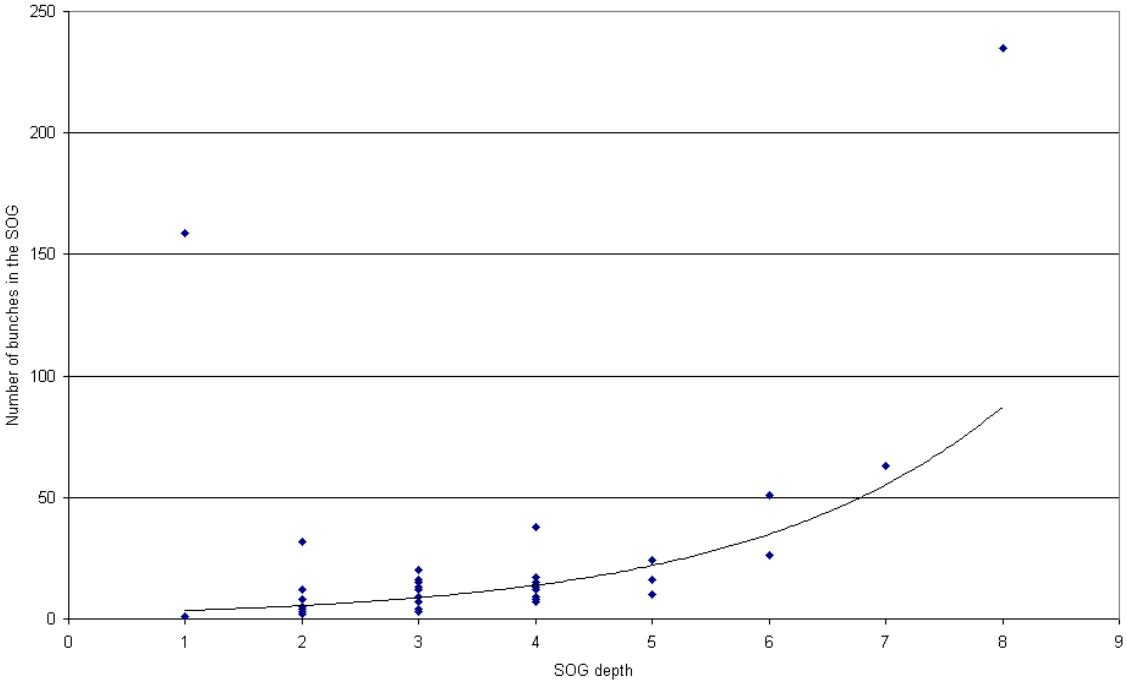
Hypothesis: is there any relationship between the depth of a project and the number of nodes belonging to it?

Experimental results: the correlation coefficient between the SOG depths and their node count is surprisingly good: if we indicate with D_i the depth of i-th SOG, and with N_i its node count, coefficient of correlation are as follows:

Type of calculated correlation coefficient	Value
$Corr(D_i, N_i)$	0.4819
$Corr(\ln(D_i), N_i)$	0.2365
$Corr(\exp(D_i), N_i)$	**0.8145**
$Corr(D_i^2, N_i)$	0.6604

Results show that the number of nodes in each SOG is well correlated to the exponential of the SOG depth. Though holding an interesting predictive power, this hypothesis must be considered not more than a by-product of this thesis, and it is not suitable to exploit and integrate outputs of bunch models and syntax object models, in order to give life to a higher-order model capable of estimating SOG sizes.

Figure 15.1: Hypothesis 5: SOG depth vs size in number of contained bunches.



15.3.6 Hypothesis 6

Hypothesis: is there any relationship between the depth of a project and its size expressed in lines of code?

Experimental results: the correlation coefficient between the SOG depths and their line count is surprisingly good, even better than ones reported for previous hypothesis.

Let D_i be the depth of i-th SOG, and L_i its total number of lines, coefficient of correlation are as follows:

Type of calculated correlation coefficient	Value
$Corr(D_i, L_i)$	0.5914
$Corr(\ln(D_i), L_i)$	0.3704
$Corr(\exp(D_i), L_i)$	**0.9826**
$Corr(D_i^2, L_i)$	0.7685

Results show that the size of a SOG expressed in lines of code is well correlated to the exponential of the SOG depth. Same considerations as the ones expressed for hypothesis 5 apply: this hypothesis holds an interesting predictive power, nevertheless it does not make use of any information coming from finer granularity models.

15.3.7 Hypothesis 7

Hypothesis: is there any relationship between the depth of a project and its size expressed in lines of code divided by the size of the top level?

This thesis is an attempt to reuse good results obtained during the analysis of previous hypotheses (especially hypothesis 6) and, in addition, it tries to open a channel to acquire data from finer granularity models. In fact it tries to normalize project sizes on the basis of the size of their top level, which is obtained by applying bunch level models and syntax object models.

Experimental results: correlation coefficients between the SOG depths and *normalized* SOG size are unexpectedly negative, but still good in their absolute values.

Let D_i be the depth of i-th SOG, L_i its total number of lines of code and L_{i_1} the number of lines of code of its level 1 (that is, the summation of the lengths of all the syntax objects belonging to bunches of top level).

Type of calculated correlation coefficient	Value
$Corr(D_i, L_i/L_{i_1})$	-0.607
$Corr(\ln(D_i), L_i/L_{i_1})$	**-0.756**
$Corr(\exp(D_i), L_i/L_{i_1})$	-0.186
$Corr(D_i^2, L_i)$	-0.461

This model is unquestionably interesting, and it would be further developed and analyzed, if only it would not have the following drawback: the amount of required knowledge is not arbitrarily variable; instead it is fixed: the designed must exactly know level 1, any additional knowledge on levels below is useless.

Possible modifications of this model to solve this limitations are left to future developments.

Another, non-negligible reason why hypothesis 7 was not deeply considered is that our final hypothesis, number 8, shows much better promises and no drawbacks.

15.3.8 Hypothesis 8

Hypothesis 8 is not thoroughly described in this paragraph like we did for any other hypothesis. The reason for this bizarre choice is that hypothesis 8 is, in practice, the basis for models SOGM0 and SOGM1, whose description occupies the following sections.

15.4 Model SOGM0

The hypothesis at the basis of model SOGM0 is the following: given a SOG of depth n, where the size of all levels is known starting from level 1 up to level k, it should be possible to find an appropriate constant value, representing the following ratio:

$$\frac{L}{\sum_{i \leqslant k} L_i} = \frac{\sum_{i \leqslant n} L_i}{\sum_{i \leqslant k} L_i}$$

This ratio expresses how many times the whole project is larger than the size of all bunches in level k and all levels above k, up to top level.

In each cell (row i, column j) of the following table we reported the average values of such ratios, namely the ratios between the size of the all nodes of levels j and above, and the size of all the projects with depth i.

For example in projects with exactly 6 levels (row 6), the number of lines of code belonging to levels 3 and above (column L3) represents the 80.8% of all the lines of code of the project.

	Last known-size level							
Number of levels	L1	L2	L3	L4	L5	L6	L7	L8
1	1	·	·	·	·	·	·	·
2	0.486	1	·	·	·	·	·	·
3	0.128	0.505	1	·	·	·	·	·
4	0.141	0.294	0.721	1	·	·	·	·
5	0.083	0.149	0.425	0.658	1	·	·	·
6	0.053	0.214	0.808	0.972	0.985	1	·	·
7	0.023	0.070	0.308	0.415	0.505	0.925	1	·
8	0.113	0.167	0.408	0.706	0.860	0.921	0.979	1

Model SOGM0 simply inverts the coefficients of the above table in order to obtain a multiplier constant that, multiplied by the size of levels k and above, should return a good estimate of the whole size of each project.

	Last known-size level							
Number of levels	L1	L2	L3	L4	L5	L6	L7	L8
	1	·	·	·	·	·	·	·
SOGM0(2,.)	2.059	1	·	·	·	·	·	·
SOGM0(3,.)	7.787	1.979	1	·	·	·	·	·
SOGM0(4,.)	7.095	3.406	1.388	1	·	·	·	·
SOGM0(5,.)	11.993	6.718	2.350	1.520	1	·	·	·
(SOGM0(6,.))	19.010	4.680	1.238	1.028	1.015	1	·	·
(SOGM0(7,.))	44.360	14.359	3.247	2.411	1.981	1.081	1	·
(SOGM0(8,.))	8.874	5.972	2.450	1.416	1.163	1.086	1.022	1

Models SOGM0(6,.), SOGM0(7,.) and SOGM0(8,.) are reported in the above table for sake of completeness, but are not used nor validated below since the low number of projects in their tuning set does not allow proper training.

15.4.1 Model SOGM0(2,1)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	910.455	910.455	0.000
Variance	592572.473	1282192.929	773924.965
Standard deviation	769.787	1132.340	879.730

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.6315

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
DLX	1112	2141	2289.613	148.613
HC11	128	2166	263.553	-1902.447
Jane	28	278	57.652	-220.348
PIC16C5X	204	877	420.037	-456.963
rd1007	68	441	140.012	-300.988
xapp328	1743	1849	3588.846	1739.846
xapp348	712	754	1466.012	712.012
xapp354	539	693	1109.804	416.804
xapp357	101	115	207.959	92.959
xapp365	155	374	319.146	-54.854
xapp369	74	327	152.366	-174.634

External validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	2656257.867	149284.288	2068223.195
Variance	1629.803	386.373	1438.132
Standard deviation	1667.333	567.256	-1100.077

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.5854

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
DLX2	417	1117	858.605	-258.395
fw09	403	4738	829.779	-3908.221
RLS	113	1329	232.668	-1096.332
SPIM-Pipe	143	478	294.438	-183.562
SPIM	70	297	144.130	-152.870
ZR36060	507	2045	1043.916	-1001.084

15.4.2 Model SOGM0(3,1)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	1117.154	1117.154	0
Variance	1123267.141	613687.205	1782176.714
Standard deviation	1059.843	783.382	1334.982

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	-0.0272

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
ADC0808	113	177	879.946	702.946
an-XC2S-USB	119	431	926.669	495.669
an-XC2S-XR16	167	801	1300.451	499.451
ax8	362	1999	2818.942	819.942
Free6502	70	803	545.099	-257.901
i8051	11	4208	85.658	-4122.342
ppx16	268	960	2086.951	1126.951
T51	115	1336	895.520	-440.480
xapp358	36	312	280.337	-31.663
xapp363	242	1518	1884.486	366.486
xapp367	149	422	1160.283	738.283
xapp370	171	773	1331.599	558.599
xapp336_8	42	783	327.060	-455.940

External validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	2087.000	6310.172	4223.172
Variance	4590721.000	18792538.737	10403708.110
Standard deviation	2142.597	4335.036	3225.478

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.6987

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
HDLLib	1110	1327	8643.716	7316.716
IEEE1149	168	428	1308.238	880.238
Manticore	1153	4506	8978.562	4472.562

15.4.3 Model SOGM0(3,2)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	1117.154	1117.154	0.000
Variance	1123267.141	770968.576	1583316.302
Standard deviation	1059.843	878.048	1258.299

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.1671

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 2} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
ADC0808	145	177	286.899	109.899
an-XC2S-USB	417	431	825.081	394.081
an-XC2S-XR16	539	801	1066.471	265.471
ax8	1534	1999	3035.188	1036.188
Free6502	181	803	358.128	-444.872
i8051	127	4208	251.284	-3956.716
ppx16	643	960	1272.246	312.246
T51	944	1336	1867.808	531.808
xapp358	187	312	370.000	58.000
xapp363	1248	1518	2469.306	951.306
xapp367	392	422	775.615	353.615
xapp370	699	773	1383.049	610.049
xapp336_8	284	783	561.925	-221.075

External validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	2087.000	3973.709	1886.709
Variance	4590721.000	16819565.935	3837819.021
Standard deviation	2142.597	4101.166	1959.035

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	1

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 2} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
HDLLib	1306	1327	2584.065	1257.065
IEEE1149	378	428	747.915	319.915
Manticore	4341	4506	8589.148	4083.148

15.4.4 Model SOGM0(4,1)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	1048.000	1048.000	0.000
Variance	159530.333	722264.586	578150.264
Standard deviation	399.412	849.862	760.362

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.4473

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
ans_RISC8	164	1606	1163.544	-442.456
TE51	36	1060	255.412	-804.588
xapp146	145	756	1028.743	272.743
xapp333	394	1508	2795.342	1287.342
xapp345	139	557	986.174	429.174
xapp355	116	728	822.994	94.994
xapp336	40	1121	283.791	-837.209

External validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	874.000	955.153	-370.271
Variance	512.000	12181.280	7698.557
Standard deviation	22.627	110.369	87.741

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.1194

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
RTC	60	858	425.687	-432.313
RTC-alt	82	890	581.772	-308.228

15.4.5 Model SOGM0(4,2)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	1048.000	1048.000	0.000
Variance	159530.333	717454.508	355486.289
Standard deviation	399.412	847.027	596.227

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.7707

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
ans_RISC8	624	1606	2125.192	519.192
TE51	83	1060	282.678	-777.322
xapp146	247	756	841.222	85.222
xapp333	701	1508	2387.435	879.435
xapp345	180	557	613.036	56.036
xapp355	191	728	650.500	-77.500
xapp336	128	1121	435.937	-685.063

External validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	874.000	955.153	-370.271
Variance	512.000	12181.280	7698.557
Standard deviation	22.627	110.369	87.741

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.0971

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 2} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
RTC	404	858	1375.926	517.926
RTC-alt	422	890	1437.229	547.229

15.4.6 Model SOGM0(4,3)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	1048.000	1048.000	0.000
Variance	159530.333	470141.090	202429.685
Standard deviation	399.412	685.668	449.922

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.7800

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 3} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
ans_RISC8	1406	1606	1951.271	345.271
TE51	200	1060	277.563	-782.437
xapp146	705	756	978.411	222.411
xapp333	1455	1508	2019.274	511.274
xapp345	324	557	449.653	-107.347
xapp355	649	728	900.693	172.693
xapp336	547	1121	759.136	-361.864

External validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	874.000	864.610	-9.390
Variance	512.000	986.130	77.006
Standard deviation	22.627	31.403	8.775

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	-0.413

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 3} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
RTC	607	858	842.405	-15.595
RTC-alt	639	890	886.815	-3.185

15.4.7 Model SOGM0(5,1)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	2398.500	2398.500	0.000
Variance	2777724.500	194444.716	4442017.986
Standard deviation	1666.651	440.959	2107.610

Since the tuning population is composed by two elements only, correlation for SOGM(5,.) models would assume only the $+1$ or -1 values, and is hereby meaningless, therefore it is not reported.

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
T80	174	3577	2086.695	-1490.305
xapp356	226	1220	2710.305	1490.305

External validation

External validation is not possible due to complete lack of 5-level projects in the external validation set.

15.4.8 Model SOGM0(5,2)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	2398.500	2398.500	0.000
Variance	2777724.500	303688.954	4918328.530
Standard deviation	1666.651	551.080	2217.730

Since the tuning population is composed by two elements only, correlation for SOGM(5,.) models would assume only the $+1$ or -1 values, and is hereby meaningless, therefore it is not reported.

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 2} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
T80	299	3577	2008.828	-1568.172
xapp356	415	1220	2788.172	1568.172

External validation

External validation is not possible due to complete lack of 5-level projects in the external validation set.

15.4.9 Model SOGM0(5,3)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	2398.500	2398.500	0.000
Variance	2777724.500	682238.477	706731.761
Standard deviation	1666.651	825.977	840.673

Since the tuning population is composed by two elements only, correlation for SOGM(5,.) models would assume only the $+1$ or -1 values, and is hereby meaningless, therefore it is not reported.

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 3} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
T80	1269	3577	2982.554	-594.446
xapp356	772	1220	1814.446	594.446

External validation

External validation is not possible due to complete lack of 5-level projects in the external validation set.

15.4.10 Model SOGM0(5,4)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	2398.500	2398.500	0.000
Variance	2777724.500	986232.692	453680.641
Standard deviation	1666.651	993.092	673.558

Since the tuning population is composed by two elements only, correlation for SOGM(5,.) models would assume only the $+1$ or -1 values, and is hereby meaningless, therefore it is not reported.

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 4} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
T80	2040	3577	3100.722	-476.278
xapp356	1116	1220	1696.278	476.278

External validation

External validation is not possible due to complete lack of 5-level projects in the external validation set.

15.5 Model SOGM1

Model SOGM1 is an attempt to improve the estimation accuracy of model SOGM0 by introducing an additive constant k_0, which should be able to reduce the estimation error variance, but not the correlation coefficient between estimated and actual values, of course.

Again, the lack of an appropriate number of tuning cases for models SOGM1(6,.), SOGM1(7,.) and SOGM1(8,.) renders a sensible identification of their parameters impossible.

For model SOGM1(5,.), where the number of tuning cases is exactly two, the estimation error variance will be necessarily equal to zero. This is an evident phenomenon of data over-fitting.

15.5.1 Model SOGM1(2,1)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	910.455	910.455	0.000
Variance	592572.473	236284.579	356287.893
Standard deviation	769.787	486.091	596.899

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.6315

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
DLX	1112	2141	1502.500	-638.500
HC11	128	2166	632.752	-1533.248
Jane	28	278	544.363	266.363
PIC16C5X	204	877	699.928	-177.072
rd1007	68	441	579.719	138.719
xapp328	1743	1849	2060.235	211.235
xapp348	712	754	1148.944	394.944
xapp354	539	693	996.031	303.031
xapp357	101	115	608.887	493.887
xapp365	155	374	656.617	282.617
xapp369	74	327	585.022	258.022

Identified model coefficients:

Coefficient	Value
k_L	0.884
k_0	519.614

External validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	1667.333	763.126	-904.207
Variance	2656257.867	27510.349	2367251.681
Standard deviation	1629.803	165.862	1538.588

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.5854

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
DLX2	417	1117	888.197	-228.803
fw09	403	4738	875.822	-3862.178
RLS	113	1329	619.494	-709.506
SPIM-Pipe	143	478	646.011	168.011
SPIM	70	297	581.487	284.487
ZR36060	507	2045	967.747	-1077.253

15.5.2 Model SOGM1(3,1)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	1117.154	1117.154	0.000
Variance	1123267.141	833.105	1122434.036
Standard deviation	1059.843	28.864	1059.450

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.0272

Identified model coefficients:

Coefficient	Value
k_L	-0.287
k_0	1158.315

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
ADC0808	113	177	1125.894	948.894
an-XC2S-USB	119	431	1124.172	693.172
an-XC2S-XR16	167	801	1110.400	309.400
ax8	362	1999	1054.452	-944.548
Free6502	70	803	1138.231	335.231
i8051	11	4208	1155.159	-3052.841
ppx16	268	960	1081.422	121.422
T51	115	1336	1125.320	-210.680
xapp358	36	312	1147.986	835.986
xapp363	242	1518	1088.882	-429.118
xapp367	149	422	1115.565	693.565
xapp370	171	773	1109.253	336.253
xapp336_8	42	783	1146.265	363.265

External validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	2087.000	925.818	-1161.182
Variance	4590721.000	25511.613	5094461.629
Standard deviation	2142.597	159.724	2257.091

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	-0.6987

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
HDLLib	1110	1327	839.839	-487.161
IEEE1149	168	428	1110.113	682.113
Manticore	1153	4506	827.502	-3678.498

15.5.3 Model SOGM1(3,2)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	1117.154	1117.154	0.000
Variance	1123267.141	31347.219	1091919.922
Standard deviation	1059.843	177.051	1044.950

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.1671

Identified model coefficients:

Coefficient	Value
k_L	0.399
k_0	891.889

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 2} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
ADC0808	145	177	949.739	772.739
an-XC2S-USB	417	431	1058.260	627.260
an-XC2S-XR16	539	801	1106.934	305.934
ax8	1534	1999	1503.910	-495.090
Free6502	181	803	964.102	161.102
i8051	127	4208	942.558	-3265.442
ppx16	643	960	1148.427	188.427
T51	944	1336	1268.517	-67.483
xapp358	187	312	966.496	654.496
xapp363	1248	1518	1389.805	-128.195
xapp367	392	422	1048.285	626.285
xapp370	699	773	1170.769	397.769
xapp336_8	284	783	1005.196	222.196

External validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	2087.000	1693.156	-393.844
Variance	4590721.000	683875.620	1731247.638
Standard deviation	2142.597	826.968	1315.769

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	1.000

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 2} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
HDLLib	1306	1327	1412.945	85.945
IEEE1149	378	428	1042.700	614.700
Manticore	4341	4506	2623.822	-1882.178

15.5.4 Model SOGM1(4,1)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	1048.000	1048.000	0.000
Variance	159530.333	31913.539	127616.794
Standard deviation	399.412	178.644	357.235

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.4473

Identified model coefficients:

Coefficient	Value
k_L	1.491
k_0	827.707

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
ans_RISC8	164	1606	1072.288	-533.712
TE51	36	1060	881.395	-178.605
xapp146	145	756	1043.952	287.952
xapp333	394	1508	1415.297	-92.703
xapp345	139	557	1035.004	478.004
xapp355	116	728	1000.703	272.703
xapp336	40	1121	887.361	-233.639

External validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	874.000	1055.956	59.593
Variance	512.000	538.235	0.328
Standard deviation	22.627	23.200	0.572

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.1118

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
RTC	60	858	917.188	59.188
RTC-alt	82	890	949.997	59.997

15.5.5 Model SOGM1(4,2)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	1048.000	1048.000	0.000
Variance	159530.333	94765.848	64764.485
Standard deviation	399.412	307.841	254.489

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.7707

Identified model coefficients:

Coefficient	Value
k_L	1.238
k_0	667.118

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 1} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
ans_RISC8	624	1606	1439.491	-166.509
TE51	83	1060	769.854	-290.146
xapp146	247	756	972.849	216.849
xapp333	701	1508	1534.800	26.800
xapp345	180	557	889.918	332.918
xapp355	191	728	903.534	175.534
xapp336	128	1121	825.554	-295.446

External validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	874.000	1150.635	304.320
Variance	512.000	248.199	47.239
Standard deviation	22.627	15.754	6.873

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.3579

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 2} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
RTC	404	858	1167.180	309.180
RTC-alt	422	890	1189.460	299.460

15.5.6 Model SOGM1(4,3)

Internal validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	1048.000	1048.000	0.000
Variance	59530.333	97064.215	62466.119
Standard deviation	399.412	311.551	249.932

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.7800

Identified model coefficients:

Coefficient	Value
k_L	0.631
k_0	571.814

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 3} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
ans_RISC8	1406	1606	1458.425	-147.575
TE51	200	1060	697.932	-362.068
xapp146	705	756	1016.380	260.380
xapp333	1455	1508	1489.323	-18.677
xapp345	324	557	776.125	219.125
xapp355	649	728	981.067	253.067
xapp336	547	1121	916.747	-204.253

External validation

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	874.000	1122.950	248.950
Variance	512.000	203.594	69.869
Standard deviation	22.627	14.269	8.359

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	1.0000

Real versus estimated project sizes in lines of code:

Project	$\sum_{i \leqslant 2} L_i$	$\sum L_i$	$\widehat{\sum L_i}$	$\widehat{\sum L_i} - \sum L_i$
RTC	607	858	1112.861	254.861
RTC-alt	639	890	1133.040	243.040

Chapter 16

Methodology validation

In this chapter, all the methodology, intended as the organic collection of models at the three possible levels of granularity (syntax object, bunch and SOG), is validated, both against internal and external validation data.

Since it would be impractical to perform validation for all the possible knowledge conditions, we choose to simulate a K2 knowledge condition on all bunches of a given level i and all levels above i, for each SOG. The following sections reports evaluation detailed results and plotted in charts.

16.1 Internal validation

16.1.1 Result summary

Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	2582.071	1826.582	427.492
Variance	14968535.587	8707892.399	1961153.910
Standard deviation	3868.919	2950.914	1400.412

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.8627

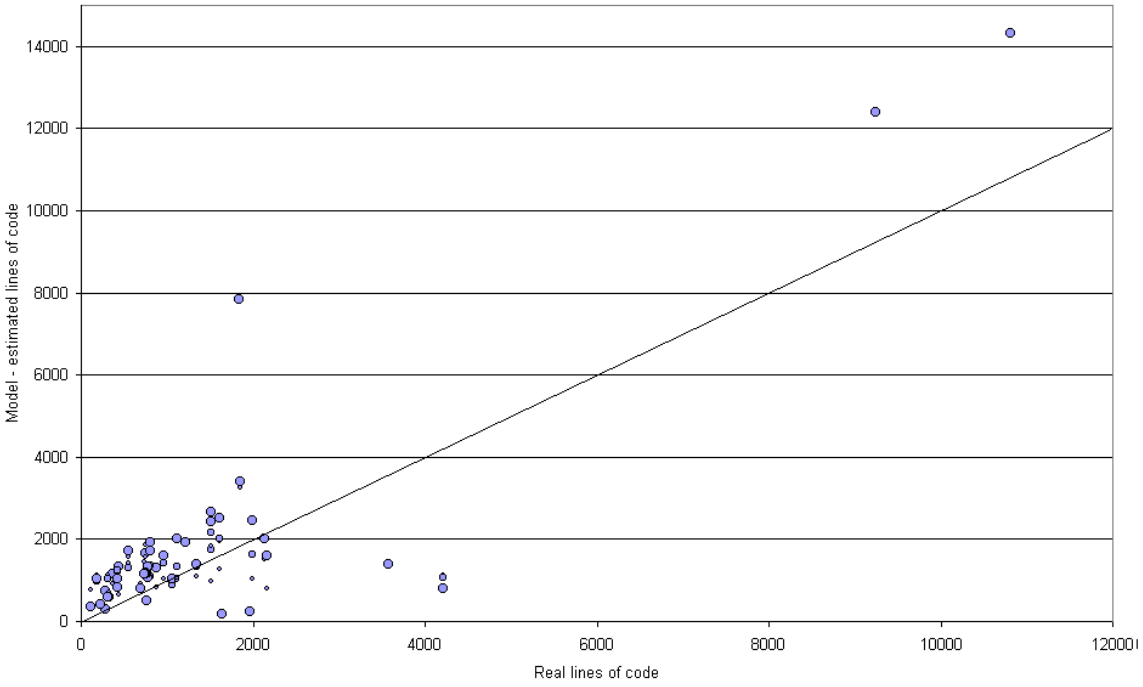
16.1.2 Detailed results

Real versus estimated project sizes in lines of code:

Project	l	n	$\sum_{i \leqslant l} L_i$	$\sum_{i \leqslant n} L_i$	$\sum_{i \leqslant l} \hat{L}_i$	$\sum_{i \leqslant n} \hat{L}_i$	$\sum_{i \leqslant n} (\hat{L}_i - L_i)$	$\frac{\sum_{i \leqslant n} \hat{L}_i}{\sum_{i \leqslant n} L_i}$
ATL18	1	1	10813	10813	14329.000	14329.000	3516.000	1.325
Am2901	1	1	286	286	307.218	307.218	21.218	1.074
gl85	1	1	1638	1638	190.299	190.299	-1447.701	0.116
i80386	1	1	757	757	500.176	500.176	-256.824	0.661

(continued on next page)

Figure 16.1: Full model aggregate: Real vs. estimated lines of code, linear axes.

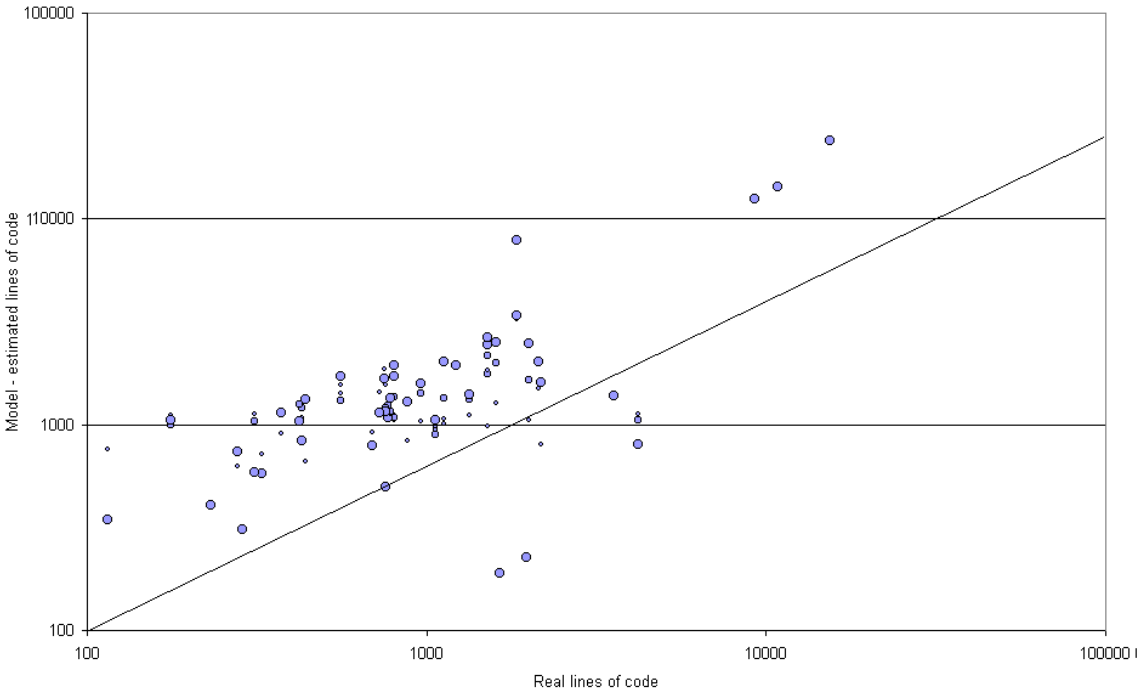


(continued from previous page)

Project	l	n	$\sum_{i\leqslant l} L_i$	$\sum_{i\leqslant n} L_i$	$\sum_{i\leqslant l} \hat{L}_i$	$\sum_{i\leqslant n} \hat{L}_i$	$\sum_{i\leqslant n} (\hat{L}_i - L_i)$	$\frac{\sum_{i\leqslant n} \hat{L}_i}{\sum_{i\leqslant n} L_i}$
SuperscalarDLX	1	1	1968	1968	226.422	226.422	-1741.578	0.115
xapp349	1	1	232	232	405.423	405.423	173.423	1.748
DLX	1	2	1112	2141	1111.893	1502.406	-638.594	0.702
DLX	2	2	2141	2141	2006.282	2006.282	-134.718	0.937
HC11	1	2	128	2166	318.567	801.193	-1364.807	0.370
HC11	2	2	2166	2166	1596.021	1596.021	-569.979	0.737
Jane	1	2	28	278	118.524	624.377	346.377	2.246
Jane	2	2	278	278	735.124	735.124	457.124	2.644
PIC16C5X	1	2	204	877	353.944	832.462	-44.538	0.949
PIC16C5X	2	2	877	877	1297.016	1297.016	420.016	1.479
rd1007	1	2	68	441	157.474	658.804	217.804	1.494
rd1007	2	2	441	441	1328.989	1328.989	887.989	3.014
xapp328	1	2	1743	1849	3110.569	3269.016	1420.016	1.768
xapp328	2	2	1849	1849	3401.480	3401.480	1552.480	1.840
xapp348	1	2	712	754	1512.188	1856.223	1102.223	2.462
xapp348	2	2	754	754	1667.472	1667.472	913.472	2.212
xapp354	1	2	539	693	444.835	912.800	219.800	1.317
xapp354	2	2	693	693	787.431	787.431	94.431	1.136
xapp357	1	2	101	115	269.985	758.251	643.251	6.593
xapp357	2	2	115	115	344.192	344.192	229.192	2.993

(continued on next page)

Figure 16.2: Full model aggregate: Real vs. estimated lines of code, log axes.

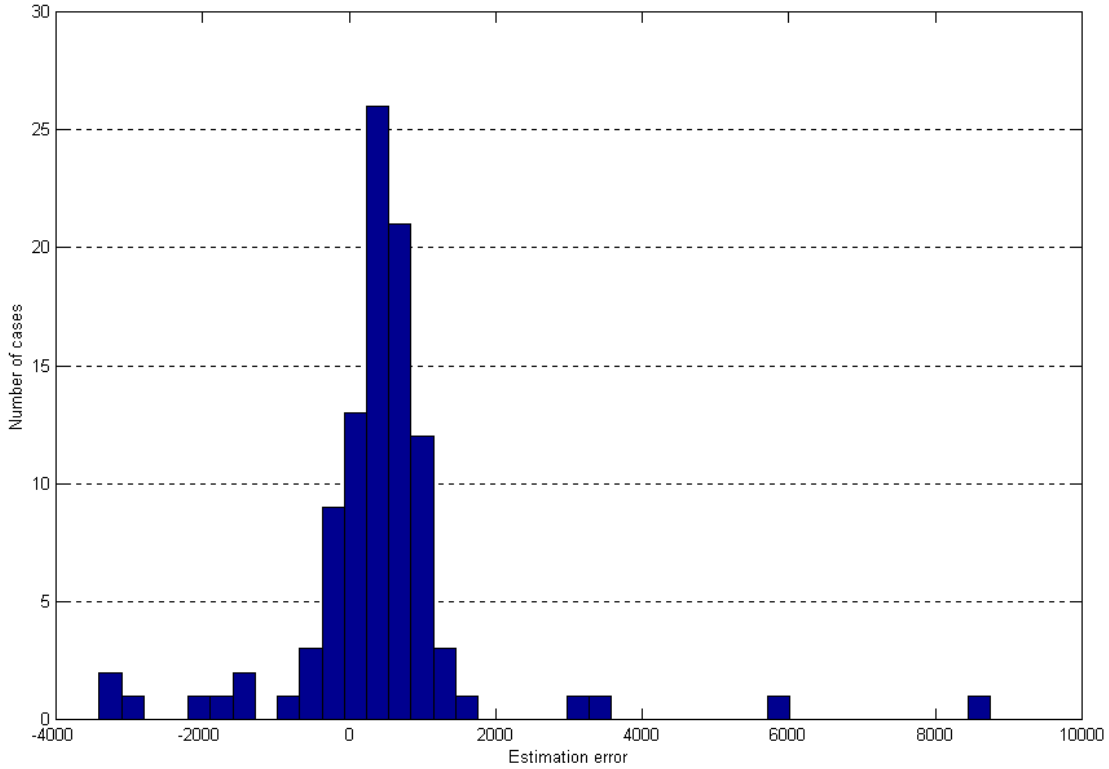


(continued from previous page)

Project	l	n	$\sum_{i\leqslant l} L_i$	$\sum_{i\leqslant n} L_i$	$\sum_{i\leqslant l} \hat{L}_i$	$\sum_{i\leqslant n} \hat{L}_i$	$\sum_{i\leqslant n} (\hat{L}_i - L_i)$	$\frac{\sum_{i\leqslant n} \hat{L}_i}{\sum_{i\leqslant n} L_i}$
xapp365	1	2	155	374	438.574	907.266	533.266	2.426
xapp365	2	2	374	374	1140.072	1140.072	766.072	3.048
xapp369	1	2	74	327	221.708	715.580	388.580	2.188
xapp369	2	2	327	327	581.221	581.221	254.221	1.777
ADC0808	1	3	113	177	160.804	1112.178	935.178	6.283
ADC0808	2	3	145	177	243.608	989.081	812.081	5.588
ADC0808	3	3	177	177	392.442	1045.718	868.718	5.908
an-XC2S-USB	1	3	119	431	253.052	1085.711	654.711	2.519
an-XC2S-USB	2	3	417	431	803.197	1212.341	781.341	2.813
an-XC2S-USB	3	3	431	431	835.161	835.161	404.161	1.938
an-XC2S-XR16	1	3	167	801	395.578	1044.818	243.818	1.304
an-XC2S-XR16	2	3	539	801	1153.508	1352.105	551.105	1.688
an-XC2S-XR16	3	3	801	801	1712.209	1712.209	911.209	2.138
ax8	1	3	362	1999	387.520	1047.130	-951.870	0.524
ax8	2	3	1534	1999	1873.600	1639.401	-359.599	0.820
ax8	3	3	1999	1999	2465.409	2465.409	466.409	1.233
Free6502	1	3	70	803	188.777	1104.152	301.152	1.375
Free6502	2	3	181	803	486.188	1085.864	282.864	1.352
Free6502	3	3	803	803	1934.759	1934.759	1131.759	2.409
i8051	1	3	11	4208	106.667	1127.711	-3080.289	0.268

(continued on next page)

Figure 16.3: Full model aggregate: Error density distribution.

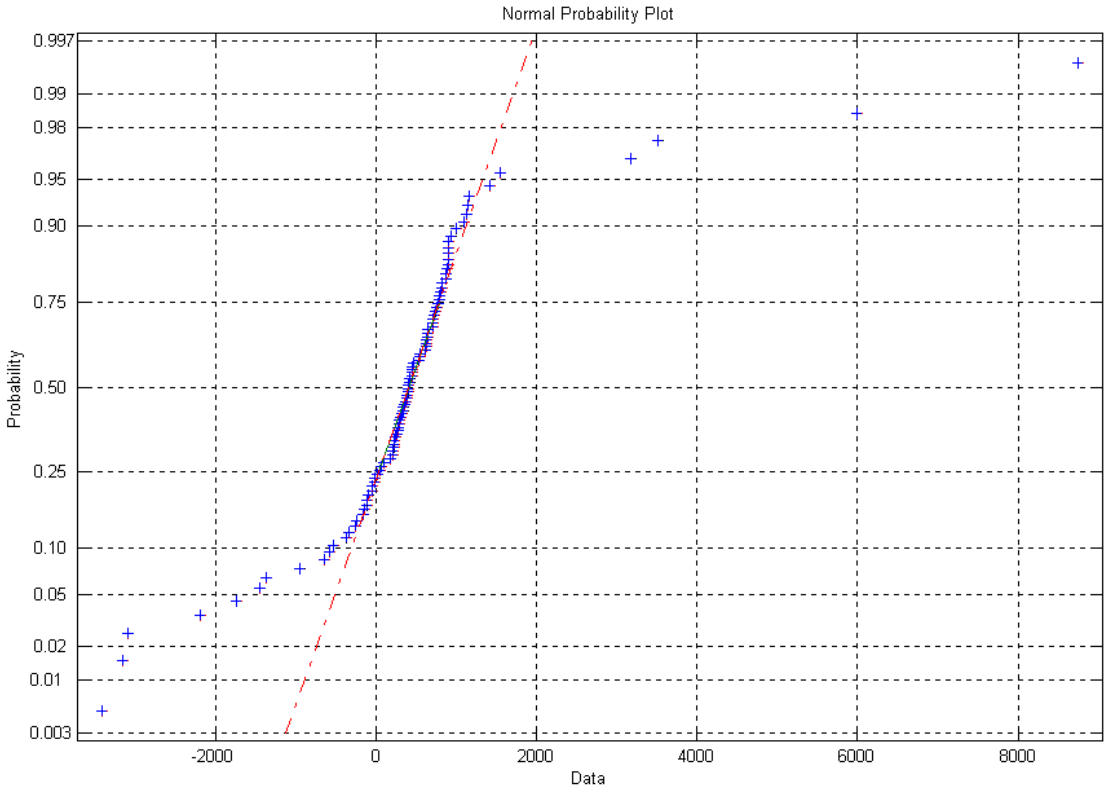


(continued from previous page)

Project	l	n	$\sum_{i \leqslant l} L_i$	$\sum_{i \leqslant n} L_i$	$\sum_{i \leqslant l} \hat{L}_i$	$\sum_{i \leqslant n} \hat{L}_i$	$\sum_{i \leqslant n} (\hat{L}_i - L_i)$	$\frac{\sum_{i \leqslant n} \hat{L}_i}{\sum_{i \leqslant n} L_i}$
i8051	2	3	127	4208	407.495	1054.467	-3153.533	0.251
i8051	3	3	4208	4208	797.023	797.023	-3410.977	0.189
ppx16	1	3	268	960	444.425	1030.803	70.803	1.074
ppx16	2	3	643	960	1305.306	1412.668	452.668	1.472
ppx16	3	3	960	960	1590.566	1590.566	630.566	1.657
T51	1	3	115	1336	185.076	1105.214	-230.786	0.827
T51	2	3	944	1336	1083.207	1324.057	-11.943	0.991
T51	3	3	1336	1336	1389.218	1389.218	53.218	1.040
xapp358	1	3	36	312	96.476	1130.635	818.635	3.624
xapp358	2	3	187	312	360.860	1035.861	723.861	3.320
xapp358	3	3	312	312	586.191	586.191	274.191	1.879
xapp363	1	3	242	1518	610.016	983.292	-534.708	0.648
xapp363	2	3	1248	1518	2162.963	1754.848	236.848	1.156
xapp363	3	3	1518	1518	2431.235	2431.235	913.235	1.602
xapp367	1	3	149	422	387.346	1047.180	625.180	2.481
xapp367	2	3	392	422	895.214	1249.053	827.053	2.960

(continued on next page)

Figure 16.4: Full model aggregate: Error cumulative distribution.



(continued from previous page)

Project	l	n	$\sum_{i \leqslant l} L_i$	$\sum_{i \leqslant n} L_i$	$\sum_{i \leqslant l} \hat{L}_i$	$\sum_{i \leqslant n} \hat{L}_i$	$\sum_{i \leqslant n} (\hat{L}_i - L_i)$	$\frac{\sum_{i \leqslant n} \hat{L}_i}{\sum_{i \leqslant n} L_i}$
xapp367	3	3	422	422	1042.508	1042.508	620.508	2.470
xapp370	1	3	171	773	324.562	1065.193	292.193	1.378
xapp370	2	3	699	773	856.957	1233.790	460.790	1.596
xapp370	3	3	773	773	1079.334	1079.334	306.334	1.396
xapp336_8	1	3	42	783	136.416	1119.175	336.175	1.429
xapp336_8	2	3	284	783	657.112	1154.057	371.057	1.474
xapp336_8	3	3	783	783	1333.720	1333.720	550.720	1.703
ans_RISC8	1	4	164	1606	301.513	1277.367	-328.633	0.795
ans_RISC8	2	4	624	1606	1045.132	1960.759	354.759	1.221
ans_RISC8	3	4	1406	1606	2270.975	2003.870	397.870	1.248
ans_RISC8	4	4	1606	1606	2509.000	2509.000	903.000	1.562
TE51	1	4	36	1060	93.326	966.888	-93.112	0.912
TE51	2	4	83	1060	224.253	944.694	-115.306	0.891
TE51	3	4	200	1060	517.092	897.887	-162.113	0.847
TE51	4	4	1060	1060	1049.136	1049.136	-10.864	0.990
xapp146	1	4	145	756	487.816	1555.209	799.209	2.057

(continued on next page)

(continued from previous page)

Project	l	n	$\sum_{i \leqslant l} L_i$	$\sum_{i \leqslant n} L_i$	$\sum_{i \leqslant l} \hat{L}_i$	$\sum_{i \leqslant n} \hat{L}_i$	$\sum_{i \leqslant n} (\hat{L}_i - L_i)$	$\frac{\sum_{i \leqslant n} \hat{L}_i}{\sum_{i \leqslant n} L_i}$
xapp146	2	4	247	756	718.513	1556.477	800.477	2.059
xapp146	3	4	705	756	1008.672	1207.873	451.873	1.598
xapp146	4	4	756	756	1156.605	1156.605	400.605	1.530
xapp333	1	4	394	1508	670.053	1826.987	318.987	1.212
xapp333	2	4	701	1508	1195.805	2147.258	639.258	1.424
xapp333	3	4	1455	1508	2515.658	2158.165	650.165	1.431
xapp333	4	4	1508	1508	2666.625	2666.625	1158.625	1.768
xapp345	1	4	139	557	495.478	1566.636	1009.636	2.813
xapp345	2	4	180	557	613.552	1426.559	869.559	2.561
xapp345	3	4	324	557	1168.484	1308.649	751.649	2.349
xapp345	4	4	557	557	1710.343	1710.343	1153.343	3.071
xapp355	1	4	116	728	410.113	1439.327	711.327	1.977
xapp355	2	4	191	728	628.669	1445.270	717.270	1.985
xapp355	3	4	649	728	918.827	1151.218	423.218	1.581
xapp355	4	4	728	728	1141.204	1141.204	413.204	1.568
xapp336	1	4	40	1121	125.537	1014.926	-106.074	0.905
xapp336	2	4	128	1121	325.714	1070.280	-50.720	0.955
xapp336	3	4	547	1121	1227.909	1346.122	225.122	1.201
xapp336	4	4	1121	1121	2021.677	2021.677	900.677	1.803
T80	1	5	174	3577	448.468			
T80	2	5	299	3577	523.380			
T80	3	5	1269	3577	709.516			
T80	4	5	2040	3577	1134.007			
T80	5	5	3577	3577	1387.871	1387.871	-2189.129	0.388
xapp356	1	5	226	1220	322.709			
xapp356	2	5	415	1220	834.763			
xapp356	3	5	772	1220	1153.271			
xapp356	4	5	1116	1220	1637.693			
xapp356	5	5	1220	1220	1936.593	1936.593	716.593	1.587
gl85struct	1	6	97	1844	280.714			
gl85struct	2	6	394	1844	2410.641			
gl85struct	3	6	1490	1844	6850.686			
gl85struct	4	6	1793	1844	7750.938			
gl85struct	5	6	1817	1844	7799.920			
gl85struct	6	6	1844	1844	7831.768	7831.768	5987.768	4.247
ERC32	1	7	347	15393	507.721			
ERC32	2	7	1072	15393	1643.329			
ERC32	3	7	4740	15393	3876.735			
ERC32	4	7	6385	15393	6450.827			
ERC32	5	7	7770	15393	9160.162			
ERC32	6	7	14238	15393	23852.070			
ERC32	7	7	15393	15393	24140.950	24140.950	8747.950	1.568
Leon	1	8	1041	9238	1365.759			
Leon	2	8	1547	9238	2188.715			
Leon	3	8	3771	9238	5464.618			
Leon	4	8	6526	9238	7724.389			

(continued on next page)

(continued from previous page)

Project	l	n	$\sum_{i \leqslant l} L_i$	$\sum_{i \leqslant n} L_i$	$\sum_{i \leqslant l} \hat{L}_i$	$\sum_{i \leqslant n} \hat{L}_i$	$\sum_{i \leqslant n} (\hat{L}_i - L_i)$	$\frac{\sum_{i \leqslant n} \hat{L}_i}{\sum_{i \leqslant n} L_i}$
Leon	5	8	7944	9238	9237.566			
Leon	6	8	8510	9238	10469.420			
Leon	7	8	9042	9238	11872.040			
Leon	8	8	9238	9238	12407.990	12407.990	3169.990	1.343

Please note that empty cells are due to models that were not tuned due to lack of proper number of cases in the tuning set.

16.2 External validation

16.2.1 Result summary

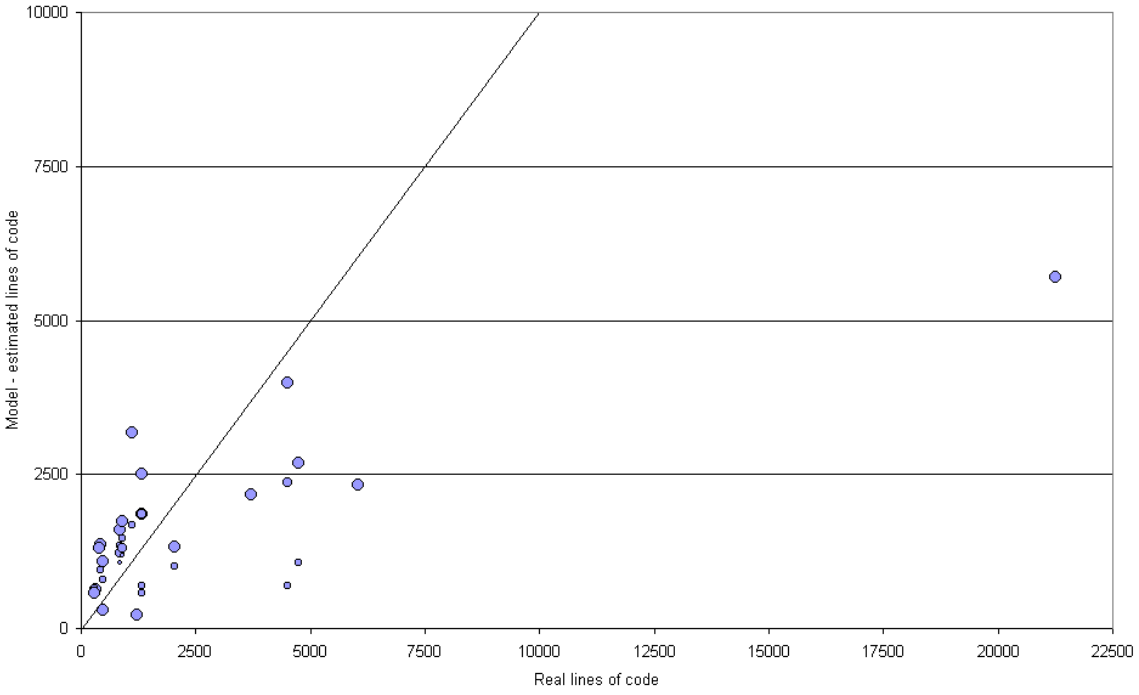
Population statistical properties and model accuracy:

	L	$\hat{L}$	$\hat{L} - L$
Average value	4016.750	2029.705	-489.674
Variance	30408828.830	9746436.318	9205966.678
Standard deviation	5514.420	3121.928	3034.134

Correlation between estimated and real values:

Correlation coefficient between L and $\hat{L}$	0.8713

Figure 16.5: Full model aggregate: Real vs. estimated lines of code, linear axes.



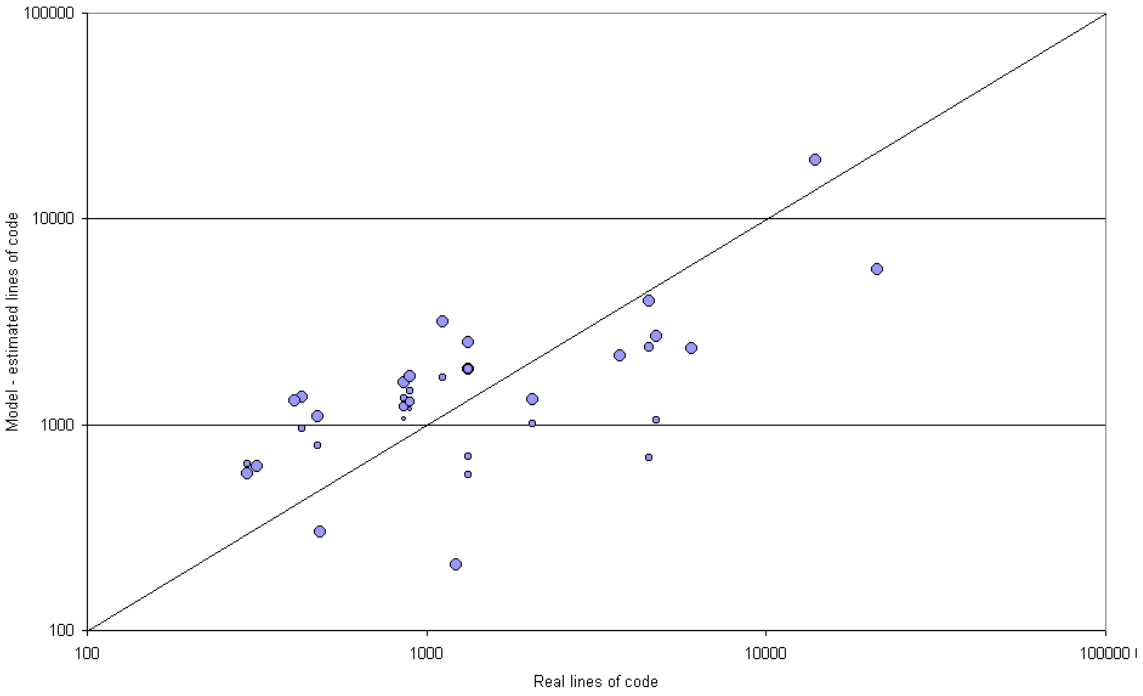
16.2.2 Detailed results

Real versus estimated project sizes in lines of code:

Project	l	n	$\sum_{i \leqslant l} L_i$	$\sum_{i \leqslant n} L_i$	$\sum_{i \leqslant l} \hat{L}_i$	$\sum_{i \leqslant n} \hat{L}_i$	$\sum_{i \leqslant n} (\hat{L}_i - L_i)$	$\frac{\sum_{i \leqslant n} \hat{L}_i}{\sum_{i \leqslant n} L_i}$
AMCC5933	1	1	486	486	301.236	301.236	-184.764	0.620

(continued on next page)

Figure 16.6: Full model aggregate: Real vs. estimated lines of code, log axes.

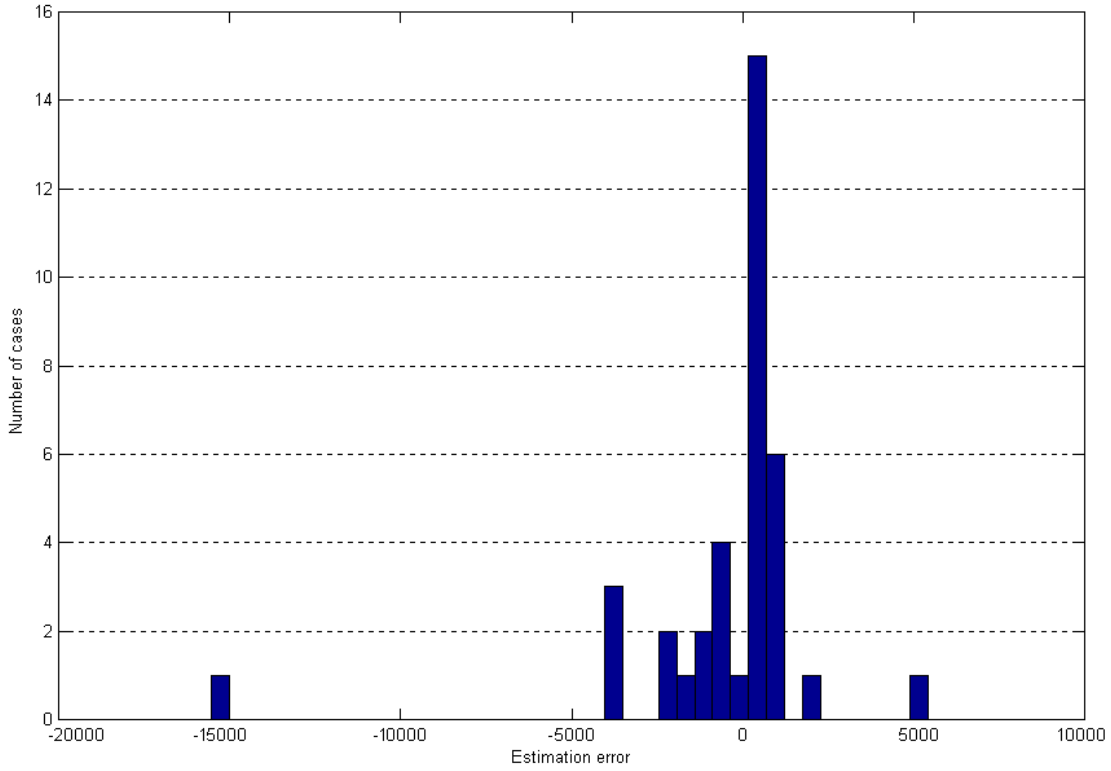


(continued from previous page)

Project	l	n	$\sum_{i\leqslant l} L_i$	$\sum_{i\leqslant n} L_i$	$\sum_{i\leqslant l} \hat{L}_i$	$\sum_{i\leqslant n} \hat{L}_i$	$\sum_{i\leqslant n} (\hat{L}_i - L_i)$	$\frac{\sum_{i\leqslant n} \hat{L}_i}{\sum_{i\leqslant n} L_i}$
DSP320VC33	1	1	3715	3715	2162.222	2162.222	-1552.778	0.582
DSP6211	1	1	6041	6041	2335.583	2335.583	-3705.417	0.387
DSP6415	1	1	21248	21248	5699.402	5699.402	-15548.598	0.268
PDP-8	1	1	316	316	625.486	625.486	309.486	1.979
STD8980	1	1	1226	1226	207.709	207.709	-1018.291	0.169
DLX2	1	2	417	1117	1315.917	1682.741	565.741	1.506
DLX2	2	2	1117	1117	3183.363	3183.363	2066.363	2.850
fw09	1	2	403	4738	608.551	1057.507	-3680.493	0.223
fw09	2	2	4738	4738	2673.698	2673.698	-2064.302	0.564
RLS	1	2	113	1329	202.305	698.430	-630.570	0.526
RLS	2	2	1329	1329	1848.701	1848.701	519.701	1.391
SPIM-Pipe	1	2	143	478	304.770	788.998	310.998	1.651
SPIM-Pipe	2	2	478	478	1090.235	1090.235	612.235	2.281
SPIM	1	2	70	297	140.448	643.755	346.755	2.168
SPIM	2	2	297	297	579.737	579.737	282.737	1.952
ZR36060	1	2	507	2045	548.149	1004.118	-1040.882	0.491
ZR36060	2	2	2045	2045	1316.070	1316.070	-728.930	0.644
HDLLib	1	3	1110	1327	2059.338	567.460	-759.540	0.428
HDLLib	2	3	1306	1327	2423.653	1858.856	531.856	1.401
HDLLib	3	3	1327	1327	2495.622	2495.622	1168.622	1.881

(continued on next page)

Figure 16.7: Full model aggregate: Error density distribution.

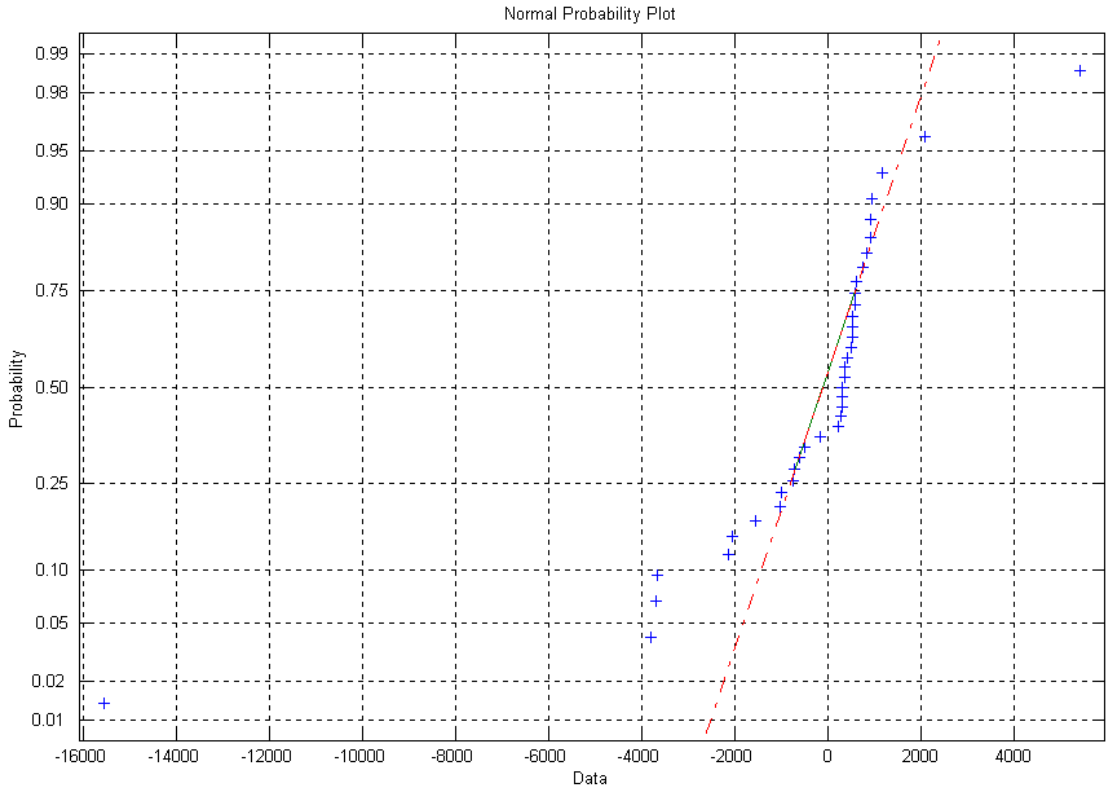


(continued from previous page)

Project	l	n	$\sum_{i \leqslant l} L_i$	$\sum_{i \leqslant n} L_i$	$\sum_{i \leqslant l} \hat{L_i}$	$\sum_{i \leqslant n} \hat{L_i}$	$\sum_{i \leqslant n} (\hat{L_i} - L_i)$	$\frac{\sum_{i \leqslant n} \hat{L_i}}{\sum_{i \leqslant n} L_i}$
IEEE1149	1	3	168	428	720.455	951.606	523.606	2.223
IEEE1149	2	3	378	428	1117.931	1337.911	909.911	3.126
IEEE1149	3	3	428	428	1354.575	1354.575	926.575	3.165
Manticore	1	3	1153	4506	1630.789	690.417	-3815.583	0.153
Manticore	2	3	4341	4506	3693.492	2365.485	-2140.515	0.525
Manticore	3	3	4506	4506	3986.181	3986.181	-519.819	0.885
RTC	1	4	60	858	158.572	1064.193	206.193	1.240
RTC	2	4	404	858	542.263	1338.319	480.319	1.560
RTC	3	4	607	858	1019.016	1214.396	356.396	1.415
RTC	4	4	858	858	1603.356	1603.356	745.356	1.869
RTC-alt	1	4	82	890	242.516	1189.382	299.382	1.336
RTC-alt	2	4	422	890	636.555	1455.031	565.031	1.635
RTC-alt	3	4	639	890	1142.182	1292.063	402.063	1.452
RTC-alt	4	4	890	890	1726.521	1726.521	836.521	1.940
LFSR	1	5	294	409	905.277			
LFSR	2	5	330	409	1016.948			

(continued on next page)

Figure 16.8: Full model aggregate: Error cumulative distribution.



(continued from previous page)

Project	l	n	$\sum_{i \leqslant l} L_i$	$\sum_{i \leqslant n} L_i$	$\sum_{i \leqslant l} \hat{L}_i$	$\sum_{i \leqslant n} \hat{L}_i$	$\sum_{i \leqslant n} (\hat{L}_i - L_i)$	$\frac{\sum_{i \leqslant n} \hat{L}_i}{\sum_{i \leqslant n} L_i}$
LFSR	3	5	353	409	1084.535			
LFSR	4	5	390	409	1243.835			
LFSR	5	5	409	409	1308.690	1308.690	899.690	3.200
leon2	1	8	1219	13993	1468.929			
leon2	2	8	1803	13993	2416.982			
leon2	3	8	4688	13993	6267.425			
leon2	4	8	8402	13993	9628.669			
leon2	5	8	10386	13993	12084.160			
leon2	6	8	12238	13993	15686.310			
leon2	7	8	13571	13993	18516.940			
leon2	8	8	13993	13993	19400.020	19400.020	5407.020	1.386

Again, empty cells are due to models that were not tuned due to lack of proper number of cases in the tuning set.

Chapter 17

Conclusions

In chapter 2 we set our primary objective: the construction of a set of models and of rules on how to apply them, able to deal with conditions of limited knowledge, capable of estimating the size of a given project with a confidence interval which becomes narrower and narrower when more information is provided and, vice versa, smoothly enlarges when the number of available input data is reduced.

After collecting an impressive amount of VHDL code, belonging to real-life, industry-strength projects, and after implementing a complete set of tools, useful to extract highly structured and complex sets of syntax-related information from that projects, we delivered a full set of models, able to estimate with a higher or lower degree of accuracy (depending on how available information correlates to data to be estimated) the size of each project, in its various conditions of knowledge.

The models we presented are obviously influenced by the actual data used to tune them, but we have the reasonable belief (sustained by external validation results) that the whole methodology would act as a good estimator in all the projects that are not extremely dissimilar from the ones used for the tuning. And since the projects belonging to the tuning set where chosen in a completely random way, and are different one from each other as much as possible, we also have the hope that our methodology can behave well in practically any project.

The methodology, considered as a whole, proves to be both accurate and robust. Accuracy is proven by a high coefficient of correlation between real and estimated data (equal to 0.8627 for internal validation cases and to 0.8713 for external validation ones) and by an acceptable error variance (standard deviation is 1400.412 lines of code for internal validation and 3034.134 lines of code for external; more than 80% of the cases have an estimation error which falls between $\pm$ the value of standard deviation). Robustness is confirmed by a really non-existent degradation of correlation and by a tolerable degradation of estimation error variance when validation is switched from internal to external sets.

A good, strongly desired, and statistically expectable phenomenon that occurred in all cases where more models were aggregated together, or used to estimate multiple instances of the similar objects, is *error compensation*: in a few words, models consisting of the composition of finer granularity sub-models usually exhibit better performances than sub-models of which they are composed. This happened when syntax object-level models are grouped and integrated together to form bunch-level models and when bunch-level models, in turn, are coalesced together to give life to SOG models.

The euphoric state of mind due to the fundamentally successful attainment of the desired goal should not move to the background a certain number of limitations from which our

methodology suffers. These consideration leaves the door wide open to future studies and equally successful enhancements and improvements: for example the method, in the way in which it is currently designed, does not deal effectively with levels known partially and in a very unbalanced way; in addition, we see the possibility of great improvements in architecture model accuracy thanks to the introduction of application-aware models.

Even current SOG models could be greatly enhanced in their estimation accuracy by using a much larger project base. Our project base, which in practice contains the vast majority of the free VHDL models that can be found on the Internet, is even superabundant as long as syntax-model tuning and validation is concerned, abundantly sufficient for bunch models, but it suffers from scarcity when it comes to SOG models.

In order to achieve better results in this last field, we believe that analysis of freely available models is not sufficient, and cooperation of large companies developing full-scale embedded design projects in VHDL is indispensable. Such an effort is certainly several orders of magnitude above the possibilities a thesis, written by an individual and without any funding.

Bibliography

[1] U. Bondi, W. Fornaciari, E. Magini, F. Salice, *Development Cost and Size Estimation Starting from High-Level Specifications*, IEEE/ACM Codes'2001 Nineth International Symposium on Hardware/Software Codesign, Copenhagen, Denmark, April 25-27, 2001. pp 86-91;

[2] Peter J. Ashenden, *Recursive and Repetitive Hardware Models in VHDL*, Technical Report TR 160/12/93/ECE, Department of Electrical & Computer Engineering, University of Cincinnati, USA;

[3] Peter J. Ashenden, *The Designer's Guide to VHDL*, Morgan Kauffmann Publishers, San Francisco, 1995, ISBN 1-55860-270-4;

[4] IEEE Standards Board, *IEEE Standard VHDL Language Reference Manual*, IEEE Standard 1076-1993, The Institute of Electrical and Electronics Engineers, Inc.;

[5] Carper Jones, Software productivity Research Inc., *What are Function Points?*, 1997 `http://www.spr.com/library/0funcmet.htm`;

[6] Carper Jones, Software productivity Research Inc., *Programming Language Table, Release 8.2*, March 1996. `http:// www.spr.com/library/0langtbl.htm`;

[7] International Software Benchmarking Standards Group Limited, *Data Disk Release 6, Demographic Data*, October 1999;

Acknowledgements

I would like to thank Professors Fabio Salice and William Fornaciari, my thesis advisors, for their unwavering support and assistance. They constantly provided guidance and support in all the knowledge domains that were required in order to accomplish the research goals.

I would like to thank Luca *Murray* Ceresoli and Alessandro *Sanna* Dester, whose friendship, companionship, comfort and encouragement rendered life at Politecnico a slightly less frightful and unbearable nightmare [1].

Finally, I would like to show my gratefulness towards Herman Hollerith, Donald E. Knuth, Leslie Lamport, Christian Schenk and Aleksander Simonic. These people taught the world how to make things that do work. And I appreciate things that **do** work, in this world of expensive, useless, vaporware things that **do not** work.

If you feel that your name should be here, feel free to send me a message at my e-mail address: `scarpaz@scarpaz.com`, I'll include you in the acknowledgements of the next degree I'll get, if any. Maybe.

[1] Note to my degree evaluation commission: please read the last statement as follows: I would like to thank Luca *Murray* Ceresoli and Alessandro *Sanna* Dester, whose friendship, companionship, comfort and encouragement rendered life at Politecnico an even more wonderful and delightful dream.