Development Cost and Size Estimation Starting from High-Level Specifications

William Fornaciari, Fabio Salice
Politecnico di Milano, D.E.I.
P.zza L.Da Vinci, 32
20133 Milano, Italy
+39-02-2399.{3504, 3636}
{fornacia, salice}@elet.polimi.it

Umberto Bondi
AlaRI, Univ. of Lugano
Via Lambertenghi, 1
Lugano, Switzerland
+41-91-9124706
bondi@alari.ch

Edi Magini
Omnitel
Italy

ABSTRACT

This paper addresses the problem of estimating cost and development effort of a system, starting from its complete or partial high-level description. In addition, some modifications to evaluate the cost-effectiveness of reusing VHDL-based designs, are presented. The proposed approach has been formalized using an approach similar to the COCOMO analysis strategy, enhanced by a project size prediction methodology based on a VHDL function point metric. The proposed design size estimation methodology has been validated through a significant benchmark, the LEON-1 microprocessor, whose VHDL description is of public domain

Keywords

Project size estimation; design reuse; VHDL; concurrent engineering; Function Point analysis.

1. INTRODUCTION

Since last decade, the importance of design reuse is steadily growing. New initiatives started, such as the VSIA [1], and third-part suppliers of IP cells are no longer confined to small market segments, but constitute a *mature* example of business-to-business cooperation among companies. Technical managers have to face with a new scenario, where the driving forces are time to market and flexibility together with the capability to keep under control the development costs. The solution of the *make-it or buy dilemma*, requires to take into account several aspects like the presence of standards, the structure of EDA design flows, direct/indirect costs, internal vs external reuse, etc, with a common problem: the necessity to predict the size of a design and to estimate the cost-effectiveness according to the number of potential reuses.

The purpose of our ongoing research is threefold: the identification of *guidelines* for VHDL-based designs to simplify their reuse, the definition of an analysis methodology providing metrics capturing the *degree of reusability* of already designed VHDL components [6] and, finally, the definition of a framework to analyze the effectiveness of design reuse as well as to predict the development effort and time. This paper addresses:

- the definition of a financial model to predict the development costs and, consequently, the potential benefits of reuse;
- the definition of a metric to predict the size of a project starting from an high-level system description;

The paper is organized as follows, Section 2 present the economical model to estimate the development effort. The adopted approach resembles the popular COCOMO model [2] developed for software projects. Section 3 presents the main impact of reuse on the cost, with particular emphasis on the concept of *equivalent size,* introduced to capture the additional overhead originated by reuse. Section 4 introduces the strategy we propose to predict the project size, which is the relevant one parameter for any planning activities. We customized the Function Point (FP) analysis to cover the peculiarity of VHDL-based designs and assessed the methodology, as reported in Section 5, by considering as an example a synthesizable 32-bits SPARC architecture [3]. Concluding remarks, and an outline of our present effort, are finally reported in section 6.

2. THE BASIS OF THE FINANCIAL MODEL

In general, to determine the impact of reuse, three models are necessary: *from scratch* associated with a design without reuse; *for reuse* to estimates the cost of reusable components and *with reuse* to consider the cost of systems partially using existing sub-components.

In addition to these models, another one associated with the potential losses due to the time-to-market (TTM) window, should be introduced, together with Return of Investment (ROI) analysis. Although the financial analyses of TTM and ROI have been considered in our overall work, due to lack of space they are not reported in this paper.

The models conceived for the software development analysis, seem to be suitable to investigate the first stages of the typical hardware design flows. In particular we considered the COCOMO [2] approach, whose main concepts are recalled in this section.

Prior to any cost trade-off, it is of paramount importance to estimate the global *development effort* (Eff), measured in person/month (pm), to realize a given system, and the time T (measured in months) to develop the project assuming a full time commitment of a properly composed group of R designers. In general, the cost C will be proportional to the effort:

$$C = K*Eff$$

To evaluate Eff, we adopted the same top-level relation of COCOMO 2.0:

$$Eff= A* S^B$$

and similarly for T:

$$T = A_2 * Eff^{B2}$$

so that:

$$R = Eff/T$$

Where the parameters are the *project size* S (Klines of code), the coefficients A, A_2 considering possible *multiplicative factors* on the effort and the *scale factors* B, B_2 accounting for economy/diseconomy originated in developing projects of different sizes. It is possible to determine the values of the parameters, according to the modality of developing the project, that is also influenced by the severity of the design constraints and the *novelty* of the application. The typical values are reported in table 1, ranging from small and simple projects (organic) to large size ones (embedded) requiring the fulfillment of stringent constraints and thus, a careful control of the development process. An extensive explanation can be found in [2].

Table 1. Values of the model parameters.

Mode	A	B	A2	B2
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

To estimate the project size, suitable metrics and reference formalism must be considered. Due to its wide diffusion for hardware projects, we selected VHDL as the representative language to quantify the size (S).
The modification on S introduced by the presence of reuse and the impact of the productivity improvements are the focus of the next section, while section four discusses the strategy to estimate S.

3. DESIGN REUSE MODELS

The rationale for reusing is that employing an available component is less expensive than designing it from scratch. However, its integration in the final design environment requires introducing some modifications. In general, the cost of reuse will be not simply proportional to the entity of the modification. In fact, it is required some extra-effort to select and understand the component to be integrated, in addition to the adaptation of the module interfaces. This section introduces some of the figures of merit and modifications to be considered in presence of reuse, with respect to the analysis presented in the previous one. Both *static* aspects and *dynamic* aspect of embedding a component in a project have to be considered.

Concerning the static issues, given a module M to be reused, it is possible to define the estimation of an *equivalent size* (S_{Mes}) to be reused, starting from the original value of S_M (size of the module M). For the following three different identified activities, the parameters and the final comprehensive expression are reported.

- **Evaluation and selection.** The activities of Assessment and Assimilation to understand the suitability of the module, are captured by the parameter AA. It is a value ranging from 0 to 8, related with the quality of the available documentation.
- **Undestanding**. The parameter CU (Code Understanding) is a percentage taking into account the increment of the code size due to the understanding of the component and the need to adapt the interfaces. Code with good documentation and high suitability with the project has CU=10% while in the opposite case CU can rise up to 50%.
  A second parameter (UNFM) captures the unfamiliarity of the designer with the module to be reused. The value ranges from 0 (completely familiar) to 1 (completely unfamiliar).
- **Modification**. The contribution AAF to the modification depends on three factors: a) the percentage of design to be modified (DM) to suit the new environment; b) the percentage of code to be modified (CM) for the same reasons of a); and c) the percentage of integration required for the modified code (IM). IM depends on the effort to embed and test the module within the overall system with respect to the case of a starting from scratch design of another module of similar size.
  $$AAF = 0.4\ DM + 0.3\ CM + 0.3\ IM$$

The value of S_{Mes} can be computed through the following formulas, in the case of $AAF \leq 0.5$ and $AAF > 0.5$, respectively. See [2] for more details.

$$S_{Mes}=0.01\ S_M[AA+AAF+(1+0.02\ CU)\ UNFM]$$

$$S_{Mes}=0.01\ S_M[AA+AAF+CU\ UNFM]$$

Using this equivalent project size, it is possible to compute the value of Eff and T for the reuse of a module M.

Some key factors in the success of a product are the costs associated with the development cycle and the time to market. An adaptive economical model should be based on the above estimations of the effort while taking into account the evolution of the designer's productivity. The goal is to predict the design cost providing quantitative estimates of the their exponential growing, together with a methodology to keep them under control via increasing levels of reuse. This *dynamic* issue cannot be addressed in this paper due to lack of space, more details can be found in [4].

4. PROJECT SIZE ESTIMATION

Unfortunately, the planning of a project requires to cope with *estimated* factors, the most important being the project size. Most of the experts, in fact, tend to underestimate (from 50% to 150%) the size of the project with catastrophic impacts on the design management.

To estimates the size of a VHDL-based project, i.e. the parameter S of the main equation of our model ($Eff = A * S^B$), we split it into a component related with purely functional aspects and the test-benches used for simulation:

$$S = S_{system} + S_{test\text{-}bench}$$

The first goal is to quantify the effort for coding the system S_{system} using a high-level formalism. A popular metric among the designers is the *Thousand Lines Of Code* (KLOC or LOC) of the specification. However, many criticism can be raised, since VHDL is inherently parallel and the different statements vary dramatically is expressiveness and complexity.

The estimation of LOC requires a well-structured and modular project to obtain reliable values. The number of lines can be determined analyzing the different contributions emerging from the architecture description, which basically are: *port (IO), signal, concurrent statements, package and library.*

The analysis can be performed trough direct measurements as well as by following the proposed high-level estimation strategy, where the requirement of achieving a fine grain analysis of the system description can be overcome.

4.1 Direct Analysis

Processes and Components are the *cornerstone* of the analysis, we assume that in a well structured project, the set of entities correspond to a graph where components are nodes and processes are leaves.

The processes typically contain the algorithmic part of the project and their sizes are strongly influenced by the number of considered signals (not only those of the sensitivity list); the type of data (structured, scalar, …) also influences the estimates. From our analysis, the estimated trend of LOC for a process is a parabolic function of the number of input/output signals, called *grade* of the process (more formally, the sum between the *outdegree* and the *indegree* of the process graph). Vectors and signals account for one, while for the records only the fields effectively manipulated by the process are considered.

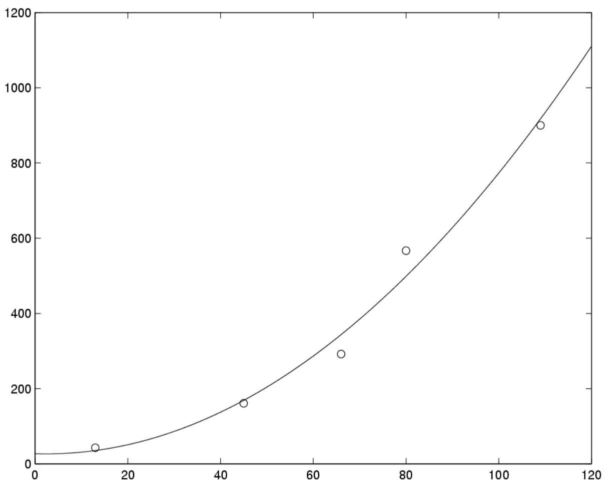

Figure 1. LOC against *grade* of a process: interpolation curve.

The fitting parabolic function, obtained via the least square method applied on the data computed on the LEON-1 VHDL description [3], is reported in figure 1.

Concerning the components, the number of lines will be of the some order of magnitude of the number of considered signal, listed in the component interface.

4.2 Function Point VHDL

The second strategy is to indirectly measure the LOC. This approach can be applied to the case of the analysis of new projects. The original idea is borrowed from the Software Engineering area [5], we properly interpreted, extended and tuned the methodology to cope with the characteristics of hardware designs. It is based on a structured, but not necessarily, detailed view of the project. From the description, some functional classes will be identified and associated with a weight depending on their complexity. In a second phase, these weights are converted in KLOC, that is the value chosen to quantify S.

At a *conceptual* level, in particular, the representation of functionality is not unique. We assume to deal with specifications that are *complete, consistent, rigorous* and *feasible*.

The elements to be developed during the project to implement the desired functionality are related to the activity of *acquisition* of information, *processing* (sequential or combinatory), *memory access,* and *emitting* of information. For the VHDL, each element of the specification falls in one of these *functional categories*: p*rimary inputs, primary outputs, basic blocks* and *internal signals*. For each of them, a contribution (FP_{VHDL}) is assigned depending on the complexity, according to the categorization of table 2.

Table 2. The weights associated with the functional categories.

Functional Category	Very Low	Low	Avg	High	Very High	Ultra High
primary inputs	1	3	4	6	10	14
primary outputs	2	4	5	7	11	15
basic blocks	2	8	20	35	52	76
internal signals	2	8	20	35	52	76

The analysis of an element of the description is composed of two steps. Initially, according to table 2, a weight is associated with each of the elements composing a given unit. The weights are function of characteristics and parameters deriving from the specification. Hence, given a functional unit *k* containing m_k elements whose weight is $w_{k,i}$, it is possible to compute:

$$FP_k = \frac{\sum_{i=1}^{m_k} w_{k,i}}{m_k} \qquad \text{(eq. 1)}$$

In other terms, the contribution is the average of those of its components. Finally, for the entire system, the associated FP_{VHDL} becomes:

$$FP_{VHDL} = \sum_{i=1}^{4} FP_k \qquad \text{(eq. 2)}$$

Let us describe in more detail how to analyze the different functional categories.

The *Primary Inputs* category contains the inputs from the specification surroundings the system, both for control (e.g. reset, clock) and data acquisition. The complexity (see table 3) depends on the:

- *Homogeneity* of data constituting the input, i.e. their nature (record, vector, scalar) and size.
- Number of involved blocks, i.e. its contribution to the internal communication of the systems after an event implying an updating of signals.

Table 3. Complexity of the *Primary Inputs.*

Involved Components	Homogeneity of data						
	1-2	3-4	5-11	12-19	20-44	45-84	84-
1	VL	VL	L	L	A	H	H
2	VL	L	A	A	H	H	VH
3-4	L	A	A	A	H	H	VH
5-6	L	A	A	H	H	VH	VH
7-	A	H	H	VH	VH	VH	UH

In a similar way, although with different values with respect to those of table 3, the complexity of *Primary Outputs* is calculated.

The functional class *Internal Signals*, involves the homogeneity of data and the control information exchanged between components and subsystems. The correspondence between the number of homogeneous data constituting an internal signal and its complexity is depicted in table 4.

Table 4. Complexity of the *Internal Signals.*

Homogeneity of data					
1-3	4-6	7-16	17-35	36-73	74-
VL	L	A	H	VH	UH

Eventually, the *Basic Blocks* are instances of blocks identifiable from the specification. It is important to identify all the functionalities of the system to be mapped, during the final VHDL coding phase, into entities with component and/or processes. Moreover, it has to be considered as a basic block also the functionality corresponding to the test-benches. The complexity of a basic block depends on the number of homogeneous data involved. In particular, if an identified basic block will be realized as a component, the estimated number of lines is equal to the number of homogeneous data processes by the component; if the basic block will be implemented as a process, the value of FP is computed by using the eq.1. Table 5 summarizes the level of complexity with respect to the number of homogeneous data arriving to the process.

Concerning the relation existing between FP_{VHDL} and LOC, some literature studies [5] suggest the coefficients to convert the LOC for different languages.

Table 5. Complexity of Basic Blocks to be implemented as processes (leaves of the hierarchy).

Homogeneous data entering into the basic block					
1-29	30-56	57-85	85-105	105-125	125-
VL	L	A	H	VH	UH

For VHDL we found sound the following conversion factors, used for local and global estimations. For a single node:

$$LOC = 19 * FP_{VHDL}$$

If the detail of the lower levels of the hierarchy are not accessible (or are unknown), the number of lines of code for a given node is estimated by introducing a *tuning factor* (L*ev*), that is the level where the node is located:

$$LOC = 19 * FP_{VHDL} * Lev$$

The value *Lev* is computed by considering a "levelized" procedure where the bottom entity assumes the minimum value “1” while the top level assumes the maximum estimated value. Hence, the value of LOC is function of the considered level used to estimate the global cost of a given module (or sub-module) when its final decomposition is unknown but its level is at least predictable. As an example, let us consider figure 2 where C can be further decomposed in sub-modules. The estimated cost of the entire project (in term of LOC) is obtained by summing up the local cost of module A, the local cost of module B and the global cost of module C, that is:

$$LOC=19*(FP_{VHDL-A} + FP_{VHDL-B} + FP_{VHDL-C} * 3)$$

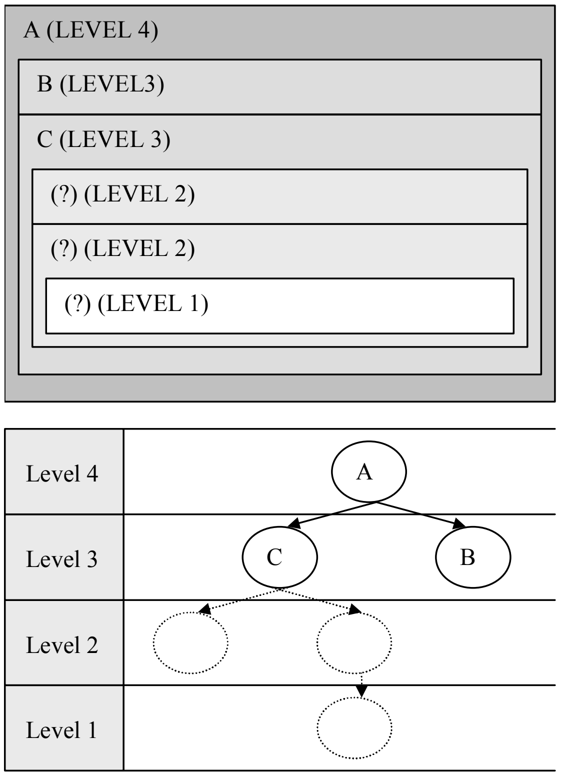

Figure 2. Example of *Lev* estimation and LOC computation.

5. Experimental Results

To provide a reliable validation of our methodology for metric estimation, we considered a real-world complex VHDL benchmark: the LEON-1 microprocessor. It implements a 32-bits SPARC V8 architecture, designed for embedded applications, with separate data and instruction caches, 32 bits memory bus, interrupt controller, two 24-bits timers, two UARTs, a power down function and a watchdog. The block diagram is reported in figure 3, the corresponding specification, composed of 20 VHDL files, can be found in [3] together with a detailed description of the functional behavior.

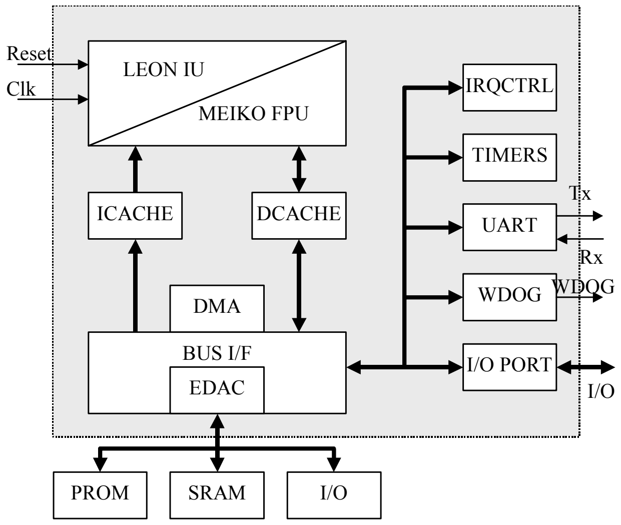

Figure 3. Block diagram of the LEON-1 CPU (gray box).

Part of this specification files have been used to validate the direct estimation of the LOC_{VHDL} and for the tuning of the interpolation function reported in figure 1. Concerning the FP analysis, let us consider, as an illustrative example, the UART subsystem depicted in Figure 4.

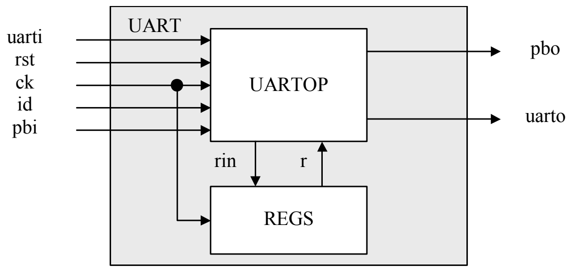

Figure 4. Block diagram of the UART.

Four primary inputs involve UARTOP while one primary input is connected to both subcomponents. All primary outputs but pbi have complexity VL; the primary input pbi is composed by 6 homogeneous data, so that its complexity is L. The Primary Outputs include pbo (2 homogeneous data) and uarto (6 homogeneous data); the complexity is VL for both signals. Internal Signals contains r and rin; both are composed by 34 homogeneous data so that their complexity is H. Concerning the basic blocks UARTOP and REGS, they are at the bottom level of the hierarchy and they are implemented as processes; UARTOP involves 45 homogeneous data (complexity L) while REGS involves 3 homogeneous data (complexity VL). See Table 6 for a quantitative computation of the UART FP.

The predictive analysis to estimate S, has been carried out for the entire LEON-1 description, whose hierarchical structure is depicted in figure 5.

Table 6 Complexity, FP_{VHDL} and its conversion to LOC_{VHDL} for the UART of Figure 3.

Category	Complexity	Weight
Primary inputs	$\frac{1(VL=3)+1(VL)+1(VL)+1(L=3)+1(A)}{5}$	1.4
Primary outputs	$\frac{1(VL=2)+1(VL)}{2}$	2
Internal signals	$\frac{1(H=10)+1H}{2}$	10
Basic Blocks	2 block (processes) 1(L=8) + 1(VL=2)	10
	Total number of FP_{VHDL}	**23,4**

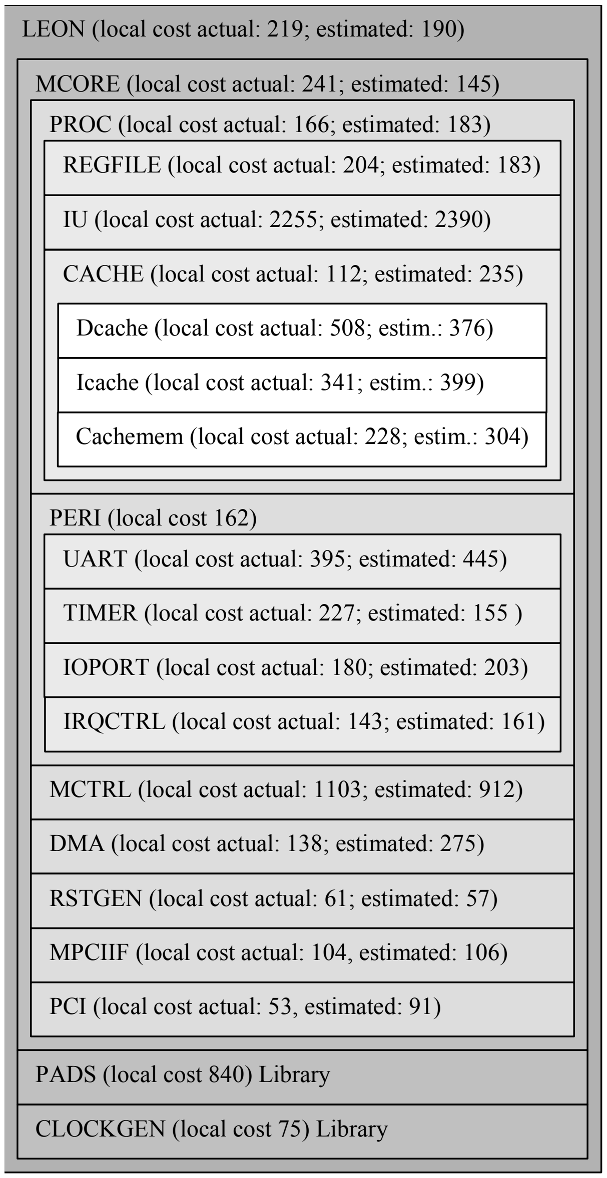
LEON (local cost actual: 219; estimated: 190)
MCORE (local cost actual: 241; estimated: 145)
PROC (local cost actual: 166; estimated: 183)
REGFILE (local cost actual: 204; estimated: 183)
IU (local cost actual: 2255; estimated: 2390)
CACHE (local cost actual: 112; estimated: 235)
Dcache (local cost actual: 508; estim.: 376)
Icache (local cost actual: 341; estim.: 399)
Cachemem (local cost actual: 228; estim.: 304)
PERI (local cost 162)
UART (local cost actual: 395; estimated: 445)
TIMER (local cost actual: 227; estimated: 155)
IOPORT (local cost actual: 180; estimated: 203)
IRQCTRL (local cost actual: 143; estimated: 161)
MCTRL (local cost actual: 1103; estimated: 912)
DMA (local cost actual: 138; estimated: 275)
RSTGEN (local cost actual: 61; estimated: 57)
MPCIIF (local cost actual: 104, estimated: 106)
PCI (local cost actual: 53, estimated: 91)
PADS (local cost 840) Library
CLOCKGEN (local cost 75) Library

Figure 5. Hierarchical decomposition of LEON-1.

The results and a comparison between estimated and actual values, for all the LEON-1 files, are summarized in table 7 (local) and table 8 (global). The data in the former one are single actual cost (KLOCs) including both BODY and ARCHITECTURE without comments and with a single statement per line.

Table 7. Experimental data obtained analyzing the LEON-1 description: local costs.

Architecture	**Local cost (actual)**	**Local cost (estimated)**	**Local error**
LEON	219	190	-13%
MCORE	241	145	-39%
PROC	166	183	10%
CACHE	112	235	110%
Dcache	508	376	38%
Icache	341	399	15%
Cachemem	228	304	5%
IU	2255	2390	0.5%
REGFILE	204	183	-6%
PERI	162	125	23%
UART	395	445	13%
TIMER	227	155	-31%
IOPORT	180	203	12%
IRQCTRL	143	161	12%
MCTRL	1103	912	17%
DMA	138	275	99%
RSTGEN	61	57	6%
MPCIIF	104	106	2%
PCI	53	91	71%
CLOCKGEN	75	114	52%
TOTAL	**6915**	**7049**	**2%**
PADS	840	library	

Table 8 contains the global estimated costs, expressed in LOC, representing the prediction of the number of lines of VHDL constituting the portion of the project included in the considered module, having the knowledge on the presumed hierarchical level but no details on the inner levels.

Table 8. Experimental data obtained analyzing the LEON-1 description: global costs.

Architecture	**Level (estim.)**	**Global cost (actual)**	**Global cost (estim.)**	**Global Error**
LEON	5	7755	11030	42%
MCORE	4	6621	6384	3.5%
PROC	3	3814	3738	2%
CACHE	2	1189	1530	28.5%
PERI	3	1107	969	-12%

6. CONCLUDING REMARKS

The paper addressed the problem modeling the development effort and time of hardware projects, to be used possibly to tradeoff between designing from scratch or reusing existing IP cells.

The analysis considered two cross-related aspects of the problem:

- The definition of a model to estimate the total effort and, thus, the required amount of resources.
- The possibility to predict in a reliable manner, possibly at the system-level, the size of the project assuming a VHDL-based realization.

The latter point is particularly important since it is the cornerstone of any planning analysis and its accuracy is very critical.

The methodology has been stressed considering a real large-size benchmark, the LEON-1 microprocessor. The obtained results are encouraging, since the average accuracy of the local estimates is around 20% with a variance of 15%. Note that the summation of local costs gives an estimated value with accuracy only around 2%, since errors tend to compensate.

Furthermore, trying to predict the size starting only from the top-level view of the project, hence with a reasonable uncertain, the estimated cost differs from the actual one of less than 40%.

For the considered LEON-1 system, the obtained effort, considering the top-level estimated cost is 255 pm, the development time T evaluates 14 months with an equivalent team composed of 17 designers. T includes all the front-end activities like simulation, testing, documentation, etc. The industrial designers we interviewed have considered these values reasonable.

Work is in progress to integrate the proposed financial analysis methodology with the metrics we developed to quantify the level of reusability of VHDL specifications, preliminarily described in [6].

7. REFERENCES

[1] Virtual Socket Interface Alliance, http://www.vsia.org.

[2] COCOMO 2.0 Model Definition man., ver 1.2, 1997.

[3] J.Gaisler, LEON-1 VHDL Model Description, TOS-ESD/JG/501, N.2.1, European Space Agency, May 2000.

[4] M.Keating, A financial model for design reuse, http://www.synopsys.com, 1998.

[5] Experiece Function Point Analysis, http:// www.sttf.fi/htnl/ant_expefpa.html, 1999.

[6] W.Fornaciari, S.Minonne, F.Salice, M.Vincenzi, Lambda-Block Analysis of VHDL for Design Reuse, In Virtual Components Design Reuse, Kluwer Acadamic Publisher, 2001. Chapter 7. pp 95-103.