Interaction in Language Based System Level Design Using an Advanced Compiler Generator Environment*

Ioannis Poulakis, George Economakos and Panayiotis Tsanakas
National Technical University of Athens
Department of Electrical and Computer Engineering
Zographou Campus, GR-15773 Athens, Greece
poulakis@cslab.ece.ntua.gr

Abstract

Computer-aided synthesis of digital circuits from behavioral level specifications offers an effective way to deal with the increasing complexity of digital hardware design. A high-level synthesis tool transforms an abstract algorithmic description into a detailed register transfer level implementation. Even though considerable research has taken place, regarding high-level synthesis, practical implementations are just emerging. This happens due to the fact that designers demand interaction at both the specification and implementation level. This paper describes an efficient implementation of an original idea, for the design of a grammar based interactive design environment, which allows designers supplement high-level synthesis optimizations and set constraints among the operators in the textual algorithmic description to meet their implementation preferences. The suggested methodology raises the feasibility for high level design space exploration by enabling synthesis results to be directly modifiable by the user.

1. Introduction

High-level synthesis (HLS) [7, 12, 13, 17], has been proven very effective in fast prototyping of VLSI circuits. HLS accepts a behavioral specification of a digital system, along with a set of constraints, and finds a structure that implements the given behavior while satisfying constraints. The behavior is usually described as an algorithm, similar to a conventional programming language description. The output structure is a register transfer level implementation, which includes a data-path and a control unit. The data-path consists of all required functional units and their interconnections. Control activates components of the data-path to realize the required behavior. The objective of synthesis is to find a structure that meets given constraints while optimizing cost functions, like the required hardware resources or the power consumed. This structural specification is technology independent and can be utilized in different circumstances and consequently, design reusability is supported.

Recently, HLS has become a hot research topic; nevertheless, designers still prefer semi-automated (with automatic optimizations starting at lower abstraction levels) or even manual methodologies. This happens because a fully automated design offers little interaction, so it is hard to verify that all constraints are indeed respected. A more feasible approach is what we call *Interactive High-Level Synthesis* (IHLS), where users can control the design process, observe the effects of design decisions and manually override synthesis algorithms at will. Recent literature presents increasing interest in interactive methodologies and synthesis systems [1, 18, 9, 5].

On the other hand, design complexity requires abstract notations to describe hardware, like *Hardware Description Languages* (HDLs) [2, 15]. HDLs are nowadays widespread among applications for the specification, verification and synthesis of digital hardware. Consequently, traditional programming language methods and tools have been applied for hardware design [6, 14, 10], their main advantages being design productivity, technology independence and greater potential for design reusability.

This paper proposes an interactive synthesis methodology, using *Attribute Grammars* (AGs) [11] as a formal unified framework over which HLS is performed, following and extending the work presented in [3, 4] and [5]. The basic idea is that the parse tree of a behavioral, high-level, language based hardware specification, is used to describe both behavioral and structural details of the design process. Attributes are used to transform behavior into structure, im-

*This work was partially funded by the Ministry of Development, General Secretariat for Research and Technology, project PENED $99E\Delta 521$

plementing widely used HLS heuristics. Furthermore, attributes are also used to support user interaction, by allowing the user to pass constraints and modifications to the automated HLS transformations, along with the algorithmic behavioral description (attributed-behavioral specifications [1]). The corresponding implementation is based on an efficient and advanced AG based compiler generator environment [8, 16]. Though HDL structural output descriptions, the proposed design environment connects HLS with lower level synthesis tools and so, complete implementation details can be obtained and evaluated repeatedly , until all constrains are met. Overall, our approach is a novel idea for IHLS, based on the declarative and modular notations of the AG computational model.

2. Related Research

The proposed methodology combines research in two recent hot design automation topics, language based design and design interaction.

Language based design is a consequence of the increased design complexity of modern digital circuits. This has motivated researchers to move towards higher abstraction levels and utilize language based processors [6]. As far as synthesis is concerned, language based processors followed the idea that by expressing the design in a higher level formalism, which is bound to a specific computational model, one can use an executable version of the model to perform synthesis. As the design specification passes through the model, constructs (crucial for synthesis) are recognized and special procedures are conducted [14]. An integrated environment for silicon compilation, was the syntax-directed system developed by Keutzer et al [10], which was based on the same ideas with our current work. However, their effort was aimed at a lower level of abstraction. It faced the problem of register-transfer level realization, that is, the optimal transformation of an FSMD architecture into netlists of digital gates, and used more than one language processors. The main disadvantage of most traditional approaches was that they used complicated design entry specifications and that they presented new heuristics which were not compared with previous ones and not tested in practical examples, with resource or power constraints.

Design interaction is a requirement for high-level design environments. From the viewpoint of an experienced human designer, "black-box" systems, where all tasks are performed automatically, are unacceptable due to the inaccessible workings of the tools and algorithms involved. Work on this area has been focused on unifying structures, which can support a design through different abstraction levels and views (behavioral, dataflow, structural, physical) [18, 9]. In [1], a methodology for interactive specification of language based designs, called the *attributed-behavioral* specification, has been presented. The work of [5] combined a unifying structure, the parse tree of the behavioral input specification, with the attributed-behavioral specification method.

This paper implements user interaction in the AGENDA [3] AG driven HLS environment, under an efficient automated compiler generator environment [8, 16]. It follows the work of [5] and the idea of the attributed-behavioral specification, where a system is described as a couple of an algorithmic specification and a set of conditions that must hold for any implementation. Taking advantage of the AGENDA methodology, where, attributes attached to the parse tree of the algorithmic specification hold implementation details, it allows user interaction with those attributes that correspond to the conditions of the attributed-behavioral input specification. By iterating over this process, all serial/parallel tradeoffs in behavioral modeling are handled in a unified, formal environment.

3. Proposed Methodology

An attributed-behavior specification consists of a textual algorithmic description accompanied by a set of conditions (attributes) that must hold in any implementation of the description. Our methodology consists of defining and manipulating attributes that offer user interaction through HLS. Consider scheduling, one of the basic problems of HLS. We have defined five attributes, five operator relationships in the temporal domain, called step, c_step, delay, group and distance. All attributes refer to an operator of the algorithmic description and are denoted themselves like referential operators. The hardware interpretations of these attributes are:

- **step:** Assigns the control step in which the referential operation will be executed. If it is impossible for the operation to be executed on that clock cycle, then a new value is estimated according to the scheduling algorithm. For example, if we want operation o_i to be executed in the n^{th} control step we add the token $[n]$ in front of o_i in the behavioral description (syntax: $[n]o_i$).

- **c_step:** Assigns the control step in which the referential operation will be executed. There is no further check for the applicability of the desired control step. The designer must be certain about the effect of this attribute. For example, if we want operation o_i to be executed in the n^{th} control step we add the token $[\&n]$ in front of o_i in the behavioral description (syntax: $[\&n]o_i$).

- **delay:** Assigns the number of control steps that the referential operation will be delayed. The control step the operation will be executed is the one estimated by the scheduling algorithm, increased by the value of delay. For example, if we want operation o_i to be delayed

by n control steps we add the token $[\wedge n]$ in front of o_i in the behavioral description (syntax: $[\wedge n]o_i$).

- **group:** With this attribute, operations in the algorithmic description can be partitioned in groups. We can have more than one groups but each operator may belong only to one. All operators in a group will be executed at the same control step. There are two variations of the group attribute. The user can either group the operators and define the control step on which they will be executed with a additional c_step attribute, or let the system choose the latest among the control steps of all operators in the group, and schedule them then. For example, if we want operation o_i to be included in group x_j we add the token $[gx_j]$ before o_i in the behavioral description (syntax: $[gx_j]o_i$).

- **distance:** This is a special case of the group attribute. It assigns minimum, maximum or exact distance, in clock cycles, between execution times of the operators in a group. In other words, operators within a group will not be scheduled in the same control step but, in a time slot whose width is given by the distance attribute. In every case, operator dependencies are examined to avoid violations. For example, if we want operation o_i, which is part of group x_j, to be scheduled within a time slot of at most n control steps width, with respect to all other operators in x_j, we add the token $[dgx_j < 3]$ before o_i in the behavioral description (syntax $[dgx_j(<,>,=)n]o_i$).

All attributes are used to describe the designer's implementation preferences and thus, have a direct impact on the operation of the current scheduling algorithm (CSA).

The implementation of the proposed methodology supplements the AGENDA formal, AG based, hardware compilation environment. AGENDA uses the parse tree of the given behavioral description as a unifying structure that holds both behavioral and structural system details. Attributes are used to transform behavior into structure, implementing widely used HLS heuristics through attribute evaluation rules. These attributes, which are values associated with non-terminal symbols of the underlying grammar, form an AG. Passing this AG over and AG evaluator generator [8, 16], produces an AG based hardware compiler, which implements the selected HLS heuristics. On the other hand, the attributes of an attributed-behavioral specification describe the conditions that must hold for any implementation. However, using AGs, such conditions can be attached to non-terminal symbols of the grammar and thus, can be expressed with attributes of the AG. Moreover, we can say that these latter attributes play in fact the role of initializations for the other, HLS transformation attributes. For example, consider the following operation parsing grammar rule, with a step relationship.

$$operation \rightarrow operand_1\ [n]operator\ operand_2 \qquad (1)$$

Assuming ASAP scheduling (for simplicity), AGENDA normally uses a synthesized attribute s_cs (whose value depends on values of successor nodes in the parse tree), denoted as $X.s\_cs$, where X is a non-terminal grammar symbol, to pass scheduling information from inputs to outputs, with the following evaluation rule, in any syntactic rule similar to (1).

$$\begin{aligned} operation.s\_cs &= MAX(operand_1.s\_cs, \\ &\quad operand_2.s\_cs) + 1 \end{aligned}$$

Considering the step condition, another synthesized attribute, called $step$, is used and the following modifications are made.

$$\begin{aligned} operation.CSA &= MAX(operand1.s\_cs, \\ &\quad operand2.s\_cs) + 1 \\ operation.step &= n \\ operation.s\_cs &= valid(operation.step)? \\ &\quad operation.step : operation.CSA \end{aligned}$$

As it can be seen, there exist a straightforward correspondence between the step conditional attribute of the attributed-behavioral input specification and the $step$ attribute of the underlying AG. In fact, constraint attributes can be regarded as the initial values (under conditions) of the scheduling attribute s_cs.

The same applies in the case of the delay relationship. The following is an operation parsing rule with a delay attribute.

$$operation \rightarrow operand_1\ [\wedge n]operator\ operand_2 \qquad (2)$$

Scheduling is performed with the following evaluation rule.

$$\begin{aligned} operation.delay &= n \\ operation.s\_cs &= operation.delay + operation.CSA \end{aligned}$$

Similar is the case of the c_step attribute.

$$operation \rightarrow operand_1\ [\&n]operator\ operand_2 \qquad (3)$$

$$\begin{aligned} operation.step &= n \\ operation.s\_cs &= operation.step \end{aligned}$$

In the case of group attribute, there exist two alternative uses, as reported above. If group is combined with step or c_step, then scheduling is performed as follows:

$$operation \rightarrow operand_1\ [\&n_1][gn_2]operator\ operand_2 \quad (4)$$

$$\begin{aligned} operation.step &= n_1 \\ operation.s\_group\_cs &= operation.step \\ operation.group &= n_2 \\ & ins\_group(operation.s\_group\_cs, \\ & \ operation.group, \\ & \ operation.flag) \\ operation.s\_cs &= get\_group\_cs(operation.group, \\ & \ operation.i\_group\_cs) \end{aligned}$$

If group is not combined with other conditions, then scheduling is performed as follows:

$$operation \rightarrow operand_1\ [gn]operator\ operand_2 \quad (5)$$

$$\begin{aligned} operation.step &= operation.CSA \\ operation.s\_group\_cs &= operation.step \\ operation.group &= n \\ & ins\_group(operation.s\_group\_cs, \\ & \ operation.group, \\ & \ operation.flag) \\ operation.s\_cs &= get\_group\_cs(operation.group, \\ & \ operation.i\_group\_cs) \end{aligned}$$

The difference between (4) and (5), is that in (4), group is combined with c_step and the value of c_step overrides the value calculated by the CSA. In both (4) and (5), two special functions are used, $ins\_group()$ and $get\_group\_cs()$.

- $ins\_group()$ is used to relate the estimated control step of the current operator with all other members of the same group. The $operation.flag$ attribute is used to specify weather rule (4) or (5) is used. If $operation.flag$ is set, then n is the value of the group's control step, otherwise the maximum control step of all members of the group is used.
- $get\_group\_cs()$ is used to retrieve the final value for the control step of the group.

Considering the above heuristics, it is obvious that control step evaluation in an operator parsing rule depends not only on local attributes of inputs $operand_1$ and $operand_2$. In (4), the control step of all the group members is set to n_1. In (5), as we have mentioned earlier, the control step for each of the group members is the maximum of the control step values calculated by the CSA, for all group members. To respect this relationship, before we decide the control step of the group (when $get\_group\_cs()$ is called), we have to know the maximum control step value. To accomplish this we have introduced two more attributes, one synthesized - s_group - and one inherited - i_group (whose value depends on values of predecessor nodes in the parse tree). As we can see in the parse tree example of figure 1, attribute s_group moves upwards in the parse tree from each grouped operator towards the root, carrying information about the control step value calculated using the CSA. When all s_group attributes reach the root, they are inserted in i_group, which carries all values downward, towards the rest nodes of the same group. When i_group reaches these nodes, all s_group attributes have been calculated and so, the maximum control step can be calculated.

Finally, in the case of the distance attribute, scheduling is performed as follows:

$$operation \rightarrow operand_1\ [dgn\ rel\ dis]operator\ operand_2 \quad (6)$$

$$\begin{aligned} operation.step &= operation.CSA \\ operation.s\_group\_cs &= operation.step \\ operation.group &= n \\ & ins\_group(operation.s\_group\_cs, \\ & \ operation.group, \\ & \ operation.flag) \\ operation.s\_cs &= (rel == '<')? \\ & \ min\_distance(operation.dis, \\ & \ step, operation.i\_group) : \\ & (rel == '>')? \\ & \ max\_distance(operation.dis, \\ & \ step, operation.i\_group) : \\ & (rel == '=')? \\ & \ exact\_distance(operation.dis, \\ & \ step, operation.i\_group) \end{aligned}$$

In (6) i_group and s_group attributes are used the same way as in (5). The functions $min\_distance()$,

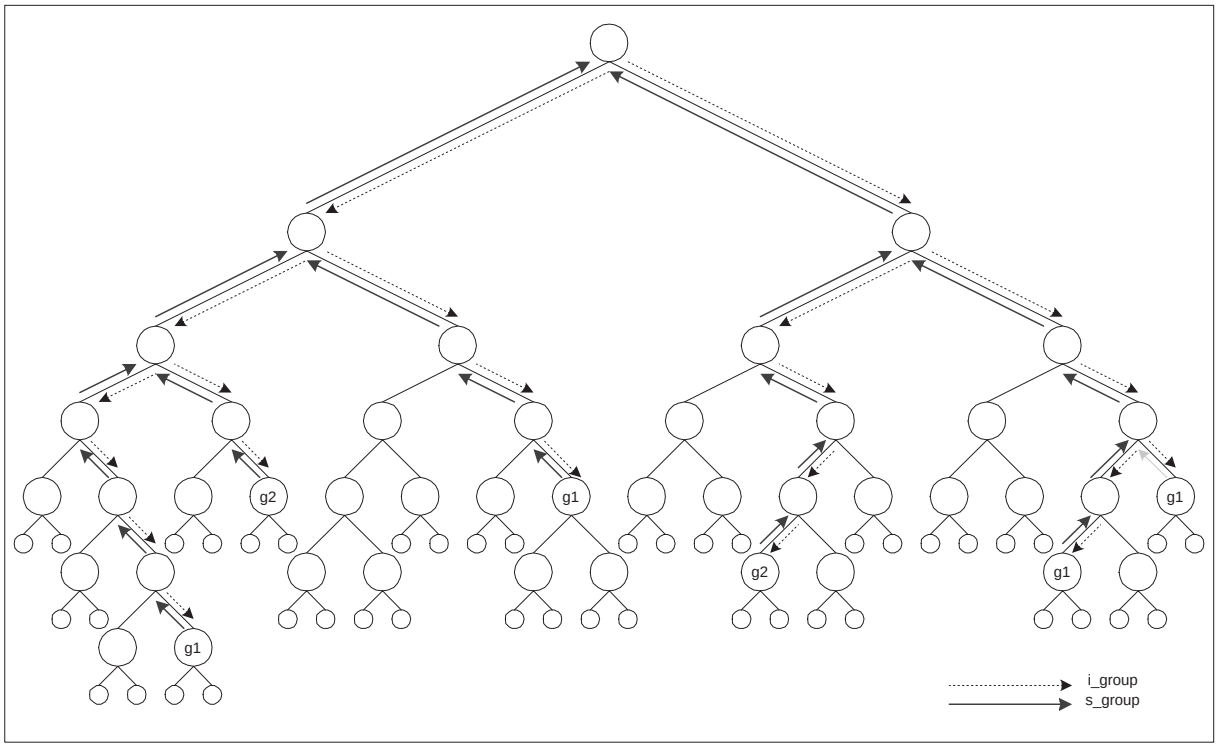

Figure 1. s_group **and** i_group **tree traversal**

$max\_distance()$ and $exact\_distance()$ are used to estimate the control step for the operation as follows:

- $min\_distance()$:The operator is scheduled so that all members of the group are within a time slot of at least $dist$ controls steps.
- $max\_distance()$:The operator is scheduled so that all members of the group are within a time slot of at most $dist$ controls steps.
- $exact\_distance()$:The operator is scheduled so that all members of the group are within a time slot of exactly $dist$ controls steps.

Rules like (1), (2), (3), (4), (5) and (6) are applied independently or combined (ex. $[\&2][g1]$) in a behavioral description. They affect the CSA selected by the HLS environment. In fact, the overall resulting scheduling algorithm is the following:

```
for each operation oi
 if c_step(oi,n)
  ScheduledStep(oi) = n
 else if step(oi,n)
  if n is a valid control step
   ScheduledStep(oi) = n
  else
   ScheduledStep(oi) = CSA(oi)
 else if delay(oi,n)
  ScheduledStep(oi) = n+CSA(oi)
 else if group(oi,xj)
  ScheduledStep(oi) =
    MAX(ScheduledStep(ok),∀ok ∈xj)
 else if distance(oi,xj)
  if relation=='<'
   ScheduledStep(oi) = min_distance() con-
sidering all ok ∈xj
  else if relation=='>'
   ScheduledStep(oi) = max_distance() con-
sidering all ok ∈xj
  else if relation=='='
   ScheduledStep(oi) = exact_distance() con-
sidering all ok ∈xj
 else /* no attribute applicable */
  ScheduledStep(oi) = CSA(oi)
```

Using the proposed methodology, designers can increase their interaction with the synthesis process. They can iterate through different implementations of the desired functionality changing the supplied step, c_step, delay, group and distance attributes until a desired implementation is generated. The overall advantage of the methodology is the declarative and modular notations presented above, which are used for both the specification and implementation of each scheduling heuristic (provided an automated compiler generator en-

vironment). With the adoption of the attributed-behavioral paradigm, user interaction is considerably increased.

4. Implementation Details

The implementation of the proposed methodology is a flexible and interactive high-level hardware compiler that produces synthesizable VHDL [2] or Verilog [15] output descriptions. The resulting environment can support top down, language based design of digital circuits by using any modern, commercial HDL synthesizer.

This new environment has been found to provide notable advantages, especially for researchers. This is due to the fact that the high-level transformation specifications needed are declarative and thus, very close to the actual description of the heuristic they implement. This makes them flexible and easy to manipulate and cause minor modifications, which is crucial for new research ideas. Another advantage is that all specifications are modular, so a problem can easily be partitioned into subproblems with separate specifications. When common subproblems are found, reusable specifications may be written. The complete source code in AG form required 51 files and 42K bytes in a Pentium 166MHz Linux based workstation.

5. Conclusion

An interactive HLS synthesis environment, following the attributed-behavior specification paradigm has been presented in this paper. It takes advantage of the capabilities supported by the AG computational model, that is, declarative and modular design specifications, and allows users to supplement scheduling heuristics with their implementation preferences. Moreover, the use of step, c_step, delay, group and distance attributes allows the designer to interfere with the scheduling algorithm and produce an efficient design. Both minor and major modifications to the scheduling algorithm can be easily and interactively achieved, depending on the skills of the human designer. This idea can be very helpful in design space exploration and with the implementation flexibility offered by AGs, can support a new paradigm for efficient system level design.

References

[1] L. F. Arnstein and D. Thomas. The attributed-behavior abstraction and synthesis tools. In *Design Automation Conference*, pages 557–561. ACM/IEEE, 1994.

[2] J. Bhasker. *A VHDL Primer*. Prentice Hall, 1992.

[3] G. Economakos, G. Papakonstantinou, and P. Tsanakas. An attribute grammar approach to high-level automated hardware synthesis. *Information and Software Technology*, 37(9):493–502, 1995.

[4] G. Economakos, G. Papakonstantinou, and P. Tsanakas. AGENDA: An attribute grammar driven environment for the design automation of digital systems. In *Design Automation and Test in Europe Conference and Exhibition*, pages 933–934. ACM/IEEE, 1998.

[5] G. Economakos, I. Poulakis, G. Papakonstantinou, and P. Tsanakas. An attribute grammar based interactive high-level synthesis tool. In *International Workshop on Logic and Architecture Synthesis*, pages 181–186. IFIP, 1998.

[6] R. Farrow and A. G. Stanculescu. A VHDL compiler based on attribute grammar methodology. In *Conference on Programming Language Design and Implementation*, pages 120–130. ACM SIGPLAN, 1989.

[7] D. Gajski, N. Dutt, A. Wu, and S. Lin. *High-Level Synthesis*. Kluwer Academic Publishers, 1992.

[8] R. W. Gray, V. P. Heuring, S. P. Levi, A. M. Sloane, and W. M. Waite. Eli: A complete, flexible compiler construction system. *Communications of the ACM*, 35(2):121–131, 1992.

[9] H. P. Juan, D. D. Gajski, and V. Chaiyakul. Clock-driven performance optimization in interactive behavioral synthesis. In *International Conference on Computer Aided Design*, pages 154–157. ACM/IEEE, 1996.

[10] K. Keutzer and W. Wolf. Anatomy of a hardware compiler. In *Conference on Programming Language Design and Implementation*, pages 95–104. ACM SIGPLAN, 1988.

[11] D. E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.

[12] Y.-L. Lin. Recent development in high level synthesis. *ACM Transactions on Design Automation of Electronic Systems*, 2(1):2–21, 1997.

[13] M. C. McFarland, A. C. Parker, and R. Camposano. The high-level synthesis of digital systems. *Proceedings of the IEEE*, 78(2):301–318, 1990.

[14] M. Naini. A dedicated dataflow architecture for hardware compilation. In *22nd Annual Hawaii International Conference on System Sciences*, 1989.

[15] S. Palnitkar. *Verilog HDL: A Guide to Digital Design and Synthesis*. Prentice Hall, 1996.

[16] W. M. Waite. An executable language definition. *ACM SIGPLAN Notices*, 28(2):21–40, 1993.

[17] R. A. Walker and S. Chaudhuri. High-level synthesis: Introduction to the scheduling problem. *IEEE Design & Test of Computers*, 12(2):60–69, 1995.

[18] C. H. Wu, T. S. Hadley, and D. D. Gajski. An efficient multiview design model for real-time interacive synthesis. In *International Conference on Computer Aided Design*, pages 328–331. ACM/IEEE, 1992.