BEHAVIORAL SYNTHESIS OF DIGITAL FILTERS USING ATTRIBUTE GRAMMARS

George Economakos, George Papakonstantinou and Panayiotis Tsanakas
National Technical University of Athens,
Dept. of Electrical and Computer Engineering,
Zografou Campus, GR-15773 Athens, GREECE
Tel: +301-7721532; fax: +301-7722496
e-mail: george@dsclab.ece.ntua.gr

ABSTRACT

Recently, a formal framework to perform behavioral synthesis using attribute grammars has been presented, its main advantages being modularity and declarative notation in the development of design automation environments. From a practical point of view, most modern behavioral synthesizers are best suited for the development of special purpose hardware to implement digital signal processing algorithms. In this paper, the attribute grammar formalism and the corresponding framework are extended to handle the development of special purpose hardware for both FIR and IIR filters. The proposed methodology is capable to construct hardware implementations in various technologies (FPGA, CPLD, ASIC) from behavioral filter specifications (difference equations or discrete convolution), utilizing the VHDL standard hardware description language. Overall, a formal specification for fast and efficient filter implementation is proposed.

1 INTRODUCTION

Attribute grammars (AGs) were devised by Knuth [8] as a tool for the formal specification of programming languages. However, in the general case, an AG can be seen as a mapping from the language described by a context free grammar (CFG) into a user defined domain. Over the years, a large number of AG based automated systems, that generate different kinds of language processors from their high-level specifications, have been constructed. The development of such systems is the main advantage of AGs over other formal specification methods; that is, they can also be used as an executable method, for the automatic construction of programs to implement the specified mapping. This has made AGs one of the most widely applied semantic formalisms.

Traditionally, AGs have been extensively used in compiler construction [1, 20]. Recently they have also been adopted in areas like dataflow computing [3, 16]. Similar to these two topics is the behavioral or high-level automated synthesis of special purpose architectures [4, 7, 21]. It is defined as the transformation of behavioral circuit descriptions into register-transfer level (RTL) structural descriptions (expressed in a special purpose Hardware Description Language (HDL) like VHDL [2]) that implement the given behavior while satisfying user defined constraints. Behavioral synthesis has been a hot research topic during the last twenty years. However, a lot of problems are still open. Attempting to overcome inefficiencies and propose a unifying formal framework, Economakos et al [12, 11, 13] proposed an AG based approach, which performs synthesis by decorating the parse tree of a behavioral circuit description with appropriate attributes.

In this paper, the whole methodology of AG-driven synthesis and the corresponding implementation are transferred to the domain of digital signal processing [15], for the implementation of digital filters. The behavioral specification method is extended to handle difference equations and discrete convolutions for both IIR and FIR filters, with different results for various VHDL description styles. The rest of the paper is organized as follows. Section 2 summarizes previous related research attempts and compares them with ours. Section 3 gives a detailed description of various aspects of AG-driven behavioral digital filter design automation and finally, section 4 gives the conclusions and proposes possible extensions.

2 RELATED RESEARCH

The design complexity of modern digital circuits has early enough motivated researchers to move towards higher abstraction levels. The main idea is that, by describing hardware in a more abstract way, larger designs can be automatically managed. Recently, language based processors have been put to use to aid in this direction. As far as automated synthesis is concerned, the idea is 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 all required optimization.

This was the idea followed by Seawright et al [17] to develop Clairvoyant, a system to perform hardware compilation using production-based specifications. As an extension to Clairvoyant, Oberg et al [14] presented

PRO-GRAM, a YACC-like synthesis environment for data communication protocols. AGs were introduced to the field of design automation by Jones et al [5], who presented an AG based solution to the incremental evaluation of properties in VLSI circuits. Another tool for silicon compilation, was the syntax directed system developed by Keutzer et al [6], which was based on the same ideas our current work is, but aimed at a lower level of abstraction.

In [12, 11], a first attempt to fully describe and implement the process of behavioral hardware synthesis using AGs was given. An extension of the formalism, supporting pure multi-pass AGs (AGs that pass some nodes of the parse tree more than one time) was given in [13]. The proposed methodology faced the whole problem (both data and control flow synthesis) in a unifying framework. However, so far it has not been tested with real world designs but rather small benchmark circuits.

In this paper, the methodology of AG-driven behavioral synthesis is transferred to the domain of digital signal processing, for the implementation of digital filters. The behavioral specification method is extended to handle difference equations and discrete convolutions for both IIR and FIR filters, with different results for various VHDL description styles. Specifically, I/O commands are added in the specification language, to handle timed input data. Description styles for both difference equations and discrete convolution of digital filters are investigated and implementation results are given. Overall, a flexible and modular environment for the behavioral design of FIR and IIR digital filters is proposed.

3 AG-DRIVEN TECHNIQUES FOR HIGH-LEVEL DIGITAL DESIGN AUTOMATION

3.1 Overview

Design entry in a high-level synthesis system is an algorithmic description written in a conventional or special purpose HDL. All HDLs exhibit some common programming language features, including constructs like data types, operators, assignment statements and control statements, supporting behavioral abstractions in different levels. Supplementary, hardware specific properties are also supported by modern HDLs with constructs like interface declarations, structural declarations, register-transfer and logic operators, asynchronous operations and constraint declarations. Finally, all HDLs define an execution ordering, with sequential and parallel threads of operation.

In an AG-driven environment, the HDL used plays a dual and crucial role. On one hand, as in all high-level synthesis systems, it is used to express design functionality. On the other hand, its syntax is used as the underlying CFG that is decorated by a synthesis AG, which implements scheduling heuristics through attribute evaluation rules. It must support a high level of specification abstraction with a strict and well-defined syntax. Such a language, used to describe hardware specifications is HardwareC [9]. A subset of HardwareC is used in our environment.

After the design is given to the synthesis tool as HDL text, high-level synthesis starts with the construction of the Control/Data Flow Graph (CDFG) of each behavioral description, then scheduling of all nodes of the CDFG into control steps (exploiting all possible parallel/serial arrangements while satisfying constraints) and finally allocation of the hardware elements that will carry each scheduled node.

Our AG-driven environment constructs the CDFG (using structures similar to [19]) and schedules nodes following well known heuristics (with both timing and resource constraints). Detailed allocation is not performed but, since a scheduled CDFG is an architecture (called Finite State Machine with Datapath, or FSMD architecture) architectural VHDL descriptions are generated that can be given as input to lower level tools to finish the design.

Both construction of the CDFG and scheduling are performed at the time the input behavioral description is being parsed by a special purpose attribute evaluation tool ([18]). Attributes are generated to hold all kinds of information that each CDFG node may need (type of operation, number and type of inputs, number and type of outputs, e.t.c.). These are attached to syntactic rules of the CFG that correspond to primitive operations. Such primitive operations are the CDFG nodes. The attributes are responsible for putting them in the correct order, to construct the CDFG. The same idea is used for scheduling.

This is the underlying methodology. With the addition of a few navigation procedures (attributes that force a specific order in the way the parse tree of the behavioral description is traversed [12, 13]), an integrated behavioral synthesis environment has been constructed.

3.2 Behavioral Synthesis of Digital Filters

For the design of digital filters, the above mentioned AG-driven environment must be enhanced to support the two types of filter behavioral entry, difference equations for IIR filters (mathematical specification in equation (1) and behavioral description in figure 1) and discrete convolution for FIR filters (equation (2) and figure 2).

$$y[k] = a_1 y[k-1] + a_2 y[k-2] + b_0 u[k] + b_1 u[k-1] + b_2 u[k-2] \quad (1)$$

$$y[k] = \sum_{i=0..L} h[i] u[k-i] \quad (2)$$

All of the coding constructs (assignments, while loops, e.t.c.) used in both figures were supported by the environment presented in [11] with the exception of the

```
block IIR;
yk1:=0; yk2:=0; uk1:=0; uk2:=0;
while (not reset)
  {
     u:=read(uin);
     yk:=a1*yk1+a2*yk2
        +b0*u+b1*uk1+b2*uk2;
     write(yout:=yk);
     yk2:=yk1; yk1:=yk; uk2:=uk1; uk1:=u;
  }
```

Figure 1: IIR behavioral description

```
block FIR(L=3);
uk1:=0; uk2:=0; uk3:=0;
while (not reset)
  {
     u:=read(uin);
     yk:=h0*u+h1*uk1+h2*uk2+h3*uk3;
     write(yout:=yk);
     uk3:=uk2; uk2:=uk1; uk1:=u;
  }
```

Figure 2: FIR behavioral description

`read` and `write` built-in functions. So, to handle digital filters, special AG-driven routines were generated to handle them. Here, only the case of the `read` function will be discussed for simplicity (the case of the `write` function is dual).

In behavioral synthesis, a call to a `read` function is functional equivalent to a simple assignment. For example, `x:=read(y)` is functional equivalent to `x:=y`. Their difference has to do with timing. The `read` function is used to describe serial protocols, that is, two consecutive `read` calls for the same input, mean sampling the input at two consecutive control steps (that may return the same or different values). On the other hand, two consecutive assignments of the same input, mean sampling the input at times determined by the scheduling heuristic (that always return the same value).

Using AGs, this can be accomplished using an attribute evaluation rule to insert in the symbol table the number N_i of `read`s encountered in the behavioral description from each input i. Each time a new `read` is parsed, N_i is increased. Then, it is scheduled as an assignment, N_i control steps later than the first `read` of i.

Since symbol tables are global data and global data are not allowed in pure AG formalisms, this is the idea that can be used to handle `read` calls from serial inputs.

	IIR	FIR
XOR2	2361	1895
OR2	4736	3800
AND2	10009	8084
INV	2537	2037
MUX	196	132
DFF	451	355

Table 1: Filter resource usage

In practice, the symbol table must be simulated using attributes and semantic rules. Specifically, all syntactic rules that may force the insertion of some element into the symbol table (in this case the syntactic rule that correspond to the `read` function), are equipped with a pair of attributes, one inherited and one synthesized. The inherited attribute holds the symbol table before the insertion and the synthesized after. The inherited attribute is passed through all left siblings of the of the insertion rule and the synthesized is passed to the right. This technique can be seen as an AG-driven floating symbol table, that supplies all relative information at the point it is needed.

After all operations of the CDFG have been extracted and scheduled, using both the above and all other techniques reported in [11, 13], an FSMD architectural description can be generated. A VHDL preprocessor tool is used to produce such descriptions following four different coding styles (for more details, see [10]). Each of them can be passed to lower level synthesis tools, presenting different advantages and disadvantages. Putting it all together, any kind of IIR or FIR filter can be designed from its behavioral specification using different implementation technologies (ASIC, FPGA, CPLD). The experimental results obtained so far are very promising. As an example, table 1 shows the resource usage from the implementation of the filters of figures 1 and 2 as netlists of primitive digital gates.

4 CONCLUSIONS AND FUTURE WORK

An AG-driven approach to the implementation of a flexible digital design automation environment has been presented in this paper, with focus on digital filter design. The results obtained show that this combination is promising. Its main advantages are the extensive use of existing tools and techniques for attribute evaluation and the incorporation of the AG formalism as a compact and modular very high level meta-language. Currently, we are working on enhancements to the environment that will allow us to handle larger designs and effectively handle all signal transformations under a unified AG formalism.

References

[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, 1986.

[2] K. C. Chang. *Digital Design and Modeling with VHDL and Synthesis*. IEEE Press, 1997.

[3] R. Farrow. Attribute grammars and dataflow languages. In *Symposium on Programming Language Issues in Software Systems*, pages 28–40. ACM SIGPLAN, 1983.

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

[5] L. G. Jones and J. Simon. Hierarchical VLSI design systems based on attribute grammars. In *13th Symposium on Principles of Programming Languages*, pages 58–69. ACM, 1986.

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

[7] D. Knapp. *Behavioral Synthesis*. Prentice Hall, 1996.

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

[9] D. Ku and G. De Micheli. HARDWAREC: A language for hardware design. Technical Report CSL-TR-90-419, Stanford University, 1990. Version 2.0.

[10] G. Economakos, P. Economakos, G. Papakonstantinou, and Tsanakas P. Integrating different VHDL coding styles in an attribute grammar driven high-level synthesis environment. In *International Workshop on Logic and Architecture Synthesis*, pages 175–184. IFIP, 1997.

[11] G. Economakos, G. Papakonstantinou, K. Pekmestzi, and P. Tsanakas. Hardware compilation using attribute grammars. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, pages 273–290. IFIP WG 10.5, 1997.

[12] 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.

[13] G. Economakos, G. Papakonstantinou, and P. Tsanakas. Incorporating multi-pass attribute grammars for the high-level synthesis of ASICs. In *Symposium on Applied Computing*. ACM, 1998.

[14] J. Oberg, A. Kumar, and A. Hemani. Grammar-based hardware synthesis of data communication protocols. In *9th International Symposium on System Synthesis*, pages 14–19. IEEE/ACM, 1996.

[15] A. Oppenheim and R. Schafer. *Discrete-time Signal Processing*. Prentice Hall, 1989.

[16] G. Papakonstantinou and P. Tsanakas. Attribute grammars and dataflow computing. *Information and Software Technology*, 30(5):306–313, 1988.

[17] A. Seawright and F. Brewer. Clairvoyant: A synthesis system for production-based specification. *IEEE Transactions on Very Large Scale Integration Systems*, 2(2):172–185, 1994.

[18] M. Sideri, S. Efraimidis, and G. Papakonstantinou. Semantically driven parsing of context-free languages. *The Computer Journal*, 32(1):91–93, 1989.

[19] D. E. Thomas, E. D. Lagnese, R.A. Walker, J. A. Nestor, J. V. Rajan, and R. L. Blackburn. *Algorithmic and Register-Transfer Level Synthesis: The System Architect's Workbench*. Kluwer Academic Publishers, 1990.

[20] W. M. Waite and G. Goos. *Compiler Construction*. Springer-Verlag, 1984.

[21] R. A. Walker and R. Camposano. *A Survey of High-Level Synthesis Systems*. Kluwer Academic Publishers, 1991.