2010 Second Pacific-Asia Conference on Circuits, Communications and System (PACCS)

XML Component-Based Modeling for Digital Circuits

Fulong Chen
Department of Computer Science and Technology
Anhui Normal University
Wuhu, Anhui Province, China
chenfulong@gmail.com

Zhaoxia Zhu
College of Computer Science
Yangtze University
Jingzhou, Hubei Province, China
zhuzx1973@126.com

Xiaoya Fan
School of Computer
Northwestern Polytechnical University
Xi'an, Shaanxi Province, China
fanxy@nwpu.edu.cn

***Abstract*— This paper presents a method of scheduler modeling for components of embedded systems, making the specification and verification process practical for real designs. In this method, designers can regard all circuits as components. A bigger component is constructed from those smaller components connecting with connectors (wires). Designers specify components in XML, and simulate their behaviors and verify their correctness. All components can be mapped into Verilog or VHDL automatically. This method supports hierarchical models, provides an efficient simple solution to model and verify our design quickly, and can be used in design and verification of embedded systems.**

***Index Terms*— Component ; XML ; digital circuits; FSM; truth table**

I. INTRODUCTION

Currently, VLSI (Very Large Scale Integration) circuits are becoming an integral part of our day-to-day lives, being embodied in various forms- microprocessors in home computers, embedded controllers in automobile fuel-injection systems, graphics controllers in video games, micro-controllers in toasters, answering machines, automated data-acquisition and manipulation components in bio-medical systems, etc. Before 2008, the IC market grew steady. In the third quarter of 2008, IC market peaked, and there were 44.1 billion of units shipped. But the global financial crisis affected the IC market. In 2009 the IC market fell. But since the second and third quarters of 2009, the IC market has been rapidly recovering. This indicates there is a strong demand in the IC market. IC Insights forecasts, as the global economy recovers in 2010[1], the IC market will enjoy very healthy growth rates.

In the development of embedded systems, especially digital circuit design, before we submit our design result for manufacturing, usually we have to face two key questions. The first one is how to specify what we want. The second one is how to make sure that what we specify is want we want. It has been widely estimated that over 70% of the design time for circuits is spent in performing various kinds of verification tasks and the effort devoted to this process eclipses all other aspects of the design process. Specification and verification have become the most important two tasks in current design flows and can have major impact on the timely delivery of a functionally correct product.

To get the final product, designers need to specify its functionalities and simulate its behaviors under the help of some simulation and verification tools. To save costly engineering effort, much of the effort of designing large logic machines need to be automated.

A. Specifying and Verifying Circuits Today

Verilog[2] and VHDL[3] are two HDLs (Hardware Description Languages) widely used to describe circuits. However, these HDLs have no enough system-level support for complex circuit design. They don't fully support the algorithm-level specification. Due to the low level of abstraction, designing Verilog/VHDL modules manually takes very skilled engineers and a significant time investment. Writing in HDLs can be more tedious and time consuming than writing a software program to do the same thing. After the design is complete, verification takes even more time. Currently, there are some expansions in traditional HDLs for enhancing their describing capability.

SystemVerilog [4] is an superset of Verilog, which extends some synthesizable design specifications. However, for the same function, the design descriptions in SystemVerilog are still much different from software programs, and designers still cannot ignore some timing constructs.

SystemC [5] has semantical similarities to Verilog and VHDL, and is emerging as a standard for high-level hardware/ software co-design and system-level modeling. However, it is neither used nor supported as widely as Verilog. Some specifications in SystemC are not synthesizable despite being valid for prototyping and simulation. To become a widely used HDL, SystemC still has a long way to go.

Recently, there are also some researches on converting from other software programming languages such as OCaml

*This work is partially supported by Project of Scientific Research for Young University Teachers of Anhui Province of China under Grant No.2008jq1057 to Fulong Chen

978-1-4244-7968-9/10/$26.00 ©2010 IEEE

[6] and C [7]to HDLs.

C-to-Verilog[7] automates circuit design and allows users to compile existing C functions into RTL Verilog codes. One blemish is that it combines data paths with FSM within an always-statement. Another one is that the only one return value is insufficient for multiple return values.

Therefore, directly programming in these HDLs is time consuming because they are in low abstraction level. Some descriptions in HDLs are only suitable for simulation purposes and cannot be synthesized into circuits. This means, we can simulate it, but we cannot get the real product. The verification is more important. It has been widely estimated that over 70% of the design time for circuits is spent in performing various kinds of verification tasks and the effort devoted to this process eclipses all other aspects of the design process. But it is also hard. Simulation is a method widely used in verification. Through simulation, we may know something is wrong in the HDL description, but it is not easy to find where and why the error happens.

B. Contributions

We want develop a method towards making the specification and verification process practical for real designs. In our method, we regard all circuits as components. Basic gates such as AND, OR and NOT are the most simple components. A bigger component is constructed from some smaller components. We specify our components in OCaml, which is a functional language, more abstract and easily used than HDLs. We can simulate the circuit behaviors and verify its correctness. And we also can map it into HDLs automatically. This is what we want to do-study on scheduling models for component-based embedded systems

Our contribution is to give an efficient simple method to model and verify embedded system quickly and make the specification and verification process practical for real designs. In this method, designers can regard all circuits as components. Each component has some ports for getting input signals from other components and generating output signals to other components.Basic gates such as AND, OR and NOT are the most simple components. Some small components - combinational circuits specified in truth tables and sequential circuits specified in FSMs(Finite State Machines),are synthesized into those interlinked gates. A bigger component is constructed from those smaller components connecting with connectors (wires). Designers specify components in XML (Extensible Markup Language)[8], which is a simple, very flexible text format derived from SGML (ISO 8879), originally designed to meet the challenges of large-scale electronic publishing, widely used for the representation of arbitrary data structures, and playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. Due to its structure style, it is a good format for specifying components of digital circuits a set of rules for encoding documents electronically, and simulate their behaviors and verify their correctness. All components can be mapped into HDLs automatically. Designers also can simulate the circuit in FPGA after translating the generated HDL codes into one bitstream file with some available synthesis tools.

This method supports hierarchical models, gives an efficient simple solution to model and verify embedded system quickly, and can be used in ASIC (Application-Specific Integrated Circuit) design and simulation.

II. Structures of Logic Circuit

Most digital systems divide into "combinational systems" and "sequential systems".

A. Combinational logic component

A combinational logic component is specified in a truth table as shown in Figure.1. In the grammar, only **TRUE**(1), **FALSE**(0) and **UNKNOWN**(x, X) are used for specifying logic values. Given an input vector, it can generate a corresponding output vector.

Truth table	**tt**	::=	< truthtable > **tterm**$^+$ < /truthtable >
Table term	**tterm**	::=	< term >
			< input > **value**$^+$ < /input >
			< output > **value**$^+$ < /output >
			< /term >
Logic value	**value**	::=	$0 \mid 1 \mid x \mid X$

Fig. 1. Grammar for the Truth Table

B. Sequential logic component

Sequential systems includes synchronous and asynchronous systems. The former change all states at once triggered by a clock signal. And the latter propagate changes whenever inputs change. Only synchronous sequential systems are supported in our specification.

A Mealy logic component generates an output based on its current state and input. In contrast, the output of a Moore logic component depends only on its current state; transitions are not directly dependent upon input. Figure 2 shows their grammar.

III. Specification for Components

A component may be a combinational logic component, sequential logic component or hierarchical component. Basic gates such as AND, OR and NOT are the most simple combinational logic components. Some small components - combinational circuits specified in truth tables and sequential circuits specified in FSMs, are synthesized into those interlinked gates. A bigger component is constructed from those smaller components connecting with connectors

(wires). Each component has some ports for getting input signals from other components and generating output signals to other components.

A complete grammar for the components is defined as Figure 3. All components are specified in some XML documents. Each document includes a head and body. The former is used to define if the current document includes other documents like head file in C language. And the latter is used to define components.

Each component has a name, a list of input ports, a list of output ports and a body. Of them, the body decides its type - combinational logic component, sequential logic component or hierarchical component. In a hierarchical component, some subcomponents are connected with connectors.

IV. Translation into Verilog

A translator is needed to deal with the translation from XML descriptions to Verilog codes.Figure.4 presents the BNF of the target Verilog grammar subset. This set is essentially a very small subset of Verilog that has structural models.

Mealy machine	**mealy**	::=	< mealymachine > **init mltran**$^+$ < /mealymachine >
Moore machine	**moore**	::=	< mooremachine > **init mrtran**$^+$ **mrout**$^+$ < /mooremachine >
Initial state	**init**	::=	< initial > **int** < /initial >
Mealy transition	**mltran**	::=	< transition > < input > **value**$^+$ < /input > < currentstate > **int** < /currentstate > < output > **value**$^+$ < /output > < nextstate > **int** < /nextstate > < /transition >
Moore transition	**mrtran**	::=	< transition > < input > **value**$^+$ < /input > < currentstate > **int** < /currentstate > < nextstate > **int** < /nextstate > < /transition >
Moore output	**mrout**	::=	< mooreoutput > < currentstate > **int** < /currentstate > < output > **value**$^+$ < /output > < /mooreouput >
Positive integer	**int**	$\in$	**Integer**

Fig. 2. Grammar for the Mealy and Moore Finite State Machine

Document	**docu**	::=	**head incls comps** | **head comps**
Head	**head**	::=	<?xml version="1.0" encoding="UTF-8" standalone=**opt**? >
Option	**opt**	::=	"yes" | "no"
Includes	**incls**	::=	< includes > **incl**$^+$ < /includes >
Include	**incl**	::=	< include > **fn** < /include >
Components	**comps**	::=	< components > **comp**$^+$ < /components >
Component	**comp**	::=	< component > **compn inp outp bdy** < /component >
Component name	**compn**	::=	< name > **id** < /name >
Input port	**inp**	::=	< inputport > **pt**+ < /inputport >
Output port	**outp**	::=	< outputport > **pt**$^+$ < /outputport >
Bidirectional port	**iop**	::=	< inout > **pt**$^+$ < /inout >
Port	**pt**	::=	< port > **id** < /port >
Body	**bdy**	::=	< body > **tt** < /body > |< body > **mealy** < /body > |< body > **moore** < /body > |< body > **subc cns** < /body >
Sub components	**subc**	::=	< subcomponents > **inst**$^+$ < /subcomponents >
Instance	**inst**	::=	< instance > < name > **id** < /name > < type > **id** < /type > < /instance >
Connectors	**cns**	::=	< connectors > **cn**$^+$ < /connectors >
Connector	**cn**	::=	< /connector > < port > {**id.**}**id** < /port > < port > {**id.**}**id** < /port > < /connector >
File name	**fn**	$\in$	$\mathbb{X}$
Identifier	**id**	$\in$	$\mathbb{X}$

Fig. 3. Grammar for the Components

Bit Constant	c	::=	$0 \mid 1$
Signal Name	s	$\in$	$\mathbb{X}$
Signals	S	::=	$s \mid s, S$
Module Name	m	$\in$	$\mathbb{X}$
Instance Name	n	$\in$	$\mathbb{X}$
Module Definition	D	::=	`module` $m(p); b$ `endmodule` $D \mid \varepsilon$
Assignment Statement	A	::=	`assign` $s = s; A$ $\mid$ `assign` $s = c; A \mid \varepsilon$
Module Body	b	::=	$V\ A\ M$
Port Type	t	::=	`input` $\mid$ `output`
Port Declaration	p	::=	$t\ s, p \mid t\ s$
Variable Declaration	V	::=	`wire` $s; V \mid \varepsilon$
Module Instance	M	::=	$m\ n(S); M \mid \varepsilon$

Fig. 4. Grammar for the Verilog subset

A. Minimization of Logic Functions

To reduce the circuit's complexity so that it has fewer errors and less electronics, and is therefore less expensive,many methods are used to minimize logic functions. Truth table-style descriptions of logic are often optimized with Electronic Design Automation (EDA)tools,which automatically produce reduced systems of logic gates or smaller lookup tables.

The most widely used simplification is a minimization algorithm like the Espresso heuristic logic minimizer[10] within a CAD system, working for those logic functions with a large number of inputs.

An automated Quine-McCluskey algorithm[12],developed by W.V. Quine and Edward J. McCluskey, slower than the former, is also used for simplifying those logic functions with less inputs. Starting with the truth table for a set of logic functions,it is a systematic and practical procedure to find the smallest set of prime implicants the output functions can be realised with. Although it is very well suited to be implemented in a computer program, its time and space complexity are so high that it is practical only for functions with a limited number of input variables and output functions. For example, adding a variable to the function will roughly double both of them, because the truth table length increases exponentially with the number of variables. A similar problem occurs when increasing the number of output functions of a combinational function block.However, it is very attractive since it can get the most optimal solution.

Moreover, Karnaugh Maps[11] and Boolean algebra have been used manually, difficult to be automated in the form of a computer program and only suited for up to 4 input functions.Both of them work in a laborious, tedious and error prone process,and therefore seldem used in practical design.

B. Minimization of State Machines

To automate costly engineering processes, we need take state tables that describe state machines and automatically produce a truth table for the combinatorial part of a state machine. However, the state machine often need optimizing because of the existence of some redundant states. Optimizing a state machine means finding the machine with the minimum number of states that performs the same function. The fastest known algorithm doing this is the Hopcroft minimization algorithm[13] used in our design.

C. Logic Synthesis

Logic synthesis is a process by which an abstract form of desired circuit description is turned into a design implementation in terms of logic gates. In some logic families, NAND gates are the simplest digital gate to build. All other logical operations can be implemented by NAND gates. Otherwise, sum-product based on NOT, AND, and OR gates is also widely used in logic mapping. Both of them are provided.

D. Hierarchical Component Translation

A hierarchical component is composed of some subcomponents and connectors. In the translator, all subcomponents are mapped into instances, and all connectors are mapped into continuous assignment statements.

V. Conclusions

To our best knowledge, this paper presents a practical way to describe circuit by providing specialized XML grammer. Our design will enable the quick development and data exchange of new IC product. Hierarchical structure helps to divide and conquer verification. A tool called XCOMP is developed for users.

References

[1] IC Insights, Inc., "2009 IC market enjoying a V-shaped recovery", http://www.icinsights.com/news/bulletins/bulletins2009/bulletin20091001.html, Oct.1, 2009

[2] IEEE Standard Board, "IEEE standard verilog hardware description language"(IEEE Std 1364-2005),IEEE, New York, 2006.

[3] IEEE Standard Board, "IEEE standard VHDL language reference manual"(IEEE Std 1076-2008), IEEE, New York, 2009.

[4] IEEE Standard Board, "IEEE standard for SystemVerilog- unified hardware design, specification, and verification lanaguage"(IEEE Std 1800-2005), IEEE, New York, 2005.

[5] IEEE Standard Board, "IEEE standard SystemC language reference manual"(IEEE Std 1666-2005), IEEE, New York, 2005.

[6] F., R.Goyal, E. Westbrook, W. Taha, "Implicitly heterogeneous multi-stage programming for FPGAs", Proceedings of 11th Symposium on Trends in Functional Programming, 2010, pp.217-235

[7] Y. Ben-Asher and N. Rotem, "Synthesis for variable pipelined function units", Proceedings of International Symposium on System-on-Chip, 2008, pp.1-4

[8] T.Bray, J. Paoli, C. M. Sperberg-McQueen, et al, "Extensible markup language (XML) 1.0" (Fifth Edition), http://www.w3.org/TR/2008/REC-xml-20081126/, November 26, 2008

[9] R. Sharp, "Higher-leverl hardware synthesis", Springer-Verlag,Berlin Heidelberg, 2004

[10] R. L. Rudell "Multiple-valued logic minimization for PLA synthesis", Memorandum No. UCB/ERL M86-65 (Berkeley), http://www.eecs.berkeley.edu/Pubs/TechRpts/1986/ERL-86-65.pdf,June 5, 1986.

[11] M. Karnaugh, "The map method for synthesis of combinational logic circuits", Transactions of American Institute of Electrical Engineers part I1953729,pp.593-599

[12] A. Popov and K. Filipova, "Genetic algorithms synthesis of finite state machines", Proceedings of 27th International Spring Seminar on Electronics Technology: Meeting the Challenges of Electronics Technology Progress, 2004, pp.388-392.

[13] J. E. Hopcroft, "An n log n algorithm for minimizing states in a finite automaton", Stanford University Stanford, CA, USA,1971.