AN INTERACTIVE DESIGN ENVIRONMENT FOR C-BASED HIGH-LEVEL SYNTHESIS

Dongwan Shin
Andreas Gerstlauer
Rainer Dömer
Daniel D. Gajski
Center for Embedded Computer Systems
University of California, Irvine, CA 92697
{dongwans,gerstl, deomer, gajski}@cecs.uci.edu

Abstract Much effort in RTL design has been devoted to developing "push-button" types of tools. However, given the highly complex nature, and lack of control on RTL design, push-button types of synthesis is not accepted by most designers. Interactive design space exploration with assistance of tools and algorithms can be more effective because it provides control of all steps of synthesis.

In this paper, we propose an interactive RTL design environment, which enables designers to control design steps. In our interactive environment, the user can control the design process at every stage, observe the effects of design decisions, and manually override synthesis decisions at will. Finally, we present a set of experimental results that demonstrate the benefits of our approach. Our combination of automated tools and interactive control by the designer results in quickly generated RTL designs with better performance than fully-automatic results, comparable to fully manually optimized designs.

1. Introduction

Automating RTL synthesis is very complicated issue. It is known that the majority of synthesis tasks are NP-complete problems. Hence, the design time becomes large, or the results are suboptimal, resulting designs cannot satisfy the performance or area demands of real-world constraints.

To develop a feasible approach for RTL synthesis, we have substituted the goal of a completely automated, "push-button" synthesis system with one that allows to maximally utilize the human designer's insights. This approach is called *Interactive synthesis methodology*. In

this approach, the designer can control the design process at every stage, observe the effects of design decisions, and manually override synthesis decisions at will. This is facilitated through a convenient graphical user interface (GUI).

Hardware description languages (HDLs) such as Verilog HDL and VHDL are most commonly used as input to RTL design. However, system designers often write models using programming languages such as C/C++ to estimate the system performance and to verify the functional correctness of the design, even to refine the design into implementation.

C/C++ offers fast simulation as well as a vast amount of legacy code and libraries which facilitate the task of system modeling. To implement parts of the design modeled in C/C++ in hardware using synthesis tools, designers must then manually translate these parts into a synthesizable subset of a HDL. This process is well known for being both time consuming and error prone. Moreover, it can be eliminated completely. The use of C-based languages to describe both hardware and software will accelerate the design process and facilitate the software/hardware migration. Hardware synthesis tools from C/C++ can then be used to map the C/C++ models into logic netlists.

The rest of the paper is organized as follows: section 2 shows related work and section 3 introduces our RTL design environment and the program flow of the proposed RTL synthesis tool. Section 4 shows the experimental results. Section 5 concludes the paper with a brief summary.

2. Related Work

Issues in RTL modeling, RTL design and behavioral synthesis, aka. High-Level Synthesis (HLS), have been studied for more than a decade now [3].

In the recent years, a few projects have been looking at means to use C/C++ as an input to current design flows [4, 6, 16]. Constructs are added to model coarse-grain parallelism, communication and data-types. These constructs can either be defined as new syntactic constructs, hence creating a new language [4]. They can also be implemented as part of a C++ class library [6]. In order to facilitate the mapping of C/C++ models into hardware, several tools exist that automatically translate C/C++ based descriptions into HDL either at the behavioral level or the register transfer level (RTL) [13, 16, 7].

Many automatic synthesis tools (also known as *push-button* synthesis) have been developed, including Olympus [10], OSCAR [11], SPARK [7], Synopsys Behavioral Compiler [15], Mentor Catapult-C [12], and Cy-

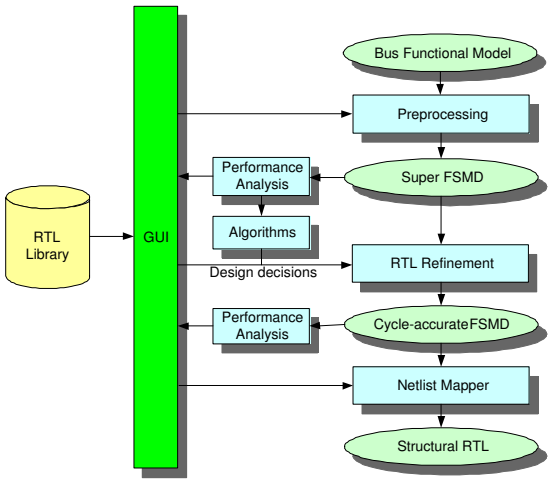

Figure 1. RTL design flow

ber [16]. However, these tools provide no means to access the intermediate design models that are created during the synthesis process and to change important decisions by designers. The designer can specify design constraints for whole designs and access the behavioral input model and the structural output model and design constraints.

Some interactive synthesis approaches [8, 9] addressed the importance of user-interaction with synthesis system. However they have a fixed design flow, that is, the designer has to perform a sequence of synthesis tasks in a predefined order and a cycle-accurate simulation model with complex components is not available for the intermediate stages.

3. RTL Design Environment

In this section, we will describe our RTL design environment integrated in a system-level design flow. The RTL design environment provides synthesis, refinement and exploration for RTL design as shown in Figure 1. It includes a graphical user interface (GUI) and a set of tools to facilitate design flow and perform refinement steps. In our flow, designers or algorithms of automatic tools can make decisions such as clock period selection, allocation, scheduling and binding. The GUI allows designers to input and change such design decisions. It also enables the designer to observe the effects of the decisions and to manually override the decisions at will. Further, the designers can make partial decisions and then run automatic tools to take care of the rest of the decisions.

We model an RTL design as a Finite State Machine with Data (FSMD) [1], which is an FSM model with assignment statements added to each state.

The FSMD can completely specify the behavior of an arbitrary RTL design. The variables and functions in the FSMD may have different interpretations which in turn defines several different styles of RTL semantics.

In addtion, in order to represent pipelined or multicycled units in a design in the cycle-accruate FSMD, we have introduced new constructs [2] such as `piped` for pipelined units, and `after` for multicycle units. The simulation speed of the cycle-accurate FSMD is significantly improved compared with that of the structural RTL description.

During preprocessing, the behavioral description of custom hardware in C/C++ will be refined into an SFSMD model where each state is a basic block of the original description. Also some presynthesis optimization techniques including constant propagation, dead code elimination, and common subexpression elimination are integrated. The generated FSMD will be the input model of the RTL synthesis.

A performance analysis tool is used to obtain characteristics of the initial design such as the number of operations, variables and data transfers in each state, which serves as the basis for RTL design exploration. It also produces quality metrics for RTL design such as the delay and power of each state and area of the design to help the designers to make decisions on clock selection, allocation, scheduling and binding.

The refinement tool then automatically transforms the FSMD model based on relevant design decisions. Finally, the structural RTL model is produced by a netlist mapper, ready to feed into traditional design tools for logic synthesis, etc.

3.1 Synthesis Decisions

The refinement engine works on directions called the RTL synthesis decisions. The synthesis process can either be automated or interactive as per the designer's choice. However, the decisions must be input to the refinement engine using a specific format. For the purpose of our implementation, we annotated the input model with the set of synthesis decisions. The refinement tool then detects and parses these annotations to perform the requisite model transformations. Based on these decisions, the refinement engine imports the required RTL components from the RTL component library and generates the cycle-accurate FSMD.

The decisions can be made by designers interactively through GUIs and/or be made through automatic algorithms. The GUIs for interactive decision-making allows designers to (a) specify decisions (b) override the decisions, which are already made by the designers or automatic

algorithms (c) partially assign decisions and automatic algorithms will fill in the rest of decisions.

The GUI also allows automatic algorithms being plugged in. Thus it is easily extendible because designers can select an algorithm from a list of plug-in algorithms such as ASAP, ALAP, list and force-directed scheduling and graph coloring for binding and so on.

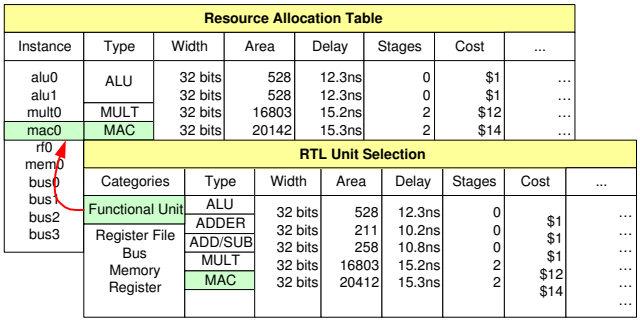

Figure 2. Allocation window

3.1.1 GUI for Interactive Decision-making. In order to help designers to make synthesis decisions interactively, we provide an *allocation window* and a *scheduling & binding window*. In allocation window as shown in Figure 2, designer can see all RTL components in the RTL component library, select them and set the parameters such as bit width, size of array and so on [5].

The scheduling & binding window displays the SFSMD in *state-operations table* format which contains a series of states, each state containing a set of operations to be performed in the state, shown in Figure 3. The state-operations table displays the behavior of a design and all design decisions made in graphical format. This is, the designer can modify all design decisions at any time in the design process in the state-operations table. In the table, *State* is the current state and *NS* is next state. *CS* is the control step of the expression which is relative to the start time of the state.

The table also shows statistics such as the lifetimes of all variables, occurrences of operations, the number of data transfers and the critical path in number of operations in each state. It also shows the ASAP and ALAP control step for each expression in each state.

All expressions are scheduled at specified control steps in the scheduling view, which will be assigned to *CS* in the state-operations table. All operations are bound to functional units and their ports, which will be specified in the *oper* column. Also all operand variables (*destination*,

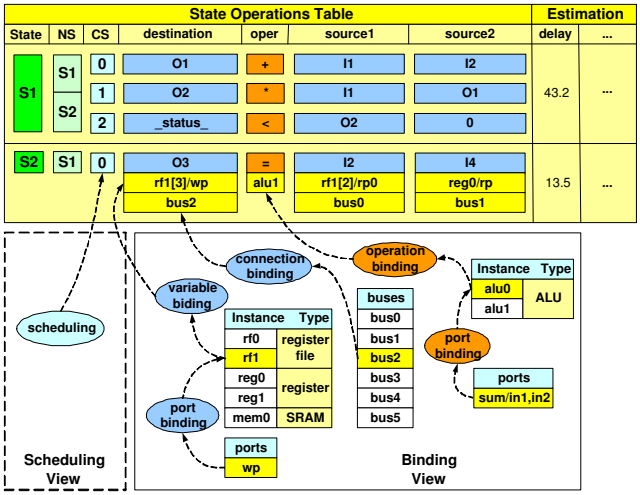

Figure 3. Scheduling & Binding window for an SFSMD

source1, *source2*) are mapped to storage units, read/write ports of the storage unit, and busses in the binding view. If the variables are mapped to memory, then the base address needs to be specified as well.

Designers can input, modify all decisions and override decisions which algorithms made through automatic tools in scheduling & binding window. Furthermore, the designers can partially specify some of the decisions and then algorithms take care of the rest of decisions still meeting the specified designer's decisions.

3.2 Performance Analysis

Several synthesis metrics are implemented to help the designer decide how to select the allocation and partition a super FSMD description into control steps. Two important metrics of design cost are operator occurrences and variable lifetimes. Operator occurrences metric shows the number of operations of each type used in each state. The maximum number of occurrences of a certain operator type over all states determines the required minimum number of functional units to perform that type of operation. Variable lifetimes metric identifies states in which state a variable holds a useful value. The maximum number of variables with overlapped lifetimes over all states determines the required minimum number of storage units. After allocation, performance estimation calculates the delay and power consumption of each state and the area of the design.

4. Experimental Results

We have implemented our interactive RTL synthesis approach in C++ (algorithms, data structure) and Python (GUI). The benchmarks used are *sra* (square root approximation), *GCD* (greatest common divisor), *DIFFEQ* (differential equation solver), from high-level synthesis benchmark suite.

The different types of implementation of discrete cosine transformation, *DCT* (2-dimensional DCT with matrix multiplication), *ChenDCT* (2-dimensional DCT implementing Chen algorithm), *MP3DCT* (1-dimensional DCT for MP3 Codec) *MP3IMDCT* (1-dimensional ImDCT for MP3 Codec) *Codebook* (codebook search block in the GSM Vocoder which is employed worldwide for cellular phone networks. The model was based on the bit-exact reference implementation of the ETSI standard in ANSI C).

Table 1 lists the characteristics of the designs used in terms of the number of operations (`#OPs`) in the input description, which is indicative of the data complexity of the design, and the number of basic blocks (`#BBs`), indicative of its control complexity. Also the type and quantity of each resource allocated to schedule and bind this design for all the experiments are given in Table 1. The resources indicated in this table are: *ALU* is used for arithmetic and logic operations ($+$, $-$, &, $|$, $\hat{}$, and $\sim$) or saturated arithmetic operations in `Codebook` example. *ADD* and *SUB* for addition ($+$) and subtraction ($-$) respectively, *ASU* for both addition and subtraction, *MULT* for multiplication ($\times$) in 1 stage pipeline fashion, *DIV* for division ($\div$) in 5 stage pipeline fashion, *SQRT* for square root operation in 5 stage pipeline fashion, *LU* for logic operations (&, $|$, $\hat{}$, and $\sim$), *SHIFT* for left/right shift operation ($\ll$, $\gg$), *CMP* for comparison ($\geq$, $\leq$, , $<$, $==$, $!=$, !)

REG is a register with 1 read port and 1 write port and *RF* is a register file with 2 read ports and 1 write port. *MEM* and *RAM* have 1 read port and 1 write port, and the *ROM* has 1 read port. The number in parenthesis indicates the size of register files and memories.

For our experiments, we implements a heuristic based on list scheduling algorithm [14] which performs scheduling and binding at the same time (`Automatic` in Table 2). The heuristic considers the allocation of units and the number of ports of allocated units and busses. After applying the heuristic algorithm, we applied some optimization techniques to improve the performance of designs on the RTL design environment (`Automatic + Manual` in Table 2), while `Manual` shows the examples designed by designers.

For `MP3DCT` and `MP3IMDCT`, a designer manually implements control pipelining by inserting pipeline registers on the output logic of the controller, which can reduce the clock period by reducing the delay from the output of output logic of the controller to its datapath. In addition, all control words are stored in program ROM.

For `Codebook`, the designer inserts registers at the output of functional units to reduce the clock period and implements data forwarding from output of a function unit to the input of other functions units. In addition, the special counters are introduced to calculate the index of arrays in the memory (inside loops), which is accessed by row and column by two indices.

We present the logic synthesis result obtained after synthesizing the RTL Verilog generated by Netlist mapper using the Synopsys *Design Compiler* logic synthesis tool. The `LSI-10K` synthesis library is used for technology mapping and components are allocated from the Synopsys DesignWare Foundation library.

The logic synthesis results are presented in terms of three metrics: the unit area (in terms of synthesis library used), the number of states (`states`) in FSM controller, the critical path length (clock period, `CP` in nanoseconds) and the number of clock cycles (`cycles`) to execute the design. The critical path length is the length of the longest combinational path in the netlist as reported by static timing analysis tool and it dictates the clock period of the design.

Table 1. Characteristics of synthesis examples

Examples	#OPs	#BBs	resources
GCD	5 (3)	9	1 SUB, 1 CMP, 1 LU, 4 REGs
DIFFEQ	11 (10)	6	1 ASU, 1 CMP, 1 MULT, 16 REGs
DCT	21 (8)	22	1 ADD, 1 MULT, 3 COUNTs, 3 REGs, 3 RAM (64), 2 ROM (64)
ChenDCT	167 (42)	27	1 ADD, 1 SUB, 1 SHIFT, 1 MULT, 1 CMP, 16 REG, 1 RAM (128)
MP3DCT	330 (0)	3	1 ASU, 1 SHIFT, 1 MULT, 1 CMP, 2 RFs (64)
MP3IMDCT	195 (178)	49	1 ASU, 1 SHIFT, 1 MULT, 1 CMP, 2 RFs (64)
Codebook	604 (253)	197	1 ALU, 1 CMP, 1 MULT, 1 DIV, 1 MAC, 1 SQRT, 16 REGs, 1 RAM (2048)

The speedup, as shown in Table 3 of a design by `Automatic + Manual` and `Manual` is defined as follows:

$$Speedup_x = \frac{CP_{Auto.} \times cycles_{Auto.} - CP_x \times cycles_x}{CP_{Auto.} \times cycles_{Auto.}} \times 100$$

Table 2. Synthesis result

Examples	Automatic			Automatic + Manual			Manual		
	area	CP	cycles	area	CP	cycles	area	CP	cycles
GCD	5154	35.7	34	5177	33.9	34	—	—	—
DIFFEQ	17484	37.6	113	16314	31.7	93	—	—	—
DCT	95408	60.1	6914	95173	50.9	5792	114090	42.6	4225
ChenDCT	32487	49.0	2469	31155	45.4	2129	—	—	—
MP3DCT	85786	61.2	565	83780	54.9	308	103357	47.7	308
MP3IMDCT	93069	55.0	382	91669	50.9	209	70239	44.3	209
Codebook	991215	71.4	56492	991065	70.2	43195	987162	55.5	33000

Table 3. Performace improvement

Examples	Automatic + Manual	Manual
GCD	5.0%	—
DIFFEQ	30.6%	—
DCT	29.1%	56.7%
ChenDCT	20.1%	—
MP3DCT	51.1%	57.5%
MP3IMDCT	49.4%	55.9%
Codebook	24.8%	54.6%

where x is either `Automatic + Manual` or `Manual`.

Through the experiments, the performace of the design by the heuristic algorithm is the worst among 3 implementations, but after modified the scheduling and binding result by users, the designs get about 30% improvement, specially in terms of clock cycles, and become close to the designs by human.

5. Conclusion and Future Work

In this paper, we proposed an interactive C-based RTL design environment which takes full advantage of the designer's insight by allowing to enter, modify, override all decisions at will.

It has been developed and integrated into SoC design environment in order to validate our approach. This allows designers to evaluate several design points during fast exploration. Our experimental results show that the proposed solution to improve behavioral modeling of RTL designs is not only feasible and practical for real-world designs, it also comes with a significant speed-up in simulation.

Future work in this direction will involve comparison between our approach and commercial tools and the scheduling of bus protocols under timing constraint in clock cycles.

References

[1] Accellera C/C++ Working Group of the Architectural Language Committee. RTL Semantics, Draft Specification. Technical report, Accellera, February 2001. available at http://www.eda.org/alc-cwg/cwg-open.pdf.

[2] R. Dömer, A. Gerstlauer, and D. Shin. Cycle-accurate RTL modeling with multi-cycled and pipelined components. In *Proceedings of International SoC Design Conference*, pages 375–378, October 2006.

[3] D. D. Gajski, N. Dutt, S. Y.-L. Lin, and A. Wu. *High Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic Publishers, 1992.

[4] D. D. Gajski, J. Zhu, R. Dömer, A. Gerstlauer, and S. Zhao. *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, January 2000.

[5] A. Gerstlauer, L. Cai, D. Shin, R. Dömer, and D. D. Gajski. System-on-Chip component models. Technical Report CECS-TR-03-26, Center for Embedded Computer Systems, University of California, Irvine, August 2003.

[6] T. Grötker, S. Liao, G. Martin, and S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, March 2002.

[7] S. Gupta. SPARK: High-level synthesis using parallelizing compiler techniques. available at http://www.cecs.uci.edu/~spark/.

[8] A. A. Jerraya, I.-C. Park, and K. O'Brien. AMICAL: An interactive high level synthesis environment. In *Proceedings of the European Design Automation Conference*, pages 58–62, February 1993.

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

[10] D. Ku and G. D. Micheli. *High-level Synthesis of ASICs under Timing and Synchronization Constraints*. Kluwer Academic Publishers, 1992.

[11] B. Landwehr, P. Marwedel, and R. Dömer. OSCAR: Optimum simultaneous scheduling, allocation and resource binding based on integer programming. In *Proceedings of the European Design Automation Conference*, February 1994.

[12] Catapult C Synthesis, Mentor Graphics Inc. available at http://www.mentor.com/.

[13] L. Séméria and G. D. Micheli. SpC: Synthesis of pointers in C, application of pointer analysis to the behavioral synthesis from C. In *Proceedings of the International Conference on Computer-Aided Design*, pages 340–346, November 1998.

[14] D. Shin and D. D. Gajski. Scheduling in RTL design methodology. Technical Report CECS-TR-02-11, Center for Embedded Computer Systems, University of California, Irvine, April 2002.

[15] Behavioral Compiler, Synopsys Inc. available at http://www.synopsys.com/.

[16] K. Wakabayashi and T. Okamoto. C-based SoC design flow and EDA tools: An ASIC and system vender perspective. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, December 2000.