Synthesizing Reconfigurable Sequential Machines Using Tabular Models

Kamlesh Rath
University of Texas at Dallas
Richardson, Texas.

Jian Li
University of Illinois
Urbana-Champaign, Illinois.

Abstract. Two common problems we face in implementing reconfigurable systems on currently available FPGA chips are: (1) fitting designs which are too big for available hardware resources on a single FPGA chip, (2) lack of synthesis tools for high-level specifications. One solution to address the first problem is partial reconfiguration or run-time reconfiguration which requires only loading a portion of the design onto a FPGA chip at one time. In this paper, we present a synthesis methodology which starts from high-level system specifications and synthesizes run-time reconfigurable systems. Our approach uses tabular models as intermediate data structures. Tabular representations provide a convenient platform for separating control and data-path, and dividing the data-path into separate control-paths. This makes our approach very useful in synthesis targeted at implementations that depend on run-time reconfiguration to fit bigger designs on currently available FPGA chips.

1 Introduction

Reconfigurable systems enable system designers to change the behavior of a system by configuring generic logic blocks and inter-connect structures of FPGAs at boot-time or run-time.

Though very promising, systems built out of FPGAs face many problems. One problem is that available hardware resources on FPGA chips are rarely large enough to synthesize entire interesting systems all at once [1]. One solution to this problem is to use partial reconfiguration [2] or run-time reconfiguration [3] instead of loading the whole configuration corresponding to a design all at once. The basic idea here is to load only part of a design which is needed right away on a FPGA chip. It is swapped out and replaced by another part of the design when it is needed.

Another problem is the compatibility issue among a range of implementations [1]. Currently designs have to be modified before they are ported even from one chip to another chip of the same family. One possible solution we see to address this problem is to provide designers with tools which synthesize high-level designs.

There has been very limited research on synthesis tools for reconfigurable systems. Most systems require input by schematic capture or vendor-specific

tools. So far to our knowledge, no work has been reported to provide synthesis tools which take high-level specifications and target run-time reconfigurable implementations on FPGA chips.

In this paper, we present a synthesis methodology which starts from behavioral descriptions and implements run-time reconfigurable systems on a given FPGA chip. Our methodology uses an intermediate tabular model called Timed Decision Table (TDT). TDT is a behavior model first introduced for presynthesis optimizations [4, 5]. These tables provide a convenient data structure for separating control and data-path, and for dividing data-path into different control paths.

The rest of this paper is organized as follows. Section 2 introduces tabular models we use in our proposed synthesis system. Section 3 presents our synthesis methodology. In Section 4, we demonstrate how our proposed methodology work on a simple example. Finally, we conclude in Section 5 with a description of our on-going and future work.

2 Tabular Representations for Hardware Modeling

Tabular models have been used for hardware modeling at different abstraction levels [6, 4]. In [6] the authors have proposed behavior tables (BTs) as a register transfer language using finite state machine model. These tables have been used to evaluate control versus data tradeoff while preserving cycle-accurate (RTL) behavior. In [4], the authors have introduced a behavior level model called Timed Decision Tables (TDT). TDT has been used in presynthesis optimizations or behavior optimizations before any synthesis tasks (tasks such as scheduling, binding, logic optimization, and technology mapping) are carried out [4, 5].

In our synthesis methodology targeted at run-time reconfigurable systems, we use TDTs as intermediate data structures. We give a brief introduction of TDTs using a simple example. Interested readers are referred to the original references for the details about TDTs.

The example we use to illustrate TDT models is given in Example 2.1. It is a HardwareC model modified from the standard GCD example in the *High-Level Benchmark Suite* [7]. HardwareC is an HDL with C-like syntax first developed at Stanford [8]. Many statements in HardwareC have similar meaning to those in the C programming language.

Example 1. This 'gcd2' process keeps sampling data on its two input port `xi` and `yi`, computing their greatest common divisor, and putting the square of this result on the output port `ou`.

```
process gcd2 (xi, yi, ready, ou)
     in port       xi[8];     /* input numbers */
     in port       yi[8];
     in port       ready;     /* restart input */
     out port      ou[8];     /* result output */

[
   static x[8] = 0;
   static y[8] = 0;
    }
   while (! ready) /* sample input numbers */
    {
      read x = xi;
      read y = yi;
    }

/* only positive numbers will work */
      if ((x > 0) && (y > 0))
        {
          repeat /* using euclid's gcd algorithm */
           {
              while (x >= y)  x = x - y;
              < x = y; y = x; >
           }
          until  (y == 0);

/* write square of the gcd results to the output */
       write ou = x * x ;
 ]
```

2.1 Timed Decision Tables (TDTs)

A TDT is behavior-level model. It closely models system specification described in behavioral HDLs. A TDT representation of system behavior consists of three major parts: (1) a control section which is a set of rules, or a list of control path segments, (2) a delay table which lists the execution delay of each action or data path operation in the model, (3) an additional table describing data-dependency, serialization, concurrency type specified between each pair of operations.

S	0	1	1	2	3	3	4	5	5	6	7	7
rst	-	Y	N	-	-	-	-	-	-	-	-	-
(x!=0)&(y!=0)	-	-	-	-	Y	N	-	-	-	-	-	-
x>=y	-	-	-	-	-	-	-	Y	N	-	-	-
y == 0	-	-	-	-	-	-	-	-	-	Y	N	
write out=0	1											
x=read(xi)				1								
y=read(yi)				1								
x= x-y								1				
x=y												1
y=x												1
T1 = x * x							1					
write ou=T1							1					
S_N	1	2	1	3	5	4	0	5	6	7	4	7

	delay
write out=0	1
x=read(xi)	1
y=read(yi)	1
x = x-y	1
T1 = x*x	4
write out=x	1

	x=read(xi)	y=read(yi)	x=y	y=x
x=read(xi)		data-parallel		
y=read(yi)				
x=y				parallel
y=x			parallel	

Fig. 1. The TDT representation of this previous of the gcd2 example.

An example of such a representation is given in Figure 1. The control section consists of four quadrants: (1) the condition stub, which is a list of input ports and internal conditions, (2) the condition entries, which list possible combinations of conditions, (3) action stub, which list all computations that appear in the HDL description, (4) action entries, which list the actions enabled in each column or segments of control path.

3 Synthesis Methodology

The most important difference between classical systems and reconfigurable systems is the temporal nature of synthesizing such systems. We use the temporal dimension of systems, that is specified by state transistions in a sequential system, to partition a system specification into several configurations. Each configuration is self-contained and has all the required hardware to execute its state transitions. New configurations are loaded into the target FPGA platform on reaching the "boundary states" between configurations.

We first translate the high-level system specification (e.g. HardwareC or VHDL behavioral specification) into Timed Decision Tables (TDTs) [9, 4]. The TDT specification of a system includes both a control description and a data path description of the system. It can be used to perform pre-synthesis optimizations on the system specification [10].

A control-flow graph representation of the TDT specification is used to perform the next step in our synthesis methodology. The main synthesis task involves partitioning this control-flow graph into connected sub-graphs. Each subgraph groups a set of transitions starting from a "root" state. The grouping is done so that the cost of all transitions in each group is less than the available

resources on the FPGA. A group is constructed by performing a breadth-first coverage of successive transitions from the root state. We start with the "start state" of the system as the root and iteratively cover the entire control-flow graph by using nodes at the "boundary" of the previous sub-graph as the root for the next iteration.

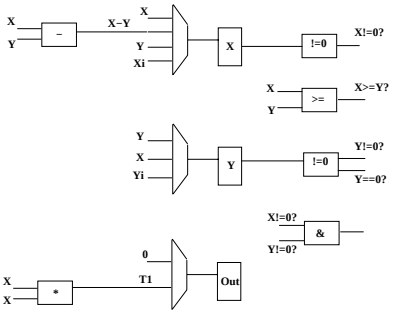
A restriction imposed on the partioning is that all out-going transitions from any state must belong to a single partition. This eliminates indeterminism in loading a new configuration on reaching a "boundary state".

The sub-graphs identified by the partitioning step form the different data path configurations that can be swapped in and out of the FPGA. The method described in this section is illustrated using the "gcd2" example in Section 4. The partitions identified using the above method are then synthesized independently using standard FPGA synthesis tools [11]. The control logic for the system is also partitioned into corresponding parts so that the control outputs required for a partition are synthesized along with the data-path. Only the memory elements in the system, e.g. data and state registers, are static and are synthesized once during startup.

4 Example

Consider the "gcd2" example. The high-level specification for "gcd2" is first translated into a TDT specification [12, 5]. The controller and datapath of a system can then be constructed from its tabular specification using well known techniques [6, 13].

The complete datapath for "gcd2" is shown in Figure 2 (a). The conditionals in the system are shown on the right side of the figure. The registers in the datapath are `X, Y, Out`. The multiplexers at the input of the registers show the different values that could be assigned to the registers based on the current state and conditional values.



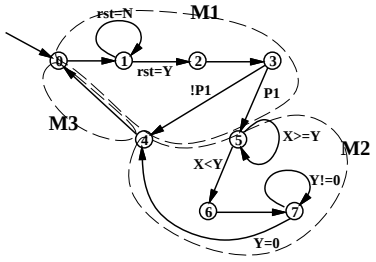


Fig. 2. **(a)** Complete gcd2 datapath **(b)** Control-flow graph for gcd2

Using the partitioning method described in Section 3, we partition the control-flow graph into three parts. Figure 3 shows the three different configurations for "gcd2". All the registers in the system (including the state registers) are static and are not reconfigured during run-time. Figure 2 (b) shows the control-flow

graph for "gcd2". The costs of implementing all operations and conditionals in every transition in a partition are added to include the maximum number of possible transitions in each partition upto the capacity of the FPGA. Going breadth-first from the start state (labeled 0 in Figure 2 (b)), we include all transitions from states labeled 0, 1, 2 and 3 in the first partition (M1). Similarly, using the state labeled 5 as the new root, we include all transitions from states labeled 5, 6 and 7 in the second partition (M2). The remaining transition, originating from state labeled 4 is included in the third partition (M3). This has the highest cost because the multiplier in the system is included in this partition. Fortunately, this partition has minimal logic for control signals, multiplexers and conditionals.

Note that each partition is much simpler to implement compared to the complete "gcd2" system shown in Figure 2 (a). The control logic for each partition is also simpler because we only need to implement the control signals for that partition. After this step in the synthesis process, each partition is then synthesized independently using available FPGA synthesis tools.

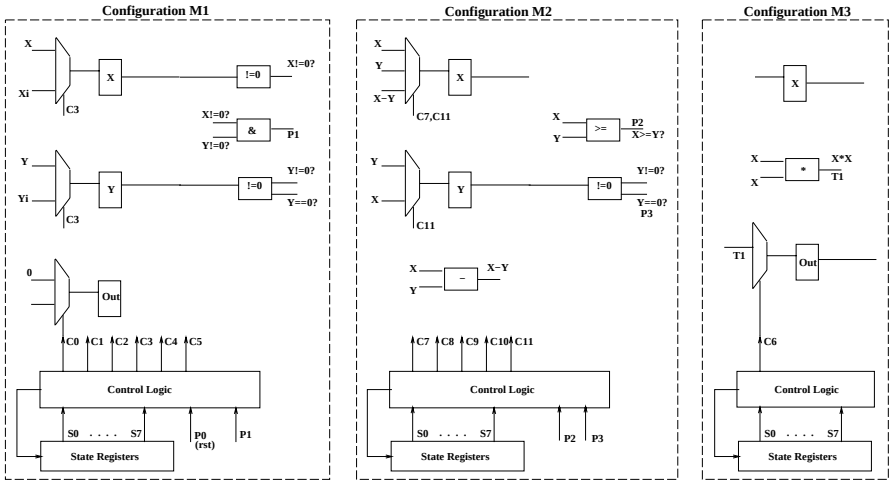


Fig. 3. Three configurations for gcd2

5 Conclusion

The synthesis method described in this paper shows a way of synthesizing sequential machines into reconfigurable implementations. This method was developed as part of our on-going effort to develop tools for designing reconfigurable systems. Our method enables designers to fit designs larger than available FPGA resources on the chip by partitioning the design into run-time reconfigurable subsystems. It also allows designers to implement reconfigurable systems starting with high-level specifications.

Our approach is well suited to design control dominated systems. Data dominated systems, e.g. data-flow machines, will probably be better designed using

techniques that partition and reconfigure along data-flow boundaries. Another limitation of our approach is that we analyze and partition the system specification at synthesis time and not at run-time. This simplifies the design problem but may not provide the most optimal reconfigurable design partitions.

We are currently targeting our tool-set for synthesizing reconfigurable systems on Xilinx 6200 series FPGAs [14]. These FPGAs allow partial run-time reconfiguration of logic blocks, an architectural feature that is required by our methodology.

We believe that good synthesis and run-time tools hold the key to the success of reconfigurable systems in the future. We will refine our methodology to enable data-dominated reconfigurable system design and add support for run-time tools and libraries to ease software development for reconfigurable systems.

References

1. J. R. Hauser and J. Wawrzynek, "Garp: A mips processor with a reconfigurable coprocessor," in *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 24–33, 1997.
2. J. Hadley and B. Hutchings, "Design methodologies for partially reconfigured systems," in *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 78–84, 1995.
3. E. Lemoine and D. Merceron, "Run-time reconfiguration of fpga for scanning genomic databases," in *IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 90–98, 1995.
4. J. Li and R. K. Gupta, "HDL Optimization Using Timed Decision Tables," in *Proceedings of the* 33^{rd} *Design Automation Conference*, pp. 51–54, June 1996.
5. J. Li and R. K. Gupta, "Limited exception modeling and its use in presynthesis optimizations," in *Proceedings of the* 34^{th} *Design Automation Conference*, June 1997.
6. K. Rath, M. E. Tuna, and S. D. Johnson, "Behavior tables: A basis for system representation and transformational system synthesis," in *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, IEEE, Nov. 1993.
7. G. D. Micheli, D. C. Ku, F. Mailhot, and T. Truong, "The Olympus Synthesis System for Digital Design," *IEEE Design and Test Magazine*, pp. 37–53, Oct. 1990.
8. D. Ku, *HardwareC – A Language for Hardware Design Version 2.0*.
9. J. Li and R. K. Gupta, "Timed Decision Table: A Model for System Representation and Optimization," Technical Report UIUCDCS-R-96-1971, University of Illinois, 1996.
10. J. Li and R. K. Gupta, "Decomposition of Timed Decision Tables and its Use in Presynthesis Optimizations," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, November 1997.
11. P. K. Chan and S. Mourad, *Digital Design Using Field Programmable Gate Arrays*. Prentice Hall, 1994.
12. J. Li and R. K. Gupta, "System modeling and presynthesis using timed decision tables," *TVLSI*, 1997. (submitted).
13. S. D. Johnson, *Synthesis of Digital Designs from Recursion Equations*. Cambridge: MIT Press, 1984. ACM Distinguished Dissertation 1984.
14. Xilinx, *XC6200 Data Sheet*, 1996.