A Hybrid Numeric/Symbolic Program for Checking Functional and Timing Compatibility of Synthesized Designs*

Chih-Tung Chen and Alice C. Parker
Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089-2562

Abstract

In this paper, we present an efficient and effective approach for checking synthesized RTL designs. This approach uses a hybrid numeric/symbolic simulation to extract the functional behavior of a design while taking into account the interaction between the data and control paths as well as the clocking scheme and delays, and employs a graph-comparison procedure to perform the checking task. The value of this work was shown by its ability to identify problems with an early version of the ADAM Control Signal Generator (CSG) software, which was then corrected accordingly.

1 Introduction

Due to the cost of engineering and fabrication and the critical marketing time, design errors of digital systems should be eliminated at all costs. Although one may argue that the synthesized designs should be *correct by construction*, in reality there is no such guarantee unless the whole synthesis process, including techniques and programs, can be formally validated. However, to validate a large software system like a high-level synthesis system formally is still impractical, if not impossible, for current formal verification techniques [9]. A more practical alternative is to verify the synthesized designs with respect to their specifications.

Although designs can be verified at various levels of abstraction, it is desirable to find any design problem as early as possible. In addition, a high-level synthesis system may produce many designs for a given set of constraints; without proper verification, there is a lack of sense of correctness while the designs are being evaluated or compared. On the other hand, pure functional validation is not enough because many design errors are also related to the control and timing of the design. Hence, we believe that there is a strong need for an automatic tool which can check both the functionality and timing of synthesized designs efficiently, to be integrated in a high-level synthesis system.

This paper presents an efficient approach for checking the RTL designs produced by the USC ADAM high-level synthesis system. This technique is also applicable to other synthesis systems incorporating a similar design flow. Our approach is motivated by the observation that the structural designs are derived in a well-defined manner from the behavioral specifications [11] in the ADAM system. We found that these RTL designs possess several common properties so that the symbolic simulation technique can be effectively utilized to perform the checking task. Using this approach, we are able to not only verify the design functionality formally but also take into account the interaction between the data path and the controller as well as the timing issues, such as delays and the clocking scheme.

2 Related Work

There are a variety of approaches for design verification (see [3, 10] for a survey). However, formal methods are still not practical at present and simulation-based ones have theoretical limitations. The RLEXT system by Knapp and Winslett [8] is the only one we have found which can achieve some of our objectives. RLEXT is a rule-based system which relies on a formal model similar to the ADAM Design Data Structure (DDS) [7]. It allows the user to modify the design structure and then check the consistency of the design with the ability to repair some design-rule violations automatically. Their approach, however, does

*This work was supported by the Advanced Research Projects Agency and monitored by the Federal Bureau of Investigation under Contract No. JFBI90092. The views and conclusions considered in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

0-8186-5785-5/94 $03.00 © 1994 IEEE

not mention the control logic, nor the interaction between the datapath and the controller. How the timing issues such as the clocking scheme and delays are handled is not described.

In our work, hybrid numeric/symbolic simulation is utilized to check the functional and timing compatibility of the *synthesized* chips. The idea behind symbolic simulation is to evaluate circuit behavior over expanded sets of signal values so that a number of operating conditions can be simulated in a single run. In late 1970's, researchers at IBM first applied symbolic simulation to hardware verification at the register-transfer level [5]. The research activities [2] on this problem only lasted till the early 1980's due to the weakness of the algebraic manipulation and the nondeterminacy caused by conditional and looping constructs [1]. Although this form of symbolic evaluation is not powerful enough for general functional verification at the RT level, we will show in this paper that it can be made very useful when applied to the synthesis domain where both the functional behavior and timing of a design are equally important and the implementation is derived from the specification in a well-defined manner.

3 Properties of Synthesized Designs

The problem which we are solving here can be briefly described as follows:

Show whether or not the RTL implementation I will perform the required computation specified in the control data flow graph $CDFG_S$ for every execution instance.

The implementation I itself is a static structure which consists of a data path and a controller. We are asked in this problem to obtain the dynamic relationship between sequences of inputs and outputs while I is physically operated under the specified clocking scheme and input/output protocol. However, even if we can faithfully obtain this dynamic behavior of I, it is still very difficult to prove the correctness if I and $CDFG_S$ are regarded as two independent entities. This is because the problem is similar to showing whether or not two dataflow graphs are equivalent, which is believed to be *intractable*. Even a simple problem like determining the equivalence of two finite-precision algebraic expressions cannot be done in P nor NP time if $NP \neq co\text{-}NP$ (see [4] for the proofs).

Fortunately, the implementation I is actually the result of a mapping from $CDFG_S$. The major tasks of this mapping involve assigning the operations to control steps (scheduling), assigning the operations and values to hardware (data path allocation and binding), and generating a controller to deliver the required control signals (control synthesis) [11]. Consequently, I, if mapped correctly, will have the following properties:

Property 3.1 *For each operation op in $CDFG_S$, there exists a functional unit u in I such that u can be configured to perform op and op is achieved by directing all the input values of op to the corresponding input ports of u.*

Property 3.2 *For each data dependence (val, op) in $CDFG_S$, there exists an interconnect path in I between the source of val and the corresponding input port of the functional unit u which is designated to perform op. The source of val can be an output port of a functional unit or a storage element. The path is set up by using buses and/or switching devices.*

Property 3.3 *$CDFG_S$ defines the required computations which have to be done in I for every execution instance.*

In fact, the information regarding these properties is generally available after the synthesis process. For example, this information is represented explicitly by means of *bindings* in ADAM's DDS representation.

4 Approach Overview

In order to produce an automatic tool for checking synthesized designs quickly and effectively, we developed an approach which combines symbolic simulation at the RT level with a behavior-comparison procedure based on the properties described in Section 3.

The motivations to use symbolic simulation for our approach are twofold. First, it provides formal results because the simulator operates over a *symbolic* domain, and at the same time we are able to take into account design timing in terms of the clocking scheme, delays, and input/output protocols. Second, the symbolic simulation results are ready for comparison with the design specification for high-level synthesis since they both can be represented in a similar form such as a CDFG.

Figure 1 shows a flow chart which briefly illustrates our approach. First, the design data is read from the DDS. It includes the behavioral specification, the structural implementation, the physical information if available, and the module library. Next, we set up the simulation parameters such as delays and clocking, and the input/output protocol. We estimate the wiring delays if the floorplan is provided. The control flow of the design is analyzed to produce a list of all possible execution paths.

The hybrid symbolic simulation performed next is event-driven. It is a hybrid model because the data

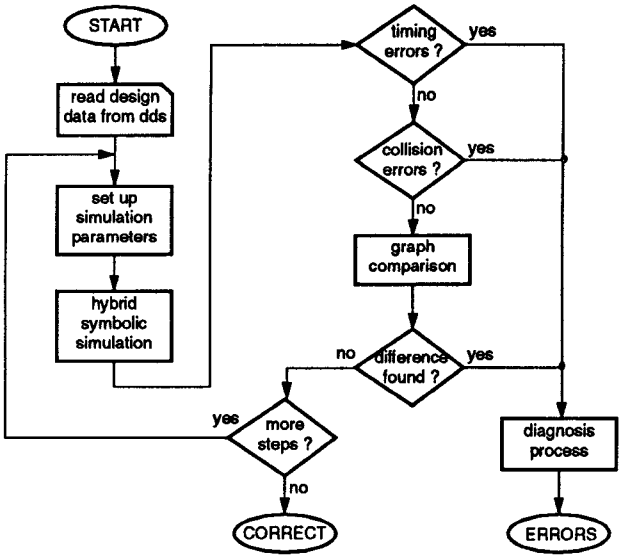


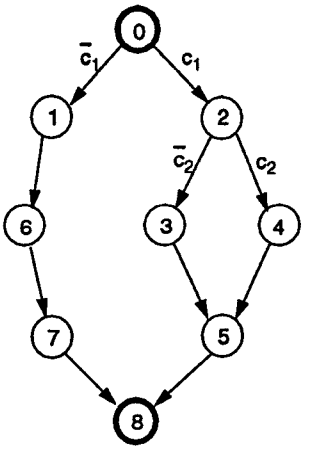
Figure 1: A flow chart of our approach

path is evaluated symbolically but the controller is simulated numerically so that all the control signals will be either 1, 0 or *unknown* throughout the simulation. A transport delay model is used in the simulation. Due to lack of space, the delay modeling and timing checking will not be elaborated in this paper. The simulator also constantly monitors the occurrences of value collision on the wires. If any problem is found during the simulation, a diagnosis procedure is called to determine the cause. The result of the simulation is represented by a dataflow graph which describes the actual data operations and data transfers done by the design.

Finally, the graphical simulation result is compared with the graphical specification. If the comparison procedure finds any difference between these two graphs, a diagnosis procedure is called to find the possible design errors. The whole process is repeated until no more execution paths are left.

5 Hybrid Symbolic Simulation

As we have discussed earlier, what is important for a synthesized design is whether or not it performs the required data operations and the correct sequencing of data transfers for each execution instance. Also, many design errors are related to the control and timing of the design. Hence, we need to be able to extract the circuit behavior in terms of the symbolic data operations and data transfers that occur in the datapath and at the same time to emphasize exercising the control and modeling the timing. The hybrid symbolic simulation to be described here performs exactly this task.



Executaion Paths	Path Condition
0-1-6-7-8	c_1=0
0-2-3-5-8	c_1=1, c_2=0
0-2-4-5-8	c_1=1, c_2=1

Figure 2: Execution paths of a state-transition graph

In our simulation model, the simulation proceeds from one execution path to another.

Definition 5.1 *An execution path is a direct path from an* initial *state to an* end *state in the state-transition graph*[1] *of an FSM controller.*

For example, the state-transition graph shown in Figure 2 contains three execution paths, each of which is associated with a *path condition* that represents the assumptions made along the path.

Definition 5.2 *A path condition is an assignment for a set of Boolean symbolic values which together determine alternative paths through the state-transition graph.*

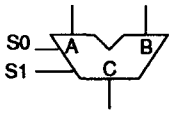
5.1 Element Evaluation

Data Path

The evaluation of a datapath module depends on its behavior model. In DDS, this information is available from the behavioral model of the component used to implement the module. We represent this information internally in the form of function tables. The function table of a datapath module defines the manipulation of symbolic data for each possible condition on the control lines. For example, a simple four-function ALU is shown in Figure 3. If the input condition of a module being evaluated is invalid, its outputs and data storage, if any, are set to *unknown*.

The datapath carriers (nets) are used for propagating the symbolic values. A carrier connecting more than one output port requires those outputs to be tristate. Normally, at most one tristate output is enabled at any time. A *value collision* occurs if two or more

[1]Currently, we require the state-transition graph to be *acyclic*. Techniques to handle cycles are under development.



S0	S1	C
0	0	A + B
0	1	A - B
1	0	A & B
1	1	A | B

propagation delay = 10 ns

Figure 3: A four-function ALU

output ports drive a carrier at the same time [8]. This type of design error can be easily detected by the simulator.

Controller

The controller is evaluated when there is a change on its clock signal. If the clock change results in a state transition, the outputs are computed and the controller moves to the next state in the current execution path. The path condition of the current execution path is updated, if necessary, at each state transition. For example, if the state transition requires the inputs i_1 and i_2 to be 1 and 0 and if the symbolic values currently appearing at i_1 and i_2 are a and b respectively, then $a = 1$ and $b = 0$ will be added to the path condition. On the other hand, if a required input for the state transition contains an *unknown* value, the simulator aborts the current execution path and reports a data-dependency violation.

5.2 Representation of Symbolic Data

In our simulation model, the symbolic values and operations produced during the simulation are used to build a bipartite data flow graph, which is the same representation used for the design specification. A vertex is created in the data flow graph whenever a new symbolic value or operation is produced during the simulation. If a symbolic value is the result of an operation performed by a functional unit, the operation becomes its direct predecessor. Similarly, the symbolic values which appear at the input ports of a functional unit become the direct predecessors of the operation being performed.

Our model is different from the early works [5, 2] on symbolic simulation at the RT level in the following ways:

1. The overhead to propagate the algebraic expressions is eliminated since we focus only on collecting the actual data operations and data transfers that occur in the data path.
2. A powerful algebraic manipulator is not required since we do not try to simplify the expressions during the simulation. Instead, the data flow graph representing the simulation result is compared with the specification using the graph-isomorphism property to be discussed later.

This difference is also the reason that symbolic simulation can be effectively applied to solve our problem.

6 Graph-Based Verification

In our approach, the behavior comparison is based on the data flow graph model. Since the design specification is already represented in this model, only the behavior of the structural implementation has to be extracted and translated into this model. The hybrid symbolic simulation described in Section 5 performs this task for us. Therefore, the verification of the RTL implementation I becomes the problem of comparing two control dataflow graphs, $CDFG_S$ derived from the input specification and $CDFG_I$ derived during the simulation. $CDFG_I$, however, is actually a set of dataflow graphs (DFG), each of which corresponds to the result of simulating I for one of its execution paths.

Because the design I is the result of a mapping from $CDFG_S$, there exists a strong relationship between $CDFG_S$ and $CDFG_I$ (see Section 3). In fact, there is an isomorphic property between them. Consequently, a graph-matching procedure based on this property has been developed to compare them efficiently.

6.1 The Isomorphic Property

From Section 3, we know that the RTL implementation I, if mapped correctly, will have several properties. In summary, I will perform the required computations specified by $CDFG_S$ for every execution instance. The computations are done by making sure the input values of each required operation are available at the corresponding input ports of the designated functional unit which is configured properly.

Let DFG_I be the result of simulating I for one execution instance under the path condition pc. If the specification $CDFG_S$ is interpreted symbolically under the same path condition pc, the result is a data flow graph DFG_S such that the predicate of each operation in DFG_S is evaluated to *true* under pc.

Before we present the isomorphic property between DFG_S and DFG_I, we first need to establish the correspondence for all their primary input/output values. This correspondence is important because it provides the starting point to compare these two graphs.

Lemma 6.1 *There exists an one-to-one correspondence between DFG_S and DFG_I for the primary input/output values.*

The detailed proofs of this lemma and the following theorem can be found in our technical report [4]. In short, this correspondence is established mainly because of the input/output protocol which determines how to apply the input values to I and obtain the corresponding output values.

Intuitively, the isomorphic property between DFG_S and DFG_I exists because I is synthesized in such a way that each required operation in DFG_S will be performed by a designated functional unit at some time and every data dependency will be preserved by establishing a proper interconnection. Hence, if I is synthesized correctly, each of the corresponding primary outputs of DFG_S and DFG_I should have similar geometric properties which will be explained by the following theorem.

Theorem 6.2 *For each pair of the corresponding primary output values (out_S, out_I) of DFG_S and DFG_I, the cones C_S and C_I[2] of out_S and out_I respectively are* isomorphic.

The isomorphic property between C_S and C_I not only implies that there is a one-to-one correspondence between their vertices and edges such that the incidence relationship is preserved, but also requires that each pair of corresponding vertices are compatible. In other words, if the corresponding vertices are operations, they must be of the same type. If they are values, they have the same bitwidths.

6.2 A Graph Matching Procedure

Knowing that there is an isomorphic property between $CDFG_S$ and $CDFG_I$, it becomes straightforward to develop a method for behavior comparison. In fact, all we need to do is to check whether or not the cones of their corresponding output values are isomorphic for all the execution paths.

Unlike the general isomorphism problem in graph theory, which is an important unsolved problem, it is much easier to check the isomorphic property between the cones of the corresponding output values because the correspondences of their primary input and output values are known in advance. Furthermore, the correspondences of two operations can be established as soon as they are of the same type and all their input values are equivalent whereas the vertices in the general problem are typeless. Based on this principle, we developed a procedure for checking the isomorphic property between the cones of two corresponding output values as shown in Figure 4. Let out_S and out_I of DFG_S and DFG_I be a pair of corresponding output values and let C_S and C_I be their respective cones to be checked. This procedure basically tries to establish the correspondences of the operations and values between C_S and C_I incrementally from the primary inputs toward the primary outputs out_S and out_I. If there is any operation or value whose correspondence is not established (no ID is tagged), this procedure returns *false*; otherwise, C_S and C_I are equivalent and *true* is returned.

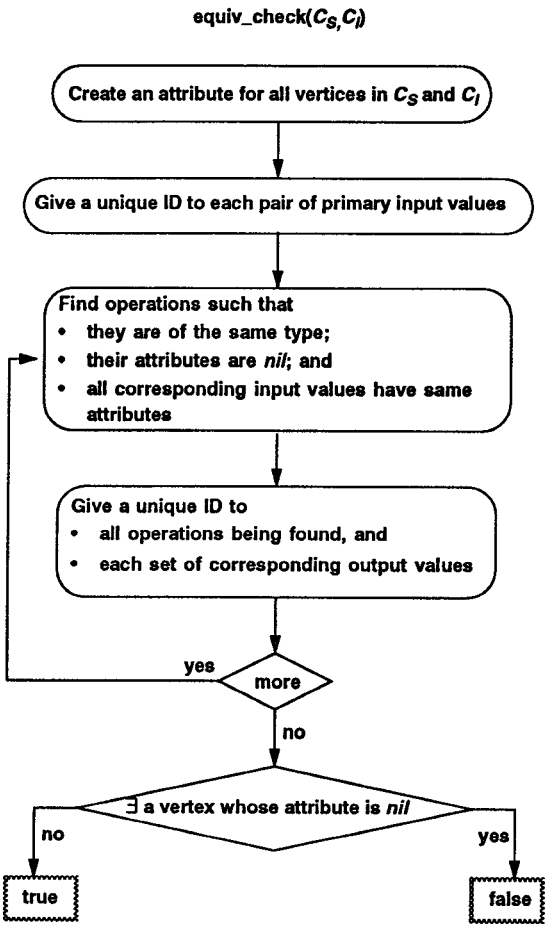


Figure 4: A procedure for the isomorphic check

7 Experiments

In order to show the effectiveness of our approach, we performed a number of experiments with the designs synthesized from the USC ADAM system. In fact, our preliminary experiments immediately identified that the controllers generated by the ADAM control signal generator (CSG) tool [13] for these designs were incorrect. CSG was then revised accordingly.

[2] A cone of a vertex v in a graph $G = (V, E)$ is a subgraph $C = (V', E')$ such that

1. $V' = \{ v \} \cup predecessors(v)$
2. for all v_1, v_2 in V' if edge (v_1, v_2) in E then (v_1, v_2) is also in E'.

Shortly after CSG was revised, we experimented with a non-pipelined AR filter. MAHA [12] was used for scheduling and MABAL [6] for datapath allocation and binding. This design is characterized as follows:

- It has 4 time steps.
- Both the input and output values are not latched.
- A two-phase non-overlapping clocking scheme is used.

The controller generated by CSG contains only one execution path with four states. The experiment was carried out by holding the input values (symbolic) at the input ports during the execution and obtaining the symbolic values from the output ports at the end of the 4th clock cycle. The cones of these output values were then extracted from the data flow graph which was built during the simulation and compared correctly with the ones specified in the original data flow graph.

We also experimented with a robot arm controller whose control flow is much more complex than the previous one. The design was synthesized in a similar way except we required the inputs values to be latched. The RTL implementation has 12 time steps and 16 possible execution paths. The controller was generated by CSG using status registers. We were able to verify this RTL implementation with the following conclusions:

- All the constant values were required to be supplied externally, which results in inefficient use of input ports.
- Conditional values were unnecessarily routed to the output ports.
- Some of the input values were not latched as specified.

We are continuing to experiment with other designs under development, including video compression chips.

8 Conclusions

In this paper, we have identified the properties of automatically synthesized RTL designs to facilitate the verification task. We also presented a hybrid symbolic/numeric approach for checking both the functional and timing compatibility of synthesized designs. Several experiments have been conducted using this approach, and the results indicate that our approach is indeed effective and efficient.

Further experiments with designs whose timing is critical should be performed to demonstrate our ability and the advantage to take into account design timing during verification. In addition, data-dependent delays and loops needs to be handled in the future.

References

[1] R.E. Bryant. Symbolic Simulation - Techniques and Applications. In *27th ACM/IEEE Design Automation Conference*, 1990.

[2] W.E. Cory. Symbolic Simulation for Functional Verification with ADLIB and SDL. In *18th ACM/IEEE Design Automation Conference*, 1981.

[3] P. Camurati and P. Prinetto. Formal Verification of Hardware Correctness: Introduction and Survey of Current Research. IEEE *Computer*, July 1988.

[4] C.T. Chen and A.C. Parker. A Symbolic Approach for Checking Functional and Timing Compatibility of Synthesized Designs. Technical Report CEng 93–26, Univ. of Southern California, May 1993.

[5] J.A. Darringer. The Application of Program Verification Techniques to Hardware Verification. In *16th ACM/IEEE Design Automation Conference*, 1979.

[6] K. Küçükçakar and A.C. Parker. MABAL - A Software Package for Module And Bus ALlocation. In *Int. Journal of Computer Aided VLSI Design*, June 1989.

[7] D.W. Knapp and A.C. Parker. A Unified Representation for Design Information. In *CHDL-85*, Elsevier, 1985.

[8] D.W. Knapp and M. Winslett. A Formalization of Correctness for Linked Representations of Datapath Hardware. In *IFIP Workshop on Applied Formal Methods for Correct VLSI Design*, November 1989.

[9] M.C. McFarland. Formal Verification of Sequential Hardware: A Tutorial. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 633–654, May 1993.

[10] M.C. McFarland. Practical Lessons in Verification and High-Level Synthesis. AT&T Bell Laboratories, Murray Hill, NJ 07974-2070.

[11] M.C. McFarland, A.C. Parker, and R. Camposano. Tutorial on High-Level Synthesis. In *25th ACM/IEEE Design Automation Conference*, 1988.

[12] A.C. Parker, J. Pizarro, and M.J. Mlinar. MAHA: A Program for Datapath Synthesis. In *23th ACM/IEEE Design Automation Conference*, 1986.

[13] J.P. Weng and A.C. Parker. CSG: Control Path Synthesis in the ADAM System. Technical Report CEng 92–03, Univ. of Southern California, April 1992.