Automated Formal Verification of Scheduling Process using Finite State Machines with Datapath (FSMD)*

Youngsik Kim, Shekhar Kopuri and Nazanin Mansouri
Electrical Engineering and Computer Science
Syracuse University
Syracuse, New York, 13244, USA

Abstract

This paper presents a methodology for the formal verification of scheduling during High-Level Synthesis(HLS). A notion of functional equivalence between two Finite State Machines with Datapath (FSMDs) is defined, on the basis of which we propose a methodology to verify scheduling. The functional equivalence between the behavioral specification and the scheduled Control-Data Flow Graph (CDFG) - that is the result of scheduling - is established using their FSMD models. The equivalence conditions are mathematically modeled and implemented in the higher-order specification language of theorem proving environment PVS[11], integrated with a HLS tool. The proof of correctness of the design is subsequently verified by the PVS proof checker.

1 Introduction

High-Level Synthesis (HLS) identifies and assigns time steps to all operations of a given behavioral description ($scheduling$) and then binds them to individual resource instances from library of components ($allocation$ or $binding$) to automatically generate Register-Transfer Level designs. Verification of synthesis can be performed by establishing equivalence between the result of each step and the design specification that is the input of the synthesis tool, if each of these is represented using the same abstract models. Verification by this method has been demonstrated to be highly scalable in [1][9][10].

FSMD, that extends the Finite State Machine(FSM) model, is known to be a simple and extremely efficient way to represent complex designs comprehensively[4]. Unlike FSMs that often model the control flow, FSMDs capture the the data-flow of a design too. Moreover, in modeling the sequential elements of the data-path, each bit is NOT directly translated into two states alleviating the state explosion problem. We use FSMD models to represent the three different design abstractions along the HLS process viz, the initial design and the design after scheduling and binding. Verification of HLS is done by establishing the equivalence of FSMDs between any two consecutive design abstractions. This approach, however, cannot always be successfully applied for two FSMD models that have the same functionality for the reason explained below.

HLS can be seen as stepwise transformation of a behavioral specification into a structural implementation as shown in Fig.1. The intermediate result produced after each step is modeled by an automaton. Each automaton is a FSMD as described in [4][2]. Three different automata represent the behavior (M), the scheduled CDFG (M_s) and the final CDFG after scheduling and binding (M_b). The scheduling process may result in movement of operations into different states. In Fig.1, for example, after scheduling, the shift-right operation in state s_j and the addition operation in state s_l of the behavior M have moved to states s_i and s_k, respectively. Hence, two states s_j and s_l present in M were removed in M_s. This causes a difference in the number of states and the behavior of the two FSMDs in each state. Similar problem does not arise between M_s and M_b as is evident by Fig.1. Since the correspondence between the states is lost after scheduling, the two FSMs M and M_s are not equivalent based on the conventional definition of FSM equivalence. This definition is, however, too restrictive as the equivalence condition might fail for designs that are *functionally* equivalent. Therefore, a definition of equivalence between FSMDs that captures the notion of functional equivalence is needed.

In this paper, we propose a novel approach to verify equivalence between FSMDs representing automata M and M_s. The notion of *functional equivalence* of FSMDs is also defined. Both FSMDs for M and M_s are converted into a set of equal number of higher-order logic predicates. The functional equivalence is then proved based on equivalence of each pair of predicates. Defining each predicate in a re-

*This work was partly supported by a grant from MDC-NYSTAR, under award number 411903-G and the New York State Center for Advanced Technology in Computer Applications and Software Engineering (CASE).

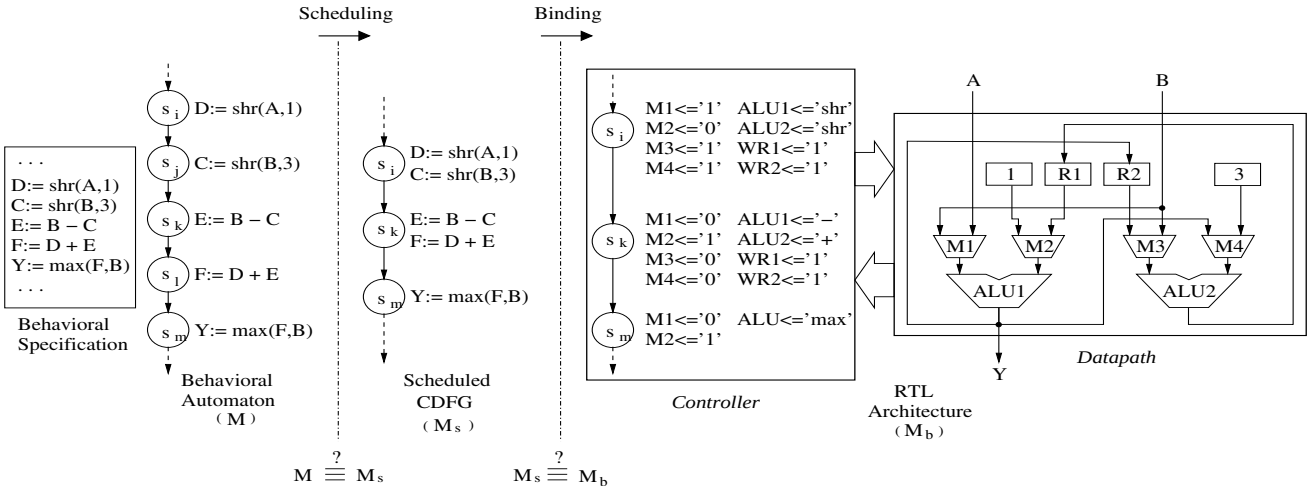

Figure 1. Stepwise Transformation during High-Level Synthesis Process

cursive way to facilitate automation of verification process is significant.

The rest of the paper is organized as follows. Related works are mentioned in Section 2. Section 3 describes the FSMD model formalization and the notion of FSMD equivalence. Section 4 describes the formalization of the functional equivalence of the two FSMDs. The implementation of the equivalence model is discussed in section 5. Section 6 presents the conclusions.

2. Related work

Model checking was used to perform equivalence checking between behavioral specification and RTL implementation of designs in [1], and also to validate the controllers generated by the HLS system (DSS) in [10]. Mansouri et al suggested a formal verification methodology for HLS process using PVS prover [9] by checking the equivalence between behavior specification and RTL implementation. FSM verification methodology based on failure state reachability of the product machine was presented in [8]. This is, however, limited to FSM models and cannot exploit the benefits of FSMD models.

Borrione et al presented the formal verification methodology of establishing equivalence between FSMDs representing scheduled CDFG and RTL implementation after binding[2]. A fully automatic method for the equivalence checking between designs before and after scheduling in HLS was presented in [3], using a custom-defined language, LLS (Language of Labeled Segments) instead of FSMDs which led to lack of design compatibility. While all these methods target a similar verification problem, we believe that a completely automated verification of scheduling process using the FSMD model is unique to this work.

3 Formalization of FSMD Equivalence

FSMD is an extension to FSM. Usually, FSM models provide the required precision for formal reasoning of designs. But, they are susceptible to state explosion problems and hence cannot be employed to large designs. FSMDs, on the other hand, have states encoded as variables, thus alleviating the state explosion problem and hence making formal verification of large designs possible. The definition of the FSMD equivalence is presented as an extension to FSM equivalence below.

3.1 FSM Model and its Equivalence

Here, we briefly review the definitions of FSM and FSM equivalence as given in [6] and [7].

1. **FSM model** : A FSM is defined as a quintuple $M = \langle S, \Sigma, \Delta, f, h \rangle$ where,
   S, Σ, and Δ are finite, nonempty sets of states, input values, and output values, respectively;
   $f : S \times \Sigma \rightarrow S$ is the next-state function;
   $h : S \times \Sigma \rightarrow \Delta$ is the output function.

2. **States equivalence** : "States s_i and s_j of machine $M = (S, \Sigma, \Delta, f, h)$ are said to be equivalent if and only if, for every possible input sequence, the same output sequence will be produced regardless of whether s_i or s_j is the initial state"[6]:

   $$s_i \equiv s_j \Leftrightarrow$$
   $$\forall (i \in \Sigma) : [h(s_i, i) = h(s_j, i) \ \wedge \ f(s_i, i) \equiv f(s_j, i)]$$

   A state s of a machine $M = (S, \Sigma, \Delta, f, h)$ and a state s' of a machine $M' = (S', \Sigma, \Delta, f', h')$ are equivalent, if given equal input values; 1)they lead to the same output values, and 2)their successor states are also equivalent:

$s \equiv s' \Leftrightarrow$
$\forall(i \in \Sigma) : [h(s,i) = h'(s',i) \ \wedge \ f(s,i) \equiv f'(s',i)]$

3. **FSM equivalence** : "Two machines, $M = (S, \Sigma, \Delta, f, h)$ and $M' = (S', \Sigma, \Delta, f', h')$ are said to be equivalent if and only if, for every state in M, there is a corresponding equivalent state in M' and vice versa."[6]:

$B_s : S \rightarrow S'$
$M \equiv M' \Leftrightarrow \forall(s \in S, s' \in S', i \in \Sigma) :$
$[B_s(s) = s' \rightarrow (h(s,i) = h'(s',i) \wedge$
$B_s(f(s,i)) = f'(s',i))]$

Note that in the above definition: (1) $B_s : S \rightarrow S'$ is a bijective mapping function between the state of two machines, (2) this definition only applies to *compatible machines*, i.e., machines which have the same input and output valuation sets.

3.2 FSMD model and its Equivalence

Our verification method uses a FSMD that is an extension/variation of [4] and [2]. It is defined as a six-tuple $M = \langle S, \Sigma, \Delta, V, f, h \rangle$ where,
S is the set of states;
Σ is the set of all valuations of the inputs;
Δ is the set of all valuations of the outputs;
$X = \{x_1, x_2, ...\}$ is the set of all variables;
$D = D_1 \times D_2 \times ....D_n$ is the variable domain (where D_k denotes the domain of variable x_k);
$V = \{\langle v_1, .., v_n \rangle \mid v_1 \in D_1 \wedge ... \wedge v_n \in D_n\}$ is the set of all valuations of variables;
$f : S \times \Sigma \times V \rightarrow S$ is the next state function;
$h : S \times \Sigma \times V \rightarrow \Delta \times V$ is the output and variable update function.

A valuation $\langle v_1, v_2, ..., v_n \rangle$ is an assignment of values to variables $x_1, x_2, ...,$ and x_n from their respective domains. The input and output valuations are similarly defined. Following state equivalence approach, the equivalence of FSMDs is defined similar to the equivalence of FSMs.

Definition Given $M = (S, \Sigma, \Delta, V, f, h)$ and $M' = (S', \Sigma, \Delta, V, f', h')$ and a mapping $B_s : S \rightarrow S'$ between their states, the two FSMDs are called equivalent if for an arbitrary input sequence applied to both automata, the same output sequence results:

$M \equiv M' \Leftrightarrow \forall(s \in S, s' \in S', i \in \Sigma, v \in V).$
$[B_s(s) = s' \rightarrow (h(s,i,v) = h'(s',i,v) \wedge$
$B_s(f(s,i,v)) = f'(s',i,v))]$

Here, compared to FSM, the compatibility is defined by having the same input and output valuation sets as well as the same variable valuation sets.

```
entity sqrt_part is
   port (A, B : in integer ;
         X : in bit ;
         Y : out integer );
end sqrt_part;

architecture arch of sqrt_part is
begin

   P0 : process
      variable C, D, E, F : integer;
      variable Timer : integer

   begin
      wait until X = 1;
      if (Timer != 0) then
         D := shr( A, 1);
         C := shr( B, 3);
         E := B – C;
         F := D + E;
         Y:= max( F, B);
      else
         Timer := Timer – 1;
      end if;
   end process P0;
end architecture arch
```

Figure 2. Sample VHDL input description

4 Functional Equivalence of FSMDs

Fig.2 shows the behavioral description of a design in VHDL, borrowed from [4]. To project the problem and illustrate our methodology, this example will be used throughout the paper. The FSMD models of the behavioral specification (M) and the scheduled CDFG (M_s) for this example are shown in Fig.3 and Fig.4, respectively. The equivalence between the FSMD models of scheduled CDFG(M_s) and bound CDFG(M_b) can be established using the definition of FSMD equivalence as defined in the previous section. The same however is not possible between the FSMD models of behavior CDFG(M) and scheduled CDFG(M_s) because: 1)More than one states of behavior (M) might get merged into a single state in M_s after scheduling due to assignment of same time step to them. For example, $sb2$ and $sb4$ in Fig.3 merge into $ss4$ in Fig.4 after scheduling, 2) A compound operation in behavior might be broken into simpler operations, each scheduled at a different time step in M_s. In other words the bijective relation between the states of the two machines is lost. This, however, does not mean that the two FSMDs are not functionally equivalent. Motivated by this, we define a less constrained equivalence relation between the FSMDs, where two machines with different number of states may be considered *functionally equivalent*.

4.1 State Transition Predicate

Our fundamental idea of functional equivalence for two FSMDs is based on introducing the predicate logic. Predicate calculus can be used to describe hardware structures [5]. The basic principles are:

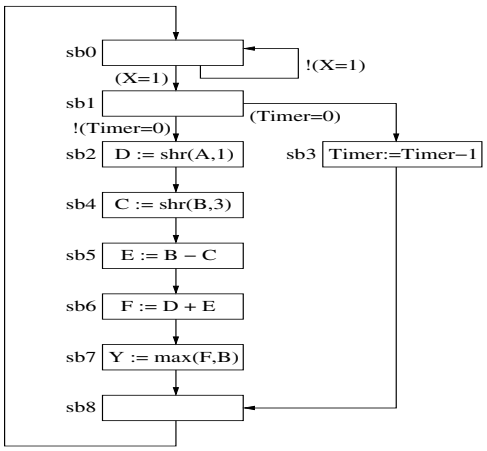

Figure 3. Behavioral FSMD Automaton

- Modules are predicates with their primary inputs and outputs form the parameters of predicates.
- Primary inputs and outputs are universally quantified while internal connections are hidden using existential quantifiers.
- Connections between modules are done by sharing names.
- Composition is done with logical conjunction.

Since the values generated by the next state function f and the output/update function h in one state transition of a FSMD are used as inputs of f and h in the next transition, we can draw an analogy between FSMD and iterative networks. An iterative network is a digital structure with cascaded identical circuit modules[6]. More precisely, one particular state transition of FSMD can be seen as one independent module. This is depicted in Fig.5.

Analogous to iterative networks, i and o form the input and output vectors between environment and module. Also,

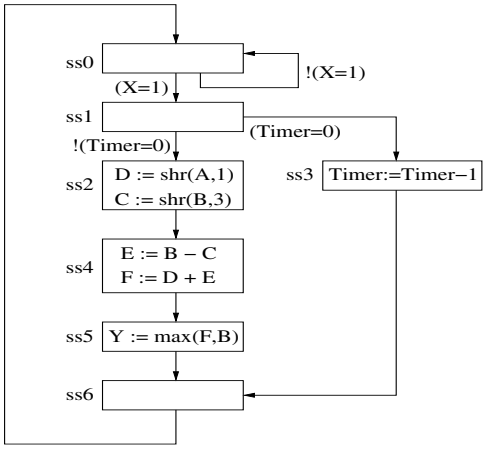

Figure 4. FSMD Automaton after Scheduling

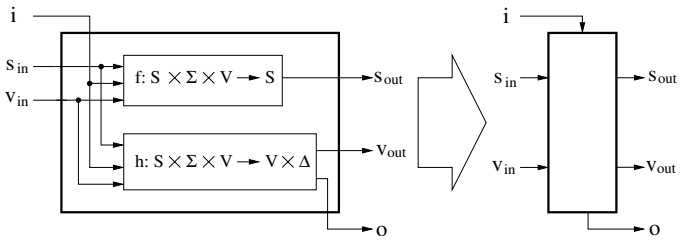

Figure 5. Module Representation of FSMD

s and v are the intermodule carriers representing a state and a valuation of the variables. We can further clarify this using s_{in}, v_{in}, s_{out} and v_{out} in Fig.5. Consecutive modules with same inputs that remain constant during the transitions, producing a single valid output at the end, can be grouped into a single larger module. Fig.6 depicts one module representing a *state transition path* consisting of three consecutive state transitions. Except the last state transition, the valuations of s and v are seen as internal connections that can be hidden by existential quantification as it is done for internal connections between submodules of hardware modules. The state transition path predicate ($path$) for this compound module is given as:

$$\begin{aligned}
&\forall(s_{in}, s_{out} \in S, i \in \Sigma, o \in \Delta, v_{in}, v_{out} \in V) : \\
&\quad path(i, s_{in}, v_{in}, o, s_{out}, v_{out}) \equiv \\
&\qquad \exists(s_1, s_2 \in S, v_1, v_2 \in V) : \\
&\qquad\quad [f(s_{in}, i, v) = s_1 \wedge v_member(h(s_{in}, i, v)) = v_1 \wedge \\
&\qquad\quad\ f(s_1, i, v_1) = s_2 \ \wedge v_member(h(s_1, i, v_1)) = v_2 \wedge \\
&\qquad\quad\ f(s_2, i, v_2) = s_{out} \ \wedge \ h(s_2, i, v_2)) = \langle v_{out}, o\rangle]
\end{aligned}$$

where, v_member is the function that takes the outputs of h function and only returns the variable valuation such that:

$$\forall(v \in V, o \in \Delta) : v\_member(\langle v, o\rangle) = v$$

It is worthwhile to note that since the behavior of FSMD in every state is defined with the same function f and h, the predicate representing a state transition path with n transitions given by a compound module can be defined recursively over the index n. When $n = 1$ we have the base case. This corresponds to the last state transition where the values of the outputs must also be checked apart from the values of the next state and variables, unlike in the case of intermediate state transitions. The definitions of the base case (single

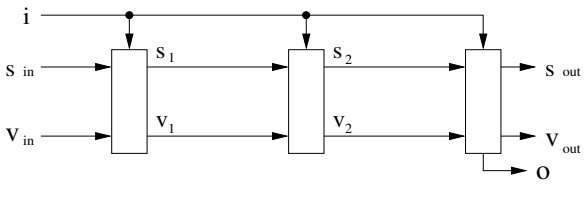

Figure 6. State Transition Path with Three Consecutive State Transitions

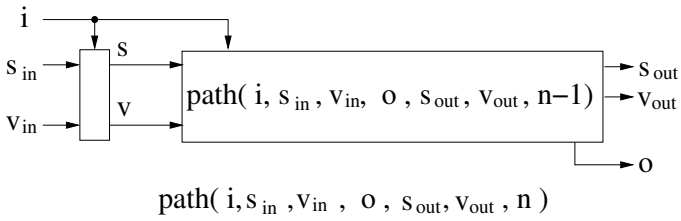

Figure 7. Recursive Definition of State Transition Path Predicate

transition) and the general case (a path consisting of a sequence of n transitions) are as follows:

$$path(i, s_{in}, v_{in}, o, s_{out}, v_{out}, 1) \equiv$$
$$[f(s_{in}, i, v_{in}) = s_{out} \wedge h(s_{in}, i, v_{in}) = \langle v_{out}, o \rangle]$$

$$path(i, s_{in}, v_{in}, o, s_{out}, v_{out}, n) \equiv$$
$$\exists (s \in S, v \in V) \,.\, [f(s_{in}, i, v_{in}) = s \wedge$$
$$v\_member(h(s_{in}, i, v_{in})) = v \wedge$$
$$path(i, s, v, o, s_{out}, v_{out}, n-1)]$$

and these are illustrated in Fig.5 and Fig.7, respectively. Note that in these definitions a new parameter is added to the path predicate to denote the number of transitions on the path. Having defined the structure of a generic state transition path predicate, we can model the entire FSMD as a set of path predicates.

4.2 Breaking FSMD into a Set of State Transition Paths

The behavior automaton M, can be decomposed into several sub-automata by breaking it in 1) the initial state, 2) the states at which I/O occur, and 3) conditional states. The state transition path between any two break-points is then formally described by a path predicate, p_i(as defined in previous section). This break-up for our example is illustrated in Fig.8. The predicate for M is constructed as a conjunction of all path predicates p_i. The predicate describing M_s, the FSMD of the scheduled CDFG, is constructed similarly using all p_i^s, the path predicates for sub-automata of M_s. It is significant to note that both M and M_s will have the same number of breaking points because conditional states and states where I/O occur are not altered during scheduling. Consequently, the two sets $P = \{p_i \mid p_i$ is a path predicate of $M\}$ and $P_s = \{p_i^s \mid p_i^s$ is a path predicate of $M_s\}$ have the same cardinality. Therefore, we can define a bijective mapping $\beta_p : P \rightarrow P_s$ between these two sets. Functional equivalence between M and M_s can now be established by verifying the equivalence amongst corresponding pairs of p_i and p_i^s.

Table 1. The Sets of State Transition Paths

M, State Transition Path	Initial State (S_{in})	Number of State (Nst)	Last State (S_{out})	Path Predicate Condition ($Cond$)
p_1	sb0	1	sb0	!(X=1)
p_2	sb0	1	sb1	(X=1)
p_3	sb1	5	sb7	!(Timer=1)
p_4	sb7	2	sb0	TRUE
p_5	sb1	3	sb0	(Timer=0)

(a) P : The set of state transition path of M

M_s State Transition Path	Initial State (S_{in})	No. of State (Nst)	Last State (S_{out})	Path Predicate Condition ($Cond$)
p_1^s	ss0	1	ss0	!(X=1)
p_2^s	ss0	1	ss1	(X=1)
p_3^s	ss1	3	ss5	!(Timer=1)
P_4^s	ss7	2	ss0	TRUE
p_5^s	ss1	3	ss0	(Timer=0)

(b) P_s : The set of state transition path of M_s

4.3 Formalization of the Verification Technique

Each path predicate is represented as a four-tuple comprising of the initial state of the path, s_{in}, the number of intermediate state transitions on the path n, its final output state s_{out}, and the condition under which the sequence of the transitions on the path will occur, $cond$;

$$\langle s_{in}, n, s_{out}, cond \rangle \Leftrightarrow$$
$$\forall (i \in \Sigma, o \in \Delta, v_{in}, v_{out} \in V) :$$
$$[cond \rightarrow path(i, s_{in}, v_{in}, o, s_{out}, v_{out}, n)]$$

The predicate sets P and P_s are constructed as shown in Table 1 for our example.
Therefore, the functional equivalence of two FSMDs is defined as;

Definition *The behavioral FSMD (M) and the scheduled FSMD (M_s) are said to be* ***functionally equivalent*** *if and only if each of their corresponding pairs of state transition path predicates are equivalent, given that the conditions for both paths are true*:

$$M \equiv M_s \Leftrightarrow$$
$$\forall (p \in P) \; : \; [\forall (i \in \Sigma, o \in \Delta, v_{in}, v_{out} \in V) \; :$$
$$[(Cond(p) \wedge Cond(\beta_p(p))) \rightarrow$$
$$(path_{beh}(i, S_{in}(p), v_{in}, o, S_{out}(p), v_{out}, Nst(p)) \equiv$$
$$path_{sch}(i, S_{in}(\beta_p(p)), \beta_v(v_{in}), o, S_{out}(\beta_p(p)),$$
$$\beta_v(v_{out}), Nst(\beta_p(p))))]]$$

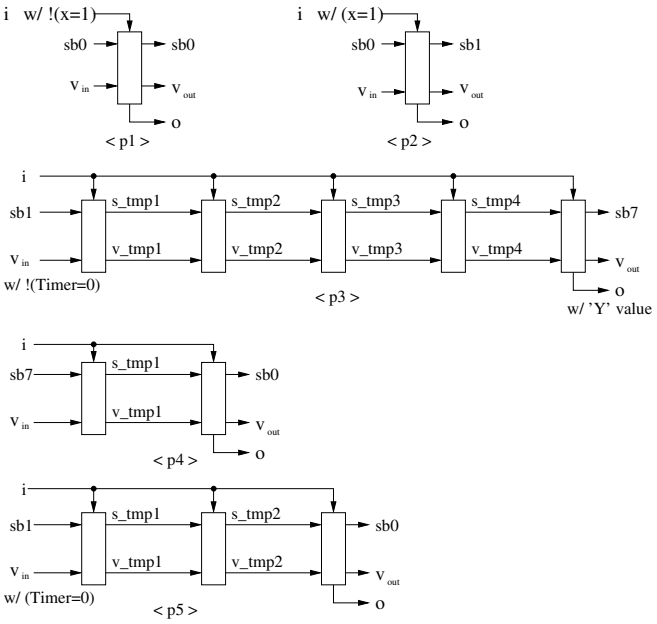

Figure 8. Break-up of FSMD Automaton of Behavioral Specification

where, for any given path predicate, S_{in} returns s_{in}, Nst returns n, S_{out} returns s_{out}, $Cond$ returns $cond$, and β_v returns the mapping between variables of M and M_s, taking into account the possibility of intermediate variables being introduced during scheduling.

5 Implementation

The FSMD models of the behavior specification of a design given in VHDL (M) and the scheduled CDFG (M_s) generated by HLS tool are extracted, and formulated in PVS language. The set of state transition paths and the set of transition conditions for the paths are then extracted as described in previous section. Also, the mapping function between the transition paths, β_p, and between the variables, β_v, of the two machines are defined. Then, exploiting the recursive nature of the path predicates, and using the standard proof template by induction, the proof script of equivalence theorem for each corresponding pair of transition paths is automatically generated.

This method has been integrated with a HLS tool. No human intervention is required from the input of behavioral specification in VHDL till the generation of the scheduled CDFG and the equivalence theorems along with their proof scripts. The equivalence theorems are then loaded into the PVS system and the proofs are performed using the generated proof scripts. The methodology has been applied for verification of synthesis results with various scheduling algorithms.

6 Conclusion

In this paper, a method for formal verification of scheduling in HLS was presented. Since automation is the key in formal verification of the synthesis, the significance of this work is the induction-based mathematical model of FSMD that makes formal verification of scheduling results amenable to automation. Based on this, we introduced a less constrained definition of functional equivalence between two FSMDs, making the equivalence checking between FSMDs applicable to more practical verification problems. However, the scope of this approach is not limited to high-level synthesis. Since FSMD is highly suitable for modeling the designs at any abstraction, designs at the same or different levels can be modeled as FSMDs, and their functional equivalence can be verified automatically.

Our future work includes the extension of the method to accommodate verification of scheduled CDFGs with pipelined control constructs, and the CDFGs that have undergone aggressive code motion (e.g. speculation, reverse speculation).

References

[1] P. Ashar, S. Bhattacharya, A. Raghunathan, and A. Mukaiyama. Verification of RTL generated from scheduled behavior in a high-level synthesis flow. In *Proc. ICCAD*, pages 517–524, 1998.

[2] D. Borrione, J. Dushina, and L.Pierre. A compositional model for the functional verification of high-level synthesis results. In *IEEE Transactions on VLSI System*, volume 8, Oct. 2000.

[3] H. Eveking, H. Hinrichsen, and G. Ritter. Automatic verification of scheduling results in high-level synthesis. In *Proc. DATE*, Mar 1999.

[4] D. Gajski and L. Ramachandran. Introduction to high-level synthesis. *IEEE Design and Test of Computers*, Winter, 1994.

[5] M. Gordon. Why higher-order logic is a good formalism for specifying and verifying hardware. Technical Report 77, Univ. of Cambridge, 1985.

[6] Z. Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, 1978.

[7] T. Kropf. *Introduction to Formal Hardware Verification*. Springer, 1998.

[8] N. Kumar and R. Vemuri. Finite state machine verification on MIMD machine. In *Proc. European Design Automation Conference*, pages 514–520, Sep 1992.

[9] N. Mansouri and R. Vemuri. Automated correctness condition generation for formal verification of synthesized RTL designs. *Formal Methods in System Design*, 2000.

[10] N. Narasimhan, R. Kalyanaraman, and R. Vemuri. Validation of synthesized register-transfer level designs using simulation and formal verification. In *Proc. High Level Design Validation and Test Workshop*, Nov 1996.

[11] S. Owre, N. Shankar, J. Rushby, and D.Stringer-Calvert. *PVS Language Reference*. SRI International, Sep. 1999.