

Synthesis for Logical Initializability of Synchronous Finite State Machines*

Montek Singh Steven M. Nowick

Department of Computer Science
Columbia University
New York, NY 10027

Abstract

We present a new method for the synthesis for logical initializability of synchronous state machines. The goal is to produce a gate-level implementation that is initializable when simulated by a 3-valued (0,1,X) simulator. We build on the approach of Cheng and Agrawal who constrain state assignment to translate functional initializability into logic initializability [4, 5]. We propose an alternative method which is guaranteed safe and not as conservative. In addition, we propose necessary and sufficient conditions on 2-level and multi-level logic synthesis to insure 3-valued simulation succeeds.

1 INTRODUCTION

It is well known that state encoding can influence the logical initializability of an FSM implementation [4, 5]. If the sole objective of an “optimal” state assignment is to minimize the amount of logic, one may end up with implementations that are logically uninitializable. That is, a 3-valued (0,1,X) logic simulator may not be able to initialize the circuit even when its FSM has a synchronizing sequence.

Traditionally, several different approaches to initializability have been used. Each of these assumes different models of initializability (such as single- or multi-vector) and of simulation (such as true-value or 3-value). Further, while the Wehbeh and Saab [3] algorithm analyzes the gate-level circuit to determine if it is initializable, there are other methods that enable one to *synthesize for initializability*. We build on the approach of Cheng and Agrawal [4, 5] who produce a state assignment to translate a functionally initializable finite state machine into a logically initializable gate-level circuit.

The property of initializability is useful for several reasons. It is needed for physically resetting machines should they go out of synchronism. Logical initializability is required for several fault simulators and non-scan ATPG’s, such as STG [2] and CONTEST [1] to work effectively.

There are two views on initializability. *Functional* initializability is a top-level property that implies existence of a synchronizing input sequence which takes the *state machine* into a unique state irrespective of the initial state. On the other hand, *logical* initializability is a gate-level property, that implies the *physical circuit* will be initialized to a unique state under *3-valued simulation*. Logical initializability requires functional initializability, and is, therefore a stronger property. Our focus in this paper will be on synthesis for logical initializability.

1.1 Contributions of this paper

In previous work [4, 5], a method was proposed which attempted to translate functional initializability into logical initializability by imposing certain constraints on state assignment. We demonstrate that these constraints are neither necessary nor sufficient. We propose an alternative method which is not as conservative and is guaranteed to be safe.

In addition, in [5], it was indicated that combinational logic synthesis influences initializability under 3-valued simulation. However, the technique of 2-level logic synthesis proposed in [5] is a heuristic one, and not guaranteed to succeed. In this work, we present *precise* (necessary and sufficient) conditions on 2-level and multi-level logic synthesis to insure logical initializability.

In short, given a functionally initializable finite state machine, our synthesis technique guarantees a logically initializable gate-level circuit. It handles both single-vector as well as multi-vector initializability.

2 PREVIOUS WORK

A typical synthesis path consists of several steps. Initializability considerations can be incorporated at various levels, as shown in Fig. 1. The figure also shows some of the recent work on initializability at different levels in the synthesis path.

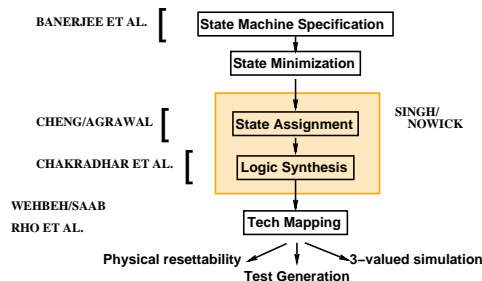


Figure 1: Synthesis for initializability

The method of Rho *et. al.* [6] analyzes a functional description of a state machine and generates *functional* initialization sequences, if any. This method uses BDD’s and produces minimum-length sequences. Wehbeh and Saab [3] present a method which determines if a gate-level implementation is initializable. It is able to generate both *functional* and *logical* initialization sequences.

The above two methods are not synthesis-for-initializability techniques; they are *analysis* techniques to

*This work was supported by an NSF CAREER Award MIP-9501880 and by an Alfred P. Sloan Research Fellowship.

find initialization sequences. The method of Chakradhar *et. al.* [7], on the other hand, is a *synthesis* method for *logical* initializability. It targets asynchronous design and is essentially a search procedure for finding initialization sequences and concomitant don't-care assignments for initializability.

Cheng and Agrawal [5] introduce a synthesis-for-initializability method for synchronous circuits. This method includes a novel state assignment step to insure *logical* initializability.

Banerjee *et. al.* [8] present a technique that targets the highest level in the synthesis path—it modifies the top-level functional specification (signal transition graph) for initializability of asynchronous circuits. However, initializability is achieved only at the cost of some reduction in concurrency.

The synthesis procedure we present in this paper builds on the state assignment method of Cheng and Agrawal [4, 5]. Additionally, our method provides for a complete combinational logic synthesis step, which is shown to be critical for logical initializability.

3 BACKGROUND

The Cheng–Agrawal Method

In this Section we review the Cheng and Agrawal state assignment method [4, 5]. The basic approach is as follows: given a finite state machine and a synchronizing sequence, constrain the state encoding step to insure logical initializability.

We first show how state encoding can affect logical (3-valued) initializability. Consider the functionally initializable machine M in Fig. 2. At startup, the machine can be in any state, i.e. the initial *state group* consists of all states: $(S_1 S_2 S_3 S_4)$. $I = 100$ is a *synchronizing sequence* of M . The following is the trace of state groups that result as the input sequence 100 is applied to M :

$$(S_1 S_2 S_3 S_4) \xrightarrow{1} (S_1 S_2 S_3) \xrightarrow{0} (S_1 S_4) \xrightarrow{0} (S_4)$$

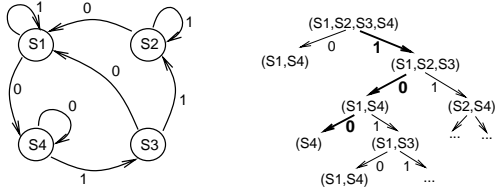


Figure 2: Example FSM and synchronization tree.

Associated with each state group, after state assignment, is its smallest containing cube, called the *group face*. Each group face is represented by a 3-valued vector. Thus, if state encoding $(S_1 : 00, S_2 : 01, S_3 : 10, S_4 : 11)$ were used, the following group faces would result: $XX \xrightarrow{1} XX \xrightarrow{0} XX \xrightarrow{0} XX$.

This sequence does not converge to a single state, therefore the circuit realized above is *logically uninitializable*. A 3-valued simulator can only simulate group faces, not state groups, so there is a loss of information. *Therefore, our aim is to produce a state assignment that allows the sequence of group faces to “track” the sequence of state groups, and therefore insure logical initializability.*

Cheng and Agrawal propose that the problem can be solved by introducing an additional set of *face-embedding*

constraints into the state encoding step. Each of these is a $k \rightarrow 1$ *dichotomy constraint*, $(G_i; s_j)$.

In general, an $n \rightarrow k$ dichotomy constraint $(X; Y)$ [10] is the stipulation that the smallest containing cubes of X and Y not intersect. Here, X and Y are sets of states of cardinality n and k respectively. A dichotomy is satisfied if some state bit has value 1 for all states in X and value 0 for all states in Y , or vice versa.

In particular, a $k \rightarrow 1$ dichotomy of the form $(G_i; s_j)$ is introduced for every symbolic state s_j not present in the state group G_i in the state group sequence. *That is, a symbolic state that does not belong to a state group is forbidden from being embedded in it.* This applies to all state groups encountered when a synchronizing input is applied to the machine.

For the above example, the non-trivial dichotomies are: $(S_1 S_2 S_3; S_4)$, $(S_1 S_4; S_2)$ and $(S_1 S_4; S_3)$. A state assignment satisfying these dichotomies is: $(S_1 : 000, S_2 : 001, S_3 : 010, S_4 : 100)$, resulting in a correct 3-valued simulation: $XXX \xrightarrow{1} 0XX \xrightarrow{0} X00 \xrightarrow{0} 100$. Fig. 3 shows graphically the state group sequence and the corresponding sequence of group faces resulting from 3-valued simulation.

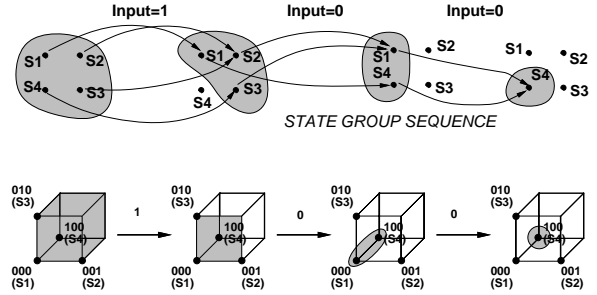


Figure 3: Groups faces “track” state groups

4 OUR SYNTHESIS METHOD: AN OVERVIEW

The technique we present in this paper is a synthesis method for logical (3-valued) initializability. Given a finite state machine and a synchronizing input sequence, our method consists of the following steps:

Step #1: constrained state assignment

- (a) generate face embedding constraints
- (b) generate don't-care intersection constraints

Step #2: combinational logic synthesis

5 CONSTRAINED STATE ASSIGNMENT

5.1 Step #1(a): Face embedding constraints

In [15] we demonstrated that when considering encoding of symbolic states, the constraints of [4, 5] are unnecessarily restrictive, and safely relaxed those constraints. In this subsection, we will review the method of [15] for producing relaxed face embedding constraints.

Again, consider the machine in Fig. 2. A synchronizing sequence for the machine was 100, resulting in state group sequence: $(S_1 S_2 S_3 S_4) \xrightarrow{1} (S_1 S_2 S_3) \xrightarrow{0} (S_1 S_4) \xrightarrow{0} (S_4)$. The dichotomy constraints produced by the Cheng-Agrawal method were $\{(S_1 S_2 S_3; S_4), (S_1 S_4; S_2), (S_1 S_4; S_3)\}$ and a minimum length state encoding satisfying these constraints

required 3 bits. In fact, the dichotomy $(S_1 S_2 S_3; S_4)$ is unnecessary. At the second time instant, the next-state value of S_4 is S_4 , which happens to belong to the next state-group, $(S_1 S_4)$. Therefore, if we let S_4 be embedded in the group-face of $(S_1 S_2 S_3)$, no harm would accrue. Thus, it is safe to delete the dichotomy $(S_1 S_2 S_3; S_4)$. The resulting state assignment is $(S_1 : 00, S_2 : 01, S_3 : 11, S_4 : 10)$, still yielding a correct 3-valued simulation: $XX \xrightarrow{1} XX \xrightarrow{0} X0 \xrightarrow{0} 10$.¹ Our solution uses only two bits to encode the states, whereas the Cheng-Agrawal solution requires three.

In sum, we prune the list of dichotomies proposed by Cheng-Agrawal—we delete the dichotomy (G_i, s_j) whenever $NS(s_j, I_i) \in G_{i+1}$, because this is a “safe” embedding. (Here, G_i is the i th state group, I_i is the input seen by G_i , and $NS(\text{current-state}, \text{input})$ is the next state function.) More formally, the original Cheng-Agrawal face-embedding constraints (FEC’s) were: $\mathbf{FEC} = \{(G_i; s_j) \mid s_j \notin G_i\}$. Our relaxed face-embedding constraints (RFEC’s) are:

$$\mathbf{RFEC} = \{(G_i; s_j) \mid NS(s_j, I_i) \notin G_{i+1}\} \quad (1)$$

It is easy to see that our RFEC constraints are less restrictive than the original constraints: $s_j \in G_i$ implies $NS(s_j, I_i) \in G_{i+1}$.

5.2 Step #1(b): Don’t-care Intersection Constraints

In this subsection, we demonstrate that the conditions of [4, 5], in general, may be unsafe. We show that even if these conditions are satisfied, the resulting circuits may *still* be uninitializable. In particular, we show that the issue of filling in don’t-care entries is important, and cannot be left entirely to the later stages of synthesis.

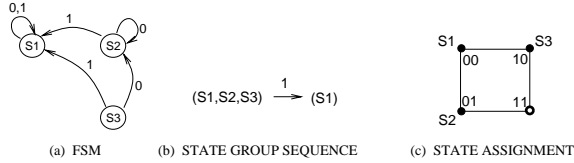


Figure 4: The issue of don’t-care NS assignment.

Consider the state machine of Fig. 4(a). It has a single-vector initialization sequence $I = 1$. Fig. 4(c) shows a state encoding satisfying the face-embedding constraints (there are none). It uses two bits to encode three states, thereby introducing a fourth state (11) that has no symbolic equivalent (a *non-symbolic* state). If a later step assigns state 11 a next-state (NS) entry of 10 on input 1, then the machine is no longer *initializable*; the 3-valued simulation trace is: $XX \xrightarrow{1} X0$. Therefore, if non-symbolic states can be assigned arbitrary NS values (during logic synthesis), a non-initializable circuit may result.

Simulation fails because the NS entry, 10, for state 11 lies outside of the state group being simulated, (S_1) , thus throwing initialization off course. However, observe that if the NS value were assigned to S_1 (00) instead, the circuit would be initializable: $XX \xrightarrow{1} 00$. The solution in this case, therefore, is to assign to the DC next-state entry a value *lying within the next group-face*.

¹In this example, we use 100 as the synchronizing sequence even though 00 is a shorter synchronizing sequence. However, the same problem can arise even starting with a minimum-length sequence.

In general, though, there are examples where using the Cheng-Agrawal method (or our RFEC method) results in a circuit that is uninitializable *for every assignment of don’t-cares*.

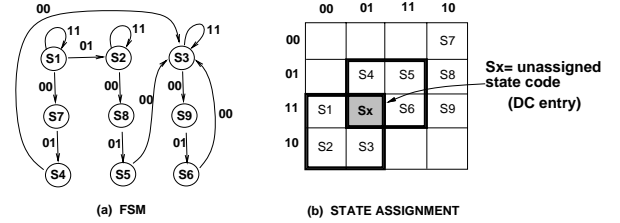


Figure 5: Second example illustrating the issue of don’t-care NS assignment.

Consider the example of Fig. 5(a).² Applying a synchronizing sequence gives the following state groups: $(S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 S_9) \xrightarrow{11} (S_1 S_2 S_3) \xrightarrow{00} (S_7 S_8 S_9) \xrightarrow{01} (S_4 S_5 S_6) \xrightarrow{00} (S_3)$. A state encoding that satisfies the above constraints is shown in Fig. 5(b). Bit-vector 0111 is a non-symbolic state with an as yet unassigned NS value. A careful analysis shows that this non-symbolic state *cannot be assigned any NS value* for input 00 while preserving initializability! In particular, state 0111 is embedded in the group-faces of $(S_1 S_2 S_3)$ as well as $(S_4 S_5 S_6)$. The latter embedding mandates the NS value of this non-symbolic state to be set to S_3 . The former embedding requires the NS value to be set to a state in the column containing $(S_7 S_8 S_9)$. *These two conditions are not simultaneously satisfiable*. For example, assuming the NS value is set to S_3 , the resulting simulation trace is: $XXXX \xrightarrow{11} 0X1X \xrightarrow{00} XXXX \xrightarrow{01} XXXX \xrightarrow{00} XXXX$.

To insure that there are no conflicting demands on the assignment to a don’t-care, we can draw up pairs of group-faces and force them to be disjoint. Then, no non-symbolic state can belong to both of them, so the above problem is circumvented. One way to do this is to introduce an $n \rightarrow k$ dichotomy constraint between each G_j and G_k in the state group sequence, represented as $(G_j; G_k)$, if G_j and G_k are followed by the same input vector but G_{j+1} and G_{k+1} do not intersect. In this case, a DC entry in $G_j \cap G_k$ may not have any consistent NS assignment. In the above example, we therefore add dichotomy $(S_1 S_2 S_3; S_4 S_5 S_6)$. This results in the state encoding of Fig. 6(b).

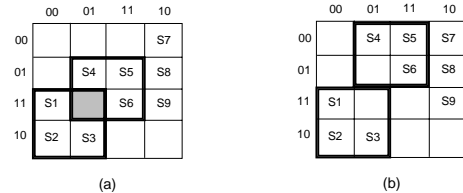


Figure 6: (a) bad state encoding, (b) good encoding

The constraints introduced above are called the “don’t-care intersection constraints” (*DCIC*) and are formalized as follows:

$$\mathbf{DCIC} = \{(G_j; G_k) \mid (I_j = I_k) \wedge (G_{j+1} \cap G_{k+1} = \phi)\} \quad (2)$$

²This state machine is incompletely-specified, but our analysis also applies to completely-specified machines.

DCIC, as well as RFEC, are dichotomies; these constraints can always be solved using existing algorithms (See [10]).

Together, RFEC and DCIC are sufficient to produce a state assignment for logical initializability.

6 STEP #2: COMBINATIONAL LOGIC SYNTHESIS
 In [5], Cheng and Agrawal mention another issue: that combinational logic synthesis influences 3-valued simulation, and hence initializability. They surmise that initializability can be preserved by separately optimizing each output, as opposed to doing multi-output optimization. However, we discovered that this is neither necessary nor sufficient.

In this section, we introduce necessary and sufficient conditions for combinational logic synthesis to insure 3-valued simulatability. In particular, we show that these conditions correspond precisely to *hazard-free* synthesis requirements (cf. Nowick [12], Eichelberger [11]). Our conditions apply to both 2-level and multi-level logic.

6.1 How logic synthesis affects 3-valued simulatability

The following example illustrates how logic synthesis can affect 3-valued simulatability, and hence initializability.

Example. Let Y be the Boolean function of three variables a, b and c shown in the K-map of Fig. 7(a). Let it be implemented in 2-level AND-OR logic using two product terms: $Y = ab + \bar{a}c$.

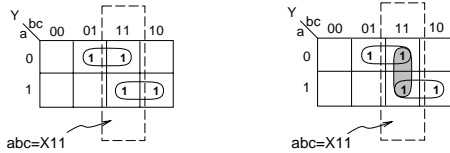


Figure 7: (a) Non-simulatable, and (b) simulatable implementations of Y

Assume at the current step in 3-valued simulation, the value of the 3-valued input vector abc is $X11$. For $abc = X11$, Y is functionally equal to 1, as seen from the K-map. However, a 3-valued simulator may evaluate Y as follows:

$$Y = ab + \bar{a}c = a + \bar{a} = X + X = X$$

Therefore, the above implementation of Y is not correctly simulatable.

The K-map of Fig. 7(b) shows an alternate implementation of Y which is 3-valued simulatable: $Y = ab + \bar{a}c + bc$ (where the shaded region represents the added product term bc).

3-valued simulation in this case yields the correct value:

$$Y = ab + \bar{a}c + bc = a + \bar{a} + 1 = X + X + 1 = 1.$$

The reason simulation succeeds in this case is that the term bc evaluates to 1 irrespective of the value of a . Therefore, this implementation of Y is correctly simulatable for the input combination $abc = X11$. \square

Thus, we have shown that logic synthesis has an impact on 3-valued simulatability.

6.2 Simulatability and Hazard-freedom

The covering requirements for simulatability presented in the previous subsection are very similar to the covering requirements for hazard-freedom (cf. [11, 12]).

Refer once again to Fig. 7. In the simulatability framework, the highlighted column corresponding to $abc = X11$ represented *indeterminacy* in the values of the inputs—the

value of a was unknown. In order for Y to be simulatable to 1 for this input combination, the requirement added was that the cube bc be covered some term of the cover.

Let us now interpret the above result from the point of view of hazard-freedom. If the same column $abc = X11$ is now regarded as representing an *input transition* $011 \rightarrow 111$ (or $111 \rightarrow 011$) which *spans* $X11$, then the requirement to insure a glitch-free output Y (static $1 \rightarrow 1$ transition) is *precisely* that cube bc be *covered* by some product term. In the hazard framework, cube bc is called a *required cube*; some product must cover the required cube for the transition to be hazard-free (see [11, 12, 9]).

Thus, we are able to relate 3-valued simulation to the transient behavior for the given example: if the implementation of Y is *not* simulatable to 1 over $\mathcal{I} = X11$, then the same implementation *has* a static $1 \rightarrow 1$ logic hazard for any transition spanning \mathcal{I} .

We now formalize and generalize this result: given an arbitrary multi-level circuit, such as the one in Fig. 8, simulatability for a given 3-valued input vector corresponds *precisely* to hazard-freedom for a spanning input transition.

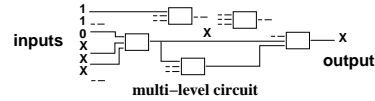


Figure 8: A general multi-level circuit

Theorem 6.1. Let f be a Boolean function implemented by an arbitrary multi-level network of gates, α be any 3-valued input vector to be simulated, and δ be any corresponding input transition that “spans” α . Then, 3-valued simulation and hazard simulation “correspond”. That is, if the result of 3-valued simulation on α is 0 (1), then the same circuit has a (function- and logic-) hazard-free $0 \rightarrow 0$ ($1 \rightarrow 1$) transition on δ . If, on the other hand, the result of 3-valued simulation on α is X , then the circuit has either (i) a static hazard, or (ii) a dynamic ($0 \rightarrow 1$ or $1 \rightarrow 0$) transition on δ .

Proof. Due to constraints of space, the proof is omitted. The interested reader can refer to [16].

What the above theorem states is that if “indeterminacy” of a 3-valued input is translated into an input transition, then indeterminacies on wires elsewhere in the circuit manifest themselves as *transients* (transitions or hazards) on those wires.

Let us now consider the special case of interest in our synthesis procedure: for the given 3-valued input, the circuit output is functionally equal to 1 (or 0), *but* 3-valued simulation is not faithful (yields a value X), as in Figure 7(a).³ We call such a scenario “non-simulatability.” It is clear that the result of hazard simulation for the output will be a “transient” (dynamic transition or static hazard), due to Theorem 6.1. Since the circuit output is functionally equal to 1 (or 0), the result of hazard simulation cannot be a dynamic transition. Therefore, the output exhibits a static hazard. This is stated in the following corollary.

Corollary 6.1 (non-simulatability \iff static logic hazard transition). If a circuit is non-simulatable for a 3-valued input α , then that circuit has a static logic

³It can easily be shown that RFEC and DCIC constraints insure that the output is functionally 1 (or 0) for each 3-valued input of interest [16].

hazard for every transition δ , where δ “spans” α . And, conversely, a static logic hazard implies non-simulatability.

Proof. See [16].

6.3 Logic Synthesis

The key result of the previous subsection was that for 3-valued simulatability, the circuit should be static hazard-free for certain input transitions. Conversely, any circuit realization that is free of static hazards for those input transitions, is also simulatable. This, then, provides us a technique for 2-level logic synthesis for simulatability.

For the special case of a 2-level AND-OR implementation, the conditions for hazard freedom were presented in [11, 12, 9]. To eliminate static logic hazards ([9, 11]), the covering requirements are of the form of cubes that must be covered by some product term of the cover—these are called *required cubes*.⁴ Techniques for minimization of hazard-free logic based on required cubes can be found in [12].

The duality between simulatability and hazard-freedom enables us to do multi-level logic synthesis for simulatability as follows: (a) do 2-level hazard-free logic synthesis, and (b) use those multi-level transformations that do not introduce any static-hazards for input transitions that span the 3-valued input vectors (see [13]). Corollary 6.1 provides the basis for the correctness of this procedure. Alternatively, direct multi-level hazard-free synthesis methods based on BDD’s can be used [14].

7 RESULTS

Table 1 presents the results of our synthesis-for-initializability method on several synchronous state machine examples. We only focus on the state assignment step (Step 1) here.

Column *I/S* indicates the number of inputs and the number of states respectively, for each example. Column *Coding Length* compares the length of the state encoding produced by three methods: (1) the *Base* method which does not incorporate initializability considerations; (2) the Cheng-Agrawal synthesis-for-initializability method (CA); and (3) *Our* method. For each of these examples, our method used the same number of state bits as the *Base* method, whereas the Cheng-Agrawal method used more bits in 2 cases. Therefore, for the examples considered, our method was able to achieve initializability without incurring any additional cost in terms of coding length.

Column *#Constraints* lists the number of encoding constraints generated by the Cheng-Agrawal method and by our method. Our method uses fewer face-embedding-constraints (FEC’s or $N \rightarrow 1$ dichotomies) than the Cheng-Agrawal method for almost all of the examples, and it never uses more. Moreover, our method needed to generate a don’t-care intersection constraint (DCIC or $N \rightarrow k$ dichotomy) for only one circuit.

The columns *Init?* report whether or not the implementations produced were actually initializable. Our method *always* produced correct initializable implementations. In fact, it guarantees initializability. On the other hand, the Cheng-Agrawal method produced incorrect results for one example: for *c-4*, the implementation produced by the CA method is not initializable. In contrast, our method used fewer face-embedding-constraints, and only one additional

⁴The transitions are *function hazard free*, since the function value is all 0 (1) throughout the transition. Therefore, the constraints for static logic hazard-freedom can always be solved [12].

Circuit Name	I/S	Coding length			#Constraints				
		Base	CA	Our Method	CA		Our method		
					FEC	Init?	RFEC	DCIC	Init?
c-1	1/4	2	2	2	6	Y	4	0	Y
c-2	1/4	2	3	2	3	Y	2	0	Y
c-3	1/3	2	2	2	0	Y	0	0	Y
c-4	2/9	4	4	4	18	N	12	1	Y
c-5	2/8	3	4	3	10	Y	3	0	Y

Table 1: Results: comparison of state encoding methods

DC intersection constraint ($N \rightarrow k$ dichotomy) and was able to guarantee initializability.

8 CONCLUSIONS

In this paper we have presented a new synthesis for logical initializability method. We have included a state assignment step and a combinational logic synthesis step. For state assignment, we first reviewed our relaxed face embedding constraints presented in [15]. We then introduced the new don’t-care intersection constraints and constraints on combinational logic synthesis. Our method guarantees logical initializability for the resulting circuit under 3-valued simulation. Preliminary results show little logic overhead.

9 ACKNOWLEDGMENTS

We thank Prof. Niraj Jha of Princeton University for discussions on initializability and testability.

REFERENCES

- [1] V.D. Agrawal, K.T. Cheng, and P. Agrawal, “A directed search method for test generation using a concurrent fault simulator,” *IEEE Trans. CAD*, vol. CAD-8, pp. 131-138, Feb. 1989.
- [2] S. Mallela and S. Wu, “A sequential circuit test generation system,” in *Proc. of ITC*, Philadelphia, PA. 1985, pp. 57-61.
- [3] J.A. Wehbeh and D.G. Saab, “On the initialization of sequential circuits,” in *Proc. of ITC*, pp. 233-239, 1994.
- [4] K. Cheng and V. Agrawal, “State assignment for initializable synthesis,” in *Proc. ICCAD*, pp. 212-215, 1989.
- [5] K. Cheng and V. Agrawal, “Initializability consideration in sequential machine synthesis,” *IEEE Trans. Comput.*, vol 41, pp. 374-379, Mar. 1992.
- [6] J.-K. Rho, F. Somenzi, and C. Pixley, “Minimum length synchronizing sequences of finite state machines,” *Proc. DAC*, pp. 463-468, 1993.
- [7] S.T. Chakradhar, S. Banerjee, R.K. Roy, and D.K. Pradhan, “Synthesis of initializable asynchronous circuits,” in *Proc. 7th Int. Conf. on VLSI Design*, pp. 383-388, Jan. 1994.
- [8] S. Banerjee, R.K. Roy, S.T. Chakradhar, and D.K. Pradhan, “Initialization Issues in the Synthesis of Asynchronous Circuits,” in *Proc. ICCD-1994*.
- [9] S.H. Unger. *Asynchronous Sequential Switching Circuits*. New York: Wiley-Interscience, 1969.
- [10] G.D. Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill (1994).
- [11] E.B. Eichelberger, “Hazard detection in combinational and sequential switching circuits,” *IBM J. Res. Develop.*, vol 9, no. 2, pp. 90-99, 1965.
- [12] S.M. Nowick and D.L. Dill, “Exact Two-level Minimization of Hazard-free Logic with Multiple-Input Changes,” *IEEE Trans. CAD*, vol. CAD-14, pp. 986-997, Aug. 1995.
- [13] D.S. Kung, “Hazard-Non-Increasing Gate-Level Optimization Algorithms,” in *Proc. ICCAD*, pp 631-, 1992.
- [14] B. Lin and S. Devadas, “Synthesis of Hazard-Free Multilevel Logic Under Multi-Input Changes from Binary Decision Diagrams,” *IEEE Trans. CAD*, vol. 14, pp 974-985, Aug. 1995.
- [15] M. Singh and S.M. Nowick, “Synthesis-for-initializability of asynchronous sequential machines,” in *Proc. of ITC*, 1996.
- [16] M. Singh and S.M. Nowick. *Logic Synthesis for Initializability of Synchronous Finite State Machines*. TR-CUCS-036-96, Dept. of Computer Science, Columbia University (to appear).