Sequential Circuit Optimization by FSM Transformation

Sanghun Park and Kiyoung Choi
School of Electrical Engineering, Seoul National University, Korea

Abstract

Sequential *circuits that are usually modeled as FSMs (Finite State Machines) can be implemented in synchronous Mealy or Moore style. Some circuits are better implemented in synchronous Mealy style than in Moore style. The opposite is true for some other circuits. Even, for some circuits, the optimum point lies in between the two extreme points. In this paper, we investigate this relatively new concept and propose an optimization algorithm, which can be implemented and integrated into the conventional synthesis flow.*

The algorithm starts from a synchronous Mealy style implementation, which is gradually transformed to a Moore style as long as there is a gain in view of area. Experimental results show that we can obtain area reduction by about 13% on the average.

1. Introduction

Sequential circuit optimization has been the subject of intensive investigation for several decades [1][2]. Sequential logic synthesis seems no longer to be a frontier of hardware design automation area. Nonetheless, many researchers still work on it because it is a necessary step in most hardware synthesis processes and better quality of the sequential logic synthesis tool is critical for better quality of the final synthesis results. It is important to make every effort to optimize circuits in every aspect of the sequential logic synthesis process.

Most of the previous work on FSM (Finite-State Machine) synthesis focuses on synthesizing a sequential circuit from a given state table [4][6]. The conventional synthesis of an FSM is accomplished by an ordered processing of sophisticated optimizing steps including state minimization, state encoding, logic minimization, and library binding. However, this synthesis flow still has room for further optimization. In this paper, we focus on generating an optimized state table maintaining the behavior intact. Our work is based on the observation that a synchronous sequential circuit can be implemented as a synchronous Mealy machine, a Moore machine, or a mixed machine, and the quality of the result considerably depends on the implementation style. We search for the optimal point in between synchronous Mealy machine and Moore machine. The search starts from all synchronous Mealy or mixed style implementation and gradually moves to Moore style implementation by splitting states, thereby making outputs to be input-independent. Currently, if the original description of the circuit is fully in Moore style, then there will be no further improvement.

This paper is structured as follows. In the next section, we briefly review the implementation styles of sequential circuits. Section III describes how FSM transformation is accomplished through state splitting. By splitting states, we can increase or decrease the area of the synthesized FSM. We need to estimate the area increase/decrease to decide whether to split states or not. In section IV, we propose an area estimation model and optimization algorithm. In section V, we discuss the experimental results. Then we conclude with future work.

2. Preliminaries

An FSM can be described by a quintuple $(X, Y, S, \delta, \lambda)$, where X is a set of primary inputs, Y is a set of primary outputs, S is a set of states, δ is a state transition function ($\delta : X \times S \rightarrow S$), and λ is an output function ($\lambda : X \times S \rightarrow Y$ for Mealy models and $\lambda : S \rightarrow Y$ for

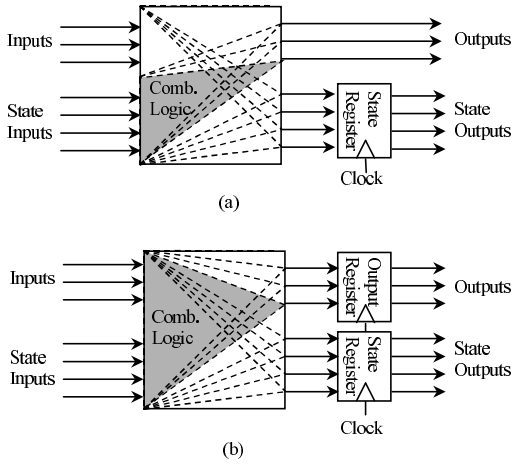


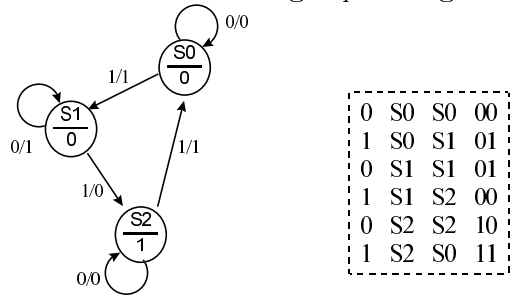
Figure 1. Block diagram of (a) Moore machine and (b) synchronous Mealy machine.

Moore models). There are three basic implementation styles for a clocked sequential circuit: Moore machine, Mealy machine, and synchronous Mealy machine. Mixture of these three styles is also possible.

The outputs of Moore machine depend only on the present state. See the block diagram in Figure 1 (a). A combinational logic block maps the inputs and the current state to the flip-flop inputs to store the appropriate next state. However, it maps only the current state to the outputs. The outputs change synchronously with state transitions and clock edges. The outputs of Mealy machine depend on the present state and the present input values. The outputs can change immediately after a change at the inputs, independent of the clock.

Glitches in the output are inherent in Mealy machines due to the asynchronous nature. The glitches are undesirable in real hardware controllers. However, Mealy machines are sometimes preferred because of the saving on state registers. This leads to alternative synchronous design styles for Mealy machines. Simply stated, to construct a synchronous Mealy machine we can break the direct connection (or connection via combinational logic only) from inputs to outputs by introducing storage elements. One way to do this is to synchronize the Mealy machine outputs with output flip-flops. See figure 1 (b). The flip-flops are clocked with the same edge as the state registers. However, the synchronous version does not have exactly the same input/output behavior as the original Mealy machine. The output timing of this machine is delayed by one clock cycle due to the output registers [4].

There is a mismatch of output timing between Mealy machine and Moore machine in nature. In addition, the plain Mealy machine may not be used for hardware controllers in practice because of the glitches. In contrast, the synchronous Mealy machine can be transformed to Moore machine while maintaining output timing intact.



0	S0	S0	00
1	S0	S1	01
0	S1	S1	01
1	S1	S2	00
0	S2	S2	10
1	S2	S0	11

Figure 2. Mixed FSM represented by (a) state transition graph and (b) state transition table.

The behavior of an FSM can be described by a state transition graph. A state transition graph is a labeled directed graph $G(V, E)$, where the vertex set V is in one-to-one correspondence with the state set S and the directed edge set E is in one-to-one correspondence with the transitions specified by δ. In particular, there is an edge (v_i, v_j) if there is an input pattern $x \in X$ such that $\delta(x, s_i) = s_j, \forall i, j = 1,2,...,|S|$. In Mealy model, each edge is labeled with $x/\lambda(x, s_i)$. In Moore model, however, each edge is labeled with x and each vertex is labeled with the corresponding output function $\lambda(s_i)$. Figure 2 shows an example of state transition graph having two outputs. First output has Moore behavior and second output has Mealy behavior.

In this paper, we consider only Moore machine and synchronous Mealy machine for the implementation of a synchronous FSM. The following theorem states that a synchronous Mealy machine implementation actually uses more or equal area for storage elements including output synchronizing flip-flops than the corresponding Moore machine implementation. This implies that Moore machine implementation consumes less area provided that the combinational logic consumes less area.

Theorem: When the number of states is minimized by merging all equivalent states, the number of states of a Moore machine implementation is greater than or equal to that of the corresponding synchronous Mealy machine implementation. However, the opposite is true for the minimum number of flip-flops used.

Proof) Let a completely specified FSM has m inputs, n outputs and k internal states. The state transition graph corresponding to Mealy machine implementation has k internal states and $2^m \times k$ transition edges. There are n output functions, each with a value per 2^m input pattern, k states. This is a minimum state machine provided that this machine does not have unreachable or equivalent states. This machine can be implemented as a synchronous Mealy machine with $\lceil \log_2 k \rceil + n$ flip-flops, through minimum length state encoding.

Now, consider transforming Mealy machine to the corresponding Moore machines through state splitting. A state in Mealy machine splits into different states if the incoming transitions generate different output values. The number of states of Moore machine obtained this way is definitely greater than or equal to that of the corresponding Mealy machine, because there are 2^n different output combinations, the maximum number of states generated by splitting one state is 2^n. Therefore, the maximum number of states generated for the entire Moore machine is $k \times 2^n$. This implies that Moore machine implementation uses no more flip-flops than synchronous Mealy machine implementation.□

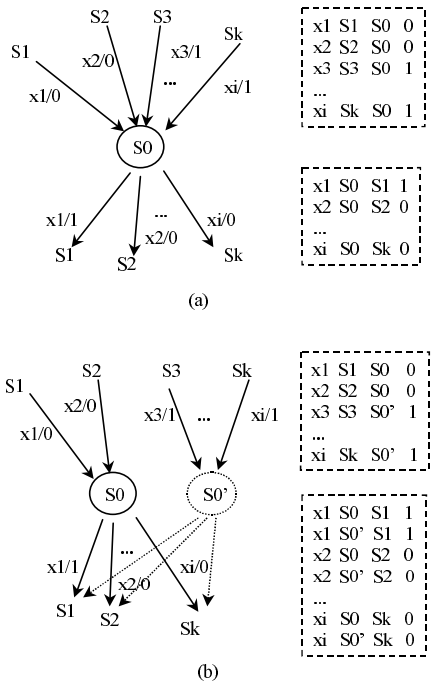


Figure 3. An example of a state transition graph (a) before state splitting and (b) after state splitting.

3. FSM Transformation by State Splitting

We define a Moore-state as a state that has the same output value for all of its incoming transitions. A Mealy-state is defined as a state that is not a Moore-state [3]. This definition can also be extended to individual outputs. Specifically, if all incoming transitions of a state have the same value for a specific output, the state is a Moore-state with respect to the output.

Figure 3 shows an example of transformation from a Mealy-state to Moore-states. A set of incoming transitions to Mealy-state S0 is divided into a set of transitions having output value "0" and a set of transitions having output value "1". The latter becomes the incoming transitions to a newly created state S0', whereas the former remains to be the incoming transitions to the original state S0. The outgoing transitions from S0' are duplicated from those of S0. After state splitting, S0 and S0' become Moore-states.

FSM transformation is accomplished for a given output by two steps: state transformation by state splitting and timing adjustment by one clock delay. Figure 4 shows an example of the output transformation for the second output of FSM shown in Figure 2. The output function shown in Figure 4 (a) has still Mealy machine behavior after state splitting. However, all output values generated by the incoming transitions to a state become the same. The output function obtained by timing adjustment has Moore machine behavior as shown in Figure 4 (b). Note that all output values originally located at the incoming transitions of a state are moved to the state itself. Physically, this change means one clock delay for the output so that the output timing must be the same as the original synchronous Mealy behavior. State splitting just creates new state S0' and two outgoing transitions from S0'. However, it enables us to change the output function by timing adjustment. The final result of the output transformation is the output function that depends solely on the current states.

This transformation simplifies the output function. For example, the cover of the function for the second output of Mealy version shown in Figure 2 consists of three cubes {(1 S0), (0 S1), (1 S2)}. In contrast, the cover of Moore version shown in Figure 4 (b) consists of two cubes {(S0'), (S1)}. Consequently, by output transformation, three literals and one cube are reduced. This means that we can obtain area gain if we make an assumption that the area cost for storing states does not change. However, this transformation decreases the number of don't cares which correspond to unused code at state encoding as many as newly created states. What is even worse is that it can increase the number of state signals/flip-flops. Moreover, the duplicated outgoing transitions from the states newly created can increase the complexity of the function. Therefore, we cannot simply say that this transformation will achieve area gain.

Under the assumption of two-level logic implementation, the minimum literal counts generated by Nova [8] for the three versions of machines are 20 literals for the original machine, 29 literals for the machine after state transformation, and 18 literals for the machine after

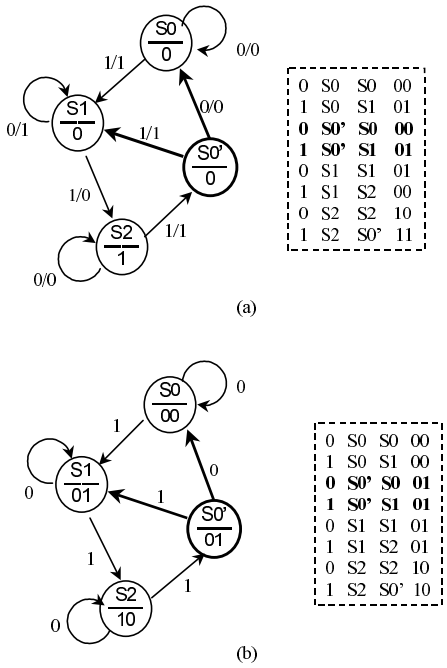


Figure 4. Change of an output function by transformation (a) after state splitting and (b) after timing adjustment.

timing adjustment, respectively. We can notice from these results that the increased states and transitions probably increase the hardware size. However, the change of output function by timing adjustment may decrease the hardware size, eventually. By changing the output function with small increase of states and transitions, we can expect sufficient area gain.

4. Optimization Algorithm

Our implementation model of a sequential circuit is depicted in Figure 5. The model contains both Moore-states and Mealy-states. Recall that an output depends either on Moore-states or on transitions to Mealy-states.

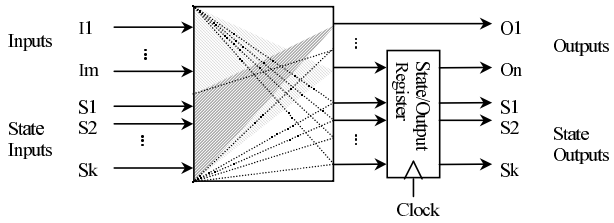


Figure 5. A model of optimized sequential circuit.

Note that an output function cone having Moore behavior consists of the combinational logic gates, which decode specific state codes. The output of a function having synchronous Mealy behavior is connected directly to flip-flop for output synchronization. On this hybrid implementation, we should be careful about timing behavior at the outputs having Moore behavior. If an output function cone contains reconvergent point, glitches can be generated at the point. So, it is better not to do output transformation for this output if glitches can cause malfunction. Another problem is that the output delay can be added to a certain path of a cascaded next block. In the worst, the delay of critical path between two flip-flops can be increased. It is rather simple to identify and prevent these timing violations generated by cascaded logic blocks and it is out of the scope of this paper

To decide whether we can obtain the area gain for a given output by the transformation, we need a method to estimate the hardware size before and after the transformation. However, it is hard to exactly estimate the hardware size in the early stage before state encoding and library binding. To simplify the problem, we consider the change of hardware size for storage part and combinational part, separately.

For the storage part, the problem can be simplified if we make an assumption of minimum length encoding. The output transformation is accomplished for one output at a time. A flip-flop attached to the output can be eliminated by splitting all the Mealy states with respect to the output. This transformation increases the number of states. Assuming that the number of states before state splitting is k and that after state splitting is k', the increase in the number of state flip-flops amounts to $\lceil \log 2^{k'} \rceil - \lceil \log 2^{k} \rceil$. The gain becomes $1 + \lceil \log 2^{k} \rceil - \lceil \log 2^{k'} \rceil$ which amounts to 0 or 1 per one output transformation.

The area change of combinational part does not seem to be estimated as easily as that of storage part. The state encoding plays a major role in minimization of hardware size. So far, many researches have been on state encoding and many of these approaches are based on symbolic minimization [7][8]. Symbolic minimization yields an encoding-independent sum of products representation of a switching function, which is minimal in the number of product terms. In general, symbolic minimization is strongly affected by the primary inputs and/or outputs of a machine [7]. The proposed output transformation plays a role of changing the symbolic cover. The simplified output functions drive the symbolic cover to be smaller. In contrast, the increased transitions drive the cover to be larger. The minimal symbolic cover guarantees that at most the number of rows in two-level PLA implementation is the same as the number of the cover cardinality. The symbolic cover cardinality reflects the final hardware size relatively well even for multi-level logic implementation [8]. Hence, it is adequate to use symbolic cover cardinality to compare the relative size of the combinational part before output transformation and that after output transformation. Under the assumption that both FSM will be encoded with the same code length, smaller symbolic cover is expected to be implemented with smaller hardware.

Algorithm 1: Area Optimizer

```
Input: A state transition graph G.
Output: The state transition graph G' optimized for area.
Begin
Output = {1,..,n}; /* output list sorted by the number of
      created states and transitions in ascending order. */
Best = area-estimation(symbolic-minimization(G));
G' = G;
Index = null; current = 0;
While (Output != null) do
    Index = remove-front(Output);
    G = mealy2moore(G, index);
    Current = area-estimation(symbolic-minimization(G));
    If (current <= Best) do
        Best = current;
        G' = G;
    End if;
End while;
Return G';
End
```

The area gain of the combinational part can be estimated by examining the change of symbolic cover.

Table 1. FSM synthesis results for IWLS'93 benchmarks

Benchmarks	Synch. Mealy				Mixed				Area Reduction	Delay Reduction
	#lit.	#reg.	Area	Delay	#lit.	#reg.	area	delay		
BBSSE	164	11	2608	17.62	136	8	2094	19.84	19.7 %	-12.6 %
CSE	321	11	4008	30.60	319	10	3984	29.80	0.6 %	2.6 %
DK14	116	8	2056	22.60	92	7	1984	22.60	3.5 %	0.0 %
DK17	68	6	1200	21.00	60	6	1144	9.00	4.7 %	57.1 %
EX1	474	24	6360	22.24	377	19	5720	17.80	10.1 %	20.0 %
EX4	59	13	2160	13.80	50	8	1628	17.80	24.6 %	-29.0 %
EX6	124	11	2664	27.64	90	5	1934	16.42	27.4 %	40.6 %
SAND	959	14	9880	55.00	930	13	9696	35.60	3.0 %	35.3 %
PMA	282	13	3928	43.00	259	12	3496	34.00	10.9 %	20.9 %
SSE	127	11	2112	21.80	111	8	1808	19.20	14.4 %	11.9 %
S1	219	11	2648	38.64	106	8	1768	21.02	33.2 %	45.6 %
S27	32	4	696	9.22	19	3	520	6.44	25.3 %	30.1 %
S386	129	11	2136	24.84	127	9	2008	20.86	6.0 %	16.0 %
S510	441	13	4936	34.00	428	11	4808	30.09	2.6 %	11.5 %
STYR	731	15	7456	48.60	704	14	7187	48.40	3.6 %	0.4 %

Algorithm 1 shows the pseudo code of the proposed area optimization algorithm. First, it greedily determines the order of outputs to be transformed. This is computed statically at initial time in increasing order of the number of states and transitions, which are created by the output transformation.

For each output transformation, we perform symbolic minimization, and then estimate the area of combinational part by counting the literals in the minimized symbolic cover. The number of literals can be computed after simple state encoding such as linear encoding or random encoding. This method improves the accuracy of the area estimation. The previous best solution is replaced with the current solution if the literal count does not increase. We used the Espresso-MV [10] for symbolic minimization, which implements a heuristic algorithm for minimizing the symbolic cover of multi-valued logic function. After it explores all output transformations, it returns the best solution.

The above algorithm iterates as many times as the number of outputs, and each iteration does one symbolic minimization. The total complexity strongly depends on the symbolic minimization.

5. Experimental Results

Experiments were accomplished on IWLS'93 benchmarks. They are state transition tables described in KISS2 format. First, we perform the proposed area optimization to these machines. Next, we synthesized the optimized machines through the conventional synthesis flow. We used SIS [6] tool for state minimization, state encoding, logic minimization, and library binding with MSU (Mississippi State University) library.

Table 1 shows the results of FSM synthesis. Comparisons are made on the number of literals, the number of flip-flops, the total area without considering interconnection, and the critical path delay. We obtained area reduction considerably in many examples by the proposed method. Moreover, the critical path delay tends to decrease. The reason is because the complexity of the combinational logic is reduced.

The area of the synthesized FSM depends on which state encoding strategy is used [7][8]. For the state encoding, we used Jedi program [5], which is suitable for multi-level logic implementation. The program supports various strategies of state encoding. The data in Table 1 reports the best of the results obtained for every encoding option. We also applied retiming [9] on the area-optimized circuits. The retiming generates speed-optimized circuits with possible increase in area. The results show that the retiming does not cause much difference in the tendency.

Figure 6 shows how area changes as we proceed to transform outputs from synchronous Mealy to Moore. In some cases, we obtain minimum area when the circuit is implemented fully in Moore style. But in other cases, the optimum point is in between the two extreme implementations.

Of course, we have experimented with large-size FSMs, such as 's298' and 'tbk' benchmarks. But, there are no improvements on these examples because there are so many states splitting. Though with so many states increasing in these examples, the proposed area optimization program finished within 1 minute at Sparc-20 machine. In many cases, the increasing of states degrades the effect of output transformation. In contrast, if output transformation is accomplished without state splitting, then the effect of transformation appears in evidence.

6. Conclusion

We investigated the possibility of further optimizing sequential circuits through FSM transformation. It was shown that by transforming a part of a sequential circuit from synchronous Mealy style to Moore style, we could effectively reduce the circuit area.

We devised an algorithm that tries to find the optimum point in between all synchronous Mealy style and all Moore style. By applying the proposed algorithm, we obtained about 13% area reduction on the average. The proposed algorithm can be simply adapted to the conventional sequential logic synthesis flow without modification. Moreover, the FSM transformation proposed in this paper will probably elevate the performance of the implementation method of low-power FSM using gated clock [3].

We are currently working on extending our algorithm so that it can also be applied to all Moore style implementations. One simple approach is to convert a Moore style description to the corresponding Mealy style description and then applies the proposed algorithm together with state minimization.

For another extension, we consider two directions for the improvement of our algorithm. One is to improve the estimation model of area gain. The accuracy of the estimation model is critical in making right decisions during the transformation.

The other direction is to improve the optimization algorithm. The current implementation adopts greedy algorithm, that is, we determine the order of outputs to be transformed statically at initial time. Consequently, the results depend strongly on the order of transformed outputs and tend to be easily trapped in a local minimum. We can consider other algorithms, such as branch and bound. The branches that have many states splitting can be pruned in early time, because the larger the increased states and transitions are, the lower the probability of getting optimum solution is. This may reduce the possibility of being stuck at local minimum at small increase of running time.

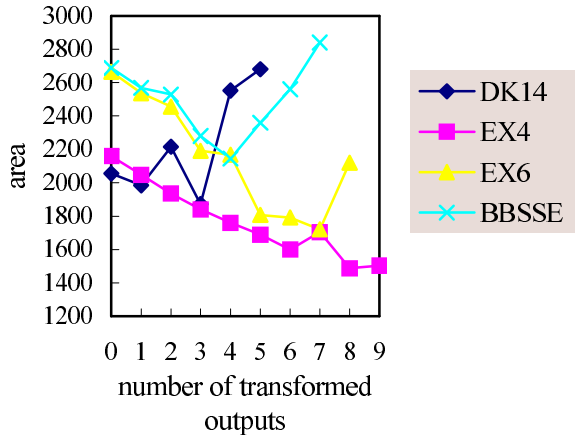


Figure 6. Area change according to incremental transformation.

References

[1] F. Hill and G. Peterson, *Switching Theory and Logical Design,* Wiley, New York, pp. 189-198, 1971.

[2] Giovanni De Micheli, *Synthesis and Optimization of Digital Circuits,* McGraw-Hill, 1994.

[3] L. Benini and Giovanni De Micheli, "Automatic Synthesis of Low-Power Gate-Clock Finite-State Machines", *IEEE Transactions on CAD/ICAS,* vol. 15, no. 6, June 1996.

[4] Randy H. Katz, *Contemporary Logic Design,* Benjamin/Cummings Publisher, 1994.

[5] B. Lin and A. R. Newton, "Synthesis of Multiple Level Logic from Symbolic High-Level Description Languages," *In Proceedings of the International Conference on VLSI,* pp. 187-196, August 1989.

[6] *Sequential Interactive System Manual,* Electronics Research Laboratory, Dept. of EECS, University of California, Berkeley, 1992.

[7] G. De Micheli, R. Brayton, and A. Sangiovanni-Vincentelli, "Optimal State Assignment for Finite State Machines", *IEEE Transactions on CAD/ICAS,* vol. 4, no. 3, pp. 269-284, July 1985.

[8] T. Villa and A. Sangiovanni-Vincentelli, "NOVA: State Assignment for Finite State Machines for Optimal Two-level Logic Implementation", *IEEE Transactions on CAD/ICAS,* vol. 9, no. 9, pp. 905-924, September 1990.

[9] S. Malik, E. Sentovich, R. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and Resynthesis : optimizing sequential networks with combinational techniques," *IEEE Transactions on CAD/ICAS,* vol. 10, no. 1, pp. 74-84, January 1992.

[10] R. L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-Valued Minimization for PLA Optimization," *IEEE Transactions on CAD/ICAS,* vol. 6, no. 5, pp. 727-751, September 1987.