Low Power Synthesis of Finite State Machines with Mixed D and T Flip-Flops

Ali Iranli
Department of EE-Systems
University of Southern California
iranli@usc.edu

Peyman Rezvani
Magma Design Automation
Cupertio CA
peyman@magma-da.com

Massoud Pedram
Department of EE-Systems
University of Southern California
pedram@ceng.usc.edu

Abstract

This paper presents a state assignment technique to reduce dynamic power consumption in finite state machines (FSM). The key idea is to decompose the state machine into a set of cycles that are collectively equivalent to the original FSM, and perform state assignment based on the cycle realization of the state machine using Gray codes. A new implementation of state machines by using a combination of D and T flip-flops is thereby proposed, which in conjunction with the proposed encoding algorithm, reduces power consumption by an average of 15%.

1. Introduction

The demand for battery-powered products has resulted in a significant interest in energy efficient design. Meanwhile, integrated-circuit densities and operating speeds have continued to increase, following the Moore's Law. The result is that VLSI circuits are becoming larger, faster, and more complex and because of this, dissipating ever-increasing amount of power.

These increases in power have created new and difficult challenges for circuit designers. Realizing that static voltage scaling was insufficie[1]nt to solve the "power problem," designers subsequently began to focus on advanced design methodologies and tools to address the power issues. Complicating designers' attempts to deal with these issues are the complexities of modern IC designs and the design flows required to build them.

In recent years, various types of design automation techniques and tools have been developed that focus on power-aware and/or power-efficient design. Many works have been reported that address power modelling at different abstraction levels, power estimation, and power optimization, and low power synthesis [6]. Low power synthesis includes state assignment, retiming, logic minimization and technology mapping.

State encoding/assignment, as a crucial step in the synthesis of the controller circuitry, has been extensively studied. Early research on state assignment was focused on finding a state encoding that minimizes area of the circuit [3, 8, 11]. More recently, a number of low power state-encoding techniques have also been proposed [5, 7, 8]. Roy et al. was the first to address the problem of reducing switching activity of input state lines of next state logic during state assignment, formulating it as a Minimum Weighted Hamming Distance (MWHD) problem [7]. Olson et al. used a linear combination of switching activity of the next state lines and the number of literals as the cost function [5]. Tsui et al. [8] used simulated annealing as a search strategy to find a low power state encoding that accounts for both the switching activity of the next state lines and switched capacitance of the next state and output logic .

In [10], Wu et al. proposed the idea of realizing a low power FSM by using T flip-flops. The authors showed that use of T flip flops results in a natural clock gating and may result in reduced next state logic complexity. However, that work was mostly focused on BCD counters which have cyclic behavior. The cyclic behavior of counters resulted in a significant reduction of combinational logic complexity and, hence, lower power consumption.

This paper proposes a novel state assignment technique, which identifies most probable cycles in the FSM and encodes states on these cycles with Gray codes. The objective function is to minimize the Weighted Hamming Distance. Notice that although techniques such as those presented in [8] use more accurate cost functions for two and multi-level logic realization of the state machines, the MWHD metric is still relevant and quite effective. In other words, although MWHD does not exhibit a high absolute accuracy, but it certainly exhibits a high degree of fidelity. This paper also teaches how a combination of T and D flip-flops as state registers can be used to achieve a low power realization of the FSM in question.

The remainder of this paper is organized as follows. Section 2 provides the theoretical background for the proposed encoding technique; the cycle-based state encoding technique is presented in Section 3, and in Section 4, the power-efficient implementation of an FSM using both T and D flip-flops is discussed. Experimental results and conclusion are given in Sections 5 and 6, respectively.

2. Background

An FSM is described by a six-tuple (X, Y, S, s_0, λ, η) where X is the set of input symbols, Y is the set of output symbols, S is the set of states, s_0 is the initial state, $\lambda : X \times S \rightarrow Y$ is the output function, and $\eta : X \times S \rightarrow S$ is the next state function.

From a probabilistic point of view, an FSM can be described by a Markov process in which p_i is the probability of being in state s_i, and p_{ij} is the conditional probability of transition from state v_i to state v_j.

The state transition graph of an FSM is a vertex/edge weighted directed graph $G(V, E)$, where the set of vertices V

1. This research was supported in part by DARPA PAC/C program under contract no. DAAB07-02-C-P302 and by NSF under grant no. 9988441.

and set of edges E correspond to the states of the FSM and transitions between them respectively:

$$V = \{v_i | v_i \in S\}$$

$$E = \{(v_i, v_j) | \exists x \in X, \eta(x, v_i) = v_j\}$$

The weight assigned to each vertex v_i is the state probability p_i, and the weight assigned to each edge (v_i, v_j) is the probability p_{ij} in the Markov process.

Assume states s_i and s_j are encoded using binary strings b_i and b_j respectively. The transition from state s_i to state s_j will have a switching activity equal to d_{ij}, the hamming distance between b_i and b_j. Since dynamic power consumption is directly related to switching activity and state transition in an FSM corresponds to the switching of the state bits, state encoding will have a major effect on the power consumption. The goal is to perform state assignment in such a way that state transitions with higher probability take place with a smaller switching activity on state bits. The objective function to minimize would then be: $\sum_i p_i \sum_{j:(v_i, v_j) \in E} p_{ij} d_{ij}$ (1)

Implementation of an FSM is usually done using D flip-flops. The input to the flip-flops would then be $D = \eta(x, s)$ where x is the input and s is the present state. One can also implement FSM using T flip-flops in which case the input to the flip-flops would be $T = \eta(x, s) \oplus s$. Since implementation using T flip-flops tends to result in more complex circuits for the combinational logic because of the additional XOR operation, in practice, it has not been heavily used. Recently, Wu et al. in [10] showed that if the circuit exhibits a cyclic behavior, use of T flip-flops as state registers can indeed result in simpler combinational logic and, hence, lower area cost and power consumption.

In our proposed technique for low power state encoding, the FSM is first decomposed into a set of cycles. These cycles are then encoded in order to minimize the total switching according to the cost function in (1). Once the cycles are encoded, the entire FSM will be implemented using D and T flip-flops.

3. Cycle-based Encoding

We first show in a systematic way that a Markov process, which defines the behavior of the FSM, can be represented by a set of weighted directed cycles. Next we introduce a new state-encoding algorithm based on this cycle representation of the FSM.

We start with a simple example of a cyclic process modeling the motion of a particle on a closed curve and focus on the particle's motion through p points of this curve at moments that are one unit of time apart; cf. Figure 1a.

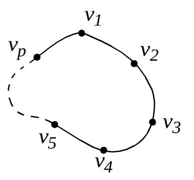

Fig. 1a: Particle's Motion on a Closed Curve

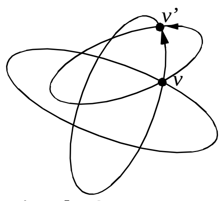

Fig. 1b: Superimposing Cycles

This leads us to a discretization of the curve into an infinite sequence of points $C = (v_1, v_2, ..., v_p, v_{p+1}=v_1, v_{p+2}=v_2, ...)$ called a directed cycle with period p. If no disturbance occurs, the passing of the particle through v_i can be codified by an infinite binary sequence $y_{v_i, v_{i+1}} = \ldots 0\, \underbrace{10\ldots0}_{p}\, 10\ldots010\ldots$ where 1 or 0 means the particle is passing through v_i or is not. The sequence $y_{v_i, v_{i+1}}$ is understood as a *non-random* sequence in the context of Kolmogorov's theory of complexities since both 1 and 0 appear periodically after each p steps [4].

Consider a set of possibly overlapping cycles $\{C_1, ..., C_r\}$ where each cycle C_i ($i \le r$) is associated with some positive number $W(C_i)$; cf. Figure 1b. Imagine that at some instance of time, the particle appears at some point v that is common to t cycles, say $C_1, ..., C_t$ ($t \le r$). The particle may continue its way to another point v', which is the intersection point of m cycles (out of that t cycles that had point v as an intersection point) say $C_1, ... C_m$ ($m \le t$). A natural measure of particle's transition when moving from v to v' can then be defined as:

$$\frac{W(C_1) + W(C_2) + \ldots + W(C_m)}{W(C_1) + W(C_2) + \ldots + W(C_t)} \quad (2)$$

Accordingly the binary sequence $y_{v,v'}$ codifying the transition of the particle from v to v' is given by a *chaotic* sequence. Furthermore, since expression (2) provides transition probability from v to v' of a Markov process ξ that behaviorally models the particle's movement, it can be concluded that:

"A collection of cycles $\mathbb{C}$ along with some weights assigned to each cycle defines a Markov process ξ."

Motivated by the above example, we proceed with a formal definition of directed cycles and cycle-based representation of Markov processes.

Definition 1. *A directed cycle over a countable set of states S is a periodic function C from the set $\mathbb{Z}$ of integers into S. Furthermore, C(i), $i \in \mathbb{Z}$ is called a vertex of the cycle and (C(i), C(i+1)) is called a directed edge of the cycle.*

Each cycle C belongs to an equivalence class of cycles $\mathbb{C}$ where $\mathrm{C} = \{C' | \forall i \in Z, C'(i) = C(t(i))\}$ where t is a translation function over $\mathbb{Z}$. Two cycles belonging to the same equivalence class are called *equivalent*.

Definition 2. *For a cycle C, passage function J_C is a binary function defined over the set of states S as follows:*

$$J_C(v) = \begin{cases} 1 & \text{if } \exists i \in \mathrm{Z}, C(i) = v \\ 0 & \text{otherwise} \end{cases}$$

The second order passage function can accordingly be defined as:

$$J_C(v, v') = \begin{cases} 1 & \text{if } \exists i \in \mathrm{Z}, C(i) = v \text{ and } \\ & \quad C(i+1) = v' \\ 0 & \text{otherwise} \end{cases}$$

Theorem 1. Let S be a finite set of states. Consider a homogeneous recurrent $|S|$-state Markov process ξ defined over a probability space with common invariant probability distribution $p_i,\ i \le |S|$; then there exists a finite set of weighted cycles $\mathbb{C}$ such that superposing the cycles will define ξ; i.e.,

$$\begin{aligned} p_i &= \sum_{C \in \mathbb{C}} W(C) J_C(v_i) \\ p_{ij} &= \sum_{C \in \mathbb{C}} W(C) J_C(v_i, v_j) / p_i \end{aligned} \qquad \forall v_i, v_j \in S \quad (3)$$

Proof. Refer to [4]. ■

Based on Theorem 1, the FSM decomposition problem can be stated as follows.

Cycle Decomposition Problem (CyDec): *Given a $|S|$-state finite state machine, find a set of weighted cycles $\mathbb{C}$ such that their superposition defines the Markov process corresponding to the given FSM.*

Solutions to **CyDec** can be classified as probabilistic or deterministic solutions depending on whether or not the weighted cycles are subjected to probabilistic interpretation. As examples of a probabilistic and a deterministic solution to **CyDec**, consider the following two approaches.

Randomized Approach: Consider a homogenous and recurrent Markov process with a countable state space, which is derived from a given FSM. The process is allowed to run along a sample path for virtually infinite time. Along this sample path, a set of cycles will be generated. Theoretically, if we let the process run for an infinitely long time, *all* possible cycles will be generated. Now, the Markov process can be decomposed into a set of cycles $\mathbb{C}$, generated in the above manner, with weights calculated by the following set of equations, which are in turn derived from (3):

$$p_i p_{ij} = \sum_{C \in \mathbb{C}} W(C) J_C(v_i, v_j) \qquad \forall v_i, v_j \in S$$

These cycle weights are unique and independent of the ordering in which the cycles were generated. The weights have a probabilistic interpretation, in the sense that $W(C)$ is equal to the expected number of times that C appears along an infinitely long sample path.

Deterministic Approach: Consider a homogenous and recurrent Markov process, which is derived from a given FSM. Pick an arbitrary state v_i. Since the process is recurrent, there exists at least one state v_j such that the transition probability P_{ij} is non-zero. Pick this new state v_j and repeat the procedure. Since the number of states is finite, a cycle will finally be created. Set the weight for this cycle equal to the minimum probability of the transitions on the cycle, decrease the probability of each transition on the cycle, P_{ij}, by this cycle weight, and proceed similarly to find other cycles until no non-zero probability transition is left.

Theorem 2: Let ξ be a homogenous and recurrent Markov process and $\mathbb{C}$ be the set of weighted cycles generated using the above-mentioned deterministic approach. The set of cycles in $\mathbb{C}$ is a decomposition of ξ that satisfies Theorem 1.

Proof: Following the cycle generation procedure mentioned above, it is trivial to show that:

$$\begin{aligned} p_i p_{ij} &= \sum_{(v_i, v_j) \in C} W(C) = \sum_{C \in \mathbb{C}} W(C) J_C(v_i, v_j) \\ p_i &= \sum_{v_j} \sum_{(v_j, v_i) \in \mathbb{C}} W(C) = \sum_{v_i \in \mathbb{C}} W(C) \\ &= \sum_{C \in \mathbb{C}} W(C) J_C(v_i) \end{aligned}$$

■

Notice that the cycles generated by the deterministic approach are not unique and depend on the policy for selecting the next state. Moreover, even if the same set of cycles is generated, the weight for each cycle will depend on the order in which the cycles were generated.

Theorem 3: Let ξ be a homogenous and recurrent Markov process and $\mathbb{C}$ be the set of weighted cycles generated using the above-mentioned deterministic approach, then $|\mathbb{C}|$ is of $O(|S|^2)$ where $|S|$ is the number of states in ξ.

Proof: Because in each iteration of the deterministic approach, after extracting a cycle, weight of at least one edge becomes zero, therefore, $|\mathbb{C}|$ is $O(|S|^2)$. ■

Having described the mathematical framework for FSM cycle decomposition, we now proceed with the state-encoding problem. As mentioned earlier, the technique for state assignment proposed here is based on the decomposition of FSM into a set of cycles. Due to the high complexity of the probabilistic method and since the number of cycles in that approach can grow exponentially large (in the number of states of the FSM), the deterministic method will be used in this paper to generate the cycles. The algorithm *generate_cycles* for generating the cycles of any given FSM ξ is shown in Figure 2.

```
algorithm generate_cycles (FSM $\xi$)
begin
1.   $\mathbb{C} = \varnothing$;
2.   while max. edge weight > 0
3.       pick ($v_i$, $v_j$) w/ max. weight;
4.       $\Phi = \varnothing$;
5.       repeat
6.           $\Phi = \Phi + (v_i, v_j)$;
7.           pick ($v_i$, $v_j$) w/ max. weight in ($E - \Phi$);
8.       until there is a cycle in $\Phi$;
9.       $C$ = cycle_of ($\Phi$);
10.      $W(C) = \min. \{w_{ij} | J_C(v_i, v_j) = 1\}$;
11.      $\mathbb{C} = \mathbb{C} \cup \{C\}$;
12.      $w_{ij} = w_{ij} - W(C), \quad \forall i, j : J_C(v_i, v_j) = 1$;
13.  return $\mathbb{C}$;
end
```

Fig. 2: Cycle Generation Algorithm

In this algorithm, the weight of each edge is the total transition probability of that edge; i.e., $w_{ij} = p_i p_{ij} \quad \forall i, j = 1..|S|$

The algorithm starts with the edge with maximum weight in line 3 and then repeatedly picks the maximum weight edges from the set of edges (line 7). This process is repeated until a cycle is generated (line 8); the cycle weight will then be set to the minimum of all the edge weights on the cycle (line 10), and the weights of all the edges on the cycle will be decreased by the cycle weight (line 12). This process is repeated till there are no more edges with non-zero weights.

Now that the FSM has been decomposed into a set of cycles, the *cycle_encode_states* algorithm of Figure 3 is employed for the state assignment step.

```
algorithm cycle_encode_states (FSM $\xi$)
begin
1.  $\mathbb{C}$ = generate_cycles ($\xi$);
2.  $\dot{C}$ = sort ($\mathbb{C}$);
3.  generate_Gray_code_table ( );
4.  while $\dot{C} \neq \varnothing$
5.      C = get the cycle w/ max. weight from $\dot{C}$;
6.      encode_cycle (C);
7.      remove from $\dot{C}$ those cycles in which
        more than t% of their states are encoded;
8.  V = get all remaining uncoded states;
9.  MWHD (V);
end
```

Fig. 3: State Assignment Algorithm

After all of the cycles are generated in line 1, they are first sorted according to their weights (line 2) and then a table of all the Gray codes for the minimum required bit count is generated. The number of such Gray codes will be $2^{\lceil lg_2|S| \rceil}$ (line 3). The cycles will then be encoded one by one according to the sorted order by using the *encode_cycle* algorithm. This algorithm assigns the codes to the states on the cycle in such a way that the hamming distance of each state from its neighboring states is minimized. However, this is not feasible for those cycles whose states are partially encoded as part of a previously encoded cycle (line 7). In fact, when the number of already-encoded states in a cycle is sufficiently large, it makes more sense to switch from a Gray coding scheme to a Minimum Weighted Hamming Distance algorithm (MWHD). That is precisely what *cycle_encode_states* does for the few uncoded states in these cycles (lines 8, 9).

Gray codes are used to encode states in a cycle because Gray codes are the optimal solution with respect to the considered cost function, i.e., the minimum weighted hamming distance. Consider a table of Gray codes shown in Figure 5; codes are divided into two sets. A code is a *high-code* if its MSB is 1, and is a *low-code* if its MSB is 0. A table of 2^{n+1} Gray codes can inductively be constructed from a table of 2^n Gray codes following a very simple procedure:

1. Write a 2^n Gray code table

2. Concatenate the above table with a copy of itself written in reverse order.

3. Add a 0 as the MSB of the entries in the first half of the table and a 1 as the MSB of the second half.

Constructing a Gray code table according to this simple procedure, low-codes will always be in top and high-codes will be at bottom. The line which separates the high-codes from low-codes is called *middle-line*. Note that the only difference between high and low codes with equal distances from the middle-line is the MSB, thus they have a hamming distance of one. Moreover, each code differs from its neighboring codes only in one bit (this is the well-known Gray code property). Given a cycle *C*, we can encode it optimally by following a *ping-pong* movement in the Gray code table starting from the very first high-code under the middle-line and choosing the codes in high, low, low, high, high, low, ... order as shown in figure 6.

Figure 4 shows the proposed heuristic for encoding cycles. For

```
algorithm encode_cycle (cycle C)
begin
1.  start = find_best_rotation (C);
2.  foreach uncoded state on C starting at start
3.      high = find first available high-code;
4.      low = find first available low-code;
5.      code(state) = find_best_code (high, low);
end
```

Fig. 4: Cycle Encoding Heuristic

those cycles, none of whose states are encoded, it makes no difference which state to start with for encoding. As we proceed with other cycles, some of the states will have already been encoded, and therefore, we must first determine what state to start with. In line 1, *find_best_rotation* returns the beginning of the largest consecutive sequence of already encoded states on the cycle. In line 2, the algorithm will proceed with all the uncoded states on the cycle.

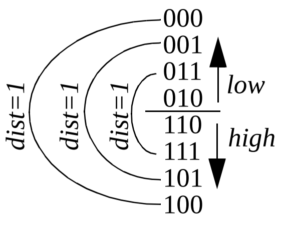

Fig. 5: Gray code table

The *high* and *low* codes are selected as candidate codes for the yet-uncoded state (lines 3, 4) and *find_best_code* is used to compare the cost for each of the two candidate codes and pick the best one (line 5). This function follows the Gray code sequence if the cycle is not partially encoded, but if some of the states in the cycle are already encoded, this function chooses the nearest code (in terms of the hamming distance cost) to the last encoded state and continues with Gray codes from there on. As a secondary tie breaking term, the low and high codes balance is used. This is useful because the ping-pong movement will always be possible if we keep a balance between the number of high and low codes that have been used.

4. Hybrid FSM Implementation

As mentioned before, D flip-flops are commonly used as state registers when implementing FSMs because the combinational logic for calculating the next state tends to be more complex for T flip-flops.[1] Wu et al. showed that using T flip-flops for BCD counters, which exhibit a purely cyclic behavior, results in a very simple combinational logic realization, thereby, very low power consumption [10]. In the case of counter's state lines, the bit-level switching activity of state lines increases as one moves from the MSB toward the LSB. So it pays off to use T flip-flops for the LSB state lines. More generally (and intuitively), if one uses T signals to implement high switching activity state lines, then the on-set of the logic function that produces the T signal would include most of the minterms in the Boolean space, resulting in simpler combinational logic realization, and therefore, lower power consumption.

This ituitive (yet inexact) observation is the key rationale behind using both T and D flip-flops for low power realization of FSMs, primarily because the encoding technique proposed in this paper is based on the cycle realization of FSM.

Of all generated cycles for a given FSM, the ones with larger weights, which are considered first, are optimally encoded, i.e., the states are encoded using consecutive Gray codes, but those with smaller weights, which are encoded later, may end up having non-consecutive codes. When FSM runs and the circuit makes transitions from one state to another, each flip-flop experiences some switching activity. The best way of implementing this FSM would be to use D flip-flops for those bits that have smaller switching activity, and T flip-flops for those that have larger activity.

Consider *encode_cycle,* which starts with the middle-line (cf. Figure 6). To keep the minimum distance of 1 between each state and its neighboring states, the algorithm experiences a ping-pong like behavior, switching between high and low codes. Assuming a cycle over all states of FSM, one can see that with the exception of the most significant bit (leftmost bit), the switching activity is higher for lower-order bits. The most significant bit (MSB) is an exception and has the largest switching activity among all bits (cf. Figure 6). .

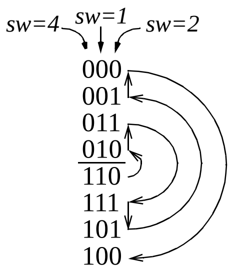

Fig. 6: Ping-pong encoding and switching activity of state bits

Based on the above observations, the high-order bits (but not the MSB itself) are implemented with D flip-flops whereas the low-order bits and the MSB, are implemented with T flip-flops. The boundary between high-order and low-order bits and hence the number of bits to be implemented using T flip-flops is chosen by the following equation:

$$T\ flip\text{-}flop\ count = \lceil lg_2 n \rceil$$

where *n* is the total number of states on those cycles that are encoded using the *encode_cycle* algorithm. Note that the cycles encoded by *encode_cycle* are the ones that end up having a consecutive set of Gray codes and are hence suitable for T flip-flop implementation.

5. Experimental Results

The cycle-based encoding algorithm was implemented in C and run on an IBM IntelliStation with a 730 MHz Pentium III processor and 256 MB memory to generate the experimental results.

The total weighted Hamming distance (WHD) for a number of FSM benchmark circuits and for different encoding techniques are reported in Table 1. For these results, we assumed uniform external input distribution and used equation (1) to calculate the WHD value for each state machine. The purpose of this table is to demonstrate the relative efficiency of the cycle-based encoding algorithm compared to a genetic search based algorithm. .

FSM	state #	cycle-based WHD	cycle-based time (s)	genetic search WHD	genetic search time (s)
dk16	28	1.89	0.6	1.83	220
dk512	15	1.35	0.4	1.42	130
donfile	24	1.45	0.5	1.39	165
ex1	21	0.72	0.7	0.70	155
ex2	19	1.61	0.6	1.60	100
ex5	9	1.33	0.4	1.29	42
ex7	10	1.46	0.4	1.35	56
planet	48	1.31	0.8	1.86	600
s208	256	0.50	1.6	3.55	3450
s298	218	3.12	1.2	4.50	2875
s820	25	0.49	0.5	0.55	170
s953	20	0.37	0.6	0.45	300
s1488	48	0.35	1.0	0.36	700
sand	32	0.65	0.8	0.82	390
sse	16	0.78	0.5	0.88	150
styr	30	0.57	0.7	0.61	250
tbk	32	1.05	0.8	1.17	400
train11	11	0.41	0.4	0.46	5

Table 1: Average Switching Activity

The first column provides names of the FSM circuits, which are all selected from LGSynth89 or ISCAS89 benchmark sets. The largest circuit used for generating experimental results has 256 states. Column 2 shows the number of states in each FSM. Columns 3 and 4 report the average switching activity per state bit line and the runtime (in seconds) for the proposed cycle-based state assignment (i.e., the *cycle_encode_states* algorithm). All FSMs were encoded in the order of one second or less. Columns 5 and 6 report the average switching activity and runtime for a genetic search algorithm that was implemented to calculate the low power state assignment based on the minimum weighted hamming distance cost. For these results, parameter *t* (cf. line 7, Figure 3) was set to 40% for all of the above experiments. The results show a significant speedup compared to the genetic search algorithm with nearly the same quality of results. The case of *s208* is a notable exceoption, where we obtain more than 70% reduction in the WHD metric. The reason for this siginificant improvement is that *s208* is indeed a 256 state counter. As a result we perform much better than the genetic search algorithm (the quality of GA solution may improve if it is given more computation time)

1. There are also legacy reasons. The use of D flip-flops is so common that almost all of the state assignment techniques published to-date support only D flip flops; this introduces a barrier to widespread and effective use of T flip-flops. A barrier that we are attempting to lower in this paper.

FSM	genetic search Power	genetic search Area	cycle-based Power	cycle-based Area
dk16	1650	97852	1760	99539
dk512	429	32976	427	32864
donfile	1106	85012	1215	84711
ex1	983	72985	968	73538
ex2	997	67862	1089	68544
ex5	572	35676	507	36606
ex7	581	40227	634	41576
planet	1326	117382	1179	110112
s208	362	63120	220	54351
s298	8943	985651	8307	917559
s820	1269	134263	1137	130741
s953	1311	113176	1047	106007
s1488	1054	159124	923	153947
sand	1623	134278	1455	129910
sse	629	48319	459	43044
styr	926	134257	840	131997
tbk	2391	173216	2135	171718
train11	310	30128	240	29533

Table 2: Genetic v.s. cycle_encode_cycle

Table 2 shows the post-mapping area and power consumption of the FSMs using genetic search generated state codes v.s. *codes generated by cycle_encode_states* algorithm. Fidelity of *WHD* cost function can be noticed by considering tables 1 and 2 side by side. For the next set of experimental results, we investigate the effect of using both T and D flip flops to implement an FSM that has been coded by the *cycle_encode_states* algorithm. *script.rugged* was then used in *SIS* to optimize the encoded circuit, which was subsequently technology mapped to an industrial 0.25 μm ASIC cell library. A gate-level power simulation tool was used to calculate the power consumption of the final circuit. We used a uniformly-distributed vector set of size 100,000 to excite each circuit from a randomly selected initial state or the reset state (if specified). The results are reported in Table 3.

FSM	Power D-FF	Area D-FF	Power D/T-FF	Area D/T-FF	Power savings
dk16	1760	99539	1364	96302	18%
dk512	427	32864	401	32369	2%
donfile	1215	84711	1177	69504	21%
ex1	968	73538	832	78801	13%
ex2	1089	68544	995	65266	15%
ex5	507	36606	429	36788	9%
ex7	634	41576	595	37151	19%
planet	1179	110112	1153	107227	4%
s208	220	54351	118	40718	25%
s298	8307	917559	7688	864484	8%
s820	1137	130741	977	102539	14%
s953	1047	106007	922	90876	12%
s1488	923	153947	716	130737	22%
sand	1455	129910	1366	121025	6%
sse	459	43044	398	40839	13%
styr	840	131997	639	125992	24%
tbk	2135	171718	1904	168050	11%
train11	240	29533	103	27011	35%

Table 3: Final Power Consumption

Columns 2 through 5 report the power consumption and mapped gate area of the resulting circuits for D flip-flop only and the hybrid flip-flop implementations, respectively. The percentage improvement in power dissipation is reported in column 6. As one can see, the hybrid implementation always results in lower power-consuming circuits compared to D flip-flop-only implementation. Experimental results demonstrate an average improvement of 15% in the total (post-mapping) power dissipation and an area savings of 10% in terms of the total gate area. Note that T flip-flops can easily be implemented using clock gating on D flip-flops as shown in [10].

6. Conclusions

A mathematical framework for the cyclic representation of FSMs is introduced. Based on this representation, a new state-encoding technique was proposed to reduce switching activity in FSMs. Since a cyclic behavior is better implemented by T flip-flops, a hybrid-flip-flop scheme for FSM implementation based on cycle decomposition was proposed to achieve further power reduction in the final circuit. Experimental results show the effectiveness of this approach.

References

[1]L. Benini and G. De Micheli, “Automatic Synthesis of Low-power Gated-clock FSM,” *IEEE TCAD*, vol. 15, pp. 630-643, Jun. 1996.

[2]L. Benini, G De Micheli, E. Macii, M. Poncino, and R. Scarsi, “Symbolic Synthesis of Clock-Gating Logic for Power Optimization of Control-Oriented Synchronous Networks,” *Proc. of ICCAD*, pp. 514-520, Nov. 1997.

[3]G. De Micheli, R. K. Brayton, and A. Sangiovanni-Vincentelli, “Optimal State Assignment of Finite State Machines,” *IEEE Trans. on CAD*, vol. 4, pp. 269-285, Jul. 1985.

[4]S. L. Kalpazidou, *Cycle Representations of Markov Processes*, Springer-Verlag, 1995.

[5]E. Olson and S. M. Kang, “Low-Power State Assignment for Finite State Machines,” *Proc. of IWLPD*, pp. 63-68, April 1994.

[6]M. Pedram, “Power Minimization in IC Design: Principles and Applications,” *ACM Trans. on Design Automation of Electronic Systems*, vol. 1, pp. 3-56, Jan. 1996.

[7]K. Roy and S. Prasad, “Syclop: Synthesis of CMOS Logic for Low-Power Application,” *Proc. of ICCD*, pp. 464-467, Oct. 1992.

[8]C. Y. Tsui, M. Pedram and A. M. Despain, “Low-Power State Assignment Targeting Two- and Multilevel Logic Implementation,” *IEEE Trans. on CAD*, vol. 17, pp. 1281-1291, Dec. 1998.

[9]T. Villa and A. Sangiovanni-Vincentelli, “NOVA: State Assignment of Finite State Machines for Optimal Two-Level Logic Implementations,” *IIEEE Trans. on CAD*, vol. 9, pp. 905-924, Sep. 1990.

[10]X. Wu, J. Wei, Q. Wu, and M. Pedram, “Low-Power Design of Sequential Circuits Using a Quasi-Synchronous Derived Clock,” *Int’l Journal of Electronics*, Taylor and Francis Publishing Group, vol. 88, no. 6, pp. 635-643, Jun. 2001.

[11]S. Yang and M. Ciesielski, “On the Relationship Between Input Encoding and Logic Minimization,” *Proc. of 23rd. Hawaii Int’l Conf. System Sciences*, vol. I, pp. 377-386, Jan. 1990.