Low-power Synthesis of FSMs with Mixed D & T Flip-Flops

Ali Iranli, Peyman Rezvani and Massoud Pedram
Department of Electrical Engineering – Systems
University of Southern California

Outline

- Background
  - FSM & Markov Process
  - State Assignment – Area / Power
  - Prior Work
- Markov Process Cycle Decomposition
- Cycle-based State Assignment
- Hybrid FSM Realization
- Conclusion

Outline

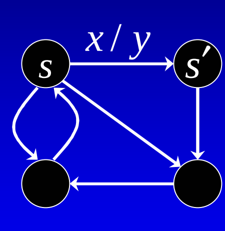
- Background
  - FSM & Markov Process
  - State Assignment – Area / Power
  - Prior Work
- Markov Process Cycle Decomposition
- Cycle-based State Assignment
- Hybrid FSM Realization
- Conclusion

Finite State Machines

- Model behavior of sequential circuits by finite state machines.
- Finite State Machine (FSM):
  - $(X, Y, S, s_0, \lambda, \eta)$
  - X: Set of input symbols
  - Y: Set of output symbols
  - S: Set of states
  - s_0: Initial state - $s_0 \in S$

    λ: Output function - $\lambda : X \times S \to Y$

    η: Next state function - $\eta : X \times S \to S$



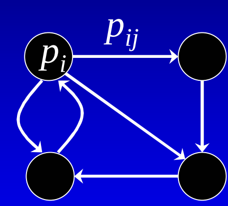
FSM & Markov Process

- Model probabilistic behavior by a *Markov Process*
  - FSM state ⇔ Markov Process state

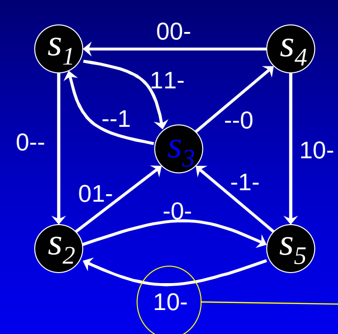
    $$p_i = \Pr(s_i)$$

  - FSM transition ⇔ Markov process transition

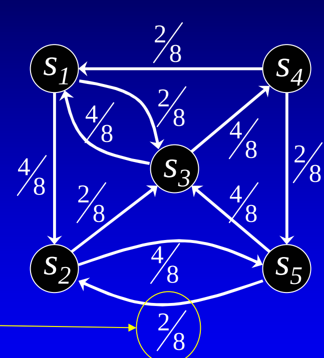
    $$p_{ij} = \Pr(s_i \to s_j | s_i)$$



FSM & Markov Process (cont'd)



2 out of 8
Input combinations



Finite State Machine

Markov Process

FSM & Markov Process (cont'd)

FSM
Input Combinations

Input Probability
Distribution

Conditional Transition
Probability (p)

Chapman-Kolmogorov

$\pi \times p = \pi$

State
Probability (π)

$$\pi = \begin{bmatrix} \cdots & p_i & \cdots \end{bmatrix} \quad p = \begin{bmatrix} \ddots & & \cdots \\ & p_{ij} & \\ \vdots & & \ddots \end{bmatrix}$$

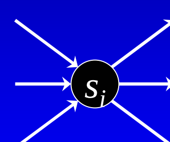
FSM & Markov Process (cont'd)

- *Irreducible* Markov Process:
  - Any state s_j can be reached from any state s_i

$$p_{ij}^{*} = \sum_{n=1}^{\infty} p_{ij}^{n} \succ 0$$

- *Recurrent* Markov Process:
  - Any state s_i is reachable from itself; i.e.,

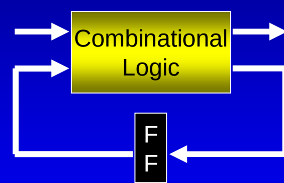
$$\sum_j p_{ij} = 1 \Rightarrow \sum_j p_i p_{ij} = \sum_j p_j p_{ji}$$



- The Markov process modeling a FSM is irreducible & recurrent

State Assignment

- Encoding Problem:
  - Transform a cover of symbolic logic function into a cover of binary logic function
  - Three classes:
    - Input Encoding
    - Output Encoding
    - Input / Output Encoding
- State Assignment
  - Input / Output Encoding
  - Difficult problem!



State Assignment

State Assignment Problem:

– Assign unique codes to each state of an FSM in order to optimize an objective function
  - Area
  - Circuit speed
  - Power dissipation

Prior Work

- Area:
  - *Armstrong '62* – graph embedding approach
  - *De Micheli, et al '85* – algebraic approach to input encoding
  - *Devadas, et al '91* – algebraic approach for output & state encoding
  - *Sangiovanni, et al '90* – graph embedding approach to state encoding (NOVA)
  - *Newton, et al '88, '91* (MUSTANG, MUSE) – state encoding for multilevel realization
- Power
  - *Roy, et al '92* – state encoding for state line switching activity
  - *Olson, et al '94* – state encoding for state line switching activity + literal count
  - *Pedram, et al '98* – low power state encoding considering switched capacitance in resulting logic

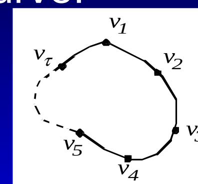
Outline

- Introduction
- Background
  - FSM & Markov Process
  - State Assignment – Area / Power
  - Prior Work
- **Markov Process Cycle Decomposition**
- Cycle-based State Assignment
- Hybrid FSM Realization
- Conclusion

What is a cycle?

- A particle's motion on a closed curve:

$$C = (v_1, v_2, \ldots, v_\tau, v_{\tau+1} = v_1, v_{\tau+2} = v_2, \ldots)$$



- Directed cycle *C* over set of states *S*:
  - Periodic function from $\mathbb{Z}$ into *S*
  - *C*(*i*) vertex -- state *i* in the cycle
  - (*C*(*i*), *C*(*i+1*)) directed edge – a transition in the cycle

Cycles - notation

- C_1 and C_2 are *equivalent* if and only if:

$C_1(i) = C_2(t(i))$ where t is a translation function over $\mathbb{Z}$

- First order passage function of *C*:

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

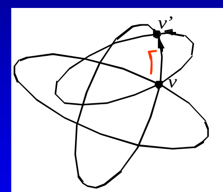
- Second order passage function of *C*:

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

Markov Process & Cycles

- Consider a set of r cycles: $C=\{C_1, C_2, \dots, C_r\}$
- Each cycles C_i is associated with a positive number $W(C_i)$.
- v lies on $C_1,\dots,C_t \quad (t \le r)$
- v' lies on $C_1,\dots,C_m \quad (m \le t)$
- A measure of transition from v to v':

$$\frac{W(C_1)+W(C_2)+\dots+W(C_m)}{W(C_1)+W(C_2)+\dots+W(C_t)}$$



- Conclusion: A collection of weighted cycles defines a Markov Process

Markov Process & Cycles

Theorem:

– Let S be a finite set of states. Consider a recurrent Markov process ξ defined over S. There exists a finite set of weighted cycles $\mathbb{C}$ such that superimposing them defines ξ:

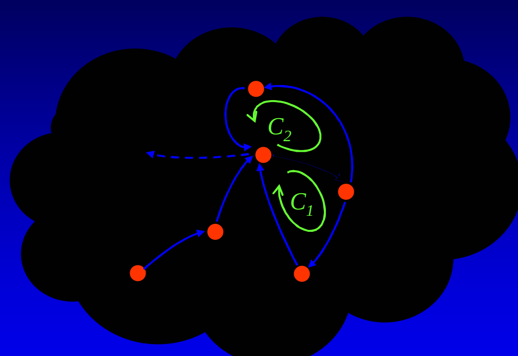
$$p_i = \sum_{C\in\mathbb{C}} W(C)\, J_C(v_i)$$
$$p_{ij} = \sum_{C\in\mathbb{C}} W(C)\, J_C(v_i,v_j) / p_i \qquad \forall v_i, v_j \in S$$

Cycle Decomposition

Cycle Decomposition (CyDec) Problem:

- Given a recurrent Markov process ξ defined over a set of states *S*, find a set of weighted cycles $\mathbb{C}$ such that their superposition defines the given Markov process.
- Solution is not unique.
- Two classes of solutions:
  - Randomized approaches
  - Deterministic approaches

Randomized Approach



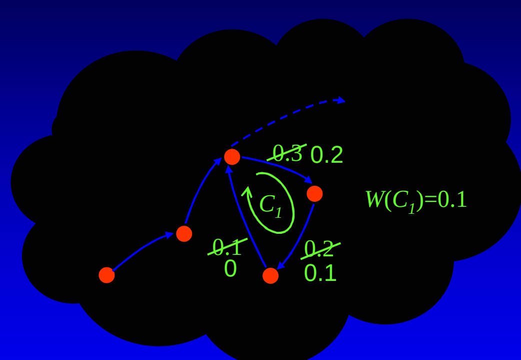
Randomized Approach (cont'd)

```
repeat forever
     pick a random state;
     make random transitions until a cycle C is
     add the cycle C to the set of cycles ℂ;
calculate cycle weights by solving set of equati
```

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

– All possible cycles are found.
– Cycle weights are unique.
– Number of cycles can grow exponentially.

Deterministic Approach



Deterministic Approach (cont'd)

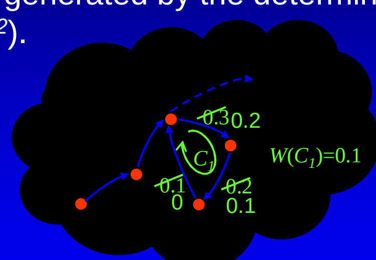
```
repeat
    pick a transition;
    make transitions until a cycle C is recog
    W(C) = min. prob. of transitions on C;
    add the cycle C to the set of cycles ℂ;
    decrease prob. of each transition on C by
until no more transition is left;
```

- Cycle set is not unique.
- Weights depend on cycle generation order.

Deterministic Approach (cont'd)

Theorem:

- Number of cycles generated by the deterministic approach is $O(|S|^2)$.



- After each cycle extraction, probability of one transition becomes zero.
- Edge count is $O(|S|^2)$.

Outline

- Introduction
- Background
  - FSM & Markov Process
  - State Assignment – Area / Power
  - Prior Work
- Markov Process Cycle Decomposition
- Cycle-based State Assignment
- Hybrid FSM Realization
- Conclusion

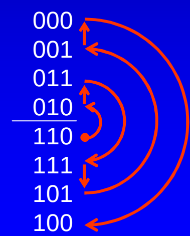
Cycle-based State Assignment

Algorithm Flow:

- Build Markov process representing the FSM
- Find cycle decomposition $\mathbb{C}$
  - Use the deterministic approach
- Encode each cycle in $\mathbb{C}$ separately

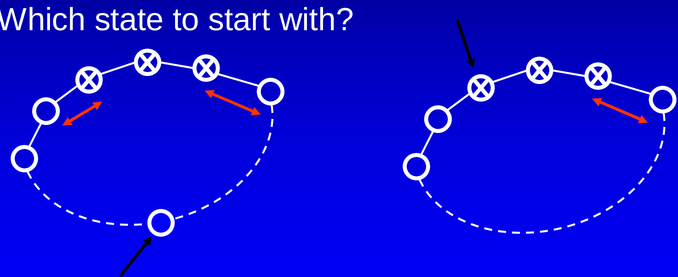
Ping-Pong Encoding

- States on a cycle can be optimally encoded using *Gray* code.
- Start with the middle Gray code.
- Encode states on cycle while maintaining a Hamming distance of 1.



Ping-Pong Encoding

- After some cycles are encoded, the rest of cycles will be partially coded; hence they may not be optimally encoded.
  – Which state to start with?



  – Start with the longest sequence of coded states.

Cycle-based State Assignment

```
build Markov process for ξ FSM;
ℂ= cycle decomposition of ξ;
sort cycles in ℂ according to their weights;
generate gray codes;
for each cycle in ℂ:
     if most of states on cycle are not coded
          ping-pong encode the cycle;
```

- Number of gray codes:

$$2^{\lceil \log |S| \rceil}$$

Outline

- Introduction
- Background
  - FSM & Markov Process
  - State Assignment – Area / Power
- Markov Process Cycle Decomposition
- Cycle-based State Assignment
- Hybrid FSM Realization
- Conclusion

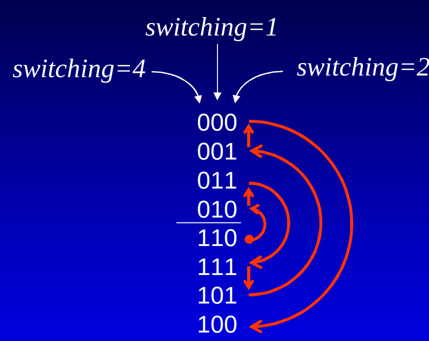
FSM Implementation

- Using D flip-flops: $D = \eta(x, s)$
- Using T flop-flops: $T = \eta(x, s) \oplus s$
  - T flip-flops usually result in more complex combinational logic realization.
  - T flip-flops are more efficient for counters. (*Wu et al.*)
- Do T flip-flops work better with cycle-based state assignment?

Hybrid FSM Realization

- Ideal:
  - Cycles with larger weights are encoded first ⇨ states are encoded with min. Hamming distance ⇨ better candidates for T flip-flop implementation.
  - Cycles with smaller weights are encoded last ⇨ states are encoded with larger hamming distance ⇨better candidates for D flip-flop.
- Impossible:
  - All states are implemented using the same flip-flops.

Hybrid FSM Realization



– High-order bits have smaller switching
  - MSB is an exception. It has the largest switching activity.
  - Use D flip-flop for high-order bits; T flip-flop for low-order bits.

ASP-DAC 2003 31/38

Hybrid FSM Realization

- States are encoded by:
  – Ping-pong encoding
  – Minimum Weighted Hamming Distance (MWHD)
- *n*: Number of states on those cycles encoded by ping-pong encoding
- Use $\lceil \log n \rceil$ T flip-flops for low-order bits (and MSB).
- Special Case – counters:
  – All states are encoded by ping-pong.
  – All bits are implemented by T flip-flop.

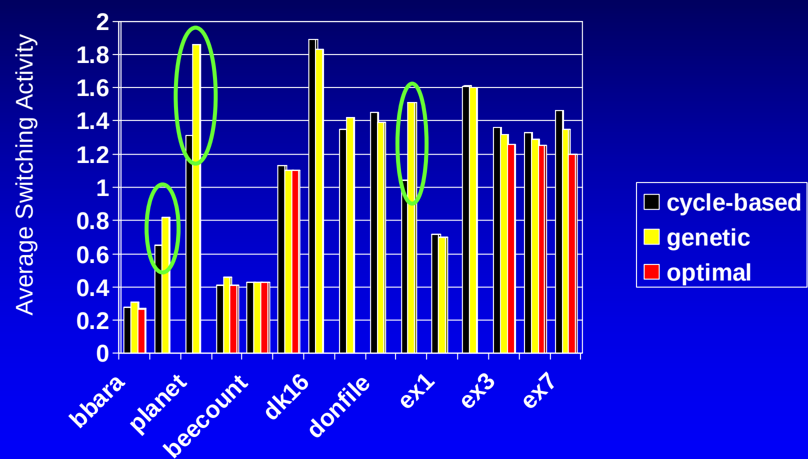
ASP-DAC 2003 32/38

Experimental Results - I

- Comparison of:
  - Average switching activity of bits on state lines
  - Runtime
- Finite state machines from LGSynth89
- Encoded using 3 techniques:
  - Cycle decomposition
  - Genetic search
  - Optimal (exhaustive search)

ASP-DAC 2003 33/38

Experimental Results - I



ASP-DAC 2003 34/38

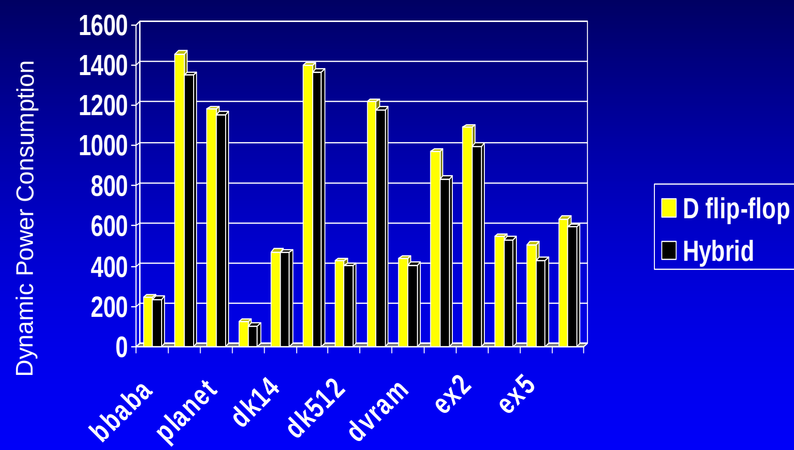
Experimental Results - I

FSM	State #	Cycle-based Runtime (s)	Genetic runtime (s)
bbara	10	0.4	10
sand	32	0.8	390
planet	48	0.8	600
train11	11	0.4	5
beecount	7	0.3	5
dk14	7	0.3	2
dk16	28	0.6	220
dk512	15	0.4	130
donfile	24	0.5	165
dvram	35	0.8	435
ex1	21	0.7	155
ex2	19	0.6	100
ex3	10	0.4	53
ex5	9	0.4	42
ex7	10	0.4	56

Experimental Results - II

- Encoded finite state machines by cycle-based state assignment method.
- Implemented using D vs. D/T flip-flops.
- Used *script.rugged* in SIS to optimize resulting circuit.
- Technology mapped to a 0.25 μm library.
- Estimated power using 100,000 uniformly distributed input vectors.
- Achieved as much as 15% reduction in the total dynamic power dissipation.

Experimental Results - II

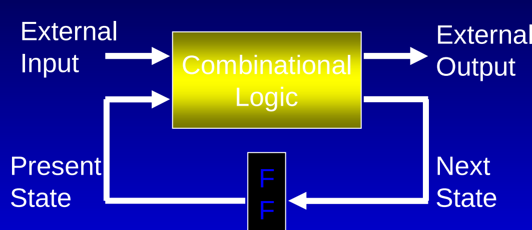


Conclusion

- Proposed a state assignment technique based on cycle decomposition of Markov processes to minimize switching activity on state bit lines.
- Proposed a hybrid implementation technique for finite state machines using both D and T flip-flops which, in conjunction with cycle-based state assignment method, significantly reduces dynamic power consumption

BACK UP SLIDES

Dynamic Power Consumption

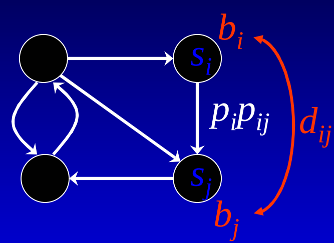


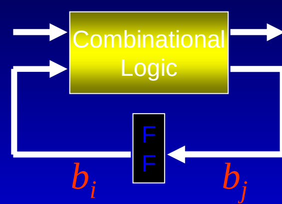
- Dynamic Power Consumption:

$$P_{ave} = \frac{1}{2} V_{dd}^2 f \sum_{\text{all gates}} C_{load} E_{switching}$$

  - V_{dd} Supply voltage
  - f Clock frequency
  - C_{load} Capacitive load of gate
  - $E_{switching}$ Average number of changes in output of gate

Power & Switching Activity





Objective: Minimize average switching activity on state bit lines

$$\min \sum_i \sum_j p_i p_{ij} d_{ij}$$

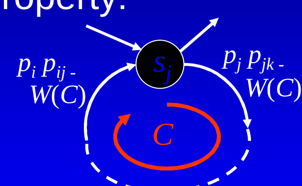
The above problem, know as Minimum Weighted Hamming Distance (*MWHD*), is NP-Complete.

Deterministic Approach (cont'd)

Theorem:

– After extracting each cycle, all state pairs s_i and s_j will still have the property:

$$\sum_j p_i p_{ij} = \sum_j p_j p_{ji}$$



- Therefore extraction of cycles in deterministic approach can be iterated.

Cycle Encoding

- States on a cycle can be optimally encoded using *Gray* code.

000
001
011
010
110
111
101
100

Hamming Distance 1

ASP-DAC 2003 43/38

Ping-Pong Encoding

Encode states such that each state has minimum distance from its neighbors.

```
find best starting state on cycle;
for each un-encoded state on cycle:
     Low = first available code above middle
     High = first available code below middle
     code (state) = pick_better (Low, High);
```

Low → 010
High → 110

000
001
011
010
110
111
101
100

ASP-DAC 2003 44/38