Wave-Steering One-Hot Encoded FSMs

Luca Macchiarulo, Malgorzata Marek-Sadowska, University of California, Santa Barbara

Abstract

In this paper we address the problem of pipelining FSMs by extending wave-steering scheme from combinational to sequential realm. A unified approach employs direct mapping of State Transition Graph into a circuit realization. Experimental results on MCNC benchmarks show performance improvement of 2 to 4 times at the cost of an average area increase of 2.9 times.

1.0 Introduction and Motivation

Pipelining is a commonly used technique when cycle time is pushed to a limit. In the majority of cases the pipelined circuits are combinational. Efforts to implement pipelining schemes in sequential circuits have been limited, and very few papers addressed [11] the issue of a general design method to pipeline FSMs. Applicability range of such machines is not at all negligible. They are needed in vast areas of communication as digital filters, convolutional and variable-length decoders [7], and also in control dominated applications [10]. In general high throughput FSMs are desirable for all applications where it is less important to reduce the latency between inputs and outputs than it is to reduce the time interval between two consecutive legal outputs. The work [11] identifies the key problem of pipelining FSMs in the presence of *iteration bound*. The basic observation is that a sequential machine, by its very nature, has a memory of the past imposing order relations between past and future signals that can't be satisfied by a naive pipelining of the combinational portion of the logic. The studies which address the problem [7][11] tend to target specific designs or classes of designs. In this work we propose a methodology that is not restricted to FSMs for a specific class of applications, and is potentially applicable to all sequential machines that can be conveniently described by State Transition Graphs (STGs). The key idea involves a direct mapping of one-hot encoded STG. The one-hot encoding scheme helps to decouple interactions between different states. In order for the methodology to be viable we need the flexibility to manipulate the pipeline at the bit level. The technique of *wave steering* [12], which is very suitable for design automation, fits perfectly into the scheme.

This work was sponsored in part by the GSRC DARPA-MARCO grant and in part by NSF CCR 9811528 grant.

DAC 2000, Los Angeles, California

2.0 Wave steered combinational circuits

Extending pipelining techniques from architectural to bit-level has led to aggressively fast designs, especially when applied to dynamic logic circuits. However, lack of a widespread design methodology has confined the use of these otherwise powerful techniques to designs where performance is the primary goal (like data-paths of processors) justifying a full custom approach.

The new approach of **wave steering** [12], that has already shown some of its potential, implements micro-pipelining in a different fashion than it is done for the classic dynamic circuits. The central idea is to allow several signal waves co-exist in a combinational circuit, each wave collecting information on different variables as it proceeds towards the output. If we consider a BDD in figure 1a we can see that, traversing it bottom up, and choosing the path according to the values of the variables, we progressively push the information about the lower variables towards the top of the diagram. The implementation consists of mapping each BDD node to

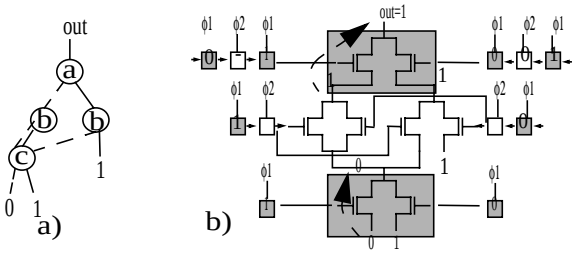


Figure 1. Function ab+c and its wave-steered implementation.

a Pass Transistor Logic multiplexer followed by a buffer (like in figure 1b, where the buffers are not shown) [3][4][14][15] and pipelining it. The pipelining is achieved not by the use of clocks in the signal evaluation path but only by skewing the inputs through appropriate fast dynamic flip-flops. The skew is obtained through the asymmetrical arrangements of flip-flops, called feeders, so that the signals arrive at the pass transistor gates in the sequence c-b-a, each delayed by half a clock period from its predecessor. In this way at each clock cycle every other level can be active and computing information relative to a different output vector. This yields an extremely fine-grained pipeline, operating at the level of a single bit, allowing for very short clock periods. On the other hand, the use of BDDs renders the method homogeneously applicable to any type of problems, with a unified methodology that requires small or no manual intervention to ensure functionality of the design. Furthermore, the attainable performance does not require any complex optimization process on transistors' dimensions or features of the layout. Recent experiments show extremely good improvements in clock frequency (2 to 8 times) for combinational designs with respect to their standard cell implementation, at the expense of an increase in area (1.2 to 3 times) that is acceptable for performance-dominated applications.

3.0 Extensions to sequential systems

The technique of wave steering cannot be applied without modifications to sequential circuit synthesis. The presence of feedback in any circuit causes the iteration bound [11].

A state machine computes the next state as a combinational function of the present state and the present input. Any pipelining scheme (wave steering included) would increase the throughput of the single partial stages of computation, leaving unchanged (or possibly worsening) the overall latency. However, the data dependency between present and future state imposes that, for the next computation to begin, the previous computation be completed (see figure 2a and 2b).

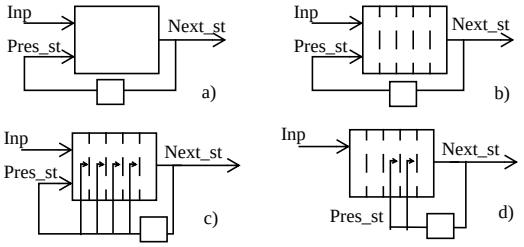


Figure 2. FSMs and pipelining.

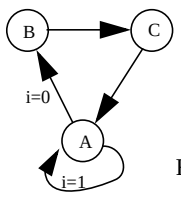
In practice, no matter how fast the partial computations of the next state are performed, the global speed of the machine is given by the latency of next state computations, and the pipeline stages would remain idle for most of the time. The iteration bound imposes a limit on the performance of any FSM implementation of $1/\tau$, τ being the time necessary to compute the next state information. A closer analysis of the problem shows that the bound is not so strict as it seems at a first sight. The value of τ in the iteration bound formula is independent of the computation time related to the primary input variables. Therefore, if there is a way to decouple the primary input variables from the state variables in the computation of the next state, the throughput can be increased. In figure 2d it is shown that this can be achieved by pushing the state variables (distributed in all stages in figure 2c) as late as possible in the pipeline stages. An advantage of the wave-steering scheme is that it offers a uniform technique to re-arrange pipeline stages. It is sufficient to choose the order of variables such that the state variables are positioned at the top of the BDDs representing the transition functions.

Even if this allows the iteration bound to be somewhat relaxed, it can be still too high to compete with classically optimized circuits. One solution is to reduce the mutual dependency between state bits, in such a way that any particular next state bit can be computed on the basis of a smaller number of present state bits. This implicitly calls for a decomposition of the monolitic transition function into smaller communicating blocks. We investigated the applicability of classical decomposition theory ([1][9]) with good results on some machines. In this paper, however, we concentrate on a different technique that can be applied to any machine (regardless its decomposability), and has the attractive feature of mapping the State Transition Graph directly to a feasible implementation.

4.0 Direct mapping of State Transition Graphs

Consider an FSM described by its State Transition Graph shown in figure 3. The transitions between states are labeled by their triggering inputs. After minimal length state encoding has taken place, this representation is only loosely related to the functional description of the State Transition Function.

This, however, is not the case when one-hot encoding is chosen. When every state is assigned its own bit, the resulting transition functions can be directly read from the STG. The relations between the present and next states can be immediately read from the STG by looking at all incoming edges of a state; e.g. the first equation is equivalent to the sentence: *The next state will be state A iff (present state is A AND input i is 1) OR present state is C.*



$$st_A(t+1) = i \bullet st_A(t) + st_C(t)$$
$$st_B(t+1) = \bar{i} \bullet st_A(t)$$
$$st_C(t+1) = st_B(t)$$

Figure 3. STG description of an FSM

If each state bit is implemented by a separate circuit, the number of variables needed by each next state computation is no longer determined by the total number of state bits, but only by those bits that are needed for that particular state. Their number equals the number of incoming arcs to the node representing the state in STG. Even if the one-hot encoding greatly increases the number of state bits (from log2(N) to N), the single iteration bound is given by the maximum number of incoming transitions, which can be (and usually are, besides some special states such as reset states) a small fraction of the number of states.

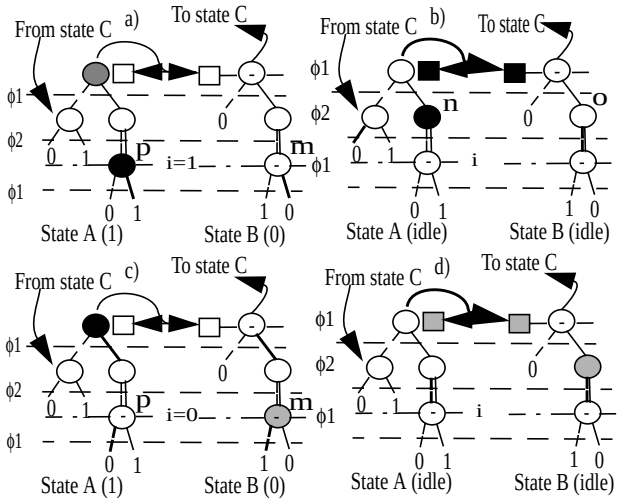


Figure 4. BDD implementation of part of previous FSM

Figure 4 shows a BDD mapped, wave steered implementation of states A and B from fig.3. Note that some signals are generated by the same circuits that use them later as inputs. In fig. 4a we depict the starting point when A is the present state and the active clock period is $\phi 1$. We use the pictorial convention that the similarly shaded areas correspond to active nodes related to the same temporal slice. In this instant, the input i has value 1. The value 1 will eventually appear at the output and from the STG in figure 3 we know that this means that in the next $\phi 1$ phase the active state will have to be A again. At the moment, the input i is such that node p evaluates to 1, node m to 0. In the next phase (fig. 4b) the information propagates to the next level (node n=1, node o=0), while state bit is stored in the two flip-flops (black squares). At the next $\phi 1$ phase the state information and the input information finally appear at the top of the diagrams. At the same time in the bottom level, next input (i=0) is evaluated and this time will propagate bit

1 to node m. This will eventually lead to state B being selected. Even if all BDDs perform simultaneously their computations, from the logical point of view the structure behaves as if a control token, representing the present state, were subsequently passed between the different states, giving them the power to fire the next transition.

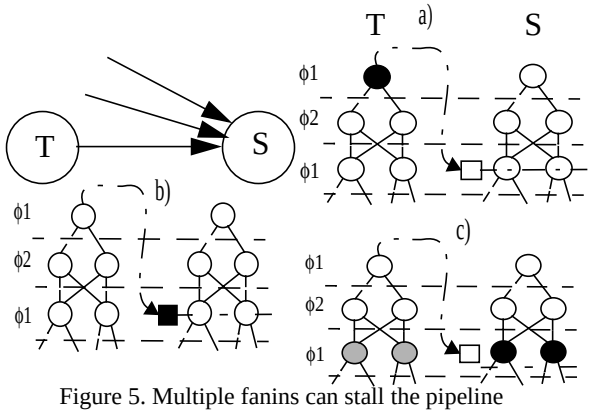


Figure 5. Multiple fanins can stall the pipeline

The delays between the input and the evaluation of its corresponding state change will not reduce the throughput. However, states with many incoming edges will stall the pipeline, as shown in fig. 5 (the equally shaded areas represent nodes evaluating signals in the same time slice). In this case the state S has a fanin of 3. This means that one of the state bits must feed the third level from the top (here the bit coming from the state T). We can follow the signals as they progress in the pipeline. State bit valid in 5(a) is stored in a flip flop 5(b), enters the third level 5(c), and only after a total of 2 clock cycles it will be able to influence the next state. This explains why the next valid level 3 input cannot appear simultaneously with the valid output (as it was the case in fig. 4a), but it has to be delayed by 1 cycle (from snapshot 5(a) to snapshot 5(c)). Only this guarantees a correct synchronization of the state and input signals. The pipeline has to be stalled by M/2 clock periods, M being the number of incoming state bits.

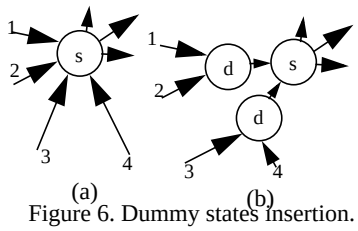


Figure 6. Dummy states insertion.

It is possible to process a pair of bits in the BDDs without stalling the pipeline. FSMs with STGs whose states have fanins of (at most) two can run at the maximum speed. Any FSM can be transformed into an FSM where this condition holds, by adding "dummy states". The concept is explained in the following figure. The diagram of figure 6a can be transformed into that of figure 6b by substituting the state s with fanin of 4 with a tree of 3 states, each with fanin of 2. This results in adding two dummy states, whose purpose is to hold the information regarding the incoming transitions for one clock cycle, before it can be completely processed. Each of the dummy nodes contains a partial information about the next state s. If and only if one of them is active, the logical next state must be s. This means that, in the next clock period, state s can be effectively computed and this information used by the rest of the machine. When any of the "dummy states" is met, the machine delays its computation by one clock period. In order to guarantee a correct behavior, an equal number of dummy states must be added in front of all states, even when this is not strictly necessary. With this methodology, the performance degradation is reduced from M/2 to log2(M)/2, M being the maximum number of incoming edges.

5.0 Circuits and Physical Design

The necessity of evaluating one state variable at a time causes that the presence of states with high "state fanin" jeopardizes all the advantages of pipelined computation. A careful analysis, though, shows that there are no fundamental reasons why all state variables have to be evaluated sequentially. Due to the choice of one-hot encoding, only one state variable can be active in a given clock period. It is possible to compute the next state information by evaluating all the incoming state bits at the same time, through the use of a precharged logic at the top of the diagrams. Spice simulations show that the circuit works at the same high clock rate as that achieved by the rest of the diagram, up to a reasonable limit of fanin states and output loading. When that limit is reached, the strategies illustrated in the previous section can be used.

6.0 Experimental results and comparisons

To evaluate the feasibility of the method we performed experiments with the .kiss format FSMs in the MCNC benchmark suite. For each of the considered state machines, we built the BDDs needed to compute each transition and mapped them into BDD circuits, which were placed and routed, and we took into account loading and electrical effects. SPICE simulations verify that the circuits can run at a clock cycle of 1.6 ns, when implemented in a 0.5μm technology and all parasitic and coupling effects are taken into account. Simulation included dynamic part of the circuit in the presence of long wires, up to 500μm, to ensure connections between blocks. Then we used a commercial tool to place and route different blocks. After obtaining the results, we checked if the loading constraints were satisfied. In the figures, areas include the logic needed to compute outputs. The comparison was made with respect to a standard cell implementation of the same machines (in the same technology)using a commercial tool, the FSMs being previously optimized for performance and mapped by SIS (using a combination of script_rugged and script_delay). The tool performed static timing analysis, providing the speed figures reported.

Table 1 shows that the standard cell implementation of the machines is always more economical area-wise, the wave-steered implementation are 1 to 5 times larger, with an average of 2.85.

The throughput of all listed machines is 1.6ns and is 2 to 4 times better (2.8 on average) than that of standard cell realizations. Due to this substantial gain, the average latency of the pipelined machines is generally very close to (and in many cases - reported in bold - less than) the standard cell delay. The significance of this result, we believe, has to be measured also on the fact that control logic usually tends to result in very shallow and fast implementations, and classical multilevel synthesis algorithms (as those con-

tained in SIS) perform extremely well on the random logic that results. It is therefore interesting that a pure mapping of transitions into a netlist, with few efforts to minimize the logic, can be compared with standard cell, at least on the performance-area tradeoffs.

FSM	#S states	S C Area ($10^3\mu m^2$)	W. S. Area (10^3 μm^2)	S C cycle	S C/ W S cycle	W.S. Latency
bbara	10	11.5	42.4	3.6	2.2	5.6
bbsse	16	26.7	72.5	5.5	3.4	6.4
bbtas	6	8.5	13.5	3.0	1.9	3.2
beecount	7	6.7	18.2	3.6	2.3	4.8
cse	16	42.5	168.6	6.6	4.1	8.8
dk14	7	20.9	56.1	6.1	3.8	**4.8**
dk15	4	15.9	42.2	5.1	3.2	**4.8**
dk16	27	49.6	145.1	8.8	5.5	**5.6**
dk17	8	13.1	34.9	6.0	3.7	**3.2**
dk27	7	7.2	12.9	2.5	1.6	3.2
dk512	15	14.6	28.2	4.5	2.8	**3.2**
ex2	19	17.4	68.6	4.8	3	4.8
ex3	10	8.1	15.2	3.8	2.4	**3.2**
ex4	14	16.3	34.7	3.9	2.4	4.8
ex5	9	14.1	13.3	3.1	1.9	3.2
ex6	8	21.7	53.5	5.4	3.4	**3.2**
ex7	10	7.4	13.3	3.7	2.3	**3.2**
kirkman	16	34.0	49.0	5.2	3.3	11.2
lion	4	8.0	8.5	2.5	1.6	3.2
lion9	9	5.2	9.3	3.0	1.9	3.2
mark1	15	19.6	66.3	4.3	2.7	**4.0**
mc	4	4.8	8.3	4.0	2.5	**3.2**
opus	10	19.5	34.3	4.5	2.8	6.4
pma	24	53.9	156.4	7.1	4.4	**4.0**
s208	18	22.2	96.4	4.1	2.6	9.6
s27	6	10.2	28.7	3.3	2.1	4.8
s386	13	28.2	79.4	4.9	3.1	**4.8**
s420	18	19.2	63.5	4.4	2.8	9.6
s1488	48	123	377.5	6.3	3.94	9.6
shiftreg	8	6.4	11.6	2.9	1.8	**1.6**
sse	16	26.2	63.9	5.1	3.2	6.4
tav	4	8.0	16.4	2.8	1.8	4.8
tma	20	34.0	179.4	6.5	4.1	**6.4**
train11	11	6.0	9.0	2.3	1.4	3.2
train4	4	8.4	13.0	3.3	2.1	**3.2**
		739	2104.1	4.47	2.8	5.01

We list only those machines for which we could guarantee the maximum timing performance. This fixed the limit on circuits dimensions. If this or a similar method is considered in the future for implementation of bigger state machines, an appropriate decomposition technique has to be devised to guarantee the "correct by construction" performance that we believe is one of the most attractive features of this methodology.

7.0 Conclusion and future work

In this work we proposed a pipeline scheme for one-hot encoded FSMs. The proposed method is simple, applicable to any machine, and yields meaningful performance improvements. However, it suffers from the problem of not being easily scalable in its original form, thus limiting its scope to small and medium machines. We think, though, that the use of decomposition techniques and other encoding schemes can alleviate and possibly solve the problem. We are currently working at these issues. We also study placement and routing schemes which can improve the performance and control the noise problem of the designs.

References

[1] P. Ashar, S. Devadas and A.R.Newton: Optimum and Heuristic Algorithm for an Approach to Finite State Machines Decomposition, IEEE TCAD, March 1991.

[2] L. Benini, E. Macii, M. Poncino, and G. De Micheli: Telescopic Units: A New Paradigm for Performance Optimization of VLSI Designs, IEEE TCAD vol. 17 no. 3, March 1998.

[3] V. Bertacco, et al.: Decision Diagrams and Pass Transistor Logic Synthesis, IWLS'97, Lake Tahoe, May 1997.

[4] P. Buch, A. Narayan, A.R. Newton, A. Sangiovanni-Vincentelli: Logic Synthesis for Large Pass Transistor Circuits, ICCAD'97, San Jose, November 1997.

[5] C. Chao and H.H. Loomis: High Rate Realization of Finite-State Machines, IEEE Trans. on Comp. vol. C-24, July 1975.

[6] G. De Micheli: Synchronous Logic Synthesis: Algorithms for cycle time minimization, IEEE TCAD, Jan 1991.

[7] G. Fettweis and H. Meyr: Parallel Viterbi decoding by breaking the compare-select feedback bottleneck, in Proc. IEEE Int. Conf. Communications, June 1989.

[8] A.D. Friedman: Feedback in Synchronous sequential Switching Circuits, IEEE Trans. on Comp. vol EC-15, June 1966.

[9] J. Hartmanis, R. E. Sterns: Algebraic Structure Theory of Sequential Machines, Prentice Hall, Englewood Cliffs, 1966.

[10] A. Hertwig, H.-J. Wunderlich: Fast Controllers for Data Dominated Applications, ED&TC 97, Paris, March 1997.

[11] H.-D. Lin, D.G. Messerschmitt: Finite state machine has unlimited concurrency. IEEE Transactions on Circuits and Systems, vol.38, (no.5), May 1991.

[12] A. Mukherjee, R. Sudhakar, M. Marek-Sadowska, S.I. Long: Wave Steering in YADDs: A Novel Non-iterative Synthesis and Layout Technique, DAC'99, New Orleans, June 1999.

[13] A. Mukherjee, M. Marek-Sadowska, S.I. Long: Wave Pipelining YADDs, CICC'99, San Diego, 1999.

[14] M. Shamanna, K. Cameron, S.R. Whitaker: Multiple-input, Multiple-output Pass Transistor Logic, Int'l J. Electronics vol. 79 n. 1, July 1995.

[15] K. Taki: A Survey for Pass-Transistor Logic Technologies, ASP-DAC'98, Yokohama, February1998.