DIRECT-MAPPED ASYNCHRONOUS FINITE-STATE MACHINES IN CMOS TECHNOLOGY

*Christos P. Sotiriou**

Institute for Computer Science (ICS),
Foundation for Research and Technology - Hellas (FORTH),
Science and Technology Park of Crete, P.O. Box 1385,
Heraklion, Crete, GR 711 10 Greece,
sotiriou@ics.forth.gr

ABSTRACT

Conventional asynchronous control circuit design is complex because potential designs must be analysed for the existence of hazards and state variable races, as these may cause incorrect circuit operation. We present the innovative *CMOS direct-mapped* asynchronous circuit design approach, which, by defining a one-to-one mapping between state diagram and CMOS circuit implementation, removes the need for race and hazard circuit analysis. It allows for regular, fast, multiple-input-change, non-fundamental mode asynchronous control circuits to be realised in CMOS technology.

I INTRODUCTION

The advances in VLSI technology pose significant challenges to the design of synchronous circuits. Sustaining high-performance, clock synchronisation, power consumption and noise emissions are problems which increase in complexity with the ever-increasing clock frequencies [1][2].

Asynchronous circuits are a potential solution to these problems as they are modular, do not require clock synchronisation, only consume power when producing useful work and have low noise emissions. The difficulty with asynchronous circuits and the fundamental reason why historically synchronous circuits dominated is their design complexity.

Typically, in order to ensure correct operation, when an asynchronous control circuit such as an Asynchronous Finite State Machine (AFSM) is designed, it must be analysed for hazards and state variable races under a delay model defined by the designer [3], *i.e.* delay-insensitive, speed-independent or quasi-delay insensitive.

After the analysis is performed, extra gates may need to be added to eliminate combinational hazards, critical paths must be identified to eliminate the possibility of sequential hazards occurring, and new state variables may be added to eliminate races. In addition, the behaviour of the environment may also need to be controlled, *i.e.* its speed and the number of inputs that may change simultaneously.

Most asynchronous circuits are constrained to single-input-change and fundamental mode operation. The former implies that only one input is allowed to change before the circuit changes state whereas the latter implies that the environment response time is shorter than the speed of the circuit.

This paper presents the *CMOS direct-mapped* approach, a novel, generic implementation approach that we developed for realising AFSMs in CMOS technology. This approach simplifies asynchronous control circuit design, as it removes the need for race and hazard analysis and allows for multiple-input-changes and non-fundamental mode operation.

II DIRECT-MAPPED AFSMS IN CMOS TECHNOLOGY

One special type of state variable encoding is the "one-hot" encoding [4], which assigns a state variable signal for each state of the state machine. This technique eliminates general races between state variables as it does not encode states. It also simplifies the circuit implementation as the logic that generates the state signals assumes a regular form.

Based on the one-hot encoding method, Hollaar proposed a direct implementation of one-hot asynchronous FSMs based on set-reset flip-flops [5]. Our innovation is a mapping directly into CMOS transistor technology. We demonstrate how direct-mapped AFSMs can be implemented using complex, dynamic CMOS gates, therefore yielding smaller, simpler and faster circuits. Our approach has been used to successfully design numerous control circuits for an asynchronous processor design [6].

* Previously with the University of Edinburgh, Scotland, where a large portion of this work was performed with the support of an EPSRC Grant.

Direct-mapped AFSMs can be implemented in CMOS technology using complex, dynamic CMOS gates as shown in Fig. 1. Each such gate corresponds to a state in the AFSM, hence the term state gates.

A state gate consists of a single or multiple p-type pull-up networks, a single or multiple p-type resolving transistors, a single or multiple n-type pull-down networks, a single or multiple n-type resolving transistors and a set of back to back inverters, acting as a storage element, at the output of the gate[1].

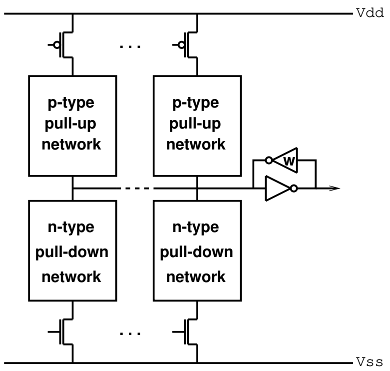

Figure 1: Asynchronous Direct-Mapped CMOS State Gate

The FSM states are the outputs of a group of such gates. When the output of a gate is high or active then the machine is in that state. Normally, only one state is active at any time, although it is possible to have parallel FSM paths, where multiple states are active simultaneously.

III CMOS STATE GATE DESIGN

In a state gate, the n-type pull-down networks detect the conditions appropriate to enter the state whereas the pull-up networks detect the conditions to leave it. The pull-down and pull-up networks are activated by the resolving transistors. The n-type resolving transistors are fed with the outputs of the previous states, enabling the n-type pull-down networks to discharge the state gate and so enter the corresponding state. The p-type resolving transistors are fed with the outputs of the following states enabling the p-type pull-up networks to leave the state. The number of n-type pull-down networks depends on the number of states from which a state can be entered and the number of p-type pull-ups on the number of states that are followed from a state. Hence, the logic for entering and leaving a state scales easily.

In the simplest case, the n-type pull-down network will correspond to a single n-type transistor fed with the signal that triggers the transition into the state. Hence, a total of two n-types will be the minimum. An n-type pull-down network may be more complex depending on the conditions for entering the state. Correspondingly, for p-types, in the simplest case there is a single p-type pull-up network, which in its simplest form is a short circuit, *i.e.* simply connects the input of the inverter to the p-type resolving transistor. The p-type resolving transistor will be connected to the inverted output of the following state.

In the case where one or more of the previous states of one state are also its following states, *i.e.* there are one or more scale-of-two loops in the state diagram, it is necessary to introduce the pull-up network to ensure that the n and p-type parts of the gate are never simultaneously ON. The structure of the p-type network in that case should be the same as the n-type network of the following state, only with inverted inputs.

The relative sizing of the p and n-type networks is important to eliminate the one-hot critical race. It must be ensured that the current, previous state pair changes are in the order 10→11→01, *i.e.* the next state must be entered before the current one is left. In this realisation, the next state is entered by its n-types, whereas the current state is left by its p-types. For correct operation, the n-type resolving transistor of the following state must be ON long enough for the following state to be entered. This n-type resolving transistor will be turned from ON to OFF by the p-types of the previous state, as the signal of the next state's variable begins to rise. For the next state to be properly entered, it must stay ON long enough for the state gate to switch. It therefore follows that the p-type pull-up must be slower than the n-type pull-down. This can be ensured by the transistor sizing by making the p-types of smaller $\frac{W}{L}$ ratio compared to the n-types. In practice however, as p-types are inherently slower that n-types (about four times for the processes reviewed), there is no need for the pull-up p-types to have different sizes from the n-types. For the case of more than two states, with their transition requirements simultaneously fulfilled, correct operation will occur if the delays of state gates and their interconnections are relatively uniform.

IV MAPPING A STATE GRAPH TO A CMOS CIRCUIT

Due to the one-hot coding, a direct-mapped AFSM is modular, *i.e.* it can be constructed as a connected set of AFSM segments. Figure 2 shows how various parts of an AFSM can be translated to a direct-mapped implementation.

Figure 2(a) shows how a linear portion of a state graph can be implemented. The first state is left when the next is entered. In Figure 2(a), when state `s3` is entered, state `s2` is left. This is achieved by the p-type transis-

[1]fed-back inverter marked **w** is of smaller width to the forward inverter.

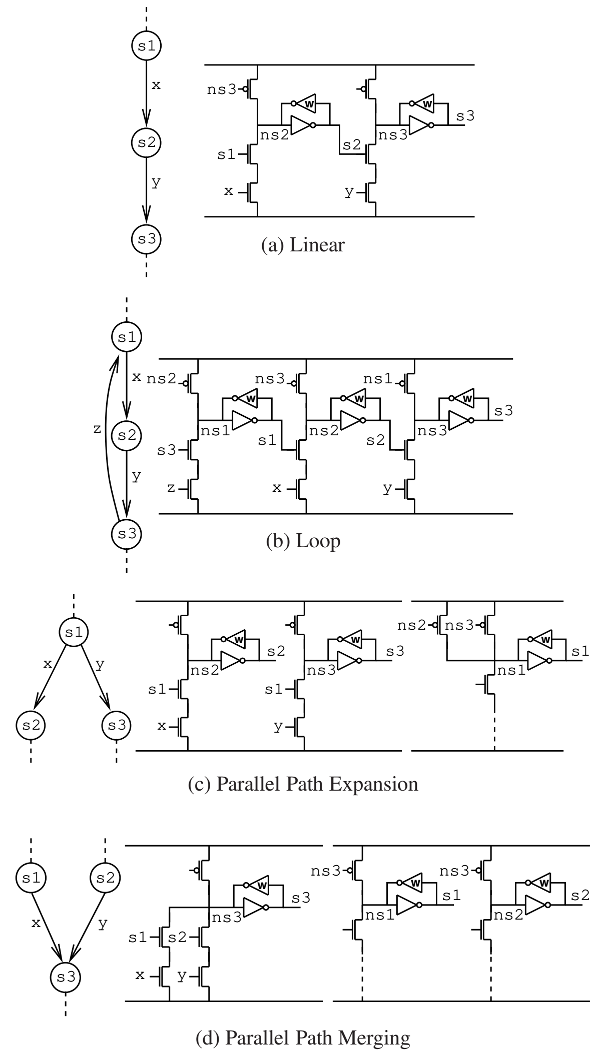

Figure 2: Mapping portions of an FSM state graph to a CMOS circuit implementation

tor of `s2`'s complex gate. It is possible for the two inputs to arrive simultaneously without the circuit malfunctioning, hence allowing for Multiple-Input-Change, non-fundamental mode operation.

Figure 2(b) shows how a loop in the state graph can be implemented. Input `z` will put the machine back in state `s1`. If all three inputs that form the loop are high simultaneously, then the machine will oscillate between the three states. As this behaviour is usually unacceptable for the interface of the FSM it should be guaranteed by the environment that at least two inputs are mutually exclusive.

Figure 2(c) shows how a parallel path in a state graph is implemented, *i.e.* the case where a state has multiple possible destinations. As states `s2` and `s3` in this example can both be entered from state `s1`, the output from state `s1` is fed to both of these states. In the case of parallel paths in the state graph, it must be guaranteed by the environment that the inputs that lead the FSM into separate paths must either be mutually exclusive, or that their value has settled before reaching the state where the machine forks. The latter is relevant in the case where one input is the inverse of the other to avoid sequential hazards..

Figure 2(d) shows how two parallel paths in a state graph may merge back into one. As state `s3` is where the two paths merge and hence states `s1` and `s2` are both state `s3`'s predecessors, it requires two pull-down networks. For the same reason, the p-types of states `s1` and `s2` are both fed by the the inverse of state `s3`.

Scale-of-two Loops

As was mentioned, in order to implement a scale-of-two loop correctly the p-type network must be made more complex than a single pull-up transistor.

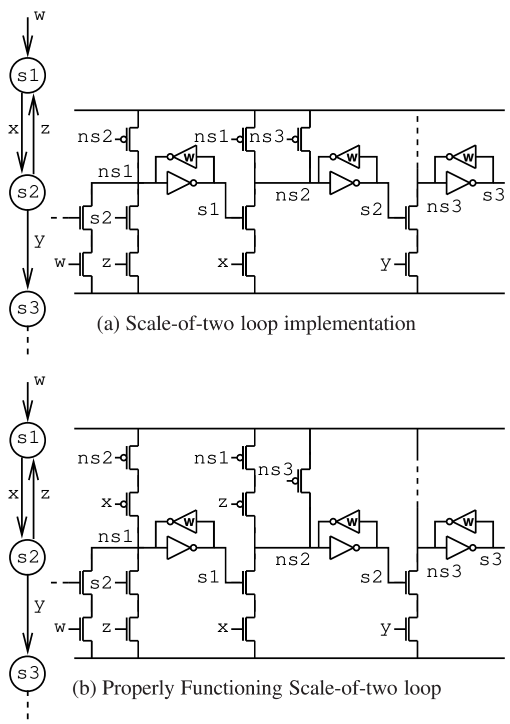

Figure 3: Implementing Scale-of-two loops

Figure 3(a) shows the implementation of a scale-of-two loop with single p-type transistors used to leave the AFSM states. This circuit will not function properly, as the n and p-types of state `s2` will be ON simultaneously as the FSM is entering state `s2` from `s1`. This happens because state `s2` is both `s1`'s successor and predecessor.

To implement scale-of-two loops correctly the departure from the states that belong to the scale-of-two loop (states `s1` and `s2` in this example) must be made sensitive to the inputs that cause the transition. This is shown in Figure 3(b), where series p-types are added to achieve this.

In the general case, where two states *m* and *n* form a state-of-two loop and n is m's successor, the p-type network of the state *m* must be identical to the n-type pull-down of state *n*. If more than one scale-of-two loop is present between state *m* and its successors, then *m* will have multiple complex p-type networks.

V COMPARISON WITH OTHER APPROACHES

In this section the performance and size of example circuits implemented using other design approaches are contrasted with their direct-mapped AFSM counterparts. The example circuits considered are semi- and fully-decoupled four-phase latch controllers [7] and the Sbuf-send-ctl finite-state machine, which was one of the control circuits of the Post Office chip [8]. Latch control circuits are ubiquitous in pipelined designs, so their performance is critical. The Sbuf-send-ctl control circuit is an example from a fabricated chip design.

All circuits have been simulated at the transistor level in HSPICE for 0.8μm, 0.35μm and 0.18μm technologies. The transistor models were obtained from [9].

A Latch Control Circuits

A latch controller is the control part of an asynchronous handshaking pipeline. A latch controller is said to be fully-decoupled when its input and output handshakes are independent of each other. If, on the other hand, the progress of the input handshake depends on the progress of the output handshake and vice versa, then a latch controller is said to be semi-decoupled.

Table 1 contrasts the performance and circuit size (in numbers of transistors) of the semi and fully-decoupled latch controllers described in [7] and implemented using the Signal Transition Graph (STG) circuit design approach [10] and of those implemented using direct-mapped AFSMs. Performance is contrasted by measuring the delays of the handshake signals of unloaded latch-controllers, *i.e.* without a buffer and a data register.

The columns labeled STG illustrate the delays of the circuit realised using the Signal Transition Graph approach, whereas the columns labeled DM illustrate the delays of the circuit implemented using the Direct-Mapped Approach.

The Reqin$\uparrow$ $\rightarrow$ Ackout$\downarrow$ delay, *i.e* the time it takes for the output handshake to complete from the initiation of the input handshake is the cycle delay of the latch. Table 1 shows that for the semi-decoupled latch controller circuit, the direct-mapped approach is close to 10% faster on average than the STG implementation and incurs a cost of 6 transistors. On the other hand, for the fully-decoupled latch controller the direct-mapped approach is significantly faster at slightly over 50% and requires roughly the same number of transistors.

Semi-decoupled Latch Controller

0.8μm	**STG**	**DM**
Reqin$\uparrow$ $\rightarrow$ Ackin$\uparrow$	0.39ns	0.3ns
Reqin$\uparrow$ $\rightarrow$ Reqout$\uparrow$	0.76ns	0.99ns
Reqin$\uparrow$ $\rightarrow$ Ackin$\downarrow$	1.59ns	1.15ns
Reqin$\uparrow$ $\rightarrow$ Ackout$\downarrow$	2.1ns	1.9ns
0.35μm	**STG**	**DM**
Reqin$\uparrow$ $\rightarrow$ Ackin$\uparrow$	0.13ns	0.10ns
Reqin$\uparrow$ $\rightarrow$ Reqout$\uparrow$	0.25ns	0.37ns
Reqin$\uparrow$ $\rightarrow$ Ackin$\downarrow$	0.56ns	0.43ns
Reqin$\uparrow$ $\rightarrow$ Ackout$\downarrow$	0.8ns	0.7ns
0.18μm	**STG**	**DM**
Reqin$\uparrow$ $\rightarrow$ Ackin$\uparrow$	55ps	40ps
Reqin$\uparrow$ $\rightarrow$ Reqout$\uparrow$	101ps	148ps
Reqin$\uparrow$ $\rightarrow$ Ackin$\downarrow$	220ps	170ps
Reqin$\uparrow$ $\rightarrow$ Ackout$\downarrow$	310ps	290ps
size (transistors)	20	26

Fully-decoupled Latch Controller

0.8μm	**STG**	**DM**
Reqin$\uparrow$ $\rightarrow$ Ackin$\uparrow$	1.09ns	0.33ns
Reqin$\uparrow$ $\rightarrow$ Reqout$\uparrow$	1.07ns	0.37ns
Reqin$\uparrow$ $\rightarrow$ Ackin$\downarrow$	2.28ns	1.26ns
Reqin$\uparrow$ $\rightarrow$ Ackout$\downarrow$	3ns	1.4ns
0.35μm	**STG**	**DM**
Reqin$\uparrow$ $\rightarrow$ Ackin$\uparrow$	0.38ns	0.11ns
Reqin$\uparrow$ $\rightarrow$ Reqout$\uparrow$	0.36ns	0.12ns
Reqin$\uparrow$ $\rightarrow$ Ackin$\downarrow$	0.74ns	0.43ns
Reqin$\uparrow$ $\rightarrow$ Ackout$\downarrow$	1.1ns	0.5ns
0.18μm	**STG**	**DM**
Reqin$\uparrow$ $\rightarrow$ Ackin$\uparrow$	150ps	42ps
Reqin$\uparrow$ $\rightarrow$ Reqout$\uparrow$	140ps	48ps
Reqin$\uparrow$ $\rightarrow$ Ackin$\downarrow$	320ps	170ps
Reqin$\uparrow$ $\rightarrow$ Ackout$\downarrow$	440ps	200ps
size (transistors)	42	41

Table 1: Latch Control circuit Comparison

B Comparison with a Burst-mode FSM example

Table 2 contrasts the performance and circuit size of the Sbuf-send-ctl FSM implemented using the burst-mode AFSM implementation approach and of the same circuit implemented using direct-mapped AFSMs. The circuit design of the burst-mode AFSM has been generated and optimised using the MEAT AFSM design tool. In this case performance is contrasted by measuring the time it takes the Sbuf-send-ctl circuit to perform a single data transfer and the cycle time, *i.e.* the time per item for multiple data transfers.

The columns labeled BM-FSM illustrate the performance of the circuit realised using the burst-mode FSM approach, whereas the columns labeled DM illustrate the performance of the circuit implemented using the Direct-Mapped Approach.

Sbuf-send-ctl AFSM

0.8μm	**BM-FSM**	**DM**
single transfer time	4ns	3.3ns
cycle time	3.8ns	3.5ns
0.35μm	**BM-FSM**	**DM**
single transfer time	1.3ns	1.1ns
cycle time	1.3ns	1.2ns
0.18μm	**BM-FSM**	**DM**
single transfer time	510ps	450ps
cycle time	540ps	490ps
size (transistors)	44	48

Table 2: Sbuf-send-ctl AFSM Comparison

Table 2 shows that for the Sbuf-send-ctl circuit the direct-mapped circuit is again faster, close to 10% on average. The circuit size is roughly the same, the direct-mapped circuit is larger by 4 transistors.

VI CONCLUSIONS

This paper presented the direct-mapped approach to AFSM design using CMOS technology. The direct-mapped approach produces regular, fast, asynchronous control circuits without the need to analyse the circuit specification to derive a state variable assignment. It allows for multiple-input-change, non-fundamental mode asynchronous operation.

The results presented for these three examples show that although direct-mapped AFSMs require more state variables, this does not necessarily imply a larger circuit size compared to other approaches. The results obtained also show that for the three examples studied the performance of the direct-mapped AFSM circuit is better. For two out of the three examples, *i.e.* the semi-decoupled latch controller and the Sbuf-send-ctl circuit the performance benefit is on average less than 10%, whereas for the fully-decoupled latch controller the average performance difference is quite significant at slightly over 50%. Hence, it can be concluded that, at least for these examples, despite its simplicity, the direct-mapped approach can potentially increase performance.

Due to its simplicity and regular nature it is possible to easily automate the direct-mapped approach so that transistor netlists can be generated directly from a circuit specification. It is also possible to incorporate Built-In-Self-Test additions.

The CMOS direct-mapped approach has been used to successfully design numerous control circuits for an asynchronous processor design [6].

VII ACKNOWLEDGEMENTS

The author would like to thank Roland N. Ibbett for his help and support during this research, Manolis G. H. Katevenis for his helpful input and EPSRC for making this research possible.

REFERENCES

[1] D. Matzke, "Will Physical Scalability Sabotage Performance Gains ?," *IEEE Computer*, Sept. 1997.

[2] M. J. Flynn, P. Hung, and K. W. Rudd, "Deep-Submicron Microprocessor Design Issues," *IEEE Micro*, vol. 19, No. 4, pp. 11–22, 1999.

[3] A. J. Martin, "The Limitations to Delay-Insensitivity in Asynchronous Circuits," in *Advanced Research in VLSI*, MIT Press, 1990.

[4] S. H. Unger, *Asynchronous Sequential Switching Circuits*. Department of Electrical Engineering, Columbia University: Wiley Interscience, 1969.

[5] L. A. Hollaar, "Direct Implementation of Asynchronous Control Units," *IEEE Transactions on Computers, Vol. C-31, No. 12*, December 1982.

[6] C. P. Sotiriou, *Design of an Asynchronous Processor*. PhD thesis, Institute for Computing Systems Architecture, Division of Informatics, University of Edinburgh, 2001.

[7] S. B. Furber and P. Day, "Four-Phase Micropipeline Latch Control Circuits," *IEEE Transactions on VLSI Systems*, vol. 4, pp. 247–253, June 1996.

[8] A. Davis, B. Coates, and K. Stevens, "Automatic Synthesis of Fast Compact Asynchronous Circuits," in *Asynchronous Design Methodologies* (S. Furber and M. Edwards, eds.), vol. A-28 of *IFIP Transactions*, Elsevier Science Publishers, 1993.

[9] S. Palacharla, N. P. Jouppi, and J. E. Smith, "Quantifying the Complexity of Superscalar Processors," technical report TR-96-1328, University of Wisconsin-Madison, Nov. 1996.

[10] T. A. Chu, "Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications," Tech. Rep. MIT/LCS/TR-393, Massachusetts Institute of Technology, June 1987.