High Speed CMOS VLSI Design

Lecture 14: Asynchronous Logic

1.0 Introduction

Asynchronous circuit design has a certain mystique and allure. Since asynchronous design is not widely understood, it is subject to many rumored advantages, some of which are true and some false. Some claimed benefits of asynchronous circuits relative to conventional circuits are:

- improved speed
- lower power
- modularity
- elimination of clock distribution

Discussing asynchronous circuits is difficult because the topic includes all timing schemes; indeed synchronous circuit design can be viewed as a specific case of asynchronous circuits with periodic, globally distributed control signals.

Many styles of asynchronous circuits are mathematically fascinating, but practically useless. In particular, “delay insensitive” circuits which make no assumptions whatsoever about the relative delays of elements are generally useless because they involve tremendous overhead determining when logic has completed. We will avoid these types of circuits and focus on circuits which make a limited number of reasonable timing assumptions. In particular, we will look at three asynchronous design styles: static register-based micropipelines, simple asynchronous domino logic, and zero-overhead self-timed domino circuits.

Since speed is a key concern, we will compare the speed of various schemes. Measuring speed is trickier than it might seem because speed could refer to latency, to throughput, or to something else. Moreover, throughput and latency can be traded off, so considering one without the other is misleading. For example, a static flop-based machine could achieve very high throughput, i.e. clock frequency, by placing only one gate between each flop. However, the latency would be terrible because the flops are in the critical path. Overall processor performance, in SPECint, is sensitive to both latency and throughput because excessive latencies slow dependent operations. In the one gate per flop example, an addition would take many cycles. A second addition using the result of the first addition would stall until the first completes, reducing overall performance even though the theoretical throughput of independent instructions could have been high. Therefore, we will try to

compare machines operating at a point where they achieve as high throughput as possible without significantly increasing latency.

We will take synchronous designs as a baseline. A static flop-based system has a cycle time:

$$T = t_{logic\text{-}static} + t_{clk\text{->}Q} + t_{setup} + t_{skew} \quad \text{(EQ 1)}$$

A skew-tolerant domino system has a cycle time

$$T = t_{logic\text{-}domino} \quad \text{(EQ 2)}$$

Both baselines involve t_{logic}, either static or domino. The flop-based system adds three sources of overhead, while the skew-tolerant domino system is better because it has no overhead.

In the next sections, we will look at how several asynchronous circuit techniques work and at what overhead appears in the cycle time.

2.0 Asynchronous Design Principles

Asynchronous designs replace clocks, which nominally reach all parts of a chip simultaneously, with control signals which only enforce sequence relationships between communicating blocks which they control.

Key concepts of asynchronous design include events and Muller C-elements which combine events. They are described in Sutherland's micropipeline paper.

3.0 Micropipelines

Sutherland describes micropipelines in his classic paper, so the discussion will not be repeated. There are numerous ways that micropipelined control circuits can be used to sequence logic. The choices are analogous to using clocks to control flip-flops, two-phase latches, and pulsed latches. The technique presented in Figure 17 of the micropipeline paper corresponds to synchronous flop-based design.

The control path uses the delay line plus the delay of two C-elements and some wiring to try to match the delay of data through the datapath logic and registers. The registers are very similar to master-slave flip-flops, but are turned on and off by events on "Capture" and "Pass" control lines rather than by rising edges of a clock.

The latency of each stage may be different, but throughput is set by the slowest stage. This stage ideally has a delay of $t_{logic\text{-}static} + t_{clk\text{->}Q} + t_{setup}$, i.e. the delay of the logic plus the delay of the register. To achieve this delay, the control circuit delays would have to be perfectly tuned to match the logic delay. If the control ran too fast, it would cause registers to capture incorrect data. Tuning the control to the data delays is a classic delay-matching

problem and requires extra margin on the control delays to guarantee they are always long enough. This margin may be 30 - 50% of the nominal delay, depending on how well the matching elements track data delays and how well the designer wants to sleep at night. Hence, actual cycle time is:

$$T = t_{logic\text{-}static} + t_{clk\text{->}Q} + t_{setup} + t_{margin} \quad \text{(EQ 3)}$$

4.0 Asynchronous Domino Logic

Dual-rail domino logic is very well-suited for building asynchronous circuits because it solves the problem of determining when a block of logic is done. Single-rail domino and static circuits often require large margin on a delay element which must indicate when the logic is done. Dual-rail domino automatically signals when the logic is done because outputs switch from both rails being 0 to one rail being 1.

A block of dual-rail domino gates consists of one or more domino gates using the same control signal to trigger precharge and evaluation. Blocks correspond to phases of logic in synchronous skew-tolerant domino. A completion detector circuit easily can be built for dual-rail domino blocks. When the block has only one output, completion is signaled by the OR of the two rails carrying the output. When a block has multiple outputs, it is complete when the latest output completes. If outputs complete at the same time or one is more critical than the others, only one output must be examined. If any of several outputs might be most critical in a data-dependent fashion, completion must be detected for each output, then completion for the block is indicated as the AND of completion of each block.

Given signals indicating the completion of each block, we can write rules determining when a block is allowed to precharge and evaluate:

1. a block may not precharge until the next block has evaluated and the previous block precharged
2. a block may not evaluate until the next block has precharged and the previous block has evaluated

A block may not precharge until its results have been consumed. This consumption is indicated by the next block signaling completion. Once the next block has used the inputs, the current block no longer must hold them. This is similar to the concept that the last gate in one phase of skew-tolerant domino may precharge when the first gate in the next phase has evaluated and consumed the result. No latches are needed between blocks or phases to hold data as long as the consumption before precharge rule is enforced.

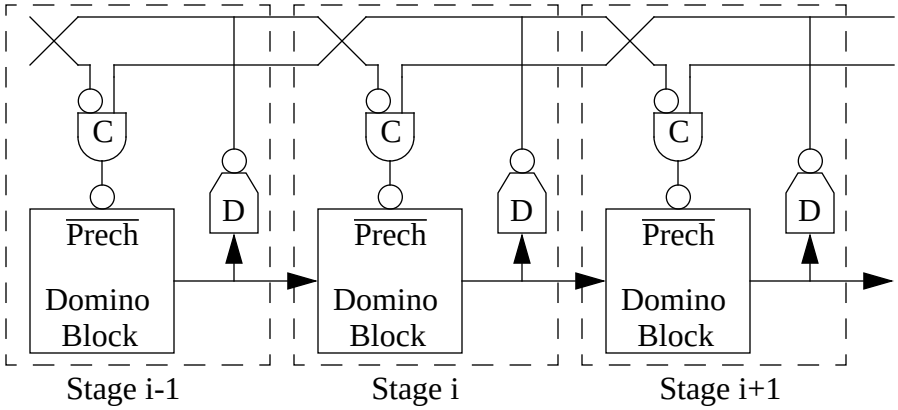
The block also should not precharge until its predecessor has precharged. This prevents racethrough which could occur if the predecessor remained in evaluation while the current block precharges, reenters evaluation, and incorrectly evaluates using the stale data from the predecessor.

Similarly, a block must not evaluate until the next block has precharged to prevent racethrough. Finally, a block should not evaluate unless its inputs are valid. A conservative

way to enforce this is to keep the block in precharge until the previous block signals completion. In the next section we will relax this rule.

Notice that these rules are exactly the same as are used to control a micropipeline. When the successor and the predecessor are different, the block gets set to the state of the predecessor. Thus, a chain of C elements can be used, as they are in a micropipeline. The major difference is that instead of dead-reckoning delays with a delay line, a completion detection circuit can be used. The scheme is illustrated in Figure 1:

FIGURE 1. Asynchronous Domino Pipeline



There must be at least three blocks in any cycle so that predecessor and successor gates can be in different states. Thus, a cycle of logic can be constructed by breaking the entire logic into three blocks, with completion detection and C elements between the blocks, just as a cycle of synchronous logic could be constructed by breaking it into two blocks with transparent latches between blocks. As long as the cycle is reasonably long, it can run at a rate equal to just the latency of data propagating forward through the path.

Since stage i+1 will not begin evaluation until stage i signals completion, there is an extra delay of the completion detection and C element between the two stages. For a cycle with three blocks, there are three such extra delays, giving a minimum cycle time of:

$$T = t_{logic\text{-}dynamic} + 3t_{completion} + 3t_{c\text{-}element} \quad \text{(EQ 4)}$$

5.0 Zero-Overhead Self-timed Domino Circuits

The asynchronous circuits of the previous section are fairly fast, but make overly conservative assumptions. By relaxing some assumptions, the circuits can run faster.

The most severe restriction is that a block cannot evaluate until the previous block has signalled completion. Since blocks consist of dual-rail monotonic logic, it is actually legal for a block to begin evaluation before its inputs have arrived, then wait until one of the input rails actually rises. This is similar to the principle in skew-tolerant domino that clocks should overlap so that a gate is in evaluation by the time its inputs become valid, so there

is no “clock blocking” in which the critical gate waits for a clock before it can process valid inputs.

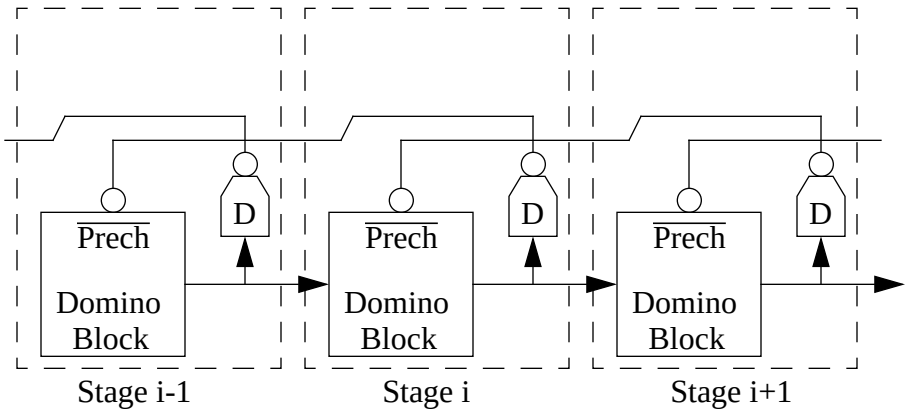
To remove this restriction, it would be convenient to be able to remove the C element which checks that the previous stage had evaluated. Unfortunately, this C element is also needed to prevent a stage from precharging before the previous stage precharges. However, if we can make a guarantee about the relative delays of precharging and evaluating stages, we can ensure the previous stage has precharged. Specifically, we can require that a all blocks precharge faster than they evaluate. This is reasonable if the block contains several gates because gates precharge in parallel, but evaluate in series.

When we make this assumption, consider the rule that a block may not precharge until the previous stage has precharged and the next stage has evaluated. Since the previous stage and next stage are activated by the same event (completion of the current stage), the previous stage is guaranteed to have precharged by the time we detect the next stage precharged.

Now we can rewrite the rules for control signals in a simpler fashion:

1. a block may not precharge until the next block has evaluated
2. a block may not evaluate until the next block has precharged

FIGURE 2. Zero-Overhead Self-timed Pipeline



Now each block is in evaluation by the time its inputs arrive, so the cycle time has no overhead:

$$T = t_{logic\text{-}dynamic} \quad \text{(EQ 5)}$$

Thus, zero-overhead self-timed domino circuits can achieve performance equal to that of skew-tolerant domino. There are certain restrictions on design to eliminate all the overhead, such as blocks with enough gates so evaluation takes longer than precharge, and at least three blocks per cycle. These are similar to restrictions on skew-tolerant domino of sufficient precharge time and of at least one gate per phase, with a minimum of three phases to avoid min-delay problems.

Self-timed circuits have certain other benefits. They can take advantage of environment and data-dependent propagation delays to have average case performance better than worst-case. They eliminate the need to route global clocks. They automatically idle and conserve power when no stimulus is applied to the inputs.

Unfortunately, self-timed circuits have other challenges. If they are used in conjunction with synchronous circuits, it is hard to take advantage of data-dependent delay because the overall delay still must be an integer number of cycles. Moreover, the results must be resynchronized with the synchronous clock, which can take time to minimize the probability of metastability. Therefore, data-dependent delays are only really useful when the overall delay is many cycles or the system is constructed entirely asynchronously.

Moreover, although global clocks are not needed, control signals must still be routed with data signals that cross the chip. The delay of the control and data lines may not track well, introducing “global control skew” which is very similar to “global clock skew.” Since there are large numbers of communicating blocks, there is also a large number of control signals which must be routed and which consume power switching.

The pictures we have considered consist of blocks taking one input and producing one output. Real circuits require fanin and fanout. When a two blocks both drive a common third block, the third block acknowledges both inputs. When one block drives a fanout of two blocks, the first block should not precharge until both receivers complete. This requires a C element to combine the completion signals. These examples are shown below. Fanout is a particular problem, because a signal fanning out to many receivers may need a slow many-input C-element which could impede the critical path. Also, sending the acknowledge signals globally can take a large amount of time and slow the path.

FIGURE 3. Circuit with fanin

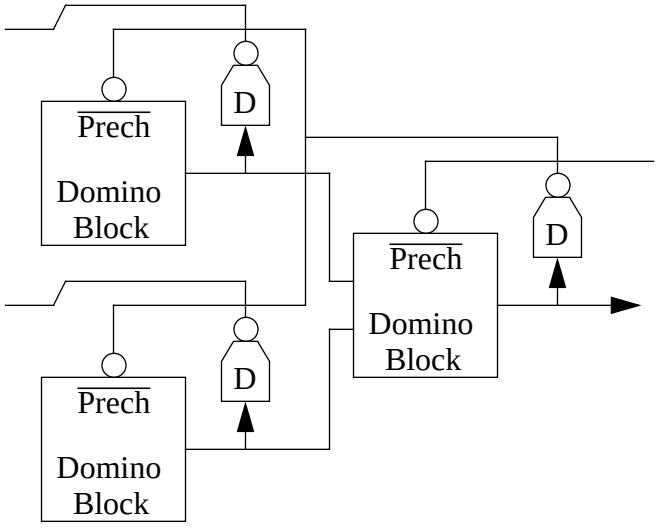
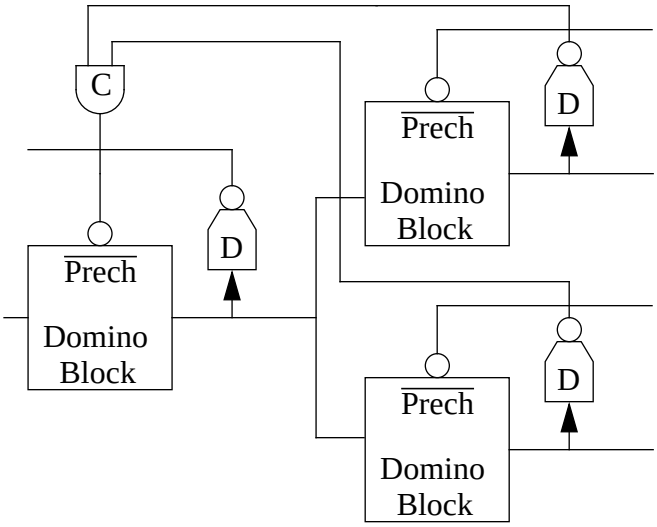


FIGURE 4. Circuit with fanout



Test is a large challenge for asynchronous circuits. Standard test methodologies involve stopping a clock and using scan signals to march other data through the latches. Since asynchronous circuits have no global clock, test must be entirely rethought if it can be done at all.

Finally, from a practical standpoint, using asynchronous circuits is difficult because few tools are available and few designers have experience in the area. These difficulties could be remedied over time if the performance benefits merit the expense.

6.0 Conclusions

Asynchronous circuit design is not yet widely understood and hence is subject to many misconceptions. This lecture has attempted to explain how a few styles of asynchronous circuits operate and to compare the potential performance of each. The results are summarized below:

TABLE 1. Latency of Clocking / Control Schemes

Scheme	Cycle Time
Synchronous Flip-flops	$t_{logic\text{-}static} + t_{clk\text{->}Q} + t_{setup} + t_{skew}$
Synchronous Skew-Tolerant Domino	$t_{logic\text{-}domino}$
Static Registered Micropipelines	$t_{logic\text{-}static} + t_{clk\text{->}Q} + t_{setup} + t_{margin}$
Asynchronous Domino	$t_{logic\text{-}dynamic} + 3t_{completion} + 3t_{c\text{-}element}$
Zero-Overhead Self-timed Domino	$t_{logic\text{-}domino}$

Micropipelines with static registers perform very much like synchronous pipelines with flip-flops. Clock skew is replaced with its asynchronous equivalent, control margin. Asyn-

chronous domino can have overhead, but when cleverly designed to overlap clocks, achieves zero overhead in much the same way as skew-tolerant domino.

Hence, arguments to use asynchronous logic for higher performance on general-purpose logic are mostly misleading. Nevertheless, there are special cases, mostly already in wide use, where asynchronous circuits are important. Self-timed RAMs and PLAs take advantage of the regularity of the structures to generate control signals exactly when needed by the logic. Long operations like with different best and worst-case delays can also benefit from signalling completion. Communication between systems with unrelated clocks inherently requires asynchronous techniques.

Many of the advantages claimed by asynchronous circuits are not a result of asynchronous control, but rather just good design practices. For example, zero-overhead self-timed domino logic was faster than slow synchronous pipelines constructed of static logic and flip-flops. The essential improvement, however, was not eliminating clocks, but rather using fast domino gates and guaranteeing that gates are in evaluation before the data arrives. Similarly, some asynchronous circuits are recommended for low power, but the essential advantage is that the activity factor is reduced by avoiding unnecessary switching. Synchronous systems can gain much of this advantage by gating clocks to unused blocks.

Synchronous circuit design faces challenges in the future, especially from clock distribution and skew. Instead of discarding all the advantages of clocks, however, it may be more effective to continue borrowing ideas from asynchronous design and integrating them into a clocked framework.

7.0 References

This lecture has taken a somewhat doubtful attitude toward asynchronous design. For a contrasting perspective, consider some of these references by authors with experience constructing practical asynchronous systems.

I. Sutherland, “Micropipelines,” *Communications of the ACM*, vol. 32, no. 6, pp. 720-738, June 1989.

C. Mead and L. Conway, *Introduction to VLSI Systems* Addison-Wesley: Reading, MA, 1980 (esp. Ch. 7 on System Timing by Chuck Seitz).

T. Williams, *Self-timed rings and their application to division*, Ph.D. dissertation, Stanford University, Stanford, CA, May 1991.

T. Williams, “Performance of Iterative Computations in Self-Timed Rings,” *J. VLSI Sig. Proc.*, no. 7, pp. 17-31, Feb. 1994.

T. Williams and M. Horowitz, “A Zero-Overhead Self-Timed 160-ns 54-b CMOS Divider,” *IEEE J. Solid-State Circuits*, vol. 26, no. 11, pp. 1651-1661, Nov. 1991.