Appeared in *Proceedings of Eighth International Symposium on Advanced Research in Asynchronous Circuits and Systems*, April 2002.

A Negative-Overhead, Self-Timed Pipeline

Brian D. Winters and Mark R. Greenstreet
Department of Computer Science
University of British Columbia
{*bwinters,mrg*}*@cs.ubc.ca*

Abstract

This paper presents a novel variation of wave pipelining that we call "surfing." In previous wave pipelined designs, timing uncertainty grows monotonically as events propagate through gates or other logic elements. We bound this dispersion by propagating a timing pulse along with the data values. Our logic elements have delays that are smaller in the presence of the pulse than in its absence. This produces a "surfing" effect: events are bound in close proximity to the timing pulse. We demonstrate this approach with the design of a 4×12 multiplier. Spice simulations from the extracted layout indicate that this design is robust in the presence of fabrication parameter variation and power supply noise. Because timing is maintained by accelerating the logic, our designs achieve lower latency than their purely combinational equivalents. Thus, the control overhead for these designs is indeed negative.

1 Introduction

This paper presents a novel variation of wave pipelining called "surfing." In surfing pipelines, a timing pulse is propagated along the pipeline, and logic elements are modified so as to have reduced propagation delays in the presence of this pulse. We show that when a few, simple conditions are satisfied, events in the data path will propagate in bounded temporal proximity to the timing pulse. This prevents timing uncertainties from accumulating in the data path, and we can implement wave pipelining on pipelines that are arbitrarily deep with a correspondingly arbitrary number of waves.

Williams and Horowitz introduced earlier the concept of "zero-overhead" pipelines [13]. If a pipeline has a total latency equal to the sum of the latencies of its stages, then no latency is introduced by control or latching, and the pipeline is said to have "zero-overhead." Our pipelines have lower latencies than the sum of the latencies of a purely combinational design. These low latencies are achieved because the delays of the logic elements decrease in the presence of the timing pulse. Thus, we say that our pipelines have *negative overhead*.

Prior to our work, Fairbanks and Sutherland observed that pipelines can operate with soft-latches where the latch does not go fully opaque between successive clock cycles [10]. Simply increasing the delay of the latch can provide proper pipeline operation in some cases. Our present work explains these observations in a more general framework and introduces an acceleration mechanism that can eliminate the need for latches entirely.

The main contributions of our paper are:

- We show how modulating the delay of logic gates can create "event attractors" (Section 3). These attractors propagate faster than the non-surfing delay of logic elements, resulting in negative overhead. Furthermore, these attractors ensure timing uncertainties remain unbounded, even in arbitrarily long pipelines. This removes the need for latches and their associated overhead.
- We describe a CMOS logic family that implements surfing (Section 4). These circuits are a simple variation of existing, self-resetting domino designs [3].
- We present a surfing multiplier (Section 5). We report extensive Spice simulation results based on a model extracted from a layout of the multiplier. Due to surfing, propagation delays of the XOR gates and multiplexors are only 11% greater than the delays of simple inverters, and about 4% faster than the corresponding, non-surfing, self-resetting domino implementations. Spice simulations indicate that the surfing pipeline is fast and robust in the presence of parameter variation and power supply noise.

2 Pipelining Methods

Figure 1 shows a traditional synchronous design. Let the period of the clock, Φ, be P, and let $\delta_{\min}$ and $\delta_{\max}$

be the minimum and maximum delay from the inputs to the outputs of the combinational logic. For simplicity, we ignore latch set-up and hold times, latch propagation delays, and clock skew, noting that the qualitative observations that we make continue to hold in more detailed models. Classical synchronous design is based on the observation that if $\delta_{\max} < P$, then the values present at the input of the latches will have settled to their proper values prior to each clock event. In other words, the minimum clock period is determined by the slowest path through the combinational logic. In general, reducing the clock period increases performance.

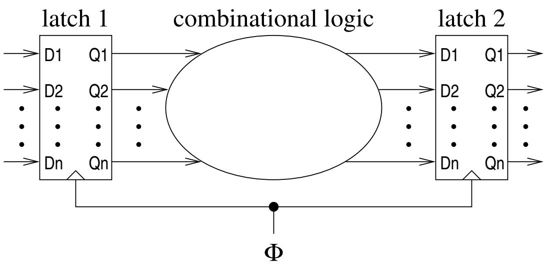

Figure 1. Synchronous Design

2.1 Wave Pipelining

With careful control of the delays in the combinational logic, wave pipelined designs [1] can achieve clock periods less than $\delta_{\max}$. For example, if $\delta_{\max} < 2\delta_{\min}$, then the circuit can operate at a clock frequency P that satisfies:

$$\delta_{\max}/2 \quad < \quad P \quad < \quad \delta_{\min} \tag{1}$$

provided that certain internal delay constraints specific for the particular logic blocks are satisfied (see [1, Section 2.3.2]). In this case, the combinational logic block operates with twice the throughput but the same latency as the classical synchronous design. Figure 2 illustrates this operation, showing three waves shortly after a clock event. At this point, the data that propagated through latch 1 at the most recent clock event is propagating through the combinational logic as wave C. Data that propagated through latch 1 one clock period earlier is also propagating through the combinational logic as wave B. The one clock-cycle head start of wave B ensures that it will arrive at latch 2 at the end of the current clock cycle without being overtaken by wave C. At each clock event, latch 2 acquires the data that propagated through latch 1 two clock periods earlier. Thus, wave A represents data from two clock cycles before wave C.

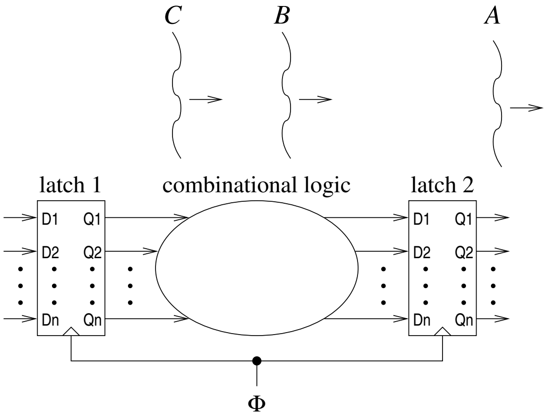

Figure 2. Wave Pipelining with Two Waves

More generally, if $(k-1)\delta_{\max} < k\delta_{\min}$ holds for some positive integer k, then the circuit can operate at a clock frequency satisfying

$$\delta_{\max}/k \quad < \quad P \quad < \quad \delta_{\min}/(k-1) \tag{2}$$

provided again that the necessary internal delay constraints are satisfied. This allows that circuit to operate with k times the throughput of classical, synchronous designs.

Timing uncertainties are the Achilles' heel of wave pipelined design. To minimize delay uncertainties, typical wave-pipelined designs arrange logic blocks into levels as shown in Figure 3.

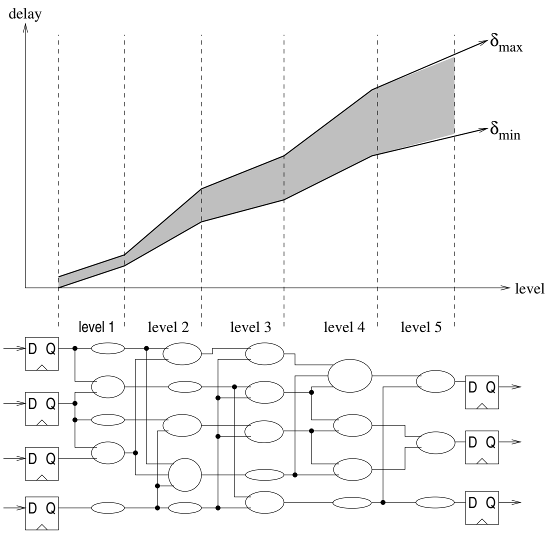

Figure 3. Timing Uncertainty

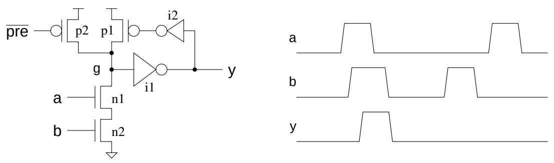

Figure 4. Self-Resetting Domino AND Gate

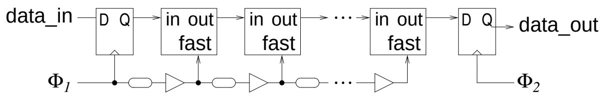

Figure 5. A Surfing Pipeline

The uncertainty at the output of a latch is determined by skew and jitter of the clock and variations of the propagation delay of the latch. This uncertainty grows monotonically as each level adds its uncertainty to the accumulated uncertainty of the previous levels (assuming all paths are sensitizable). This accumulation of uncertainty presents a fundamental limit to wave pipelining. In particular, Equation 2 implies:

$$P \quad > \quad \delta_{\max} - \delta_{\min} \tag{3}$$

Thus, uncertainty in the timing leads directly to a limit on the operating frequency. In practice, most wave pipelined designs have only supported two to four waves. To achieve this, most wave pipelined designs employ logic restructuring and extra delay padding to minimize delay variation. This gives these designs greater latency than their classical equivalents, and the throughput is improved by a factor much smaller than the degree of wave pipelining. The surfing technique introduced in section 3 avoids this accumulation of delay. We first describe self-resetting domino circuits which resemble wave-pipelined designs in their need for well matched delays. We use self-resetting domino as the starting point for our designs presented in section 4.

2.2 Self-Resetting Domino

Self-resetting domino circuits [3] are a variation of domino circuits [6] where the precharge control signal for each gate is derived from the gate's output. As an example, figure 4 shows a self-resetting domino, two-input AND gate. Transistors *p1* and *p2* are the precharge transistors. After precharge, node g is high. If the a and b inputs both go high, then node g is pulled low and output y goes high. If either a or b remains low, then the y output remains low as well. Asserted values in self-resetting domino are represented by pulses. After the output y goes high, inverter *i2* drives its output low, enabling transistor *p1* to precharge node g high which returns output y to a low value. Between input pulses, the precharge control signal, $\overline{\text{pre}}$ is low, and node g is held high by transistor *p2*. This maintains the level of g when the gate is operated at low frequencies and improves noise immunity.

Self-resetting domino circuits offer performance advantages because the only P-channel devices on the forward path are those for the output inverters of the gates. The switching networks that implement logic functions are constructed entirely of N-channel transistors with their higher carrier mobilities. The self-resetting operation allows the gate to precharge immediately after the completion of evaluation, minimizing the cycle time.

Input pulses to a multi-input self-resetting gate must have sufficient overlap to allow the N-channel network to fully discharge the precharged node (node g in Figure 4). Furthermore, input pulses must be short enough to avoid fights during the self-resetting precharge. These considerations place two-sided timing constraints on the operation of self-resetting domino circuits. Typically, blocks of self-resetting domino gates are arranged in levels, similar to those shown in Figure 3. This makes wave pipelining a natural technique for use in conjunction with self-timed domino designs [3]. As with other wave-pipelined designs, accumulated timing uncertainty limits the depth of logic in self-resetting domino designs. In particular, the accumulated uncertainty must be sufficiently less than the pulse width to ensure full triggering of the self-resetting gates.

3 Surfing

Consider again the synchronous circuit depicted in Figure 1. Due to timing uncertainties, the signals at the inputs of latch 2 may settle at different times. For proper operation, the latch must be triggered after the last data input has settled. Viewed from a slightly different perspective, latches bound timing uncertainty by slowing down events that propagate too fast. Recognizing this property of latches, Dooply and Yun [4] refer to latches as "roadblocks" when deriving timing constraints for self-resetting domino circuits.

Instead of *slowing down* the fast signals, we propose to *speed up* the slow ones. Thus, our pipelined circuits have lower latency than their unclocked combinational equivalents. This is the basis for our claim of negative overhead for our self-timed pipelines. This section describes how selective acceleration of slow paths provides a high-performance mechanism for bounding timing uncertainty.

Figure 5 depicts a simple surfing pipeline. Each logic

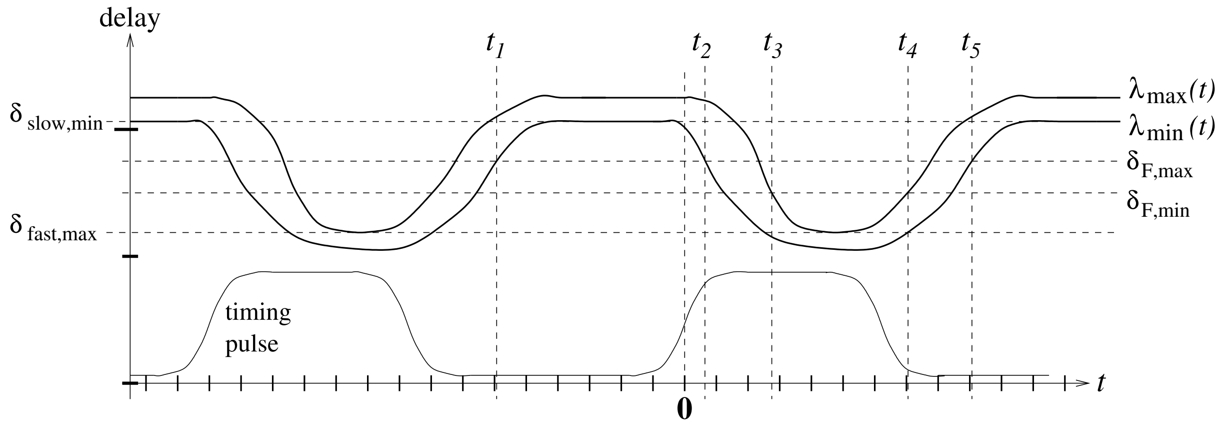

Figure 6. Timing for Surfing

block in the pipeline has a special input labeled fast. Asserting fast decreases the delay of the block. Let $\delta_{slow,min}$ be the minimum delay of a logic block when fast is not asserted, and let $\delta_{fast,max}$ be the maximum delay of a logic block when fast is asserted. The delay and buffer chain in Figure 5 generates the fast signal for each logic block. Let $\delta_{F,min}$ and $\delta_{F,max}$ denote the minimum and maximum delay between fast signals for consecutive stages of the pipeline. To ensure proper operation, we require:

$$\delta_{fast,max} < \delta_{F,min} < \delta_{F,max} < \delta_{slow,min} \qquad (4)$$

When the constraints of Equation 4 are satisfied, events in the chain of logic blocks are attracted to the leading edge of the pulses of the fast signals. To see this, consider what happens if the outputs of a logic block change before fast is asserted for that block. In this case, the propagation delay for the next block will be at least $\delta_{slow,min}$ which is greater than $\delta_{F,max}$. Therefore, the timing pulse will catch up (or partially catch up) with the logic events. Conversely, if the output of a logic block changes after fast is asserted, then the propagation delay for the next block will be at most $\delta_{fast,max}$ which is less than $\delta_{F,min}$. In this case, the data events propagate faster than the timing pulse and eventually catch up.

As a metaphor, we view the propagation of data events as a swimmer in the ocean, and propagation of the timing events as a wave. Unassisted, the swimmer cannot swim as fast as the wave. However, there is a region on the leading edge of the wave where the swimmer is accelerated by the wave to travel at the same rate as the wave. Accordingly, we refer to this mechanism as "surfing."

To examine surfing in more detail, we need to consider the continuous variation of the propagation delay of the logic block under the influence of the timing pulse. We will say that an input event to a logic block is an enabling event if it is the last input required to enable a transition on at least one output of the block. Let $\lambda_{min}(t)$ be the minimum delay from an enabling input event to the corresponding output event if the input event occurs t time units after the arrival of the timing pulse. Likewise, let $\lambda_{max}(t)$ be the maximum delay from an enabling input event to the corresponding output event if the input event occurs t time units after the arrival of the timing pulse. Figure 6 shows $\lambda_{min}(t)$ and $\lambda_{max}(t)$ for a prototypical surfing logic block. We have also drawn the timing pulse in this figure to illustrate the relationship between the timing pulse and the varying delay of the logic block.

Figure 6 illustrates the timing properties of a surfing pipeline in greater detail. The bottom trace depicts the timing pulse (i.e. the fast signal) at a particular stage of the pipeline. The upper pair of solid curves show the maximum and minimum delays of the logic block for inputs that change at the time indicated on the horizontal axis – when the fast signal is high, delays are decreased compared with the delays when fast is low. The horizontal dashed lines show the quantities that appear in Equation 4. The tick marks on the axes indicate that the plot is drawn with much greater time resolution for the vertical axis than the horizontal one.

Equation 4, used the quantities $\delta_{slow,min}$, and $\delta_{fast,max}$. These are related to $\lambda_{min}(t)$ and $\lambda_{max}(t)$ by the relations:

$$\begin{aligned} \delta_{slow,min} &= \max_t \lambda_{min}(t) \\ \delta_{fast,max} &= \min_t \lambda_{max}(t) \end{aligned} \qquad (5)$$

Now, define t_1, t_2, t_3, t_4, and t_5 as indicated below:

- t_1: The time at which $\lambda_{min}(t)$ crosses above $\delta_{F,max}$ in response to the falling edge of the previous timing pulse.

t_2: The time at which $\lambda_{\min}(t)$ crosses below $\delta_{F,\max}$ in response to the rising edge of the current timing pulse.

t_3: The time at which $\lambda_{\max}(t)$ crosses below $\delta_{F,\min}$ in response to the rising edge of the current timing pulse.

t_4: The time at which $\lambda_{\max}(t)$ crosses above $\delta_{F,\min}$ in response to the falling edge of the current timing pulse.

t_5: The time at which $\lambda_{\min}(t)$ crosses above $\delta_{F,\max}$ in response to the falling edge of the current timing pulse.

In Figure 6, dashed vertical lines depict these times.

The key properties of surfing are:

- If the enabling input events for one stage arrive in the interval $[t_2, t_3]$ at one stage, then all input events will be in interval $[t_2, t_3]$ at all subsequent stages.
- If the enabling input events for one stage arrive in the interval (t_1, t_4) at one stage, then the input events at the next stage will be in a smaller interval contained in (t_1, t_4). The sequence of such intervals for successive stages converges to $[t_2, t_3]$.

In other words, the interval (t_1, t_4) is the "capture interval" for surfing. The interval $[t_2, t_3]$ is the steady state event uncertainty; we call this the "surfing interval." We omit the proofs of these properties due to space limitations, noting that they are straightforward. Events that arrive in the interval $[t_4, t_5]$ could surf with the current timing pulse, or they could "fall-off" and slip to the next pulse. Events in this interval are timing violations that could give rise to metastable behaviors [2] and related malfunctions. In practice, the steady state interval and the violation interval are both much smaller than the capture interval – this gives rise to the robustness of surfing.

Note that surfing gates are *faster* when the timing pulse is asserted. With this negative overhead, performance is improved by implementing every gate on the critical timing paths as a surfing gate. By using surfing on every gate, timing uncertainty is minimized. Typically, such extreme pipelining is unacceptable for traditional, latched designs because of the latency overhead of the latches. In contrast with latched designs, surfing simultaneously lowers latency and bounds timing uncertainty.

4 Surfing Circuits

To achieve surfing, we designed gates where asserting the fast input causes the output of the gate to shift slightly in the direction of making a transition. For self-resetting domino logic, active transitions are always in the low-to-high direction. Thus, we shift the low output of a gate slightly upwards in response to assertion of the fast input. We call this shift *preswitching*.

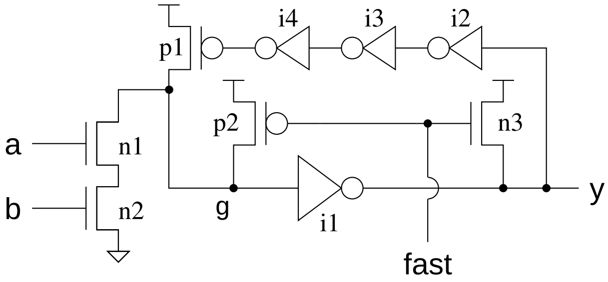

Figure 7. A Surfing AND Gate

4.1 Preswitching for Self-Resetting Domino

As an example of our approach, Figure 7 shows a self-resetting domino AND gate with a fast input implemented as described above. When the fast input is low, transistor p2 is conducting and functions as a keeper for the internal node g. To minimize the capacitance on node g, we implement p2 with a minimum width device. Accordingly, if inputs a and b are both high while fast is low, transistors n1 and n2 can overpower p2 and trigger an output pulse. In this situation, the hindering current sourced by p2 slightly delays the transition, an effect that increases the delay in the non-accelerated regime, therefore increasing the timing margins for surfing.

When fast is high, transistor p2 is turned off, and transistor n3 is conducting. If node g is high, then n3 pulls up against the N-channel pull-down of inverter i1. This raises the voltage of node y slightly above ground and decreases the delay for a subsequent rising transition of y if node g later goes low. Otherwise, if node g is low, then inverter i1 is already pulling node y high. If node y is in transition, then the extra current from n3 simply accelerates the transition. Thus, rising transitions of y are faster when the fast signal is high than the rising transitions of an otherwise equivalent, non-surfing gate.

In practice, we draw transistor n3 with the same shape factor as the N-channel pull-down of inverter i1. This design exploits the fact that N-channel devices make poor pull-ups. With equal sized devices, node y moves about 20-25% of the way to Vdd when node fast is high, and the delay of the gate decreases by about 30% compared with the delay when node fast is low. Because the fight pits an N-channel pull-up against an N-channel pull-down, our design also enjoys excellent device matching. Traditional, "five-corner" Spice simu-

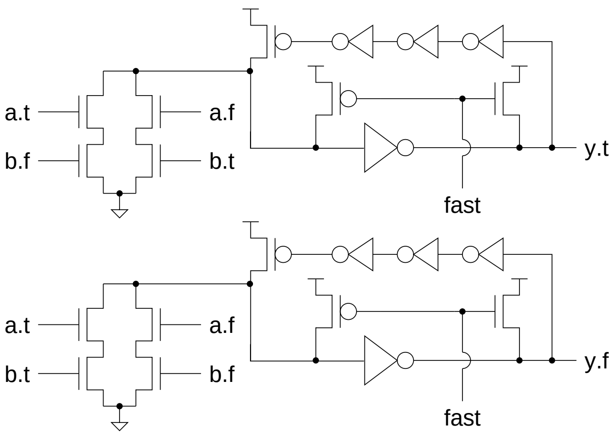

Figure 8. A Dual-Rail Surfing XOR Gate

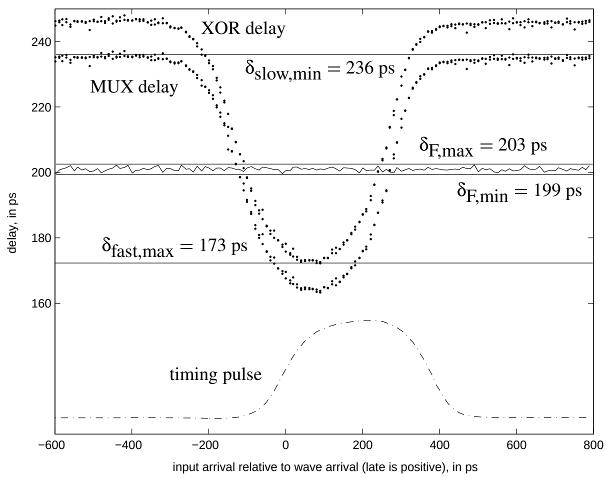

Figure 9. Measured Delays at Typical Process Point

lations show robust operation over the full range of device parameters for the 0.35μ process that we are using (see Section 5.4).

The sizing of transistor n3 presents some interesting trade-offs. Increasing the width of this transistor pulls node y higher while waiting for node g to fall and further decreases the gate delay. By making the dip in the timing curve deeper, widening transistor n3 *increases* the robustness of the design to timing variations. On the other hand, pulling node y higher moves y closer to the switching threshold of the next gate. Thus, widening transistor n3 *decreases* the voltage noise margin of the design. This concern is exacerbated by our use of dynamic logic. If the inputs to the next gate are pulled higher than the threshold voltage for N-channel device, then charge is drained from node g of that gate, even if that gate should not be enabled to switch. If this leakage persists long enough, node g will drop below the switching threshold of inverter i1, and the gate will produce a spurious output pulse.

Our simulations indicate that we obtain a fast and robust design when the width of transistor n3 is equal to that of the pull-down in inverter i1. For typical process parameters, node y is pulled slightly higher than the threshold voltage for N-channel devices. However, the leakage is quite small, and node g remains comfortably above the switching threshold of inverter i1 for the duration of the pulse on the fast input. We are currently exploring other circuit variations to better understand this trade-off between timing robustness and noise immunity.

4.2 Dual-Rail Gates

Because domino logic is non-inverting, we use a dual-rail encoding: each signal, x, is produced in *true* and *false* versions with a pulse on the x.t wire indicating a value of true and a pulse on the x.f wire indicating a value of false. In the simple multiplier presented in Section 5, the critical path consists entirely of XOR gates and multiplexors. Figure 8 shows our dual-rail surfing domino XOR gate. The multiplexor has exactly the same transistor topology with different signals connected to the inputs. By using the same circuit for both functions, we achieve close matching of the delays through the two gates.

Figure 9 shows the delays that we observed in hspice simulations for the XOR gate and for the multiplexor. The rising edge of the timing pulse occurs *after* the dip in gate delay because we measured delays based on input arrival times. The delay from data inputs to the gate output is greater than the delay from the fast signal to the output. Thus, the fast signal modulates the delay of inputs that arrived somewhat earlier.

For simplicity, we measured delays from the 50% point of rising edges. We realize that delays depend on the shape of the rising waveform as well as the time of the 50% point, and these wave shapes are significantly altered by preswitching. Also, we collected these datapoints using a small subset of circuits from the multiplier, rather than simulating the entire multiplier, to reduce the time necessary to construct Figure 9 from weeks to only hours. Thus, the curves in Figure 9 should be viewed as approximate; yet, we still found such plots to be very effective for debugging and optimizing our design.

The upper curve shows the delay of the XOR gate as

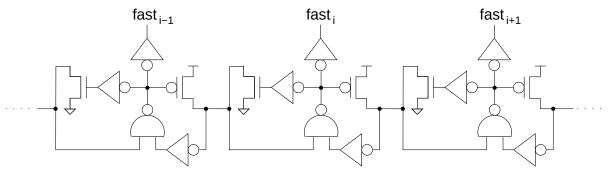

Figure 10. GasP Backwards Control

a function of the time of arrival of the last input for an exhaustive enumeration of data values, and a wide range of arrival times of the earlier input. The lower curve similarly shows the delay of the multiplexor. The horizontal line that cuts through the dip shows the measured delay of the timing chain for a large number of runs. Observe that the depth of the dip is nearly 30% of the delay without preswitching, while the vertical spacing between the minimum and maximum delay curves is much smaller. Thus, the surfing effect strongly dominates the timing variations of the gate and ensures proper operation of the pipeline.

4.3 GasP Backwards Control

In order for surfing to occur, our timing signals must propagate at about the same rate as or a little faster than our fastest domino element. Self-resetting domino logic is very fast. The forward latency is less than two inverter delays, and when preswitching is being used the forward latency is only slightly greater than one inverter delay. We avoided using a simple inverter chain because pulses can be lost. A very fast self-timed chain is required. The self-timed style we have found to be best suited to our purposes is GasP [11].

In the configuration given by Sutherland and Fairbanks [11], GasP has four inverter delays in the forward direction and two inverter delay for the backward latency. In the self-timed designs for which they created GasP, the extra forward latency matched the delay of their data paths with latches, and the smaller backward latency provided a small cycle time. Our designs have no latching overhead, and applying preswitching to every gate in the critical path improves both performance and timing margins. GasP pipelines closely resemble self-resetting domino designs, and we found the shorter, backward latency of GasP ideal for propagating our timing pulses. Figure 10 shows our "GasP backwards" control. (Note that the NAND gates are self-resetting.)

5 A Surfing Multiplier

To evaluate surfing domino logic, we want a deep pipeline with several different types of gates in order We

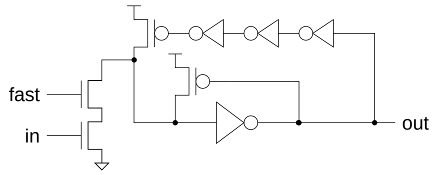

Figure 11. Pseudolatch Self-Resetting Domino Pseudolatch

have chosen to implement an integer multiplier for our tests. Our design is implemented in a 0.35μ, 3.3 volt CMOS process. Although this is far from bleeding edge technology, it is an economical process for prototyping. Even in this process, our chips should be fast enough to strain our test equipment.

5.1 High-Level Design Choices

We decided upon a radix-2, add-pass design. More aggressive multiplier designs exist (e.g. [7]), but since our goal was to create a nontrivial deep pipeline, rather than advance the state of the art in multiplier design, we chose to keep our design simple. Radix-2 gives us flexibility in testing a variety of pipeline depths, and add-pass ensures that our pipeline will be very deep.

All of the gates in our multiplier use self-resetting domino logic. There are no latches anywhere in the multiplier array. The multiplier is composed of cells which perform a single bit of multiplication and addition. Figure 12 shows one cell, where x and y are the numbers to be multiplied, s and c are sum and carry outputs of multiplier cells, *i* and *j* are the bits in x and y which are being multiplied in a cell, and *t* indicates the pipeline stage.

5.2 Timing

We used the theory of Logical Effort [12] as a starting point for optimizing our gates and matching the delays of the data path to those of the GasP backwards control. For conciseness in this section, we report delays measured using typical process parameters. Section 5.4 describes our validation of the design using five-corner Spice simulations. Surfing is used to control timing along the critical path. Along noncritical paths, we employ two additional mechanisms to control timing: pseudolatches and generous pulse widths. Each of these three mechanisms as applied in the multiplier are described in greater detail below.

The critical timing path is through surfing domino gates XOR_1, XOR_2, and MUX_1. The gates use the circuit topology shown in Figure 8 with the timing shown

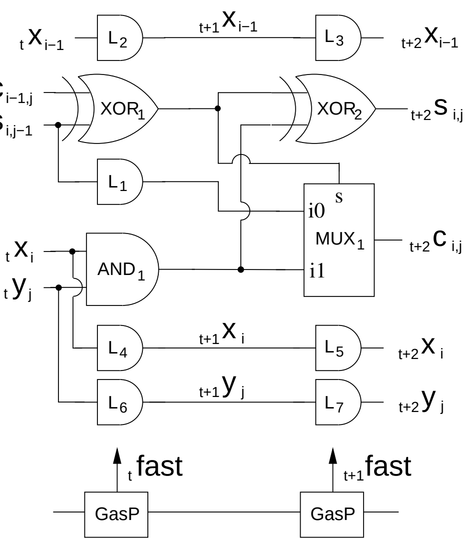

Figure 12. Multiplier Cell

in Figure 9. From Figure 9 we observe that for the XOR gate $\delta_{\text{slow,min}} = 245$ ps, and $\delta_{\text{fast,max}} = 173$ ps, with the MUX having a slightly lower delay. The forward delay of the timing pulse is on average 206 ps.[1] Thus, the inequalities of Equation 4 are satisfied.

The bits of the x and y arguments are passed along with the results of the computation, using pseudolatches L_2 through L_7. As shown in Figure 11, the pseudolatch is a self-resetting domino AND gate with the timing pulse as one of the inputs to keep signals aligned with the wave when not part of the current computation. The pseudolatch is not state holding; it only serves to prevent the events off the critical path from outpacing the events on the critical path. The pseudolatch design is simpler and uses less power than deliberately constructing a slow buffer and then accelerating it with surfing. To ensure proper operation, we must show that pulses from the critical path logic and pulses from the pseudolatches have sufficient overlap to properly trigger the gates for which they are inputs. In particular, we must establish overlap of pulses at the $\mathsf{i0}$ and s inputs of multiplexor MUX_1. The timing of both pulses can be determined relative to the timing pulse for XOR gate XOR_1 and pseudolatch L_1. The two paths are:

$$_{t}\mathsf{fast} \xrightarrow[\mathsf{XOR}_1]{180\text{ ps}} \mathsf{MUX}_1[\mathsf{s}]$$

$$\text{and} \quad _{t}\mathsf{fast} \xrightarrow[\mathsf{L}_1]{170\text{ ps}} \mathsf{MUX}_1[\mathsf{i0}]$$

The width of the XOR's output is around 345 ps, and the width of the pseudolatch's output is 350 ps. This ensures an overlap of about 340 ps, which is more than sufficient for the correct operation of MUX_1. This approach of widening input pulses for multiple input gates to ensure adequate overlap is essentially the same as that taken by Chappell *et al.* [3].

Paths through gate AND_1 introduce their own timing issues. The *true* side of gate AND_1 is the surfing domino AND gate shown in Figure 7. It has delays that are just slightly less than those for XOR gates due to lower parasitic capacitances on internal node g. The *false* side of gate AND_1 is a self-resetting domino OR gate as shown in Figure 13. This gate has a delay that is substantially lower than the target interstage delay. When exactly one of $_{t}\mathsf{x}_i$ or $_{t}\mathsf{y}_j$ is 0, the delay through the *false* side of AND_1 is typically 161 ps; when both inputs are 0, the delay drops to 112 ps. Rather than modifying the gate topology, we observed that the fast *false* side does not impair the function of our multiplier.

To verify the timing of paths through gate AND_1, we consider paths that start with the timing pulse for XOR_2, MUX_1, L_3, and L_7 of the previous stage, and propagate through these gates and gates XOR_1 and AND_1 of the current stage to reach inputs s and $\mathsf{i1}$ of MUX_1 respectively. The delay to the s input of the multiplexor is given by:

$$_{t-1}\mathsf{fast} \xrightarrow[{}_{t-1}\mathsf{XOR}_2]{180\text{ ps}} {}_{t}\mathsf{s}_{i,j-1} \xrightarrow[{}_{t}\mathsf{XOR}_1]{206\text{ ps}} \mathsf{MUX}_1[\mathsf{s}]$$

or, roughly, 386 ps. The path through $_{t-1}\mathsf{MUX}_1$ is equivalent. The delay to the $\mathsf{i1}$ input of the multiplexor through the *false* side of gate AND_1 is given by:

$$_{t-1}\mathsf{fast} \xrightarrow[{}_{t-1}\mathsf{L}_3]{170\text{ ps}} {}_{t}\mathsf{x}_i \xrightarrow[{}_{t}\mathsf{AND}_1]{[112\text{ ps},161\text{ ps}]} {}_{t+1}\mathsf{MUX}_1[\mathsf{i1}]$$

The total delay on this path is between 282 ps and 331 ps, Thus, the pulse for the $\mathsf{i1}$ input of MUX_1 can arrive as much as 104 ps before the pulse for the s input. Pulses output from gate AND_1 have a width of at least 359 ps. This provides a minimum overlap of rougly 255 ps, which is still sufficient to ensure correct operation of the multiplexor.

The analyses for the other paths are similar to those described above.

5.3 Layout

We performed our tests on a 4×12 version of the multiplier design. We chose 4×12 because it offers a deep pipeline (36 stages of computation) while keeping simulation time and memory usage reasonable. We used the layout editor Magic [8] to create a physical layout of our design. The layout is 0.7 mm $\times$ 1.8 mm. We emphasized ease of design; thus, in many regions the layout is

[1] The average GasP delay of 206 ps differs from the measurements in Figure 9 because the simulation conditions were different.

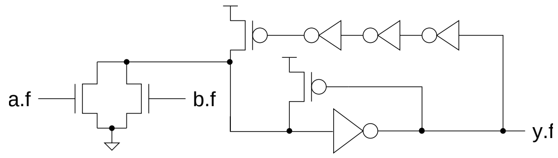

Figure 13. The *False* Side of a Dual-Rail Self-Resetting Domino AND Gate

Supply	Induct.	Current	Noise Ampl.
Logic	1 nH	286 mA	76 mV
Preswitching	3 nH	33 mA	111 mV
GasP	3 nH	52 mA	68 mV
Timing Dist.	3 nH	104 mA	195 mV

Table 1. Typical Power Supply Characteristics

not very dense, and it is certainly not optimized to minimize wiring capacitances.

We extracted capacitances for our Spice simulations from the layout. We were unable to extract wire resistances; thus, they were omitted from the model. We built the multiplier with four separate power supplies: for logic, preswitching, GasP, and timing signal distribution. The separate supplies allow us to measure the power consumption due to each class of elements. To simulate power supply noise we placed inductors in series with each of the four supplies.

5.4 Results

Our tests were performed in Spice using parameters from a 0.35μ, 3.3 V process. All timing measurements are taken at the fifty percent point. Under typical process parameters, the average measured FO4 delay [5] is 185 ps.

We tested both the speed and robustness of the design. We observed correct multiplier operation at all five process corners. Under typical process parameters the average interstage GasP delay is 206 ps. That gives an end-to-end latency of 7.4 ns. We can issue at 1.11 GHz, for eight waves in flight. At the fast/fast corner, our average interstage delay is 154 ps, and we can issue at 1.3 GHz. At the slow/slow corner, our average interstage delay is 269 ps, and we can issue at 850 MHz.

Table 1 lists power supply parameters. We used an inductance of 3 nH per supply pin. Since the logic uses by far the most power, we allocated three pins for that supply. These results were observed under typical process parameters, while driving the pipeline at 1.11 GHz. Current measurements include the input drivers (standard clocked domino buffers), but the outputs were not driving any load. Observed currents are maximums, and observed noise amplitudes are from lowest peak to highest peak. Note that the power consumed by preswitching is an order of magnitude lower than that consumed by the multiplier logic.

Our design exhibited some sensitivity to power supply noise. Initially we tested using 3 nH inductors on all four supplies. The multiplier consistently failed at the fast/fast corner. We observed a prolonged period of operation when the logic supply was 0.2 V lower than the GasP supply. As a consequence the multiplier logic ran slower relative to GasP, narrowing the gap between $\delta_{F,min}$ and $\delta_{fast,max}$ (see Section 3), reducing our tolerance for late signals. While we believe that the separate power supplies almost certainly exacerbated the noise problem, and a design with a single global power supply would be more robust, we have simulated with separate supplies to stress the design with respect to power supply noise. We are continuing our Spice simulation studies to determine the version that we will fabricate.

6 The Future of Surfing

We have designed a surfing multiplier in a 0.35μ process to the point of a layout. Spice simulations of the extracted layout show that the multiplier achieves very low gate delays and is robust with respect to variation of fabrication parameters and power supply noise. The obvious next step is to add the necessary test structures to this layout, fabricate the design, and test it. We will be doing these in the near future.

As mentioned in section 4.1, our approach to surfing introduces a trade-off between timing margins and noise immunity. Clearly much more extensive analysis and testing must be done to examine the noise sensitivity of surfing domino logic. Furthermore, we are exploring variations of the basic surfing gate design presented in section 4.1 to determine if designs that are even faster and/or more robust are feasible.

When we first considered surfing designs, we were concerned about the added power consumption due to preswitching. In some of our early designs that we have since rejected, preswitching increased power consumption by factors of four or greater. Our current design is much cooler. As shown in Table 1, preswitching current accounts for only 7% of the total power budget of our multiplier. All timing circuits (GasP + timing pulse distribution + preswitching) account for 40% of the total budget. While this number is comparable to many high-performance synchronous designs, it is an obvious area to look for improvements.

The design of the multiplier was simplified because its critical paths consist of chains of identical, or nearly identical, gates. We expect that surfing can be employed profitably in other structures as well. For such designs to be practical, we need to find practical design methodologies that will ensure sufficient matching of forward delay of the control chain to the propagation delay of the data path. Logical effort [12] is an obvious place to start. Determining a consistent effort model for preswitched gates and developing the rest of a design methodology are key areas for future work.

The analysis in Section 3 indicates that computations can surf through an arbitrarily deep logic circuit without accumulating timing uncertainty. Although the multiplier is a straight-line pipeline, we believe that our techniques can readily be extended to latch-free, surfing ring structures as well. For such a structure, computations could progress through an arbitrary number of stages without being slowed by latches. More general structures, such as the multi-rings of Sparsø and Staunstrup [9], pose additional challenges. Tokens at join nodes can stall, waiting for tokens from other branches. We have not determined an attractive way to handle such stalls in the context of surfing.

Testing is another major issue that we have yet to address. For example, scan testing relies on stopping the device under test while loading or unloading the scan registers. While stopping can be relatively straightforward with latch based designs, surfing seems much less amenable to stopping: once a wave is launched, it traverses the entire pipeline. We confess that we haven't a clue as to how one might perform production test of surfing designs. On the other hand as we gain a better understanding of the design trade-offs and opportunities of surfing, we hope that we will discover novel approaches to address testing issues.

7 Conclusions

We have presented surfing pipelines and described their implementation using a simple variant of self-resetting domino. These pipelines achieve negative overhead: the latency of the pipeline is less than delay of an purely combinational logic implementation. Furthermore, the event attractors created by surfing support arbitrarily high degrees of wave-pipelining without latches or other road-blocks.

In our surfing pipeline, the delays of logic elements are modulated by timing pulses that propagate along with events in the pipeline's data path. We use self-timed, GasP pipelines to propagate these pulses. The use of a self-timed design was motivated by the high-speed of GasP that is well matched to the propagation delays of surfing logic elements. By using self-timed handshaking, GasP ensures that pulses are not lost in the timing chain due to timing imbalances, while avoiding the need for elaborate pulse-shaping circuitry.

To demonstrate this approach, we have designed a small multiplier.Spice simulations from the extracted layout indicate that the pipeline can operate with an issue rate of 1.11 GHz with typical process parameters and 1.3 GHz at the fast corner. The latency of the critical path is reduced by 4% compared with the corresponding, purely combinational design. This shows that surfing does indeed achieve negative overhead as promised.

We have examined robustness issues and the design appears to be tolerant of process parameter variation and power supply noise. Our next step is to fabricate the multiplier to experimentally verify the simulation results and perform further tests. We also intend to apply surfing techniques to other pipeline structures.

References

[1] W. P. Burleson, M. Ciesielski, et al. Wave-pipelining: A tutorial and research survey. *IEEE Trans. on VLSI Systems*, 6(3):464–474, Sept. 1998.

[2] T. Chaney and C. Molnar. Anomalous behavior of synchronizer and arbiter circuits. *IEEE Trans. on Computers*, C-22(4):421–422, Apr. 1973.

[3] T. I. Chappell, B. A. Chappell, et al. A 2-ns cycle, 3.8-ns access 512-kb CMOS ECL SRAM with a fully pipelined architecture. *IEEE J. of Solid-State Circuits*, 26(11):1577–1585, Nov. 1991.

[4] A. E. Dooply and K. Y. Yun. Optimal clocking and enhanced testability for high-performance self-resetting domino pipelines. In *Proceedings of the Twentieth Anniversary Conference on Advanced Research in VLSI*, pages 220–214, Mar. 1999.

[5] M. Horowitz, C.-K. K. Yang, and S. Sidiropoulos. High-speed electrical signaling: Overview and limitations. *IEEE Micro*, 18(1):12–24, Jan./Feb. 1998.

[6] R. Krambeck, C. Lee, and H. Law. High-speed compact circuits with CMOS. *IEEE J. of Solid-State Circuits*, SC-17:614–619, June 1982.

[7] M. Olivieri. Design of synchronous and asynchronous variable-latency pipelined multipliers. *IEEE Trans. on VLSI Systems*, 9(2):365–376, Apr. 2001.

[8] J. K. Ousterhout, G. T. Hamachi, et al. Magic: A VLSI layout system. In *Proceedings of the 21th ACM/IEEE DAC*, pages 152–159, Albequerque, NM, June 1984.

[9] J. Sparsø and J. Staunstrup. Delay-insensitive multi-ring structures. *INTEGRATION*, 15(3):313–340, Oct. 1993.

[10] I. Sutherland. personal communication, Nov. 2000. based on SUN technical memo from Sept. 1996.

[11] I. Sutherland and S. Fairbanks. GasP: A minimal FIFO control. In *Proc. 7th Intl. Symp. on Adv. Research in Asynchronous Circuits and Systems*, pages 46–53, Apr. 2001.

[12] I. Sutherland, B. Sproull, and D. Harris. *Logical Effort: Designing Fast CMOS Circuits*. Morgan Kaufmann Publishers, Inc., Jan. 1999.

[13] T. E. Williams and M. A. Horowitz. A zero-overhead self-timed 160ns 54b CMOS divider. *IEEE J. of Solid-State Circuits*, 26(11):1651–1661, Nov. 1991.