

Notes On Pulse Signaling

Jo Ebergen, Steve Furber, Arash Saifhashemi,
Naela Nissar, Alex Chow *
Sun Microsystems Laboratories
Menlo Park, USA

Abstract

This paper reports results of a study on pulse signaling. In pulse signaling, components communicate by means of pulses instead of voltage transitions. The functionality of the components is very similar to the functionality of components used in so-called asynchronous transition signalling. In asynchronous transition signalling, communication events are represented by voltage transitions, whereas in pulse signaling communication events are represented by pulses. We describe various implementations of pulse-signaling components, report on the energy efficiencies of our implementations, and look at some robustness aspects.

1. Introduction

This paper reports on a study of asynchronous systems where components communicate by means of pulse signaling instead of transition or level signaling. In pulse signaling, communication events are implemented by means of pulses, instead of voltage transitions or voltage levels. The general idea is that each component receives a set of input pulses and produces a set of output pulses based on the set of input pulses received. We present some implementations for pulse modules and investigate the energy efficiency and robustness of these implementations.

The main inspiration for this study comes from the research by Bainbridge et al [1]. In [1], the authors propose a set of circuits for building networks on chip, called CHAIN. CHAIN uses a four-phase protocol with level signaling. The circuits in CHAIN inspired us to look at a family of circuits that use the faster two-phase protocol, but communicate by means of pulses instead of transitions. The main goal of this research was to find out if these pulse circuits are more

cost effective than the CHAIN circuits, where costs can be expressed in delay, energy, area, or a combination thereof.

We had another reason to look at pulse circuits. In the literature, various authors have proposed pulse-mode circuits, like GasP [13], APL [8], STFB [4]. These circuit families offer high speeds, but come with the disadvantage of higher noise sensitivity. In particular, these circuit families connect components by means of tri-state wires, which weakly keep their states with keepers when floating. Such tri-state wires are susceptible to noise through capacitive coupling on neighboring wires, in particular when these wires are long. Often these noise-sensitive wires are connected to a single pull-down or single pull-up stack of transistors, which have smaller noise margins than static gates. These smaller noise margins exacerbate the problem. In order to obtain a circuit family that is more immune to noise, we wanted to explore a family of circuits that connect components with wires that are always strongly driven. Yet, we also wanted to retain the pulse-mode signaling, because it seemed to offer simple and fast implementations. Golani and Beerel also address the issue of robustness, but for STFB circuits, in [5].

In our study we wanted to answer a number of obvious questions when exploring pulse signaling. First, does there exist a simple implementation template for pulse modules in a CMOS technology? Second, how efficient are these implementations with respect to delay, energy, and area? As a comparison we decided to use the pipeline circuits suggested in CHAIN [1]. Third, how robust is pulse signaling? Although long wires may arbitrarily delay a pulse or vary the pulse width, the pulse itself must still be detectable at the receiver. This condition may put constraints on the connection wires. We wanted to find out how onerous those constraints can be. The following provides our answers to these questions.

2. Related Work

The desire to find more efficient implementations for the networks on chip in [1] served as our main inspiration.

*This work was performed while some of the authors were at Sun Labs. Steve Furber is with the University of Manchester, England. Arash Saifhashemi is with the University of Southern California, USA. Naela Nissar is with the University of Waterloo, Canada.

Asynchronous systems based on pulse signaling have been suggested before by many researchers, although with very different implementations. The earliest mention we found of pulse signaling in asynchronous systems is in [7], which focuses more on correctness conditions than on implementation details. In [6], the authors also employ pulse signaling in asynchronous systems, but in a very different implementation technology. Plana and Unger propose a set of pulse-signaling components in [10] and present some systems designed with these components. The authors do not address implementation details, energy efficiency, or robustness issues with respect to long wires.

Our implementations produce fixed-width pulses. In order to produce these fixed-width pulses, we use techniques from self-resetting logic, also called post-charge logic by its inventor Bob Proebsting [11]. Self-resetting techniques have been applied mostly in combinational logic, such as address decoders for example, to achieve high-speed and low energy implementations.

In IBM's Interlocked Pipeline CMOS (IPCOS)[12], the authors describe a control circuit for a pipeline where adjacent stages of the pipeline "interlock" by means of a component that one could call a pulse Join or C-element. Their implementation of a pulse Join, however, is very different than the one proposed here.

Asynchronous Pulse Logic of Nystrom and Martin [9], GasP circuits of Sun Labs [13], as well as Beerel and Ferretti's Single-Track Full Buffer circuits [4] all are based on the same type of pulse-mode circuits. Internally each circuit module produces a fixed-width pulse when the module is supposed to "fire." Instead of communicating pulses, however, these circuits communicate by means of a two-phase protocol on a tri-state wire, also called single-track signaling. None of these pulse-mode circuits have outputs that are always actively driven by the module. Instead, the communication wires are tri-state wires, which may be susceptible to noise. The outputs of our pulse circuits are always actively driven.

Recently [5] introduced various techniques to improve the robustness of STFB circuits. At a small cost in extra energy, Golani and Beerel show that STFB circuits can be made very robust by adding special keepers at the sender and receiver so that sender or receiver always actively drive the state wire between them. The techniques only apply to state wires with a single sender and a single receiver. Our pulse circuits allow connections between a single sender and an arbitrary number of receivers.

3. Signaling protocols

Two common signaling techniques in asynchronous circuit design are transition signaling and level signaling. In transition signaling, components communicate by means of

voltage transitions. Each communication cycle using transition signaling has a two-phase protocol, where during the first phase the sender sends a request transition on the request wire and during the second phase the receiver sends an acknowledge transition on the acknowledge wire.

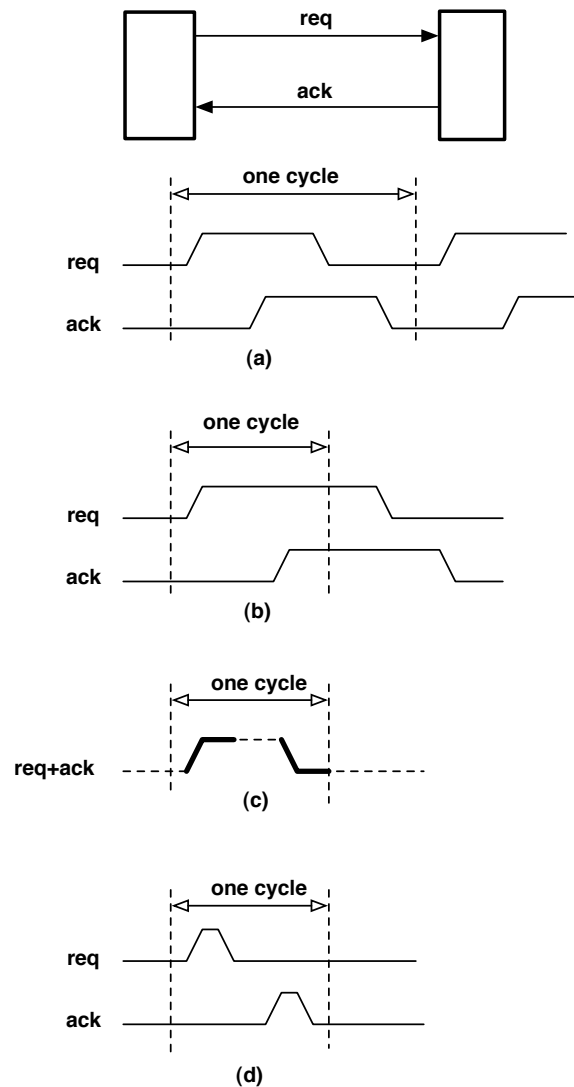


Figure 1. Four signaling protocols. (a) level signaling (b) transition signaling (c) single-track signaling (d) pulse signaling

In level signaling, primitive components communicate by means of levels. Each communication cycle using level signaling has a four-phase protocol, first there is a request and acknowledge phase for the rising transitions and then another request and acknowledge phase for the falling transitions. Thus, after the fourth phase the primitive compo-

nents are back in their initial states. Level signaling has the advantage that it leads to simpler circuits at the cost of four phases per communication cycle. Transition signaling has the advantage of only two phases per communication cycle, but at the cost of more complex circuits. Figure 1(a) illustrates level signaling and Figure 1(b) illustrates transition signaling.

If the request and acknowledge wires are combined, one obtains two-phase signaling on a tri-state wire, also called single-track signaling [2]. This signaling scheme has become popular recently, because it combines the advantages of the two and four-phase signaling schemes. These advantages, however, come at a cost of a higher noise sensitivity, because of the tri-state wires. Figure 1(c) illustrates single-track signaling.

A fourth signaling scheme is pulse signaling, where each request and acknowledge event is a pulse on the respective wire. Pulse signaling shares with transition signaling and single-track signaling the advantage of conceptual simplicity and small cycles and shares with level signaling the advantage of dealing with levels instead of transitions. The potential problem that any pulse signaling implementation must solve, however, is the robustness of sending pulses over long wires.

4. Implementations

Our implementation template for pulse modules makes use of self-resetting logic to produce fixed-width pulses. Let us start with a simple example to explain the basic ideas. The example concerns the identity gate for pulses, also called a pulse repeater [9]. Figure 2(a) shows a schematic and implementation of a pulse repeater.

This pulse repeater receives a positive pulse of fixed width on its input. The NMOS transistor in the first stage converts the positive input pulse to a low voltage level on the internal node X . Once the internal node is low, a reset loop starts with fixed delay. The reset loop contains two inverters and a PMOS transistor, each sized so that the loop delay has a fixed delay. In our implementations, we size each gate for equal delay, so the loop delay is three gate delays, where the nominal value of the gate delay can be chosen before sizing the gates. Although equal gate delays are convenient, they are not necessary to make this pulse repeater work.

The PMOS transistor drives the internal node X high, three gate delays after the NMOS transistor drives the internal node low. In order to avoid any fighting, the input pulse must have a width at most equal to the loop delay, but must also be wide enough to drive the internal node low. Thus, the NMOS transistor stops driving before the PMOS transistor starts driving. The PMOS transistor drives the internal node high for only three gate delays, because of the feedback loop of three gate delays. The keeper, labeled K ,

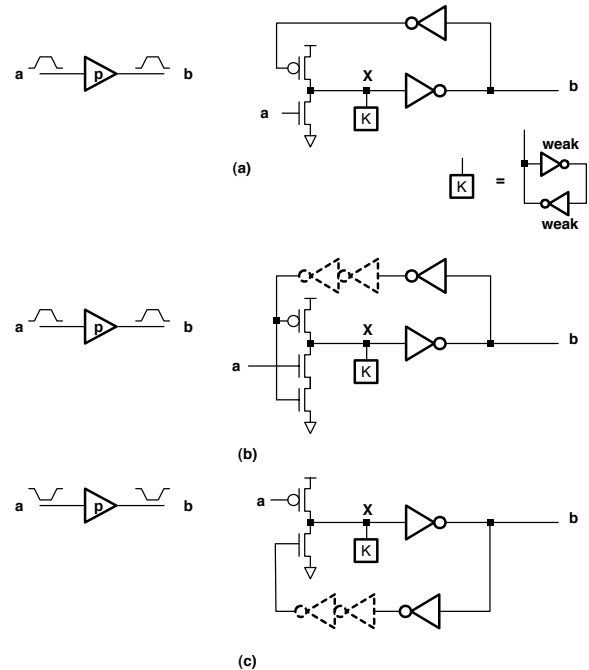


Figure 2. Some pulse repeaters

will keep the internal node high after the PMOS stops driving the internal node. When the internal node goes high, the pulse repeater produces the trailing edge of the pulse at output b .

An advantageous property of these implementations of the pulse modules is that a pulse module automatically restores input pulses that deviate from the standard width, as long as the deviation is not too large [9]. Independent experiments at Sun Labs have confirmed this property. For a small price in complexity, one can make the implementation more robust against too wide input pulses. Figure 2(b) shows an implementation that avoids any prolonged fights between the pull-up and pull-down transistors at the cost of an extra NMOS transistor.

The reset loop delay determines the width of the output pulse. For a loop delay of three gate delays, the pulse width will be three gate delays. For a loop delay of five gate delays, the pulse width will be five gate delays. Loops of five gate delays or more are more robust and are preferable for long wire connections. For the sake of convenience all drawings in this paper assume a pulse width of three gate delays. Longer loop delays are obtained by adding inverters in the reset loop as illustrated by the dashed inverters in Figure 2(b). The output of the pulse module can be taken from the output of the first or third inverter in the reset loop.

Besides repeaters that receive and produce positive pulses, there are also pulse repeaters that receive and pro-

duce negative pulses. Figure 2(c) shows a repeater that receives and produces negative pulses.

Pulse repeaters, and in general pulse modules, need to be initialized. For a positive pulse repeater, for example, the internal node X must be initialized high. There are several ways to do this. An obvious initialization method precharges the internal node through the keeper, for example with an extra PMOS transistor or replacing an inverter in the keeper by a NAND or NOR gate. Similar initialization methods can be applied to modules operating on negative pulses.

Besides the pulse repeater there are many other pulse modules realizing different functions. The pulse Merge, the pulse Join are obvious components that come to mind, but also other functions like AND, OR, XOR, or MUXes can be implemented using 1-of- N inputs. These pulse modules are more complicated in behavior than the pulse repeater, but all modules are based on the same principle: upon receipt of a number of input pulses, the module produces an output pulse, and after an output pulse the environment may provide a new set of input pulses. All our implementations must satisfy the following conditions.

pulse width All input and output pulses have about the same width W and the pulse width must be large enough so that each pulse module can detect a pulse. For large variations in the pulse width, one should use the more robust implementation in Figure 2(b). If a good transistor sizing tool can be used then all pulses will have about equal width and implementation Figure 2(a) will be more energy efficient. If we measure pulse width W in fanout-of-4 inverter delays (FO4 for short), we suggest to use $W > 3\text{FO4}$ and larger minimum width if pulse widths have large variations or long wires must be driven.

pulse separation Because each pulse module must be able to reset itself after firing, the minimum separation between the leading edges of two successive input pulses is at least $2W$. This condition implies that the minimum cycle time for each pulse module is at least $2W$.

4.1. The Pulse Merge

Let us look at some other simple pulse components. Figure 3 shows a pulse Merge. The pulse Merge receives a pulse on either input a or input b and produces a pulse on output c . The behavior then repeats. The right side of Figure 3 shows an implementation of the pulse Merge. The specification of the pulse Merge stipulates that the environment is prohibited from giving two pulses concurrently, either on the same input or on different inputs, before an output pulse appears. The implementation of the pulse Merge is a straightforward generalization of the pulse repeater.

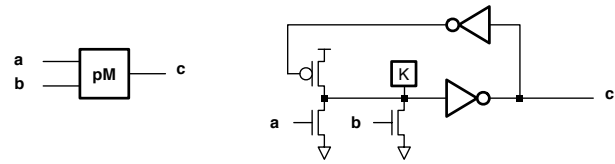


Figure 3. An implementation of a pulse Merge

4.2. The Pulse Join

A pulse Join appears in Figure 4(a). A pulse Join receives a pulse on input a and input b and then produces a pulse on output c . This behavior then repeats. It is important to note that the pulse Join produces the output pulse only after an input pulse has been received on both inputs and that these input pulses may arrive at arbitrary times. A second important property is that an environment is forbidden to provide two input pulses in a row on the same input before an output pulse is produced. (A Join is like a C-element with special restrictions on its use.)

An implementation of the pulse Join appears in Figure 4(b), including keepers. Notice that the NOR gate produces the leading edge of the pulse only after both internal nodes have become low. The reset loop then resets the internal nodes high concurrently and finally produces the trailing edge of the pulse.

Because both inputs to the NOR gate always go high concurrently, a transistor implementation can exploit this by halving the drive strength of the NMOS transistors in the NOR. This reduces the logical efforts of the inputs and outputs of the NOR gate and renders a more energy-efficient implementation.

Figure 4(c) shows an implementation for the pulse Join for negative pulses. Figure 4(d) shows an erroneous implementation of a pulse Join. This implementation fails when input pulses arrive at very different times.

4.3. n -by- m Joins

A generalization of a Join is called a n -by- m Join, also called n -by- m Decision-Wait. Figure 5(a) shows a symbol of a 2-by-1 pulse Join with inputs $a0$, $a1$, and c and outputs $b0$ and $b1$. The Join receives a pulse on either input $a0$ or $a1$ and a pulse on input c . When the 2-by-1 Join receives a pulse on inputs $a0$ and c , it produces a pulse on output $b0$. When the Join receives a pulse on inputs $a1$ and c , it produces a pulse on output $b1$. After producing a pulse on output $b0$ or $b1$, the environment may provide a new set of input pulses.

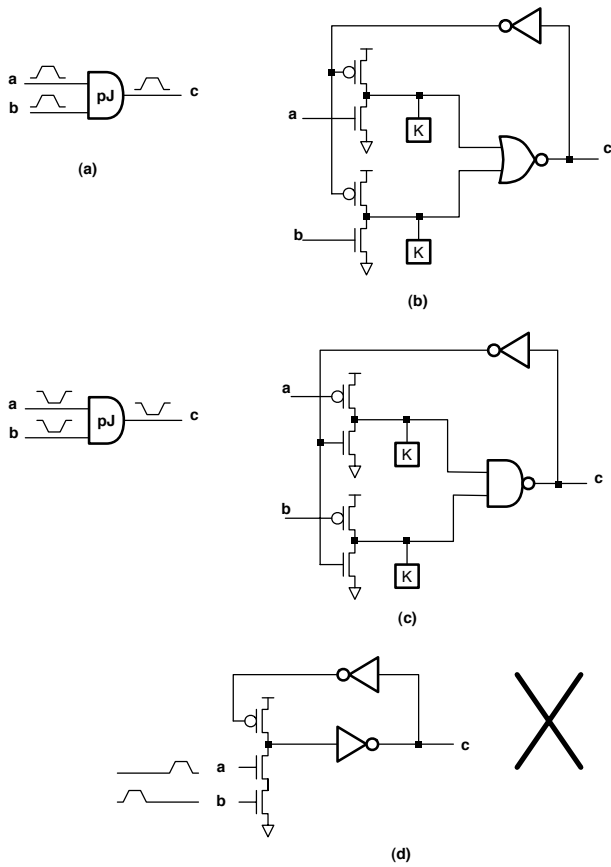


Figure 4. Pulse Joins and some implementations. (a) circuit symbol (b) a positive pulse implementation (c) a negative pulse implementation (d) an incorrect implementation

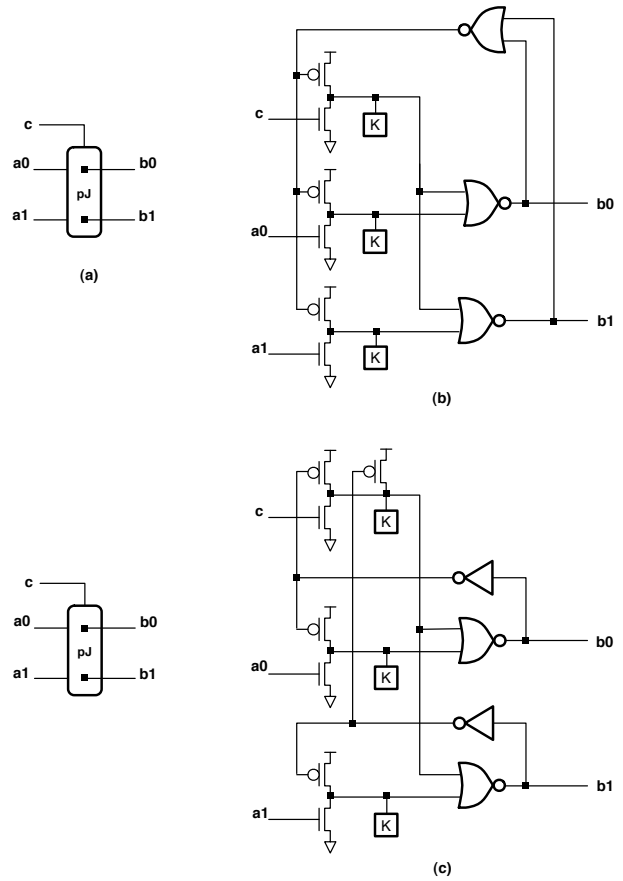


Figure 5. Pulse 2-by-1 Joins and their implementations

An implementation for positive pulses appears in Figure 5(b). Figure 5(c) shows an alternative implementation for positive pulses. Negative pulse implementations follow straightforwardly from the positive pulse implementations and are omitted here.

These implementations can be generalized to n -by-1 Joins, and even n -by- m Joins. An n -by- m Join has one set of n inputs and one set of m inputs and a set of $n \times m$ outputs, one output for each combination of two inputs. It receives one pulse on only one of its set of n inputs and one pulse on its set of m inputs. It then produces a pulse on the appropriate output. Implementations of n -by- m Joins follow straightforwardly from the implementations of the 2-by-1 Join.

Although we have presented several implementations for the same pulse module, we have not said which is the better implementation. For example, we proposed a positive pulse and a negative pulse implementation of the pulse Join. We also proposed two implementations for the n -by- m Join in Figure 5. Which implementation is better? Here “better” means that the circuit can be better in some respect for the same “cost.” For example, the circuit can be faster for the same amount of energy consumption or layout area, or the circuit can be more energy efficient for the same speed. A logical-effort analysis often quickly reveals the better implementation [3]. Although a logical-effort analysis is outside the scope of this paper, we mention that negative pulse implementations are often better than positive pulse implementations and Figure 5(c) is a better implementation of the 2-by-1 Join than Figure 5(b).

5. An Application: A Pipeline

Figure 6(a) illustrates an application of pulse circuits. The figure shows a network of pulse components forming a one-stage pipeline. This particular stage uses 1-out-of-4 encoding to communicate 2 bits: a pulse communicated on one wire encodes for one value out of four. The minimum cycle time of this stage comprises the delay of two 4-by-1 pulse Joins and a pulse Merge. If each component has a forward latency of 2 gate delays, then the minimum cycle time is six gate delays. This minimum cycle time satisfies the condition for the minimum pulse separation of pulse modules with reset loops of 3 gate delays. If the pulse modules use reset loops of five gate delays, the minimum cycle time must be at least ten gate delays, so at least two pulse repeaters need to be inserted somewhere in the circuit cycle.

Figure 6(b) shows the equivalent implementation of the pipeline stage when using level signaling and 1-out-of-4 encoding as proposed in CHAIN [1]. In order to transfer one data item, the cycle consisting of two C-elements and one NOR gate has to be executed twice: once for a rising tran-

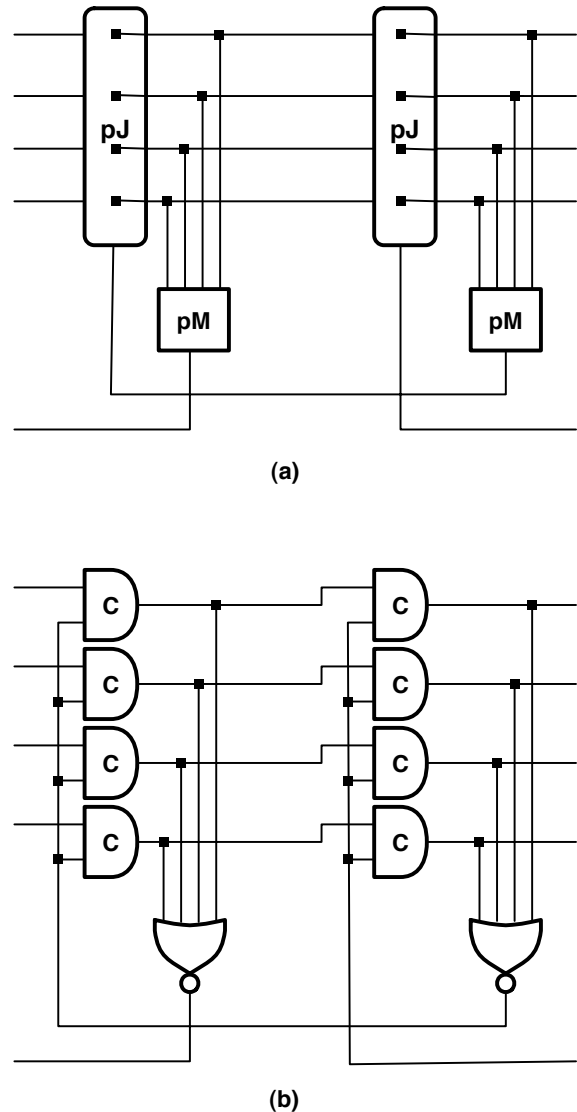


Figure 6. A pipeline stage using 1-out-of-4 encoding. (a) Implementation for pulse signaling (b) Implementation for level signaling

sition and once for a falling transition. The total cycle time for this circuit is roughly 10 gate delays.

We performed a study into the energy-delay trade offs of various pulse-signaling implementations and level-signaling implementations of simple pipelines in a CMOS 180nm technology. The pipelines contained two quad-rail data paths and assumed 500μ wires between stages and 100μ wires before the four-input Merges to generate the acknowledgement. We then calculated the estimated energy consumption per cycle for each given minimum cycle time, assuming that all gates are sized for equal delay. These calculations render the trade-off curves for energy consumption versus cycle time. The trade-off curves in Figure 7 show that the implementations for negative pulse signaling have the best energy-delay trade off, followed by the implementations for positive pulse signaling, and then the implementations for level signaling. The units for delay and energy in Figure 7 are normalized to those of a minimum-sized inverter, τ and ϵ respectively. In 180nm TSMC CMOS technology $\tau = 14.5\text{ps}$ and $\epsilon = 2.8\text{fJ}$. The results suggest that the cycle time for a given energy consumption for the negative pulse-signaling implementation is almost a factor two better than the cycle time for the level-signaling implementation. A similar experiment can be performed for the area-delay trade off curves with similar results.

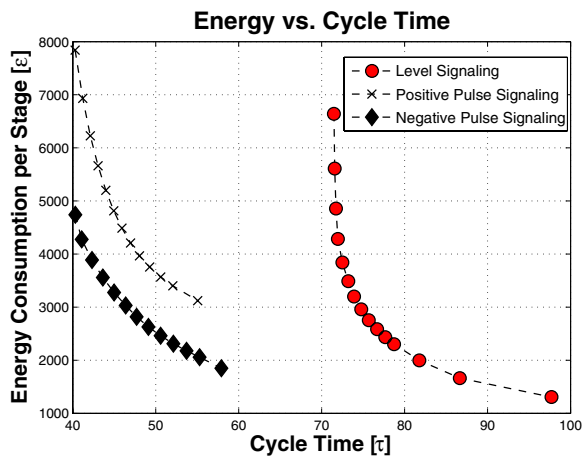


Figure 7. The energy-delay trade offs for various pipeline implementations

6. Robustness Issues

Sending pulses along long wires can be hazardous. Long wires attenuate pulses, and capacitive coupling between wires may deteriorate the pulses even more. Some long

wires may deteriorate a pulse so much that the pulse will not be detected by a receiving pulse module.

In order to find the influence of long wires on pulses we conducted two experiments: one experiment to find the maximum wire length between pulse modules for a given pulse width and no capacitive coupling between wires and one experiment to find the influence of capacitive coupling on pulse widths.

To find how long a wire can be before it deteriorates a pulse beyond detection, we constructed a ring with eleven pulse repeaters each connected with a long wire to its neighbor. For different pulse widths and slew rates we found the maximum wire length for which a pulse keeps circulating along the ring. For this experiment we used the more aggressive repeaters of Figure 2(b). For each wire length, we sized the gates to obtain the proper pulse width and then sent a pulse around the ring. We varied the slew rates by varying the stepup, or gain, of the gates in the forward path. The stepup of a gate is the ratio of its total output load to its own drive strength. We carried out the experiments in a 180nm CMOS TSMC technology. The results appear in Figure 8.

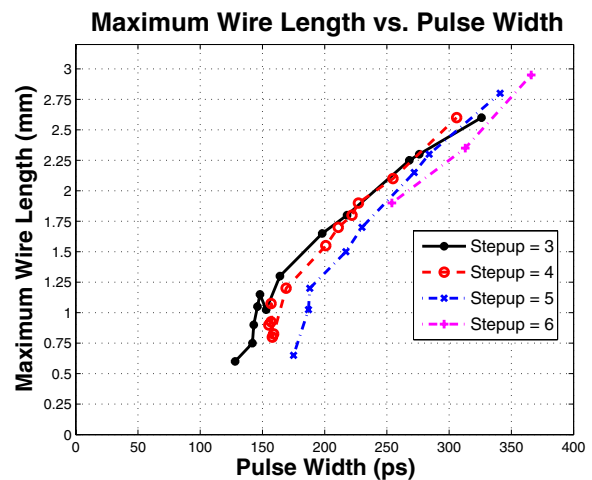


Figure 8. The maximum wire length between pulse modules as a function of the pulse width

The figure shows that there is a rough first order relationship between pulse width and maximum wire length. The maximum wire length varies from roughly 1.25mm for a pulse width of 200ps (2.66FO4 delays) to 2.75mm for a pulse width of 375ps (5FO4 delays). These wire lengths are sufficiently long to allow the construction of a large network. The figure also shows that pulses with larger slew rates lead to slightly smaller maximum wire lengths. For a stepup of 6 we had to use a sufficient number of gates in

the reset loop to get a viable pulse, hence the larger pulse widths for a stepup of 6.

These results are very conservative considering that we assumed a simple RC model, called the pi model, for the long wires. A pi model gives a longer delay than a distributed delay model, as illustrated in Figure 9, and the difference gets larger as the wire gets longer. Figure 9 shows typical wave forms of a pulse at the beginning and at the end of a 1mm wire given by a pi model. Also shown is the wave form at the end of a 1mm wire, but now modeled by a distributive RC model.

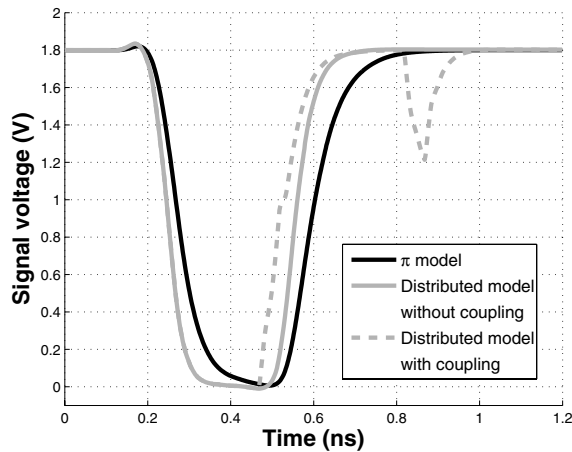


Figure 9. Typical wave forms of a 300ps pulse at the end of a 1mm wire modeled by different RC models

For several wire lengths we looked at the effect of capacitive coupling between two wires. For this experiment we took two pairs of pulse repeaters, each pair connected with a long wire. This time we modeled the two wires by distributive RC models and we calculated the capacitance between the two wires with a field solver assuming a worst-case minimum spacing over the total length of the minimum-width wires. To create the worst-case coupling between aggressor and victim, we generated a negative pulse on the victim and a positive pulse on the aggressor with different offsets.

Figure 9 shows typical waveforms as a result of capacitive coupling between aggressor and victim. Notice that not only can the pulse width be reduced by capacitive coupling, but also spurious pulses can be generated. Because the wires are always strongly driven in our designs such spurious pulses remained small and did not create any errors. For weakly driven tri-state wires, spurious pulses created by capacitive coupling can create havoc.

Figure 10 shows the variation in the pulse width on the victim wire. All experiments were done with a stepup of 5.

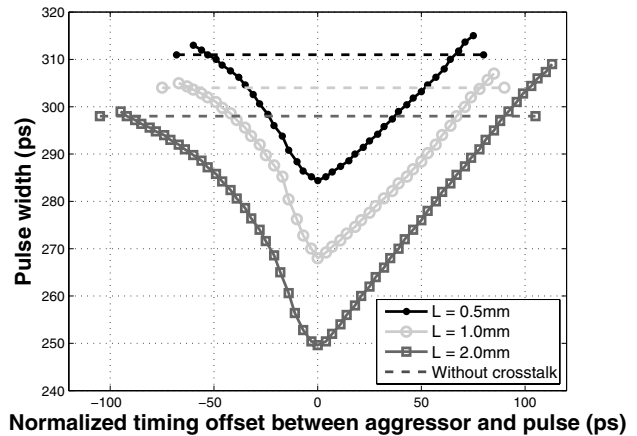


Figure 10. The effect of capacitive coupling on pulse width

The dashed lines show the pulse width at the end of a long wire, absent any capacitive-coupling effects. The solid lines show the effect of capacitive coupling on the pulse width as a function of the offset between the pulses. All curves are centered around the minimum pulse width due to capacitive coupling. The figure shows that in the worst case the pulse width reduces by 9-16%. These results are encouraging considering that we assumed minimum spacing and minimum width over the total length of the wires and that we did not include any ground planes in our capacitance calculations, which reduce the effects of capacitive coupling.

7. Protocol Conversions

In order to communicate pulses over very long wires, one can either use many pulse repeaters or simply switch to communicating transitions. Communicating a transition over a long wire rather than a pulse can be more energy efficient, because there is only one transition to communicate rather than two per event. The energy savings of sending a single transition instead of two, however, has to outweigh the extra energy and delay costs in converting between pulses and transitions. Let us examine this comparison more closely.

Here are two converters between transition signaling and pulse signaling. Figure 11(a) shows an implementation to convert a positive pulse to a transition. The implementation consists of a pulse latch and an inverting feedback loop with sufficient delay. Figure 11(b) shows an implementation to convert any transition to a positive pulse. In case the input to the transition-to-pulse converter has poor slew rates because of very long wires, one can improve the noise immunity by

putting an extra inverter in front of the input. The keeper in this implementation can actually be a half keeper that keeps a node high. Converters between negative pulses and transitions are similar.

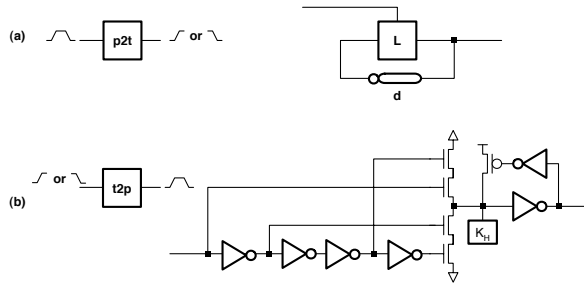


Figure 11. Conversions between pulse and transition signaling

In order to compare the costs of sending a pulse or a transition over a long wire, we examine the implementations in Figure 12. Figure 12(a) shows a module $P0$ sending a pulse over a long wire to module $P1$. In case the wire load C_w becomes large, a more energy efficient scheme is to use an extra pulse repeater to amplify the signal over several stages before sending the pulse over the wire. Figure 12(b) shows this implementation. Figure 12(c) shows the implementation where each pulse is converted to a transition first before being sent over a long wire. At the end of the long wire, a deconverter converts each transition back to a pulse. Notice that the converter in this scheme serves also as an amplifier.

What are the costs of each scheme in terms of energy and delay as a function of the wire length? In order to simplify the comparison, we assume that all gates are sized for the same delay, say a gate delay $s * \tau$, where s is the so-called stepup and τ is the technology-dependent time unit defined earlier. A common practice is to choose a FO4 delay, which corresponds to 5τ . Furthermore, we assume that the input loads of module $P1$ and $t2p$ are small loads compared to the wire load C_w . Under these assumptions we observe that each of the three schemes have common terms with respect to delay. Each scheme has the same delay due to driving the wire load C_w and the wire RC delays, because all wire drivers drive the wire with the same drive strength and each wire sees the same small load. For this reason we only look at the extra delay experienced by schemes (b) and (c) over scheme (a). When we choose a gate delay of one FO4 for each gate, scheme (b) has an extra delay of 2FO4 and scheme (c) has an average extra delay of $(2+2.5)FO4$. Notice that the deconverter alternates between a delay of 2FO4 and 3FO4, hence the average of 2.5FO4. Although the ex-

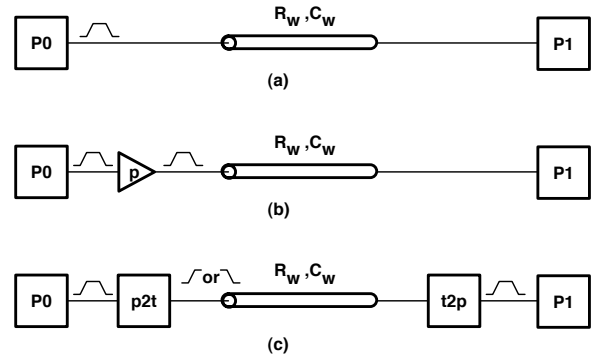


Figure 12. Comparing three schemes to communicate an event over a long wire. (a) Sending a pulse without extra amplification (b) Sending a pulse with extra amplification (c) Sending a transition using converters of Figure 11

tra delays are constant, these extra delays become relatively smaller as the wire length increases, because the wire delay increases quadratically with wire length.

What about the energy costs? Without proof, we state that driving a load C costs approximately $(sC/(s - s_c))\epsilon$ in energy when choosing a gate delay $s\tau$ [3]. Here, the unit for capacitance C is the input load of a minimum-sized inverter in a particular technology. This energy includes the energy spent by the amplification stages. The most energy-efficient amplifications have a value $s_c = 2$. For example a string of inverters achieves $s_c = 2$. Schemes (b) and (c) may approach a value of $s_c = 2$. The addition of any complexity in the amplification stages will cause this energy cost to increase. Scheme (a) most likely will include more complexity, like a Join in module $P0$. Because scheme (a) is less efficient at amplifying, we assume scheme (a) to approach a value $s_c = 3$.

To first approximation, scheme (a) and (b) spend the energy $(sC/(s - s_c))\epsilon$ twice with each pulse sent over the wire. Scheme (c) will spend this energy only once, because it sends a single transition. But scheme (c) also spends energy in the deconverter. The energy spent by a small converter is proportional to the chosen stepup s . A reasonable implementation spends about $10s\epsilon$ for a gate delay of $s\tau$. For a FO4 delay, or $s = 5$, these energy costs are about 50ϵ .

Consequently, if we choose a gate delay of one FO4 delay, or $s = 5$, we obtain an extra energy spent by scheme (a) over scheme (c) of

$$(2 \frac{5C}{5-3} - \frac{5C}{5-2} - 50)\epsilon = (\frac{10C}{3} - 50)\epsilon$$

The extra energy spent by scheme (b) over scheme (c) is

$$\left(2\frac{5C}{5-2} - \frac{5C}{5-2} - 50\right)\epsilon = \left(\frac{5C}{3} - 50\right)\epsilon$$

In 180nm CMOS technology, $C \approx 0.14l$, where l is given in microns of minimum-width wire. Thus a rough first-order estimation tells us that scheme (c) is more energy efficient than scheme (a) for wire lengths over 110μ at an extra average delay of 4.5 FO4 per wire crossing. Scheme (c) is more efficient than scheme (b) for wire lengths over 220μ at an extra average cost of 2.5FO4 per wire crossing.

8. Concluding Remarks

Let us summarize the main points of this paper. We discussed some implementation templates for components employed in pulse signaling and studied their energy efficiency and robustness.

One advantage of our implementations of the pulse circuits is that the outputs are always actively driven as opposed to tri-stated as in the pulse-mode circuits of [4, 9, 13]. Actively driven wires have a better noise immunity than tri-stated wires. This is particularly important if the inputs of your modules lead to single pull-down or single pull-up stacks, which have smaller noise margins than static gates.

As for energy efficiency, logical-effort analyses indicated that negative pulse implementations are better than positive pulse implementations, where “better” means less delay for the same amount of energy or less energy for the same amount of delay. A similar logical-effort analysis suggests that pulse-signaling pipelines are better implementations than the level-signaling pipelines in [1] by almost a factor of 2.

A study on the effects of long wires revealed that there is roughly a linear relationship between the pulse width and maximum wire length between pulse modules, where the maximum wire length varied from 1.25mm for a pulse width of 200ps to 2.75mm for a pulse width of 375ps. These lengths are conservative. Furthermore, we learned that capacitive coupling can at worst reduce the pulse width by 16% over a 2mm wire.

Pulse modules from one family must produce pulses of about the same width. How well can we control these pulse widths? It has been our experience that pulse widths can be well controlled when you can automatically size gates. We believe that with reset loops of five gates or more, the pulse width can be well controlled even when sizing gates by hand. Furthermore, implementations of pulse modules that use an extra disabling transistor as in Figure 2(b) are even more robust under pulse variations.

References

- [1] J. Bainbridge and S. Furber. CHAIN: A delay-insensitive chip area interconnect. *IEEE Micro*, 22:16–23, 2002.
- [2] K. v. Berkel and A. Bink. Single-track handshaking signaling with application to micropipelines and handshake circuits. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 122–133. IEEE Computer Society Press, Mar. 1996.
- [3] J. Ebergen, J. Gainsley, and P. Cunningham. Transistor sizing: How to control the speed and energy consumption of a circuit. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 51–61. IEEE Computer Society Press, Apr. 2004.
- [4] M. Ferretti, R. Ozdag, and P. Beerel. High performance asynchronous ASIC back-end design flow using single-track full-buffer standard cells. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 95–105. IEEE Computer Society Press, Apr. 2004.
- [5] P. Golani and P. Beerel. High-performance noise-robust asynchronous circuits. In *Proceedings of the 2006 Emerging VLSI Technologies and Architectures (ISVLSI06)*, 2006.
- [6] Y. Kameda, S. Polonsky, M. Maezawa, and T. Nanya. Primitive-level pipelining method on delay-insensitive model for RSFQ pulse-driven logic. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 262–273, 1998.
- [7] R. M. Keller. Towards a theory of universal speed-independent modules. *IEEE Transactions on Computers*, C-23(1):21–33, Jan. 1974.
- [8] M. Nyström. *Asynchronous Pulse Logic*. PhD thesis, California Institute of Technology, May 2001. Caltech Computer Science Technical Report 2001.011.
- [9] M. Nyström and A. Martin. *Asynchronous Pulse Logic*. Kluwer Academic Publishers, 2002.
- [10] L. A. Plana and S. H. Unger. Pulse-mode macromodular systems. In *Proc. International Conf. Computer Design (ICCD)*, pages 348–353, Oct. 1998.
- [11] R. Proebsting. Speed enhancement techniques for cmos circuits. US patent 4985643, 15 Jan 1991, and US patent 5343090, 30 Aug 1994.
- [12] S. Schuster and P. Cook. Low-power synchronous-to-asynchronous-to-synchronous interlocked pipelined cmos circuits operating at 3.3–4.5 ghz. *IEEE Journal of Solid-State Circuits*, 38(4):622–630, Apr. 2004.
- [13] I. Sutherland and S. Fairbanks. GasP: A minimal FIFO control. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 46–53. IEEE Computer Society Press, Mar. 2001.