Single-Track Asynchronous Pipeline Templates Using 1-of-N Encoding

Marcos Ferretti, Peter A. Beerel
Department of Electrical Engineering Systems
University of Southern California
Los Angeles, CA 90089 – USA
ferretti@usc.edu, pabeerel@usc.edu

Abstract

This paper presents a new fast and templatized family of fine-grain asynchronous pipeline stages based on the single-track protocol. No explicit control wires are required outside of the datapath and the data is 1-of-N encoded. With a forward latency of 2 transitions and a cycle time of 6 for most configurations, the new family can run at 1.6 GHz using MOSIS TSMC 0.25 μm process. This is significantly faster than all known quasi-delay-insensitive templates and has less timing assumptions than the recently proposed ultra-high-speed GasP bundled-data circuits.

1 Introduction

As CMOS manufacturing technology scales into deep and ultra-deep sub-micron design, problems with clock skew, clock distribution, and on-chip communication in high-speed synchronous designs are becoming increasingly difficult to overcome [1], warranting the exploration of alternative design approaches. In particular, asynchronous design is emerging as an increasingly viable alternative.

Among the numerous asynchronous design styles being developed, template-based fine-grain pipelines have demonstrated very high performance [5][7][8][9]. Template-based approaches also have the advantage of removing the need for generating, optimizing, and verifying specifications for complex distributed controllers, which is both difficult and error-prone [2]. Various templates tradeoff latency, cycle time, and robustness to timing. The most robust is the quasi-delay-insensitive (QDI) templates proposed by Lines [5]. One of most aggressive is the ultra-high-speed GasP [7]. GasP offers high throughput but requires a bundled data design style that involves additional timing margins and assumptions that must be verified during physical design and that introduces higher latency through the data path than even the QDI templates, possibly yielding lower system performance.

The single-track full-buffer (STFB) templates proposed in this paper use 1-of-N data encoding and two-dimensional pipelining instead of single-rail encoding and fine-grain pipelining used by GasP. They have two key advantages. First, they remove the GasP bundling constraint, making them easier to design and verify. Second, they reduce forward latency by 58% at the cost of a 26% slower cycle time compared to GasP. The overall performance impact of this tradeoff depends on characteristics of the system. In particular, if the system is latency-critical, where the performance is determined by how fast an individual data token flows through the system, a STFB system can be significantly faster than the comparable GasP system despite having local cycle times that are somewhat larger.

The remainder of this paper is organized as follows. After Section 2 provides relevant background information, Sections 3 through 6 describes our proposed 1-of-N templates in detail. Latency and throughput simulation results of STFB buffers with both QDI and GasP buffers are compared in Section 7 followed by some conclusions drawn in Section 8.

2 Background

In the absence of the clock, providing global synchronization, masking logic hazards, and signaling the end of each computation step, asynchronous circuits operate using event-driven logic. In particular, asynchronous circuits are often decomposed into processing blocks that communicate data (called *tokens*) through asynchronous channels. This decomposition facilitates re-using asynchronous blocks and simplifies the design of complex systems.

2.1 Asynchronous channels

An asynchronous channel is a bundle of wires and a protocol to communicate data across the wires from a sender to a receiver. Figure 1 shows three different types of channels.

The bundled-data channel has the advantage that the data is single-rail encoded (the same used in synchronous design) but is dependent on the timing assumption that the data is valid when the request signal is asserted. The request signal is typically driven by a matched delay line that is larger than the sender's computation delay plus some margin.

Alternatively, in a 1-of-N channel, the token value is 1-of-N encoded, meaning that N wires are used to transmit N possible data values by asserting exactly one wire at a time. A *blank* or NULL data is encoded by de-asserting all wires. 1-of-2 (dual-rail) and 1-of-4 encodings are most common and both effectively use two wires per bit to encode the data.

In the 1-of-N channel, the receiver detects the presence of the token from the data itself and, once it no longer needs the data, acknowledges the sender. In the typical four-phase protocol, the sender then removes the data by resetting all wires and waits for the acknowledgement to be de-asserted before sending another token.

In the 1-of-N single-track channel, the receiver detects the presence of the token as in the 1-of-N channel but is also responsible for consuming it (by resetting all the wires). The sender detects that the token was consumed before sending another token.

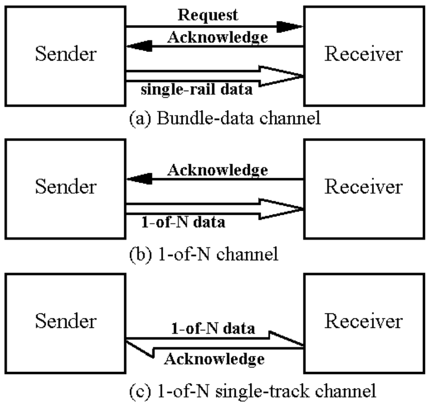


Figure 1. Asynchronous channels.

Berkel et al. [4] proposed single-track handshake circuits to control medium-grain bundled-data pipelines. Sutherland et al. [5] later developed faster single-rail GasP circuits to control fine-grain bundled-data pipelines. Nyström [8] recently also proposed a dual-rail (1-of-2) single-track template based on self-resetting pulsed-logic circuits like GasP but which requires significantly more transistors and is significantly slower.

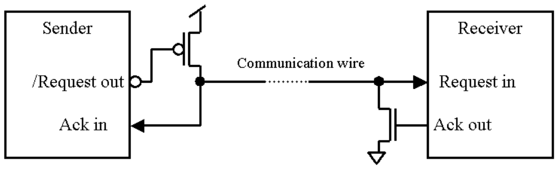


Figure 2. Single-track protocol typical connection.

Figure 2 illustrates a single-wire single-track channel. The sender waits for the wire to be low ("ready") before sending a request by driving the wire high ("busy"). After the receiver detects the wire is high and consumes the data, it drives the wire low.

Note that "transceivers" can also be implemented using the single-track wire to transport data in both directions if, for every communication event, it is well defined which block will send and which will receive [4]. Similarly, mutually exclusive transmitters and receivers may be connected to the same wire [4].

2.2 Weak-condition half-buffer (WCHB)

Figure 3 illustrates a well-known dual-rail buffer implementation called weak-condition half-buffer (WCHB) in [5]. L and R identify the left and right environments, 0 and 1 identify the zero and one rails, and "e" identifies the enable signals (high means "ready" and low means "acknowledge"). After reset, L0, L1, R0 and R1 are low while Le and Re are high. Data arrives by one of the left inputs (Lx) rising. This will cause Sx to go low, which will drive the corresponding output Rx high and the left enable Le low. The left environment then will lower Lx while the right environment receives the data Rx and lowers Re. The buffer then raises Le and lowers Rx. The cycle completes when the right environment re-asserts Re. Note that for clarity reset circuitry and staticizers are not typically shown. Note also that the generation and reset of the output token implies that the corresponding input token has been consumed and reset, respectively, a property called weak conditioned in [10] and weak indicatability in [11].

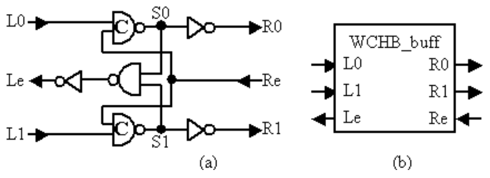


Figure 3. WCHB buffer.

We can derive an estimate of cycle time by counting the number of gate delays or *transitions* in a cycle of operation. The WCHB buffer is faster than other QDI buffers, having a cycle time of only 10 transitions. For more complex processing blocks with many inputs, however, WCHB is not recommended because it generally requires too many stacked PMOS transistors, making it slower than alternative templates.

2.3 GasP bundled data

Figure 4 shows the GasP circuit where, after reset, L, R, and A are high. When L is driven low by the left environment, the self-resetting NAND will fire, driving A low. This will restore L, activate the data latches, and drive R low, propagating the signal and avoiding re-evaluation until after R is restored high by the right environment. The self-resetting NAND will restore itself by driving A high after 3 transitions. The output of the NAND controls the latches in a parallel single-rail datapath.

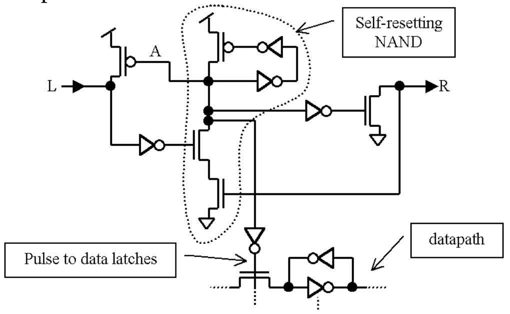


Figure 4. GasP diagram.

GasP circuits take 4 transitions to forward data and 2 transitions to reset, i.e., move a "bubble" or blank backwards. Of the 4 transitions forward latency, approximately two transitions are required for latency through the latches and satisfying setup/hold times leaving approximately two transitions for computation. Note that the control circuit itself makes up the delay line and that it is the datapath designer's responsibility to pipeline the datapath to match the control circuit delay while satisfying all setup/hold times.

2.4 Fine-grain vs. two dimensional pipelining

The WCHB and GasP templates represent a fundamental dichotomy in pipelining philosophy. The GasP design targets standard datapath widths of, for example, 32-bits. In fact, GasP circuits can be viewed as a complex method of distributing a clock that naturally facilitates gated clocking. Consequently, the bundled timing constraint captures many of the same problems as clock distribution and clock skew. The WCHB template, on the other hand, is generally applied to small datapaths, say 4 bits, and wider datapaths are made up of a two-dimensional array of communicating blocks [6]. The motivation of limiting individual QDI templates such as the WCHB to small datapaths is to keep the completion-sensing overhead to a minimum, thereby facilitating reasonable throughput while preserving robustness to timing. The completion of a wide datapath, if needed, can be pipelined across several pipeline stages using a technique called pipeline completion sensing [6]. Similarly, the broadcasting of a control signal affecting the entire datapath can be pipelined to avoid having a large completion tree for the acknowledgement signals. In this way, two-dimensional pipelines can have a cycle time that is independent of datapath width.

Moreover, the WCHB, along with other QDI templates, generally have significantly lower latency than their GasP template counterparts because they do not suffer from the latch delay and setup/hold times. Replicating the control circuits for each row (slice of bits) of the two-dimensional array, however, may result in increased area and power.

3 STFB buffers

In asynchronous design, buffers are used to balance pipelines for performance-driven *slack matching* [5] or simply storing data. Figure 5 illustrates our 1-of-N STFB buffer template.

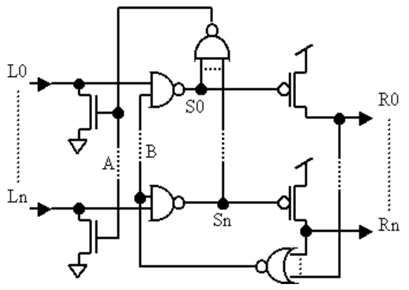


Figure 5. 1-of-N STFB buffer.

When one of the n inputs (Lx) is driven high by the left environment, the corresponding NAND gate will drive Sx low, thereby driving both the corresponding Rx and A (the "**A**cknowledgement" signal) high. A going high causes Lx to reset low, enabling the left environment to send a new token. Meanwhile, Rx going high causes the B ("**B**usy") signal to lower, restoring Sx high and preventing the NANDs to re-fire even if a new token arrives. The restoring of Sx, in turn, resets "A". The cycle completes when the right environment lowers Rx, resetting B low, and allowing a new data token to be processed. Since distinct tokens can simultaneously be at the left and right environments, the template is said to be a full buffer and have slack of 1.

The gate that drives "A" (Acknowledge) is called a state completion detector (SCD) because it detects that the internal state of the template has captured the input token. The gate that drives "B" is called a right completion detector (RCD) because it detects that the output token has been sent to the right environment. Note that the generation of the output token *indicates* [11][13] that the corresponding input token was valid and consumed. However, the reset of output tokens is caused by the right environment and does not indicate that the input tokens have reset. Consequently, we call the STFB buffer, along with most STFB logic templates, *semi-weak-conditioned*. As such, there is a timing assumption that the template must reset the input channel before "A" resets.

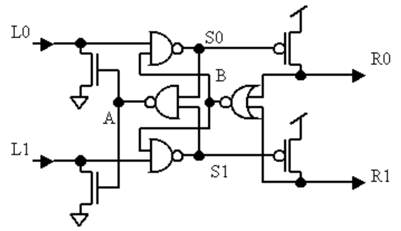


Figure 6. Dual-rail STFB buffer.

Figure 6 shows, as an example, a dual-rail STFB buffer. Figure 7 shows an optimized version in which the static NAND gates driving S0 and S1 are merged into one dual-rail dynamic gate that is reset only by the B signal. Figure 8 shows a similarly optimized 1-of-4 STFB buffer circuit and symbol.

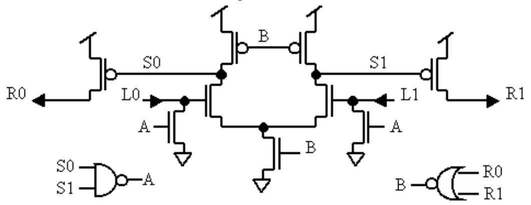


Figure 7. Optimized dual-rail STFB buffer.

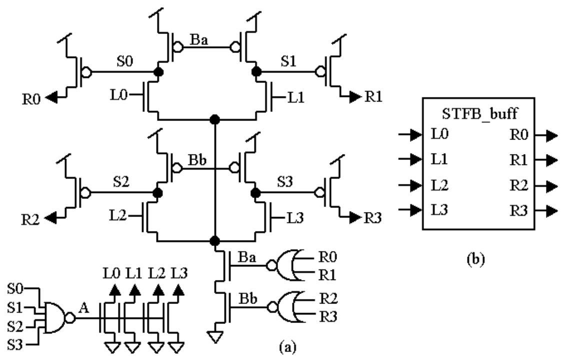


Figure 8. Optimized 1-of-4 STFB buffer.

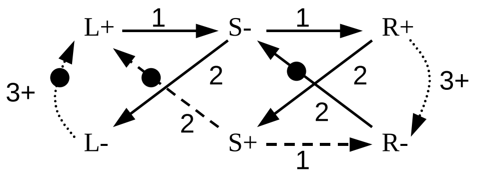
STFB buffers have an estimated cycle time of 6 transitions. This is 40% faster than WCHB and the same as GasP. The latency is 2 transitions, which is the same as WCHB but half that of GasP.

The STFB buffer, however, has higher complexity than both WCHB and GasP buffers. Compared to WCHB buffer, including required staticizers and reset circuit (Figure 20), the STFB buffer has 7 more transistors. This increased complexity, however, is mitigated by the fact that the proposed STFB buffer is a full buffer (i.e., has slack of 1), while WCHB is a half buffer (slack of ½). Moreover, the STFB buffer does not require the acknowledge wires (Le/Re), which may represent a significant saving in area and routing effort. In addition, the power consumption per communication of STFB buffer is potentially lower than WCHB buffer because each communication requires half the number of wire transitions.

Compared to a GasP buffer with a standard 32-bit datapath, the area and power consumption of a STFB pipeline may be higher because the two-dimensional STFB pipeline will be made up of many buffers in parallel and each buffer will have control circuit overhead.

Figure 9 shows the handshaking expansion (HSE) equation and the signal transition graph (STG) for the presented buffers. The notation "+", "↑" and "-", "↓" represent the rising and falling of the signals respectively. The left and right environments drive the dotted arrows and the dashed arrows represent timing constraints. The arrows are annotated with delays in terms of transitions.

$$\text{STFB buffer} \equiv *[[\neg R \wedge L \rightarrow R\uparrow];\ L\downarrow] \qquad (1)$$



(2)

Figure 9. STFB buffer: (1) HSE and (2) STG.

As can be deduced from the STG, the STFB buffer has somewhat tight timing constraints. In particular, the timing margin between the tri-stating of an output wire (one transition after S+) and the earliest time the environment can reset the wire (R-) is zero. Moreover, the timing margin between tri-stating of an input wire (two transitions after S+) and the earliest time the left environment can drive the wire (L+) is also zero. In particular, if these margins are violated, significant short circuit current may occur during the transitioning of the line. In addition, it is assumed that three transitions are sufficient to fully discharge/charge a line. This poses significant constraints on both transistor sizing and post-layout analog verification, but efforts to solve these problems are on going [5]. Unless otherwise noted, these timing constraints apply to all subsequent examples.

4 STFB forks and joins

This section covers a variety of non-linear pipelines stages that involve multiple input and/or multiple output channels and can perform more complex logic functions. While we focus on two dual-rail (1-of-2) inputs/outputs, templates that handle more channels and/or 1-of-N encoding are natural extensions.

4.1 Dual-rail STFB semi-weak-cond. AND

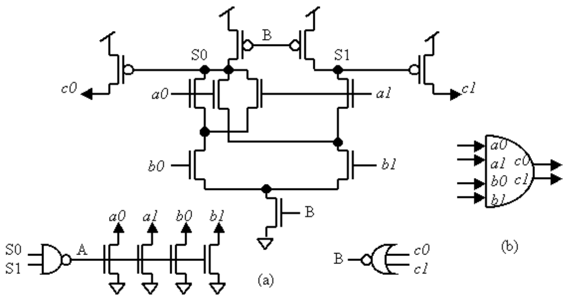


Figure 10. SFTB semi-weak-conditioned AND.

Figure 10 illustrates an STFB AND stage that performs *c* = *a*b*, where *a* and *b* are dual-rail single-track inputs and *c* is the dual-rail single-track output.

All the inputs are "acknowledged" by the signal A as soon as S0 or S1 goes low. For S1, this happens when *a1* and *b1* are high. For S0, *a0* or *b0* driven low is sufficient to define the logic result, but the circuit explicitly waits for one of the three input combinations 00, 01, and 10 to arrive before lowering S0. In this way, the evaluation of S0 implies that both tokens (*a* and *b*) arrived, guaranteeing that the acknowledgement does not precede the arrival of a late token, making this gate semi-weak-conditioned.

4.2 Dual-rail STFB non weak-conditioned AND

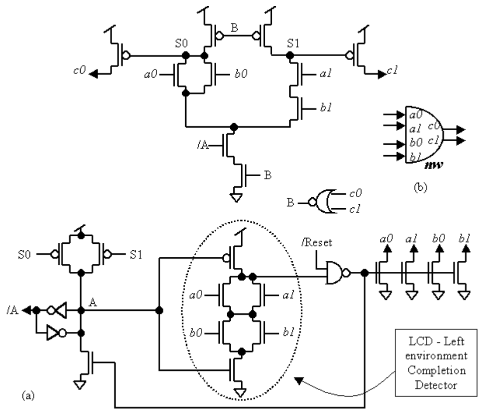


Figure 11. Non weak-conditioned STFB AND.

Figure 11 shows a non weak-conditioned AND. This circuit generates a zero result token as soon as one of the inputs is zero even if the other input has not arrived. When all the inputs are finally present, however, the stage sends an acknowledgement to all inputs.

To do this, while forwarding the early zero result, the gate's SCD (State Completion Detector) sets A high, which will disable the logic for future evaluations by keeping /A low and will hold the information that an acknowledge is pending. When the LCD (Left environment Completion Detector) detects that all input tokens are present, the acknowledge signal is passed to the transistors that will "consume" the data at the inputs and A is reset to zero. This will restore /A high and the gate will be ready to evaluate again. This LCD structure adds two transitions to the cycle time but loosens the timing margin between S- and resetting the inputs (corresponding to L- in Figure 9) by two gate delays.

Notice that for multiple inputs, this gate has a much simpler NMOS transistor stack than the weak-conditioned STFB AND.

4.3 Dual-rail STFB OR and STFB XORs

A dual-rail STFB OR performs the logic operation: *c* = *a*+*b*, where *a* and *b* are dual-rail single-track inputs and *c* is the dual-rail single-track output. This function can be implemented either with semi-weak-conditioned logic or with non-weak-conditioned logic simply by rearranging the transistors in the NMOS stack of the AND circuits presented in Section 4.1 and 4.2 .

Similarly, the dual-rail STFB XOR performs the logic operation: $c = a \oplus b$, where *a* and *b* are dual-rail single-track inputs and *c* is the dual-rail single-track output. The STFB XOR, however, must be semi-weak-conditioned, because all input token values must be known before the output value is known.

4.4 Dual-rail STFB non-conditional merge

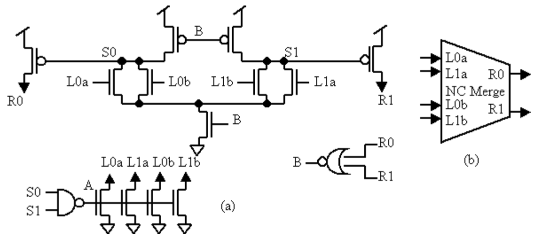


Figure 12. STFB NCMerge.

The non-conditional merge operation concatenates the incoming data from different mutually exclusive input channels. Figure 12 shows a 2-to-1 non-conditional merge circuit and symbol.

4.5 Dual-rail STFB copy

The copy operation consists of replicating the incoming data to several different paths if all output paths are ready. Otherwise, the input data must wait.

Figure 13 shows the 1-to-2 copy. Notice that the four-input NOR gate (with a stack of four PMOS transistors) driving B slows down the STFB Copy performance. To speed-up the B signal, however, we can use 2 two-input NOR gates to generate Ba and Bb and replace the B NMOS transistors with stacked Ba and Bb NMOS transistors (see Figure 8).

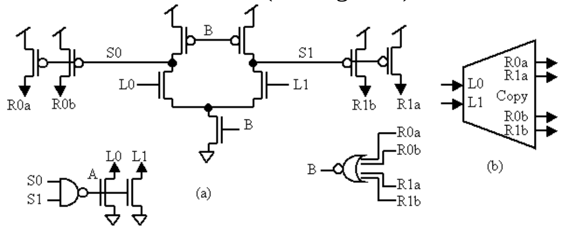


Figure 13. STFB copy.

4.6 Dual-rail STFB full adder

To implement a full adder (STFB FA) we need to compute the sum and the carry out before resetting the inputs. As illustrated in Figure 14 and Figure 15, this can be done with a three-input XOR and a three input majority (MAJ) gate. The XOR generates the sum (*s*=*a*+*b*+*ci*) and the MAJ generates the carry out (*co*=MAJ(*a*,*b*,*ci*)).

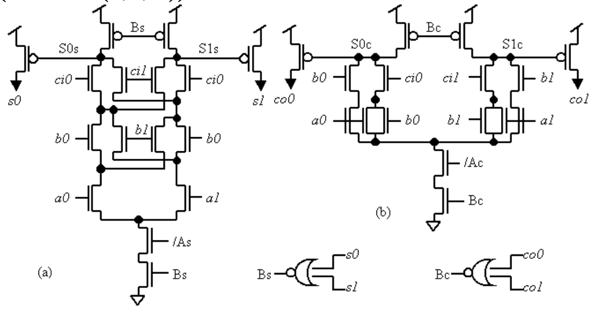


Figure 14. STFB FA: (a) XOR and (b) majority gates.

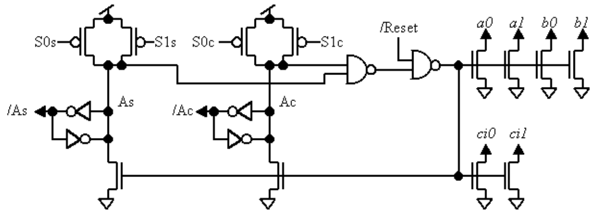


Figure 15. STFB FA acknowledgement circuit.

In this structure, the carry evaluates as soon as enough inputs arrive to define the correct output value but the acknowledgement waits for both outputs to be generated which, because the sum is an XOR gate, implicitly means that all inputs have arrived. Note that the acknowledgement circuitry adds two gate delays to the cycle time but also loosens the timing margin between S- and resetting the inputs by two gates.

The long nmos stacks in the sum and carry circuits can be reduced by one transistor by removing the transistors controlled by /As and /Ac and making As and Ac new inputs of their respective RCD NOR gates.

5 STFB conditional stages

This section covers a variety of stages in which input and/or output channels are conditionally read or written.

5.1 Dual-rail STFB split

The split operation consists of forwarding incoming tokens to one of two output channels based on the value of a control (C) channel. If the chosen output path is busy, the data must wait. Note that the micropipeline version of this block, which samples the control signal rather than consuming it, is called a *select* [12].

Figure 16 shows the 1-to-2 STFB split circuit and symbol. In this example, when C is low, L is directed to Ra and, when C is high, to Rb. Interestingly, the STFB split allows a token to be forwarded to one channel even if the other channel is busy, which increases the degree of parallelism.

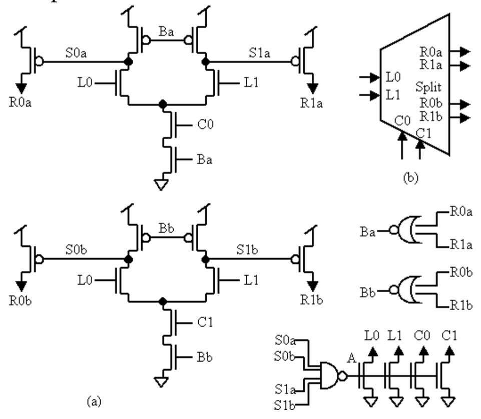


Figure 16. STFB split.

5.2 Dual-rail STFB merge

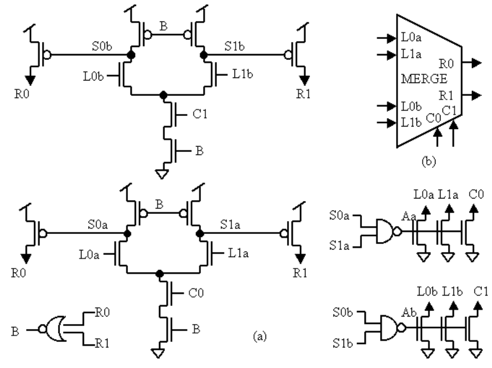


Figure 17. STFB Merge.

The merge operation consists of choosing one of the incoming tokens based on the value of a control (C) input. If the output path is busy, the input and control must wait. After forwarding the data, the control token is also consumed.

Figure 17 shows the 2-to-1 merge circuit and symbol. When C=0, La is directed to R and, when C=1, Lb is directed to R.

6 Auxiliary stages

This section covers bit generators used to generate a stream of tokens, bit buckets to consume unwanted tokens, converters between single-track and four-phase protocols, and staticizer/reset circuitry.

6.1 Four-phase to STFB converters

The "transmitter" circuit, illustrated in Figure 18, is our proposed interface between four-phase asynchronous logic and STFB. In this circuit, if Le is high and the right environment is ready, a data from the left environment will be transmitted and the signal Le will be set low. This also disables the buffer, avoiding re-transmitting the same data after the right environment consumes it. Le will remain low until both inputs return to zero (four-phase protocol). When this happens, Le is set high and the transmitter is ready for the next data.

The "receiver" circuit, illustrated in Figure 19, is our proposed interface between STFB and four-phase asynchronous logic. In this circuit, if Re is high (the right environment is ready), a data from the left environment will be received and the buffer will wait for the signal Re to be set low. When Re goes low, a three gate-delay pulse is generated to consume the left environment data and the receiver is reset (R0 and R1 goes low). While Re is low, R0 and R1 are reset and no new data is received (four-phase protocol). When Re returns to high, the receiver is ready for the next data.

The cycle time of these converters is 9 transitions when connected to WCHB buffers.

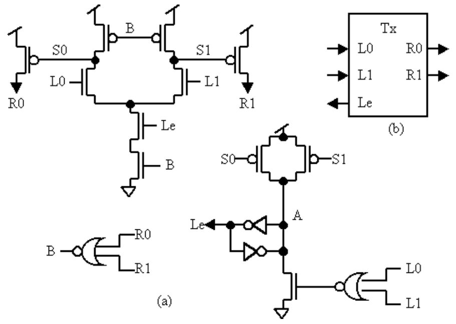


Figure 18. STFB Tx

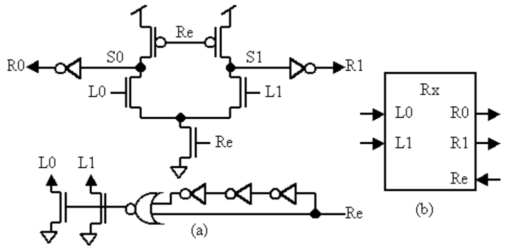


Figure 19. STFB Rx

6.2 Staticizers and reset circuitry

As shown in Figure 20, in order to preserve the state of the circuit during long periods of inactivity, weak PMOS transistors (M1 and M2) may be added to hold S0 and S1 high despite leakage currents. Moreover, staticizers (implemented with two inverters) may be placed at each output wire to mitigate both leakage current and cross-talk. Also, to further reduce crosstalk, the transistors M3 and M4 can be added to help hold one wire low while the other is driven high.

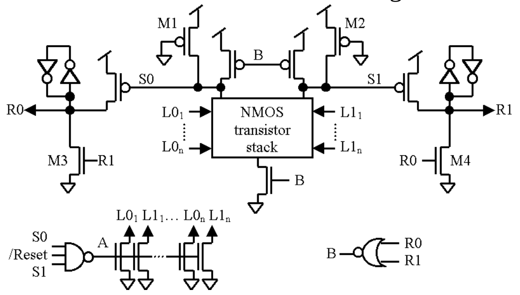


Figure 20. STFB staticizer and reset circuitry.

The reset procedure is simple: after power up, if any output wire remains high, the respective B signal will automatically disable the PMOS drivers. Then, lowering the /Reset signal will force the "A" signal high resetting all the wires.

For simplicity, the reset input, the staticizers, M1, M2, M3 and M4 have typically not been illustrated but should be assumed present.

6.3 Dual-rail STFB bit generators and buckets

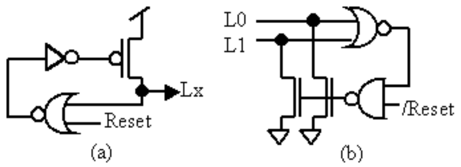


Figure 21. STFB bit generator (a) and bucket (b).

A bit generator creates a data token every time the line is empty, while a bit bucket consumes unwanted tokens. Both are also useful in test circuitry. The proposed STFB bit generator and bit bucket are shown in Figure 21.

7 Simulation results

The MOSIS TSMC 0.25µm (Vdd = 2.5V and T = 25°C) process parameters were used for the following hspice simulation results.

The transistor sizing strategy we adopted to compare STFB, WCHB, and GasP buffers was very simple. The smallest widths were Wn = 0.6 µm and Wp = 1.4 µm for the NMOS and PMOS transistors respectively. The output (line-drivers) transistors where designed two times the smallest size. The width of each stacked transistor was the smallest size multiplied by the number of transistors in the stack. To operate properly, the PMOS of the GasP self-resetting NAND needed to be increased to Wp = 2.8 µm and the NOR used in the STFB buffer was implemented with Wn = 0.9 µm. Different transistor sizing may yield better performance, but a significant change in the relation between the three approaches is not expected.

We simulated three different pipelines implemented with 10 buffer stages, a bit generator at the source, and a bit bucket at the sink, using dual-rail STFB and WCHB buffers and 1-bit GasP control. Table 1 compares their throughput and latency per stage. The single-rail GasP does achieve the highest throughput but at the expense of significantly larger latency. Compared to WCHB, the proposed STFB achieves both faster latency and throughput. We also simulated the STFB pipeline where each stage was separated by 0.5 mm length of wires. The pipeline worked correctly, demonstrating robustness to wire load, but at a reduced throughput of 1.1 GHz.

Table 1. Buffers throughput and latency

Buffer	Number of transitions			Throughput	Latency
	Forward	Backward	Cycle		
STFB	2	4	6	1.6 GHz	165 ps
GasP	4	2	6	2.2 GHz	261 ps
WCHB	2	3	10	1.0 GHz	210 ps

Non-linear STFB pipeline templates have similar cycle times. Unfortunately, we could not perform detailed hspice simulation comparisons of non-linear pipeline stages because GasP implementations for many of these stages were not presented in [7]. However, we believe we would see a similar throughput-latency tradeoff as seen in the buffers. In addition, since the fastest QDI templates for these non-linear structures are significantly slower than WCHB buffers, we expect the advantage of our non-linear templates over QDI is even more significant.

8 Conclusions

STFB templates were proposed for high-speed asynchronous non-linear pipeline design. The templates use 1-of-N data encoding rather than single-rail encoding to avoid the bundling constraint that is hard to design and verify. The templates have lower latency and 60% higher throughput than the fastest known QDI templates and have 58% lower latency than the most aggressive GasP templates. Consequently, for systems that are latency-critical, STFB templates may yield a significant performance advantage. Quantification of this advantage is part of our future work. We also plan to formally verify the presented timing constraints.

References

[1] W. J. Dally and J. Poulton, *Digital Systems Engineering*, Cambridge Univ. Press, Cambridge, UK, 1998

[2] K. Y. Yun, P. A. Beerel, V. Vakilotojar, A. Dooply, and J. Arceo, “The Design and Verification of a Low-Control-Overhead Asynchronous Differential Equation Solver”, IEEE Transactions on VLSI, Dec. 1998

[3] A. Davis and S. M. Nowick, “An Introduction to Asynchronous Design”, Univ. of Utah Tech. Rep., Dept. of Computer Science, UUCS-97-013, Sept. 19, 1997.

[4] K. van Berkel, and A. Bink,, “Single-Track Handshake Signaling with Application to Micropipelines and Handshake Circuits”, Proc. ASYNC, pp: 122–133, 1996.

[5] A. M. Lines, “Pipelined Asynchronous Circuits”, Master Thesis, California Institute of Technology, June 1998.

[6] A. J. Martin, A. Lines, R. Manohar, M. Nyström, P. Penzes, R. Southworth, U. Cummings, and T. K. Lee, “The Design of an Asynchronous MIPS R3000 Microprocessor.” Proc.of ARVLSI, pp. 164-181, 1997.

[7] I. Sutherland and S. Fairbanks, “GasP: A Minimal FIFO Control”, Proc. of ASYNC, pp: 46 – 53, 2001.

[8] M. Nyström, “Asynchronous Pulse Logic”, PhD Thesis, California Institute of Technology, May 14, 2001.

[9] M. Singh and S. M. Nowick, “High-Throughput Asynchronous Pipelines for Fine-Grain Dynamic Datapaths”, Proc. of ASYNC, pp: 198 – 209, 2000.

[10]C.L. Seitz. “System timing,” in C. A. Mead and L.A. Conway, editors, *Introduction to VLSI Systems*, chapter 7. Addison-Wesley, 1980.

[11]C.D. Nielsen. “Evaluation of Function Blocks for Asynchronous Design,” Proceedings of ACM, pp:454-459, September 1994.

[12]I.E. Sutherland, “Micropipelines”, Communications of the ACM, vol. 32, #6, pp: 720-738, June 1989.

[13]V.I. Varshavsky (editor), *Self-Timed Control of Concurrent Processes : The Design of Aperiodic Logical Circuits in Computers and Discrete Systems*, Kluwer Academic Publishers, Dordrecht, The Netherlands, January 1990.