High-Speed QDI Asynchronous Pipelines

Recep O. Ozdag
EE Dept. – Systems Division, USC
Los Angeles, CA 90089
ozdag@usc.edu

Peter A. Beerel
EE Dept. – Systems Division, USC
Los Angeles, CA 90089
pabeerel@usc.edu

Abstract

This paper introduces two new high-speed quasi delay insensitive (QDI) asynchronous pipeline templates. These new high throughput templates support complex non-linear pipeline structures and are well suited for fine-grain pipelining. Timing analysis and HSPICE simulations show that these templates are 20% and 40% faster than known QDI counterparts.

1. Introduction

Asynchronous design is increasingly becoming an attractive alternative to synchronous design because of its potential for high-speed, low-power, reduced electromagnetic interference, and faster time to market [1], [2], [3]. To support these design efforts, numerous design styles and supporting CAD tools have been developed. For low-speed low-power applications, Phillips Research has developed a comprehensive CAD-supported design flow based on a high-level language called Tangram [4]. University of Manchester has developed a similar system based on Balsa [5] to support low-power medium-speed applications including embedded microprocessors. For high-speed applications, numerous fine-grain pipelining techniques have been developed [6], [7], [8], [9], [10], [11], [12]. Most of these pipeline strategies, however, achieve high-performance at the cost of introducing numerous types of timing assumptions, ranging from ultra-aggressive to easily-satisfiable, that complicate the design process. In particular, these timing assumptions must be accounted for during floorplanning, placement, transistor sizing and verified pre- and post-layout. In fact, the more aggressive timing assumptions are expected to become increasingly difficult to satisfy in ultra-deep-submicron design because of increased variance in nominal delays.

The design methodology proposed by Caltech is perhaps the most robust using delay-insensitive communication between quasi delay insensitive (QDI) pipeline templates [11] [14]. Although focused on ease of design using quasi delay insensitive templates, this methodology achieves reasonable performance through fine-grain pipelining and parallelism [13].

This paper proposes two new templates that provide significant performance improvements over those proposed by Caltech without sacrificing quasi delay insensitivity. The key idea is to reduce the complexity of internal circuitry by intelligently reducing concurrency and using an additional wire for communication between pipeline stages. We propose two templates: one that is a half-buffer which requires two pipeline stages to hold one data token and one full-buffer template that can itself hold one data token.

The remainder of this paper is organized as follows. Section 2 gives background on asynchronous pipelines, reviews several existing QDI pipeline templates, and discusses several issues associated with non-linear pipelines. Sections 3 and 4 present the new templates in detail, including the protocol, implementation, and timing analysis. Finally, experimental results, comparisons, and conclusions are given in Sections 5 and 6.

2. Background

This section first gives background on commonly used asynchronous data representation schemes. Then, it reviews three asynchronous pipelining styles: Caltech's Weak-Conditioned Half Buffer (*WCHB*), Precharged Half Buffer (*PCHB*), and Precharged Full Buffer (*PCFB*) templates [11]. Lastly, issues with non-linear pipelines and the role of input completion sensing are reviewed.

2.1 Data Representation Schemes

An asynchronous communication channel is a bundle of wires and a protocol to communicate data between a sender and a receiver. The encoding scheme in which one wire per bit is used to transmit the data and an associated request line is sent to identify when data is valid is called *single-rail encoding* and is shown in Figure 1. The associated channel is called a bundled-data channel. Alternatively, if the data is sent using two wires for each bit

Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems (ASYNC'02)
1522-8681/02 $17.00 © 2002 **IEEE**

IEEE COMPUTER SOCIETY

of information, the encoding is called a *dual-rail channel*. Extensions to 1-of-N encoding also exist.

Both single-rail and dual-rail encoding schemes are commonly used, and there are tradeoffs between each. Dual-rail and 1-of-N encodings allow for data validity to be indicated by the data itself and are often used in QDI designs. Single-rail, in contrast, requires the associated request line, driven by a matched delay line, to always be longer than the computation. This latter approach requires careful timing analysis but allows the reuse of synchronous single-rail logic.

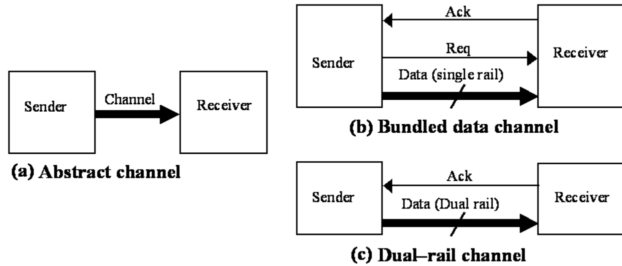


Figure 1. Pipeline Channels

2.2 WCHB

Figure 2 shows a WCHB template for a linear pipeline with a left (L) and right (R) channel and an optimized WCHB dual-rail buffer. L0 and L1, R0 and R1 identify the false and true dual rail inputs and outputs, respectively. *Lack* and *Rack* are active-low acknowledgment signals. Note that we do not show staticizers that are required to hold state at the output of all C-elements.

The operation of the buffer is as follows. After the buffer has been reset, all data lines are low and acknowledgment lines, *Lack* and *Rack,* are high. When data arrives by one of the input rails going high, the corresponding C-element output will go low, lowering the left-side acknowledgment *Lack*. After the data is propagated to the outputs through one of the inverters, the right environment will assert *Rack* low, acknowledging that the data has been received. Once the input data resets, the template raises *Lack* and resets the output.

Since the L and R channels cannot simultaneously hold two distinct data tokens, this circuit is said to be a *half buffer* or has *slack* ½ [11]. This WCHB buffer has a cycle time of 10 transitions, which is significantly faster than buffers based on other QDI pipeline templates.

Another feature of the WCHB template is that the validity and neutrality of the output data R implies the validity and neutrality of the corresponding input data L. This is called *weak-conditioned* logic [15] and is the same logic used in fine-grain pipelines proposed by Theseus logic [16]. We will discuss its advantages and disadvantages after we discuss non-linear pipeline templates.

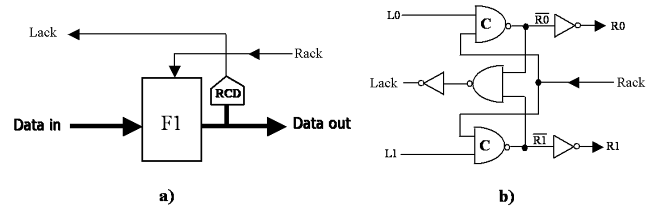


Figure 2. WCHB

2.3 PCHB and PCFB

Figure 3(a) shows the template for a pre-charged half-buffer (PCHB). Unlike the WCHB, the test for validity and neutrality is checked using an input completion detector. The input completion detector is denoted as LCD and the output completion detector as RCD.

The function block need not be weak-conditioned logic and thus can evaluate before all the inputs have arrived (if the logic allows). However, the template only generates an acknowledgment signal *Lack* after all the inputs have arrived *and* the output has evaluated. In particular, the LCD and the RCD are combined using a C-element to generate the acknowledgment signal.

A few minor aspects of this template should also be pointed out. First, because the C-element is inverting the acknowledgment signal is an active-low signal. Second, the *Lack* signal is often buffered using two inverters before being sent out. Another two inverters are also often added to buffer the internal signal *en* that controls the function block. For simplicity, these buffering inverters will not be shown in the figures in this paper.

The protocol for a PCHB pipeline stage is captured by the STG [17] for a three-stage pipeline illustrated in Figure 4(a). From the STG, it is possible to derive the pipeline's analytical cycle time:

$$T_{PCHB} = 3.\ t_{Eval} + 2.\ t_{CD} + 2.\ t_c + t_{prech}$$

Due to the extra buffering and bubble shuffling, the cycle time generally amounts to 14 gate delays or *transitions*.

The PCFB template and its STG are shown in Figure 3(b) and Figure 4(b). The PCFB is more concurrent than the PCHB because its L and R handshakes reset in parallel at the cost of requiring an additional state variable. The PCFB analytical cycle time is:

$$T_{PCFB} = 2.\ t_{Eval} + 2.\ t_{CD} + 2.\ t_c + t_{prech}$$

which generally amounts to 12 transitions. Here t_{CD} takes two transitions, one of the C-elements takes one transition, and the other takes two transitions.

2.4 Non-Linear Pipeline Structures

Recently many new asynchronous pipelines have been introduced. However most of them have been targeted for linear pipeline applications such as FIFOs. Real designs, however, require more complicated non-linear pipeline structures. In particular, linear pipeline stages have only a

1522-8681/02 $17.00 © 2002 **IEEE**

IEEE COMPUTER SOCIETY

single input and a single output channel, where as non-linear pipelines stages can have multiple input and output channels. The QDI templates are easily extended to non-linear pipelines and now we review the underlying issues.

To introduce these issues we focus on *forks* and *joins*. A join is a pipeline stage with multiple input channels whose data is merged into a single output channel. A fork is a pipeline stage with one input channel and multiple output channels. Complex forks and joins can involve conditionally reading from or writing to channels based on the value of a control channel that is unconditionally read, as in a merge or split channel. Abstract illustrations of these channels are shown in Figure 5.

Since a fork has multiple output channels, it must receive an acknowledgment signal from all of them before it precharges. A join, on the other hand, receives inputs from multiple channels and must broadcast its acknowledgment signal to all its input stages.

A join acts as a synchronization point for data tokens. The acknowledgment from the join should only be generated when all the input data has arrived. Otherwise a stage feeding a join, referred to as A, that is particularly slow in generating its data token may receive an acknowledgment signal when it should not, violating the 4-phase protocol. If the acknowledgment signal is deasserted before the slow stage A generates its token, the token is not consumed by the join, as it should be. In fact, this token may cause the join to generate an extra token at its output, thereby corrupting the intended synchronization.

A conditional split is a combined fork and join where a control channel is used to determine which output is generated. The control may indicate to send the input data to any of the output channels, any combination of the output channels, or none of them. The third option is also known as a *skip*.

A conditional join is a join where the control signal, select, comes from another pipeline stage. The select signal controls which incoming channel should be read.

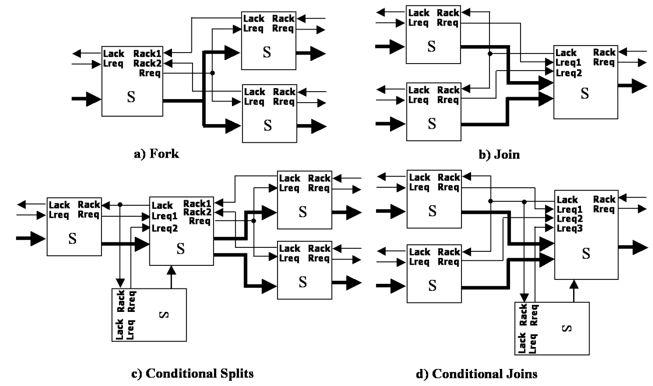


Figure 5. Non-linear pipelines.

2.5 Why Input Completion Sensing?

As mentioned above, validity should be checked on all input channels before the acknowledgment signal is asserted to prevent the incorrect insertion of a token caused by a slow/late input channel. Neutrality should be checked to guarantee that the previous stages have been precharged, so that the acknowledgment signal is not deasserted too early, thereby violating the four-phase protocol on any stage slow to precharge.

The templates presented in this section check validity and neutrality in different ways. Because the function block in WCHB template is weak-conditioned, the output completion detector implicitly checks validity and neutrality of the input data token. In the WCHB buffer the weak conditioned function block is a simple C-element. However, for more complex non-linear pipelines, weak-conditioned function blocks unfortunately require complex NMOS and PMOS networks. This results in slower forward latency and bigger transistor sizes. As an example, a weak-conditioned dual-rail OR is shown in Figure 6.

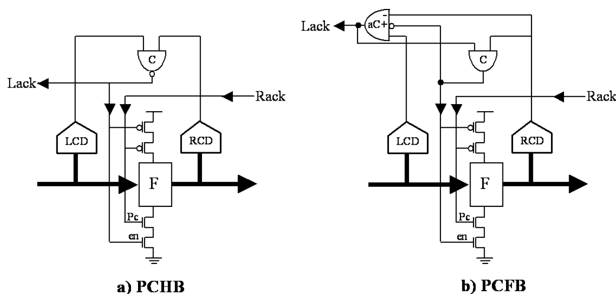


Figure 3. a) PCHB and b) PCFB templates

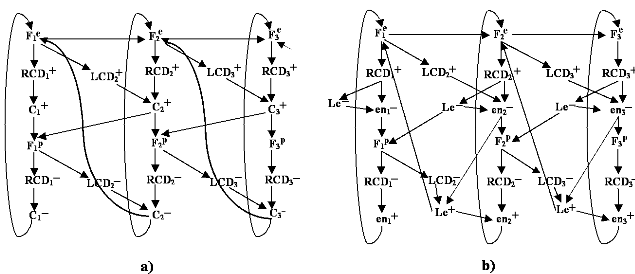


Figure 4. a) PCHB and b) PCFB STG

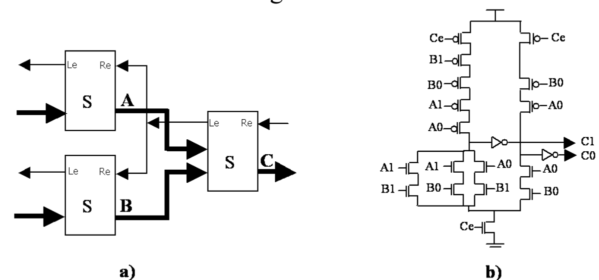


Figure 6. An OR gate implementation using weak conditioned logic

Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems (ASYNC'02)
1522-8681/02 $17.00 © 2002 IEEE
IEEE COMPUTER SOCIETY

In contrast, the PCHB and PCFB check the validity and neutrality of the input data explicitly with a distinct input completion detector. This enables the function block to use smaller and faster pre-charge logic function blocks. The cost of this alternative is that it requires more transistors and has more transitions per cycle.

3. New QDI Templates

One optimization that can be applied to the PCHB and PCFB templates is to merge the LCD of one stage with the RCD of the other by adding an additional request line to the channel. This is shown in Figure 7 for a PCHB template.

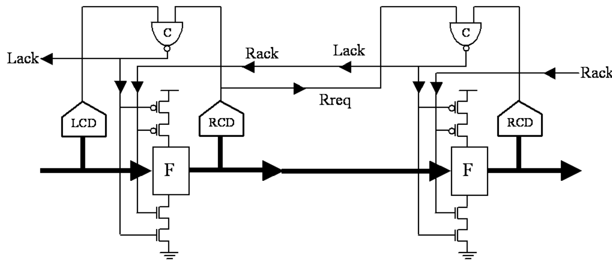


Figure 7. Optimized PCHB for a 1-of-N+1 channel

The request line indicates the assertion/deassertion of the input data, as in the bundled-data channel. However in contrast to a bundled-data channel, the data is sent using 1-of-N encoding, yielding what we call a *1-of-N+1 channel*. The request line, at least from the channel point of view, may appear redundant. However, the request line enables the removal of the input completion detector thereby saving area and reducing capacitance on the data lines. Moreover, the request line does not significantly impact performance, the template is still QDI, and the communication between stages remains delay-insensitive.

In this section we propose two new 1-of-N+1 QDI templates that intelligently reduce concurrency to reduce the stack size of the function blocks and thereby improve performance.

3.1 RSPCHB

We propose a new pipeline template that eliminates the need of the internal *en* signal of the PCHB template, thereby reducing the transistor stack sizes in the function block. We refer to this new QDI pipeline template, illustrated in Figure 8(a), as a *Reduced Stack Precharged Half Buffer* (RSPCHB). A specific form of this template for dual-rail data is shown in Figure 8(b). Notice that we optimized the RCD block by tapping its inputs before the output inverter and using a NAND gate instead of an OR gate.

The RSPCHB facilitates the removal of the internal enable signal by reducing concurrency that effectively does not improve performance. More specifically, in the PCHB template the output of the LCD and RCD are combined using a C-element to generate the acknowledgment signal *Lack*. This supports the integration of the handshaking protocol with the validity and neutrality of both input and output data, which removes the need for the function block to be weak-conditioned, but also requires the use of the *en* signal. It is this replacement however that introduces more concurrency than is necessary.

In particular, in the case of a join, the non-weak-conditioned function block may generate an output as soon as one of the input channels provides data. In response, the RCD of the join will assert its output. Meanwhile, any subsequent stage can receive this data, evaluate, assert both its LCD and RCD outputs, and assert its acknowledgment signal. Although the join can receive this acknowledgment, it will not precharge until after *en* is asserted. The *en* signal delays the precharge of the circuit until after the acknowledgement to the input stages has been asserted. This delay is critical to prevent the precharge from triggering the RCD to deassert which would prevent the C-element from ever generating the acknowledgment.

If only the generation of the acknowledgment signal from any stage subsequent to the join was delayed until all input data to the join has arrived and been acknowledged, then the *en* signal could be safely removed. In fact, such a delay of the acknowledgement would not generally impact performance because the join is the performance bottleneck for the subsequent stages. Therefore, this added concurrency is essentially unnecessary.

The unique feature of the RSPCHB is that it derives the request line from the output of the C-element instead of the RCD. (In particular, since the output of the C-element is active low and the request line is active high, the output of the C-element is sent through an inverter before driving *Rreq*.) The impact of this change is that the assertion/deassertion of *Rreq* is delayed until after all *Lreq*'s are asserted/deasserted. As a consequence, the acknowledgment from a subsequent stage of the join may be delayed until well after its data inputs and outputs are valid. More specifically, the stage will delay the assertion of its acknowledgment signal until all *Lreq*'s are asserted which can occur arbitrarily later than the associated data lines becoming valid. This extra delay, however, has no impact on steady-state system performance because the join stage is the bottleneck, waiting for all its inputs to arrive before generating its acknowledgement. In fact, this change yields a template with no less concurrency than WCHB.

The advantage of this generation of the request line is that the function block does not need to be guarded by the enable signal. In particular, it is now sufficient to guard the function block solely by the *Pc* signal because the *Pc* signal now properly identifies when inputs and outputs are valid. Namely, the function block is allowed to evaluate when *Pc* is deasserted which occurs only after all inputs and outputs data lines are reset. Similarly, it is allowed to precharge

when *Pc* is asserted which occurs only after all input and output data lines are valid.

The RSPCHB, depicted in Figure 8a and 8b is still QDI, however, the communications along the input channels become QDI instead of delay-insensitive. In particular, the relative timing assumption that must be satisfied is that the data should reset before *Rack* is de-asserted. This timing assumption is necessary to prevent re-evaluation of the block with old data. If we assert that the fork between the function block, the RCD, and the next stage is *isochronic* [18], this assumption is satisfied. In particular, the data line at the receiver side is then guaranteed to reset before the request line *Rreq* resets because only after the data lines reset can the RCD trigger the C-element, subsequently triggering *Rreq*. The analytical expression for the timing margin associated with this isochronic fork assumption can be derived from the abstract STG of the RSPCHB shown in Figure 9. In particular, the delay difference between the resetting of the data and the associated request line should be less than:

$$T_{Margin} = 2.\ t_{Inv} + 1.\ t_{CD} + 3.\ t_c$$

This margin is between 6 and 8 gate delays depending on buffering and is easily satisfied with modern routers.

As an alternative, the completion sensing circuitry can also be tapped before the output inverters, as shown in Figure 8c, reducing the output capacitance and enabling the use of a NAND gate rather than a NOR gate for completion sensing. Interestingly, this optimization preserves QDI properties when applied to the original PCHB template. In our design, however, because of the 1-of-N+1 channel, this optimization breaks QDI. In particular, the design will fail if the output buffers are tremendously slow in comparison with the associated request line. The same relative timing constraint that *Rack* is de-asserted after the dta is reset still guarantee correct operation. The associated timing margin, however, is slightly reduced.

The analytical cycle time of the RSPCHB can be derived from the STG shown in Figure 9 as:

$$T_{RSPCHB} = Max(\ 3.\ t_{Eval} + 2.\ t_{CD} + 2.\ t_c + \ t_{prech},\ t_{Eval} + 2.\ t_{CD} + 4.\ t_c + \ t_{prech})$$

With bubble shuffling, RSPCHB and PCHB have equal numbers of transitions per cycle. The advantage of RSPCHB is that the lack of an LCD and reduced stack size of the function block, which reduces capacitive load, and yields significantly faster overall performance. The cost of this increase in performance is that it requires one extra communicating wire between stages.

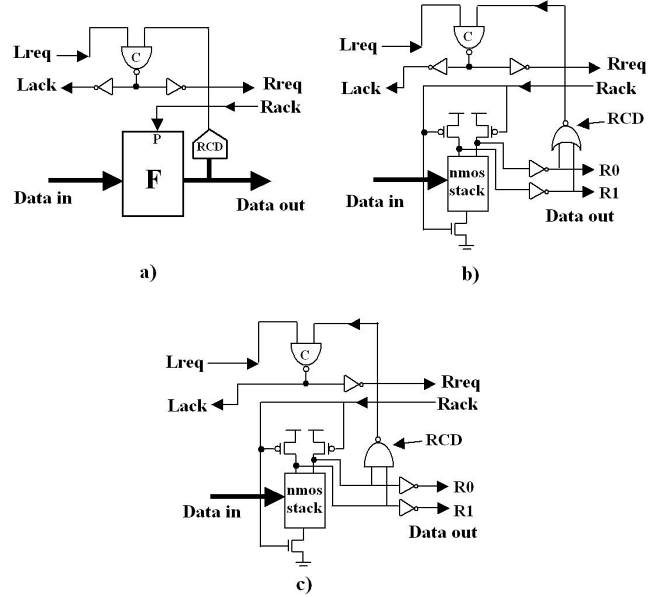


Figure 8. a) Abstract and b) detailed QDI RSPCHB pipeline template. c) Non-QDI optimized alternative.

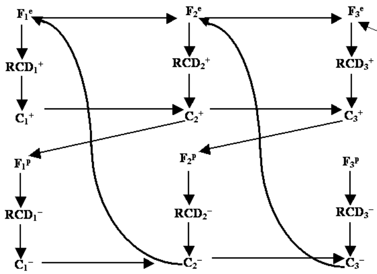
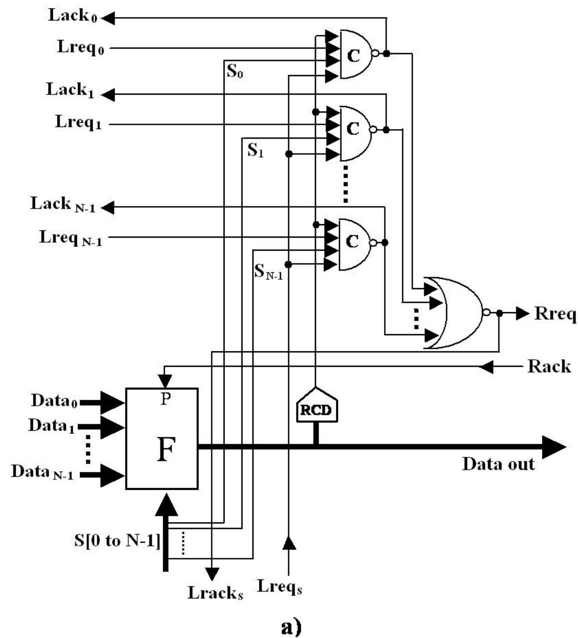


Figure 9. The STG of the RSPCHB

Note that there is a striking similarity between the control of our base RSPCHB and the basic micropipeline structure proposed by Sutherland [19]. Both pipelines use a C-element and a single inverter as the control. However, our proposed template uses 1-of-N signalling and output completion sensing instead of matched delay lines to generate the done signal. In addition, our templates use a four-phase handshaking protocol instead of a two-phase protocol.

A fork can be implemented easily by either using a C-element to combine the acknowledgment signals from the forking stages or by combining them by increasing the stack size of the function block. Similarly a join can be implemented, by combing the request lines in the C-element and forking back the acknowledgment signal.

IEEE COMPUTER SOCIETY



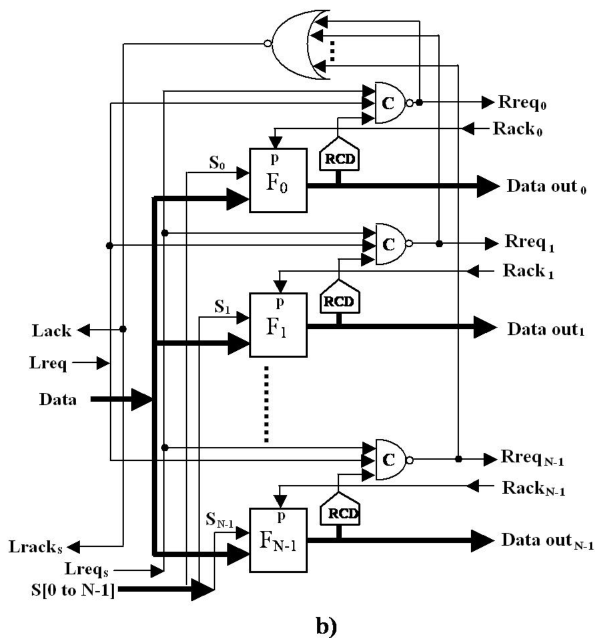


Figure 10. Conditional a) join and b) split using RSPCHB

Consider the slightly more complicated template for a conditional join in which a control channel S is used to select which input channel to read and write the read data token to the single output channel illustrated in Figure 10(a). Note that unlike micropipeline-based architectures, the control and data channels are indistinguishable in nature. (Section 3.3 describes how an FSM may generate such control channels.) The template has one C-element per input channel, each responsible for generating the associated acknowledgement signal. Each C-element is triggered by not only the RCD output, but also the corresponding control channel bit. The collection of C-elements are combined using a NOR gate to generate the $Lrack_s$ because the C-elements are mutually exclusive. This template can be easily extended to handle more complex conditionals in which multiple inputs can be read for some values of the control.

The template for the conditional fork is shown in Figure 10(b). Here, the functional block, the RCD and the C-element are repeated for each output channel. The select data lines ensure only one function block evaluates. All C-elements are combined using an AND gate to generate the acknowledgement for the select channel. (This is because both the C-element outputs and the acknowledgement signal are active low.) This template can easily be extended to handle the generation of multiple outputs in response to some values of the control.

A common example of a conditional fork is a *skip* in which depending on the control value the input is consumed but no output is generated. The implementation has a skip output acting as an internal N+1 output rail that is not externally routed and is triggered upon the skip control value.

Figure 11 shows a one-bit memory implemented using a RSPCHB template. A and C represent the input and output channels. B is the internal storage. S is an input control channel that selects the write or read operation. When S0 is high, the memory stores the value at the input channel A to the internal storage B. When S1 is high, on the other hand, the memory is read, that is, the stored memory value is written to the output channel C. For a write, both input data and control channels are acknowledged, while for a read, only the control channel is acknowledged.

The write and read operations are as follows. After reset, the memory, stored in the dual-rail *Memory Unit*, MU (similar to [11]) is initialized to some value and one of the rails of the internal signal B is high. When an input A is applied and S0 is high, if a value opposite to the value already stored in the memory is written, then first both rails of B are lowered and then one of them is asserted high, thereby storing the data. On the other hand if the value to be written is the same value already stored in the memory then the there is no transition and value remains the same. The *Memory Completion Detector, MCD*, detects that the value in the memory is updated, and asserts its output. The output of the MCD as well as the request lines from the data and control channel drive a C-element, which generates the acknowledgment signal $Lack_A$. When S1 is high, on the other hand, the internal data stored in B is sent to the output channel C. When an acknowledgment is received from the output channel C, the outputs are reset but the data stored remains unchanged. The control channel S is acknowledged for both write and read operations using an AND gate driven by the two C-element outputs.

1522-8681/02 $17.00 © 2002 **IEEE**
IEEE COMPUTER SOCIETY

Notice that the memory is actually implemented by merging two RSPCHB units. The first one is used to store data (write), and the second one to send it to the outputs (read). The first unit has an MCD that detects the completion of the write operation and resets when all inputs are lowered.

The MCD can be simplified by replacing the PMOS transistors driven by A0 and A1 with a PMOS transistor driven by $Lack_A$. However this requires that the delay difference between the data lines of channel A and its associated request line is not long enough to cause short circuit current. This restriction can be removed by also controlling the NMOS stack by also adding one more NMOS transistor driven by the $Lack_A$ signal. The overall benefit however is not clear.

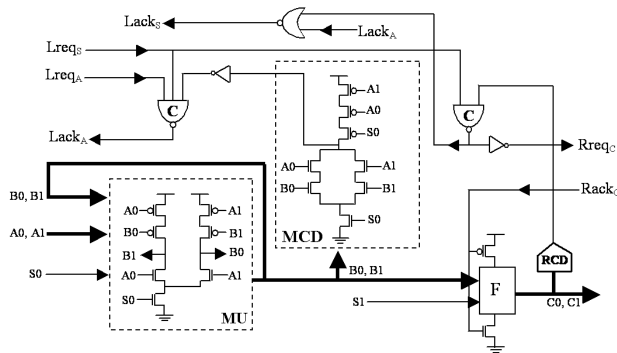


Figure 11. A RSPCHB 1-bit memory

3.2 RSPCFB

Our second new *1-of-N+1* QDI pipeline template is a full buffer constructed by merging our RSPCHB with a modified WCHB. An abstract illustration of this *reduced stack pre-charged full buffer* (*RSPCFB*) is shown in Figure 12(a) and a more detailed implementation for dual-rail data is shown in Figure 12(b).

The RSPCFB has two new features. First, the inverters from both of the half buffers have been removed to keep the forward latency of the new template at two gate delays. We assert that the inverters between the two half buffers can safely be removed because the RSPCHB has little gate load and wire load can be minimized by placing/routing this template as a single unit. The output inverters are only necessary if this unit is driving a significant load and can be added as necessary. (However a staticizer, not shown, is still necessary.) Second, the WCHB has to be modified to accept an input request signal and generate an output request signal. This input request signal drives a C-element whose other input is the RCD output. This C-element then triggers the internal acknowledgement to the RSPCHB part instead of the RCD alone. In addition, the output request signal is implemented by simply tapping of a signal from the RCD output. One other difference is that the request signal is now active low because the inverters have changed locations (i.e., *bubble shuffling* [18]).

The circuit operates as follows. The RCD of the RSPCFB part detects the evaluation of the function block and asserts its output. The output of the RCD drives the C-element, which generates the acknowledgment signal *Lack* to the previous stage after all the request lines associated with the data also arrive. If the next stage is ready to accept new data, the acknowledgment signal *Rack* should already be deasserted, allowing the C-elements in the forward path to pass the data to the next stage. Subsequently, the WCHB's RCD will assert its output asserting the request signal to the next stage. The output of the RCD also drives the C-element C_b, which asserts the internal acknowledgement back to the RSPCFB part, allowing the function block to precharge. When the acknowledgment signal *Rack* is deasserted, the C-element in the forward path will deassert its outputs. This will trigger the WCHB's RCD to deassert *Rreq*, the C-element C_b to deassert the internal acknowledgement back to the RSPCHB, and thereby enable the function block to re-evaluate.

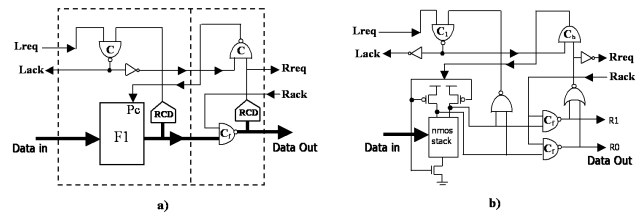


Figure 12. a) Abstract and b) detailed RSPCFB

Notice that the *Rreq* of the RSPCFB is taken from the output of the RCD instead of the C-element, unlike the RSPCHB. This is because the WCHB part has weak-conditioned logic, which will not reset until all inputs, including inputs from the RSPCHB part, have reset. This implicitly avoids the problem of preventing the assertion of the acknowledgement back to the RSPCHB part that delaying *Rreq* solved. The advantage of this is that the *Rreq* can be generated earlier. The disadvantage is that this reduces are timing margin on input channels to joins to 5 to 7 gate delays, depending on buffering.

The RSPCFB has 10 transitions per cycle, less than Caltech's PCFB, which has 12 transitions. The analytical cycle time, using the STG in Figure 13, can be expressed as:

$$T_{RSPCFB} = Max(\ 3.\ t_{Eval} + 2.\ t_{CD} + 2.\ t_c + \ t_{prech}\ , \\ 2.\ t_{Eval} + t_{CD}\ +\ 3.\ t_c + \ t_{NAND})$$

The RSPCFB can be extended to handle non-linear pipeline structures in the same way as the RSPCHB without any additional timing assumptions.

IEEE COMPUTER SOCIETY

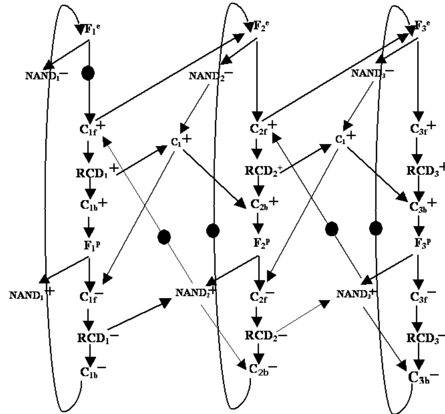


Figure 13. The STG of the RSPCFB

3.3 FSM Design

One of the most important aspects of a complete system design is the implementation of the controller. An FSM is actually a state holding circuit, which only changes its state when the expected inputs for that state are available. One way to build an asynchronous FSM is to feed the outputs of the pipeline stage back to its inputs using buffers to hold the data (also proposed in [11]). This technique is similar to the synchronous case. In addition it requires no new circuits and can be easily applied to template-based design. Figure 14 shows an abstract FSM.

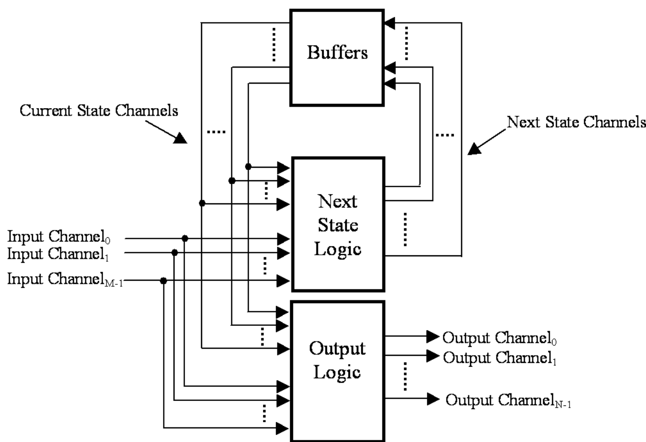


Figure 14. An abstract asynchronous FSM

Each channel either is an input, an output, or holds state. The next and current state channels can be implemented with either 1-of-N+1 channels, ideally suitable for one-hot state encoding of the FSM. The next state and the output logic blocks are complex QDI pipeline stages, which can have multiple function blocks inside. These multi-input multi-output conditional blocks are implemented the same way as the conditional read and write blocks shown previously.

The simplicity of this method for designing FSMs allows all known synchronous design techniques for generating Boolean next state and output expressions directly to be applied. Also the next state logic can be implemented as several stages of pipelined logic, reducing the number of necessary feedback buffers. Aside from using feedback buffers, which for a high number of states can yield a large circuit there are also other ways to design circuits that hold state.

4. Simulation Results

Both formal verification and HSPICE simulations were performed to check the correctness of functionality and to measure performance of all the proposed linear and non-linear pipelines.

We used the relative-timing verification tool RTCG [20] to verify the QDI property and the proposed relative-timing constraints of both RSPCHB and RSPCFB base templates. A structural blif circuit was generated for each template along with a signal transition graph (STG) description of the environments. For the optimized RSPCHB template, the tool automatically generated relative timing constraints that were sufficient to guarantee correctness, but more complex than the relative-timing constraint that we manually derived. Using the tool interactively, however, we also were able to verify the correctness of our manually-derived relative timing constraints.

HSPICE simulations were performed using a 0.25 TSMC process with a 2.5V power supply at 25°C. The purpose of these simulations was to confirm the results obtained by the Verilog simulations, and to compare the throughputs of the proposed pipelines with the bubble-shuffled PCHB and PCFB pipelines presented in the background section. Since the goal was comparison, no attempt was made to fine-tune the transistor sizing to achieve optimum performance. In particular, all transistors were sized in order to roughly achieve a gate delay equal to a small inverter (WNMOS=0.8um, WPMOS=2um, and L=0.24um) driving a same-sized inverter. For the purposes of this comparison, wire delay also has been ignored.

For the half buffers, the PCHB and the RSPCHB, a linear dual-rail pipeline of buffers with 60 stages has been constructed to achieve a static slack of 30, which means that it can hold 30 distinct data tokens. For the full buffers, the PCFB and the RSPCFB, 30 stages have been used to achieve the same static slack. All pipelines can hold 30 distinct tokens. Figure 15(a) shows throughput versus tokens triangles for the half buffers and Figure 15(b) shows them for the full buffers. The triangles for the PCHB and PCFB are indicated with the dotted lines. Approximately 15 distinct points have been obtained per pipeline for the triangle graphs using HSPICE simulation. One key result

1522-8681/02 $17.00 © 2002 **IEEE**
IEEE COMPUTER SOCIETY

obtained from this simulation is the *dynamic slack* of each pipeline, which is the number of tokens required to achieve maximum throughput [10], [11].

The PCHB achieves a maximum throughput of 772MHz with a dynamic slack of 7.3. The RSPCHB is faster with a maximum throughput of 920MHz and a dynamic slack of 8.25. The throughput improvement is approximately 20%. For the full buffers, the PCFB achieves a maximum throughput of 707MHz and a dynamic slack of 3.7. The RSPCHB is faster with a maximum throughput of 1000MHz and a dynamic slack of 5.9. The speed improvement is approximately 40%, however due to the C-elements in the forward path of the RSPCFB, the forward latency is about 15% slower. In both the half and full buffer, we achieved higher dynamic slack. This means that our templates support more system-level concurrency and higher stage utilization.

Notice that although the PCFB has 12 and the PCHB has 14 transitions per cycle, the PCFB was slower. This is partially due to the heavier load on the internal wiring in the PCFB compared to the PCHB. Clearly, careful transistor sizing and buffering can improve the performance of all pipeline templates, however, we expect the relative performances to remain approximately the same.

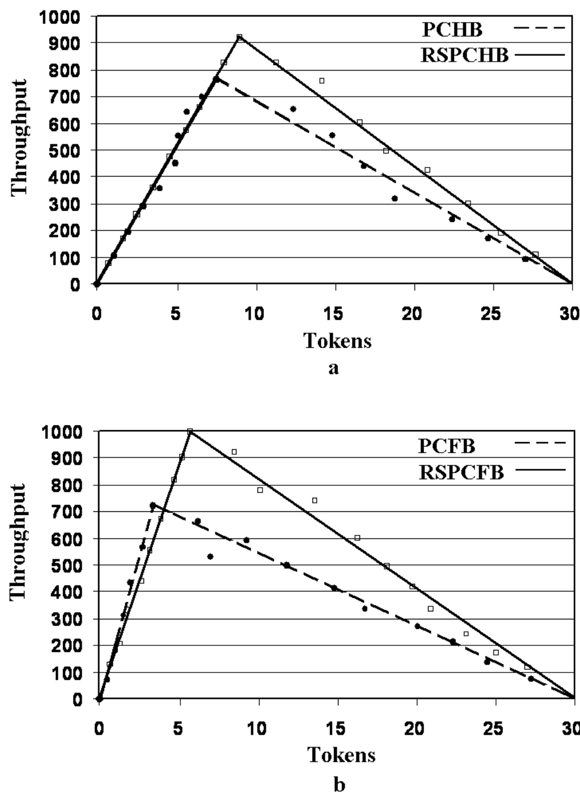


Figure 15. Throughput versus tokens for a) the PCHB and RSPCHB and b) the PCFB and RSPCFB linear pipelines

5. Conclusions

This paper has introduced new high-speed QDI asynchronous pipeline templates for non-linear dynamic pipelines, including forks, joins, and more complex configurations in which channels are conditionally read and/or written. Timing analysis and HSPICE simulation results demonstrate that our new RSPCHB achieves ~20% throughput over its PCHB counterpart and our new RSPCFB achieves ~40% throughput improvement over the PCFB counterpart.

6. Acknowledgements

This research has been partially supported by NSF Grant CCR-0086036 and gifts from both TRW and Fulcrum Microsystems. We would also like to thank Andrew M. Lines for clarifying several points in regards to his thesis and Jim Garside for shepherding this paper through the revision process.

References

[1] K.S. Stevens, S. Rotem, R. Ginosar, P.A. Beerel, C.J. Myers, K.Y. Yun, R. Kol, C. Dike, M. Roncken, “An asynchronous instruction length decoder,” in *IEEE JSSC,* Volume: 36 Issue: 2, pp. 217–228, Feb. 2001.

[2] W.S. Coates, J.K. Lexau, I.W. Jones, S.M. Fairbanks, and I.E. Sutherland. “FLEETzero: an asynchronous switching experiment,” in *Proc. of ASYNC,* pp. 173–182, March, 2001.

[3] J.V. Woods, P. Day, S.B. Furber, J.D Garside, N.C. Paver, S. Temple, “AMULET1: an asynchronous ARM microprocessor”, *IEEE Transactions on Computers,* Volume: 46 Issue: 4, pp. 385 – 398, April 1997.

[4] J. Kessels, A. Peeters “The Tangram framework: asynchronous circuits for low power ”, *Proceedings of the ASP-DAC* 2001, pp. 255 –260, 2001.

[5] A. Bardsley, D. A. Edwards, “The Balsa Asynchronous Circuit Synthesis System”, *Forum on Design Languages*, Sept. 2000.

[6] I. E. Sutherland, and S. Fairbanks. “GasP: a minimal FIFO control,” in *Proc. of ASYNC, 2001,* pp. 46–53, March 2001.

[7] S. Schuster, W. Reohr, P. Cook, D. Heidel, M. Immediato, and K. Jenkins. “Asynchronous interlocked pipelined CMOS circuits operating at 3.3-4.5 GHz,” in *IEEE ISSCC Digest of Technical Papers,* pp. 292–293, Feb. 2000.

[8] M. Singh, and S.M. Nowick. “High-throughput asynchronous pipelines for fine grain dynamic datapaths,” in *Proc. of ASYNC,* pp. 198–209, March 2000.

[9] M. Singh, and S.M. Nowick. “Fine-grain pipelined asynchronous adders for high-speed DSP applications” in *Proc. of IEEE Computer Society Annual Workshop on VLSI, Orlando, FL*, pp. 111–118, April 2000.

IEEE COMPUTER SOCIETY

[10] T.E. Williams, and M.A. Horowitz. "A Zero-overhead self-timed 160ns 54b CMOS divider," in *ISSCC Digest of Technical Papers,* pp. 98-296, 1991.

[11] A.M. Lines. *Pipelined Asynchronous Circuits*. M.Sc. Thesis, California Institute of Technology, June 1995, revised 1998.

[12] T.E. Williams. *Self-Timed Rings and their Application to Division.* Ph.D. Thesis, Stanford University, May 1991.

[13] A.J. Martin, A. Lines, R. Manohar, M. Nystroem, P. Penzes, R. Southworth, U. Cummings and T. K. Lee. "The Design of an Asynchronous MIPS R3000 Microprocessor". *In Proc. of 17th Conference on Advanced Research in VLSI*, pp. 164-181, 1997.

[14] A.J. Martin. "The Limitations of Delay-Insensitivity in Asynchronous Circuits". *Sixth MIT Conference on Advanced Research in VLSI*, ed. W.J. Dally, pp. 263-278, 1990.

[15] C. L. Seitz. "System Timing," in Carver A. Mead and Lynn A. Conway, editors, *Introduction to VLSI Systems,* Chapter 7. Addison-Wesley, 1980.

[16] K. Fant, R. Stephani, R. Smith and R. Jorgensen. "The Orphan in 2 Value Null Convention Logic", Internal Technical Report, Theseus Logic, 1998.

[17] T.-A. Chu. "Synthesis of Self-Times VLSI Circuits from Graph-Theoretic Specifications", Internal Report: MIT/LCS/TR-393, June 1987.

[18] A. J. Martin, "Programming in VLSI: From Communicating Processes to Delay-Insensitive Circuits" in Hoare, C. A. R. editor, *Developments in Concurrency and Communication*, UT Year of Programming Series, pp. 1-64, 1989.

[19] I.E. Sutherland. Micropipelines. *Communications of the AC*M, vol.32, no.6, pp. 720–738, June 1989.

[20] H. Kim, K. Stevens, and P.A. Beerel. "Relative Timing Based Verification of Timed Circuits and Systems". To appear in *Proc. of ASYNC,* April 2002.

1522-8681/02 $17.00 © 2002 **IEEE**

IEEE COMPUTER SOCIETY