In *Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems (“Async2000”), April 2–6, 2000, Eilat, Israel.*

High-Throughput Asynchronous Pipelines for Fine-Grain Dynamic Datapaths[*]

Montek Singh and Steven M. Nowick

Department of Computer Science
Columbia University
New York, NY 10027
{montek,nowick}@cs.columbia.edu

Abstract

This paper introduces several new asynchronous pipeline designs which offer high throughput as well as low latency. The designs target dynamic datapaths, both dual-rail as well as single-rail. The new pipelines are latch-free and therefore are particularly well-suited for fine-grain pipelining, i.e., *where each pipeline stage is only a single gate deep. The pipelines employ new control structures and protocols aimed at reducing the handshaking delay, the principal impediment to achieving high throughput in asynchronous pipelines.*

As a test vehicle, a 4-bit FIFO was designed using 0.6 micron technology. The results of careful HSPICE simulations of the FIFO designs are very encouraging. The dual-rail designs deliver a throughput of up to 860 million data items per second. This performance represents an improvement by a factor of 2 over a widely-used comparable approach by Williams [16]. The new single-rail designs deliver a throughput of up to 1208 million data items per second.

1. Introduction

In this paper, several new asynchronous pipeline designs are introduced for dynamic datapaths. Both dual-rail and single-rail designs are presented. All of these designs are targeted towards achieving very high throughput, without degrading latency. The high performance is brought about using novel protocols which reduce the impact of handshaking overhead.

We focus on dynamic datapaths for a variety of reasons. Dynamic logic is increasingly being used in industry due to its potential for high speed and small area. Also, dynamic pipelines can be designed without the need for latches between pipeline stages; with clever control sequencing, the gates themselves can function as implicit latches. In spite of these advantages, however, there is a lack of pipeline designs which are tailored for, and take full advantage of, dynamic logic. This paper attempts to fill this void.

A particular focus of this work is on fine-grain pipelining, even though the designs presented are also well suited for coarser granularity datapaths such as processor pipelines. For very high throughput, the datapath needs to be sectioned into fine stages. In the limit, the highest throughputs are achieved when each pipeline stage consists only of a single gate, *i.e.*, pipelining is at the *gate-level.* At this granularity, latch-free datapaths are especially desirable, therefore dynamic logic is a good match. So far, there has been a lack of approaches that really push throughput to this granularity.

Our pipeline designs are based on a set of novel protocols that reduce the impact of handshaking overhead. The key strategy is one of *anticipation*: anticipating the arrival of certain critical events based on a richer class of observations of the state of the pipeline. Consequently, they are named ***L**ookahead **P**ipelines* (abbreviated **LP**).

This paper contributes five new pipeline designs. Of these, three designs are for dual-rail datapaths with completion detection, and two are for single-rail bundled-datapaths. Williams' PS0 pipeline is used as a starting point for the dual-rail designs, and several throughput-oriented protocol optimizations are then applied. These include: (i) *early evaluation*, (ii) *early done* and (iii) a combination of both. The optimizations are then adapted to single-rail. An additional contribution of this paper is a technique for interfacing the new pipeline designs, as well as Williams' PS0 design, with *arbitrary speed environments;* to this extent it fixes a major shortcoming of PS0.

Simulation of a 4-bit FIFO design provided an effective means of quantifying the throughput. The three new dual-rail pipelines have throughputs of 590, 760 and 860 million data items per second respectively, which represent 40%, 79% and 102% improvements over the throughput of Williams' PS0 [16]. The two new single-rail pipelines have throughputs of 1050 and 1208 million per second. Not only are these throughputs competitive with the best reported in

[*] This work was supported by NSF Award No. CCR-97-34803.

the literature [10], but our pipelines also have significant area and latency benefits.

The paper is organized as follows. After providing background on Williams' PS0 pipelines, Section 2 introduces the new dual-rail designs, including their operation and timing constraints. Section 3 then introduces the new single-rail designs. Section 4 provides a detailed comparison of one of the dual-rail designs (LP3/1) with a related design by Williams, and highlights the new contribution. Section 5 discusses the issue of interfacing our pipelines with the environment. Section 6 discusses some issues in the design of gate-level pipelines, and Section 7 discusses how the pipelines are initialized. Section 8 presents detailed simulation results, and finally, Section 9 gives conclusions.

2. Dual-Rail Pipelines

2.1. Previous Work

The classic work on dual-rail pipelines with completion detection is by Williams [16], which proposed a number of alternative implementation styles. Several of these designs have the advantage of very low forward latency. However, they are all *throughput-limited.*

Recently, several variants of Williams' dual-rail schemes have been proposed [13, 9, 8].

Renaudin *et al.* [13] introduce a scheme aimed at improving the storage capability of the pipeline. These pipelines use novel latch structures to enable data tokens to be packed more closely, and are hence more compact area-wise. However, the throughput of this scheme appears to be even worse than Williams' PS0, though it is an improvement over the more conservative PC0 scheme.

Matsubara and Ide [9] propose a scheme targeted at reducing power consumption and chip area. The savings are brought about by combining single-rail static circuits with dual-rail dynamic pipelines. However, since the underlying protocol is essentially that of Williams' PS0, the throughput is not improved.

Finally, Martin *et al.* [8] present the design of a complete microprocessor using very fine-grain pipelining techniques similar to Williams'. The pipeline circuits are based on the very conservative and robust QDI model, yet have high performance. Their best cycle time is approximately 8 times the delay through a single stage. In contrast, the new pipeline schemes introduced in this paper have even higher performance, with much shorter cycle times: between 4.4 and 7 stage delays.

2.2. Background: Williams' PS0 Pipeline

We now give background on Williams' PS0 dual-rail pipeline, which is the starting point for our new designs. The next subsections give details on our new dual-rail pipelines, LP3/1, LP2/2 and LP2/1.

PS0 Pipeline Structure. Figure 1 shows Williams' PS0 pipeline. Each pipeline stage is composed of a dual-rail function block and a completion detector. The completion detectors indicate validity or absence of data at the outputs of the associated function block.

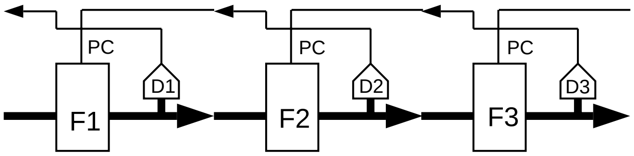

Figure 1. Block diagram of a PS0 pipeline

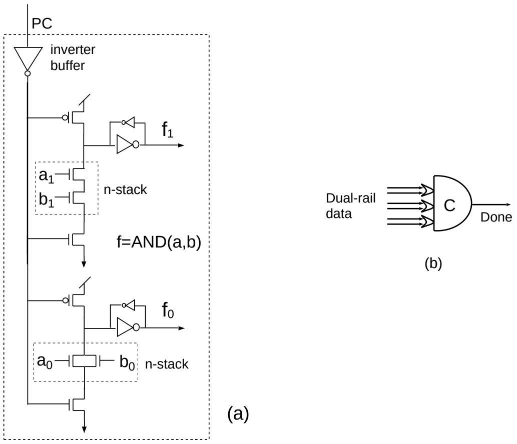

Figure 2. (a) A dual-rail AND gate in precharge logic, and (b) a dual-rail completion detector

Each function block is implemented using dynamic logic. The precharge/evaluate control input, PC, of each stage is tied to the output of the next stage's completion detector. Since a precharge logic block can hold its data outputs even when its inputs are reset, it also provides the functionality of an implicit latch. Therefore, a PS0 stage has no explicit latch. Figure 2(a) shows how a dual-rail AND gate, for example, would be implemented in dynamic logic; the dual-rail pair, f_1 and f_0, implements the AND of the dual-rail inputs $a_1 a_0$ and $b_1 b_0$.

Each completion detector verifies the completion of every computation and precharge of its associated function block. Validity, or non-validity, of data outputs is checked by OR'ing the two rails for each individual bit, and then using a C-element to combine all the results (see Figure 2(b)).

PS0 Pipeline Protocol. The sequencing of pipeline control is quite simple. Stage N is precharged when stage $N+1$ finishes evaluation. Stage N evaluates when stage $N+1$ finishes reset. (Of course, the actual evaluation will commence only after valid data inputs have also been received from stage $N-1$.) This protocol ensures that consecutive data tokens are always separated by reset tokens (or "spacers").

The complete cycle of events for a pipeline stage is derived by observing how a single data token flows through an initially empty pipeline. The sequence of events from one evaluation by stage 1, to the next is: (i) Stage 1 evaluates, then (ii) stage 2 evaluates, then (iii) stage 2's completion detector detects completion of evaluation, and then (iv) stage 1 precharges. At the same time, after completing step (ii), (iii)' stage 3 evaluates, then (iv)' stage 3's completion detector detects completion of evaluation and initiates the precharge of stage 2, then (v) stage 2 precharges, and finally, (vi) stage 2's completion detector detects completion of precharge, thereby releasing the precharge of stage 1 and enabling stage 1 to evaluate once again. Thus, there are six events in the complete cycle for a stage from one evaluation to the next.

PS0 Pipeline Cycle Time and Latency. The complete cycle for a pipeline stage, traced above, consists of 3 evaluations, 2 completion detections and 1 precharge. The analytical pipeline cycle time, T_{PS0}, therefore is:

$$T_{\text{PS0}} = 3 \cdot t_{\mathit{Eval}} + 2 \cdot t_{\mathit{CD}} + t_{\mathit{Prech}}$$

where, t_{Eval} and t_{Prech} are the evaluation and precharge times for each stage, and t_{CD} is the delay through each completion detector.[1]

The per-stage forward latency, L, is defined as the time it takes the first data token, in an initially empty pipeline, to travel from the output of one stage to the output of the next stage. For PS0, the forward latency is simply the evaluation delay of a stage:

$$L_{\text{PS0}} = t_{\mathit{Eval}}$$

2.3. Overview of New Designs

Before introducing the new dual-rail pipeline designs, we give a brief overview of our approach. The new pipeline designs use Williams' PS0 as the starting point, and apply certain optimizations to its protocol, targeted towards reducing the overall cycle time.

The basic strategy is one of *anticipation*: anticipating the arrival of certain critical events based on a richer set of observations of the state of the pipeline. In particular, such a strategy has been widely used recently in many different contexts: from timing-optimal variants of Sutherland's basic micropipeline [2], to more aggressive timing optimizations based on *kiting* [10] and *relative timing* [14, 1].

In this paper, two specific optimizations are used: (i) *early evaluation*, and (ii) *early done.* In "early evaluation," a pipeline stage uses control information not only from the subsequent stage, but also from stages *further down* the pipeline. This information is used to give the stage a headstart on its evaluation phase. In the second optimization, "early done," a stage signals to its previous stage when it is *about to* precharge or evaluate, instead of after it has completed those actions. This information is used to give a pipeline stage a headstart both on its evaluation phase as well as its precharge phase.

The net result of applying these two optimizations is a significant reduction in pipeline cycle time, and consequently a dramatic increase in throughput, with no net increase in latency.

The remainder of this section presents the three new pipeline designs, LP3/1, LP2/2 and LP2/1 in detail. The LP3/1 pipeline uses early evaluation, the LP2/2 pipeline uses early done, and the LP2/1 is a hybrid which combines both optimizations.

2.4. The LP3/1 Pipeline

LP3/1 is the first of our dual-rail high-throughput pipelines. In this design style, an *early evaluation* protocol is used, in which a pipeline stage receives control information not only from the subsequent stage, but also *from its successor.* As a result, LP3/1 pipelines have shorter cycles than Williams' PS0 pipelines: a complete cycle for a stage of an LP3/1 pipeline consists of 4 events, as opposed to 6 events for a stage of a PS0 pipeline.

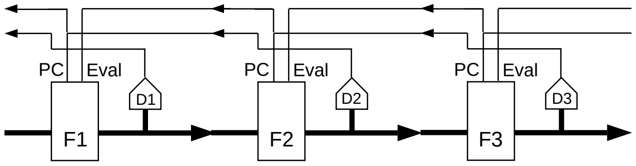

Figure 3. Block diagram of an LP3/1 pipeline

The new pipeline derives its name from the fact that 3 out of the 4 events in every stage's cycle fall in its evaluation phase, and 1 event falls in its precharge phase. Using this terminology, Williams' PS0 would be 3/3.

It is pointed out later that in [16], Williams briefly introduces a PA0 pipeline design that, in essence, has a similar underlying protocol as our LP3/1 pipeline. However, PA0 uses a different implementation of the control which is not able to fully take advantage of the new protocol. A detailed presentation of the LP3/1 design is given in this section, while the key differences between PA0 and LP3/1 are discussed in Section 4.

Pipeline Structure. Figure 3 shows the block diagram of an LP3/1 pipeline. The key difference is that, unlike PS0 stages, an LP3/1 stage has two control inputs. The first control input, PC, comes from the next stage, as in PS0. The second control input, EVAL, comes from the completion detector two stages ahead. This second input is the key to achieving a shorter cycle time.

[1] To simplify the presentation, this paper will sometimes assume that all the pipeline stages have the same evaluation delay and the same precharge delay, and that all the completion detectors are equally fast. More realistic HSPICE simulations are presented in the Results section.

Pipeline Protocol and Implementation. The key idea of the new protocol is that stage N can evaluate as soon as stage $N+1$ has started precharging, instead of waiting until stage $N+1$ has completed precharging. This idea can be used because a dynamic logic stage undergoing precharge is insensitive to changes on its inputs.[2] Therefore, as soon as stage $N+1$ begins to precharge, stage N can proceed with its next evaluation. Now, since stage $N+1$ begins precharging when stage $N+2$ completes evaluation, the new condition for evaluation is: *Evaluate* N *when* $N+2$ *completes evaluation*. The condition for precharge remains unchanged: *Precharge N when* $N+1$ *completes evaluation*. Therefore, stage N needs inputs from both the completion detector of $N+1$ as well as from that of $N+2$.

Figure 4 shows the implementation of one output of an LP3/1 stage. The figure shows how the two control inputs are combined inside the implementation of one stage. Evaluation is enabled when either EVAL is asserted high, or PC is de-asserted low, or both. The former condition, EVAL=high, corresponds to stage $N+2$ completing its computation (*i.e.*, stage $N+1$ *starting* its precharge; see Figure 3). The latter condition, PC=low, is identical to the evaluation condition of PS0; its role in LP3/1 is explained below. Precharge is enabled when both PC is asserted high and EVAL is de-asserted low.

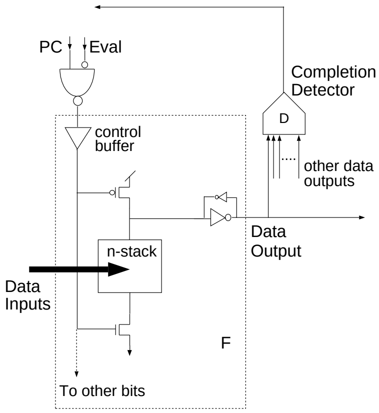

Figure 4. Implementation of an LP3/1 stage

Detailed Comparison with the PS0 Protocol. In LP3/1, there are now two distinct control inputs for a stage N, which are outputs from stages $N+1$ and $N+2$. The precharge phase of stage N begins after stage $N+1$ is done evaluating (PC asserted high), much like PS0. However, this phase is now shortened: precharge terminates when stage $N+2$ is done evaluating (EVAL asserted high). In contrast, in PS0, precharge terminates only stage $N+1$ is done precharging. At this point, stage N enters its *evaluate* phase.

In LP3/1, th evaluate phase continues until two distinct conditions hold, which drive the stage into the next precharge: (i) stage $N+1$ has completed evaluation (as in PS0: PC asserted high) and (ii) stage $N+2$ has completed precharging (EVAL de-asserted low). The NAND gate in Figure 4 (with a bubble on its EVAL input) directly implements these two conditions.

Interestingly, during LP3/1's evaluate phase, the early EVAL signal from stage $N+2$ may be *non-persistent:* it may be de-asserted low even before stage N has had a chance to evaluate its new data! However, one-sided timing constraints of Section 2.7 are imposed to insure a correct evaluate phase: PC=low will arrive in time to *take over* control of the evaluate phase, which will then be maintained until stage N has completed evaluating its inputs (as in PS0).

Pipeline Cycle Time and Latency. The complete cycle of events for a stage, say stage 1, from one evaluation till the next can be derived from Figure 3: (i) Stage 1 evaluates, (ii) stage 2 evaluates, (iii) stage 2's completion detector detects completion of precharge, and then (iv) stage 1 precharges. At the same time, after completing step (ii), (iii)' stage 3 evaluates, and (iv)' stage 3's completion detector detects completion of stage 3's evaluation, thereby enabling two subsequent events: both the precharge of stage 2 and the next evaluation of stage 1 ("early evaluation"). Thus, there are only four events in the complete cycle for a stage, from one evaluation to the next, down from the six events in PS0. This reduction by two events has been brought about by eliminating the two events of stage 2's precharge phase from stage 1's cycle.

The cycle time of the pipeline is therefore:

$$T_{\text{LP3/1}} = 3 \cdot t_{Eval} + t_{CD} + t_{\text{NAND}}$$

where t_{NAND} is the delay through the NAND gate for the early evaluation signal. Thus, the LP3/1 cycle time is $t_{Prech}+t_{CD}-t_{\text{NAND}}$ shorter than that of PS0.

The per-stage forward latency is simply the evaluation delay of a stage, as in PS0:

$$L_{\text{LP3/1}} = t_{Eval}$$

Loading Issues. The above analysis does not take into account the fact that the completion detectors in LP3/1 will be somewhat slower than those in PS0 due to greater capacitive loads. The increased loading is due to the need to fork off "done" to two preceding stages instead of one. Section 8 provides more refined results based on HSPICE simulations of the actual pipeline circuits. The simulation results indicate that, in spite of the overhead due to increased loading, the LP3/1 pipeline has significantly higher throughput than PS0.

[2] In general, this property is only true of *fully-controlled* (or "footed") dynamic logic. All of our pipelines presented in this paper use fully-controlled dynamic logic. Therefore, this property of precharge can indeed be exploited.

2.5. The LP2/2 Pipeline

The second new pipeline design is called LP2/2. The key feature is that a pipeline stage is now allowed to signal its previous stage when it is *"about to* evaluate (or precharge)" instead of after it has completed those actions. Thus, this pipeline uses an *early done* protocol.

LP2/2 pipelines have shorter cycle times than PS0: like LP3/1, the cycle of an LP2/2 stage consists of four events. Moreover, these pipelines have another desirable feature: unlike LP3/1, the stages of an LP2/2 pipeline have only one control input as opposed to two, thereby reducing loading on the completion detectors.

Pipeline Structure. Figure 5 shows a block diagram of an LP2/2 pipeline. The stages are similar to those used in PS0, but with one key difference: completion detectors are placed before their functional blocks. The idea is to let the previous pipeline stage know when the current stage is about to evaluate (or precharge).

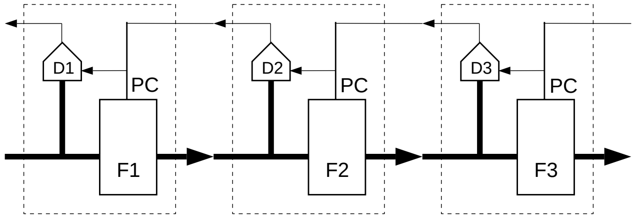

Figure 5. Block diagram of an LP2/2 pipeline

A modified completion detector is needed in order to generate the "early done" signal. The completion detector now requires an extra input: the stage's PC control input. The functionality of the completion detector is as follows. The completion detector asserts *Done* (high) when the stage is about to evaluate: the stage is enabled to evaluate (PC deasserted low), and it has valid dual-rail inputs. The completion detector de-asserts *Done* (low) when the stage is about to precharge: PC is asserted (high). Thus, the done signals are produced in parallel with the actual precharge or evaluation by the associated function block, instead of after its completion. Note that these conditions are asymmetric: only a single condition (PC asserted high) enables the stage to precharge and its completion detector to indicate that precharge is complete.

This new completion detector is implemented using an *asymmetric C-element* (Figure 6). From the figure, it is clear that this particular asymmetric C-element is a degenerate special case: it can be regarded as simply a precharged dynamic gate, which de-asserts Done (low) whenever PC is asserted high.

Pipeline Protocol and Performance. A complete cycle of events for stage 1 can be traced in Figure 5. From one evaluation to the next, it consists of four events: (i) Stage 1 evaluates, (ii) stage 2's completion detector detects "early done" of stage 2's evaluation (in parallel with stage 2's evaluation), thereby asserting the precharge control of stage 1, and then (iii) stage 1 precharges. At the same time, after completing step (i), (ii)' stage 2 evaluates, (iii)' stage 3's completion detector detects "early done" of stage 3's evaluation (in parallel with stage 2's evaluation), thereby asserting the precharge control of stage 2, and (iv) stage 2's completion detector detects "early done" of stage 2's precharge (in parallel with stage 2's precharge), thereby enabling stage 1 to evaluate once again in the next step.

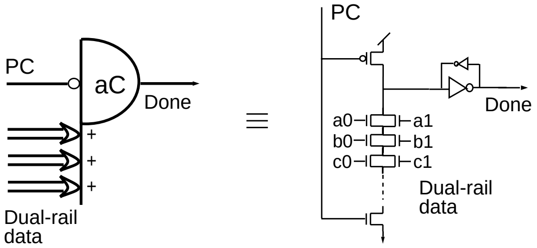

Figure 6. LP2/2 Completion Detector

Thus, the cycle time of the pipeline is:

$$T_{\text{LP2/2}} = 2 \cdot t_{Eval} + 2 \cdot t_{CD}$$

which is $t_{Eval} + t_{Prech}$ shorter than that of PS0. The latency is identical to that of PS0 and LP3/1:

$$L_{\text{LP2/2}} = t_{Eval}$$

2.6. The LP2/1 Pipeline

LP2/1 is the last new dual-rail design style. This design is basically a hybrid: it combines both the "early evaluation" of LP3/1 and the "early done" of LP2/2. Consequently, an LP2/1 pipeline has the shortest analytical cycle time of the three LP design styles: a cycle of a stage consists of only three events.

Pipeline Structure. Figure 7 shows the implementation of an LP2/1 pipeline. Each stage uses information from two succeeding stages (as in LP3/1), and also employs early completion detection (as in LP2/2).

Pipeline Protocol and Performance. A complete cycle of events for stage 1 can again be traced in the figure. From one evaluation to the next it consists of three events: (i) Stage 1 evaluates, (ii) stage 2's completion detector detects "early done" of stage 2's evaluation (in parallel with stage 2's evaluation), thereby asserting the precharge control of stage 1, and then (iii) stage 1 precharges. At the same time, after completing step (i), (ii)' stage 2 evaluates,

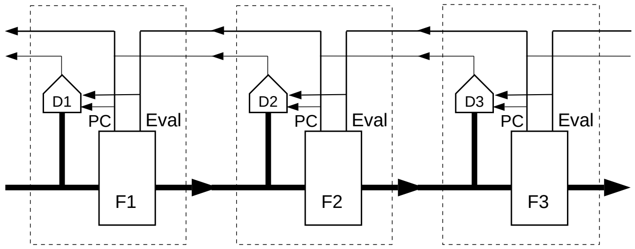

Figure 7. Block diagram of an LP2/1 pipeline

and (iii)' stage 3's completion detector detects the "early done" of stage 3's evaluation, thus enabling the evaluation of stage 1 in the next step. Thus, the cycle time is:

$$T_{\text{LP2/1}} = 2 \cdot t_{Eval} + t_{CD} + t_{\text{NAND}}$$

which is $t_{Eval} + t_{Prech} + t_{CD} - t_{\text{NAND}}$ shorter than that of PS0. Once again, the latency is identical to that of PS0:

$$L_{\text{LP2/1}} = t_{Eval}$$

2.7. Timing Constraints

Each of the LP pipeline designs requires certain one-sided timing constraints to be satisfied for correct operation. We found, in practice, that all of these timing constraints are easily satisfied.

Precharge Width. LP3/1 and LP2/1 pipelines have a shorter precharge phase than PS0 pipelines, since the start of the evaluation phase is advanced by two time steps.[3] For correct precharge, the precharge of a stage should be complete before the stage receives the evaluation signal, EVAL=high. That is, a *minimum precharge width* must be enforced.

The appropriate timing constraint for the LP3/1 pipeline is now formally derived. Using as reference the instant stage $N + 1$ finishes evaluating, stage N receives the precharge signal at time $t_{CD_{N+1}\uparrow}$, where $t_{CD_{N+1}\uparrow}$ is the time it takes for stage $N + 1$'s completion detector to switch high.[4] Also, from the same reference, the EVAL signal for stage N goes high at time $t_{Eval_{N+2}} + t_{CD_{N+2}\uparrow}$. Therefore, for correct precharge, the precharge width t_{Prech_N} must satisfy:

$$t_{Prech_N} \leq t_{Eval_{N+2}} + t_{CD_{N+2}\uparrow} - t_{CD_{N+1}\uparrow}$$

Assuming that all stages are similar and that both transitions of a completion detector are equally fast, the constraint can be rewritten as:

$$t_{Prech} \leq t_{Eval}$$

This condition is satisfied if a stage's precharge is not slower than its evaluation. In practice, the precharge phase benefits from the additional inverter delay which the EVAL=high signal must go through at the inputs of the NAND gate. This constraint effectively means that "precharge should be fast enough," and, in our experience, is quite easily satisfied. Similar constraints on the precharge width for LP2/1 pipelines can be derived.

Safe Takeover. For correct operation of the evaluation phase in LP3/1 and LP2/1, it is required that the *"takeover"* signal, PC=low, arrive at the inputs of the NAND gate before the *non-persistent* EVAL goes low. This requirement is needed to insure that the control maintains a glitch-free evaluation phase whenever early evaluation is used (i.e. LP3/1 and LP2/1). We now focus on the case of LP3/1; similar constraints can be derived for LP2/1.

The following analysis calculates the time at which stage N's EVAL is de-asserted low, and the time at which stage N's takeover signal appears. The reference time 0 is set at the point when stage $N + 2$ has just completed evaluation, which will start the early evaluation of state N. The time instant when EVAL for stage N is de-asserted low (from stage $N + 2$) is given by:

$$t_{Eval_{N+3}} + t_{CD_{N+3}\uparrow} + t_{Prech_{N+2}} + t_{CD_{N+2}\downarrow}$$

Similarly, the takeover signal, PC, of stage N is asserted low at time:

$$t_{CD_{N+2}\uparrow} + t_{Prech_{N+1}} + t_{CD_{N+1}\downarrow}$$

Therefore, to maintain uninterrupted evaluation, the takeover should arrive at least a setup time, t_{setup}, before EVAL is de-asserted:

$$\begin{aligned} t_{CD_{N+2}\uparrow} + t_{Prech_{N+1}} + t_{CD_{N+1}\downarrow} + t_{\text{setup}} \quad &\leq \\ t_{Eval_{N+3}} + t_{CD_{N+3}\uparrow} + t_{Prech_{N+2}} + t_{CD_{N+2}\downarrow} \end{aligned}$$

Assuming all stages are similar, this constraint becomes:

$$t_{Eval} \quad \geq \quad t_{\text{setup}}$$

This constraint is also easily satisfied since the setup time of a transistor is usually less than the evaluation time of a stage.

Input Hold Time. In LP2/2 and LP2/1, the data inputs to an evaluating stage must be held valid long enough for the stage to complete evaluation, before the inputs are reset. That is, the "early done" path through the completion detector must not reset the previous stage before the current stage has effectively absorbed its data inputs. If the time for a precharge-released dynamic gate to absorb its inputs is t_{hold}, then the input hold time constraint is:

$$t_{\text{hold}} \;\leq\; t_{CD_N\uparrow} + t_{Prech_{N-1}}$$

Assuming all stages are identical, this constraint becomes:

[3] The "1" in their designation indicates precisely this fact: their precharge phase is only 1 "unit" long, where a "unit" is approximately the amount of time for one stage evaluation, or one stage reset, or one completion detection.

[4] Note that, here, for a more precise analysis, we make a distinction between $t_{CD_{N+1}\uparrow}$ and $t_{CD_{N+1}\downarrow}$, which are the delays associated with detection of stage $N + 1$'s evaluation and reset, respectively.

$$t_{CD\uparrow} \geq t_{\text{hold}} - t_{Prech_{N-1}}$$

According to this constraint, the completion detectors cannot be "too fast." This constraint is also easily satisfied in practice.

3. Single-Rail Pipelines

While dual-rail datapaths allow variable-speed completion and have been effectively used in a number of applications, the area penalties are often unacceptable and the power overhead may be large. Single-rail design has much wider applicability in the synchronous world, and several asynchronous groups have recently moved from dual- to single-rail design [12]. Therefore, the focus of this section is now on single-rail pipelines using dynamic bundled datapaths.

3.1. Related Work and Overview

Synchronous Pipelines. Several novel synchronous pipelining techniques have been proposed for high-throughput applications. In *wave pipelining* [17], multiple waves of data are allowed at any time between two latches. Other quasi-asynchronous approaches include *skew-tolerant domino* [6, 3] and *self-resetting circuits* [11, 3]. These styles have partial asynchronous behavior (e.g., precharge control or waves of data). All of these styles have complex timing requirements which are difficult to verify, lack elasticity and still require global clock distribution.

Asynchronous Pipelines. The classic approach to designing asynchronous pipelines is called *micropipelines* [15]. These use an elegant control structure but have slow and complex latch control (*i.e.,* capture-pass latches). Most recent designs—including our own—can be regarded as derived from this pipeline, using variations on both latches and protocol.

Several variants have been proposed using *transition signaling.* One design [18] uses dual-edge-triggered flipflops, and another [10] uses a double (parallel) pipeline with pairs of capture-pass latches per stage.

Other variants have been proposed using *four-phase handshaking.* In [2], transparent latches replace capture-pass latches and phase converters are used to accommodate them. In contrast, both Furber and Liu [5] and Molnar *et al.* [10] (a second design: asp*) modify the original micropipeline control, introducing both asymmetric protocols and decoupled implementations.

All of these designs have limitations for fine-grained dynamic applications. Two of the highest performance ones (asp* [10] and Yun *et al.*'s 4-phase design [18]) will not function correctly on a dynamic pipeline unless there are *explicit* latches, due to overlapped activity in adjacent stages. In addition, while the former design is elegant, it has complex relative timing assumptions which are not explicitly formalized; in fact, an early version was unstable due to timing issues. The second Molnar design, using two parallel pipelines, has significant area penalty; and another design [7] has both area and throughput penalties due to double latches. Only a few asynchronous pipeline styles have been proposed which are specially tailored to single-rail dynamic pipelines [5, 4], but these also require explicit latches and have synchronization overheads.

Overview. We now present our two new single-rail pipeline designs. The first design, LP$_{SR}$2/2, essentially uses a straightforward micropipeline control for these datapaths, but modified with an *early done* optimization similar to that of Section 2.5. The second design, LP$_{SR}$2/1, adds the further improvement of *early evaluate*, similar to the LP2/1 design of Section 2.6. The designs operate correctly under simple, explicit and easily satisfiable one-sided timing constraints.

3.2. The LP$_{SR}$2/2 Pipeline

LP$_{SR}$2/2 is the first of the new single-rail pipelines. This design can be thought of as a derivative of LP2/2, or PS0, adapted to a single-rail bundled datapath.

Pipeline Structure. Figure 8 shows the structure of the pipeline. Each pipeline stage has a function block and a control block. The function block alternately evaluates and precharges. The control block generates the bundling signal to indicate completion of evaluation (or precharge). The bundling signal is passed through a suitable delay, allowing time for the dynamic function block to complete its evaluation (or precharge). This signal is communicated to two stages: (i) to the previous stage, to indicate a "done" (*Done*), and (ii) to the next stage, to indicate a "request" (*Req*) (*i.e.,* valid data).

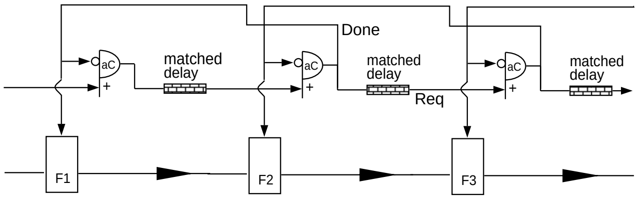

Figure 8. Block diagram of an LP$_{SR}$2/2 pipeline

Pipeline Protocol. The pipeline protocol is very similar to that of PS0. When a stage is done evaluating, it tells the previous stage to precharge. Similarly, when a stage is done precharging, it tells the previous stage to evaluate. In addition, the *Done* signal is passed forward to the next stage, indicating that the evaluation (or precharge) is complete.

However, there are two subtle optimizations that take advantage of the innate property of dynamic logic. The first is aimed at reducing the cycle time; the second is aimed at decreasing latency.

The first optimization is to *tap off* the *Done* signal for the previous stage from *before* the matched delay, instead of after the matched delay. In spirit, this optimization is similar to the "early done" of LP2/2. The same justification applies. For footed dynamic logic, it is safe to indicate *completion of precharge* as soon as the precharge cycle *begins:* during precharge, the stage is effectively isolated from changes at its inputs. Likewise, for a dynamic stage, it is safe to indicate *completion of evaluation* as soon as the stage begins to evaluate on valid inputs; once the stage has evaluated, its outputs are effectively isolated from a reset at the inputs.[5] This early tap-off optimization has a significant impact on the pipeline performance: the cycle time is reduced by an amount equal to *two* matched delays.

The second optimization is to allow an *early precharge-release*. In dynamic logic, unlike static logic, the function block can be precharge-released *before* new valid inputs arrive. Once data inputs arrive, the function block starts computing its data outputs. Similarly, once the matched bundling input arrives, the bundling output (*Req*) is also generated. Thus, in our design, precharge release of the function block is completely decoupled from the arrival of the inputs. In contrast, in other recent pipeline designs, the function block is precharge-released only after the bundling input has been received [5]; this latter requirement typically adds extra gates to the critical forward path in the pipeline. In LP$_{SR}$2/2, the optimization results in a reduction in the forward latency.

Pipeline Cycle Time and Latency. A complete cycle of events for a stage in LP$_{SR}$2/2 is quite similar to that in PS0. From one evaluation of stage 1 to the next, the cycle consists of four events: (i) Stage 1 evaluates, (ii) stage 2 evaluates, (iii) stage 3 evaluates, asserting the precharge input for stage 2, and finally, (iv) stage 2 precharges, enabling stage 1 to evaluate once again.

The following notation is used for the various delays associated with this pipeline:

t_{Eval} = time for a stage evaluation

t_{gC} = delay of the control block (generalized C-element)

t_{delay} = magnitude of the matched delay. For correct operation, $t_{delay} \geq t_{Eval} - t_{gC}$. For ideal operation, we will assume that $t_{delay}$is no larger than necessary, $t_{delay} = t_{Eval} - t_{gC}$.

In this notation, the delays of steps (i) and (ii) in the cycle traced above, are each t_{Eval}. The delays of steps (iii) and (iv) are each t_{gC}. Therefore, the pipeline cycle time is:

$$T_{\mathrm{LP_{SR}}2/2} = 2 \cdot t_{Eval} + 2 \cdot t_{gC}$$

The per-stage forward latency of the pipeline, $L_{LPSR22} = t_{Eval}$.

3.3. The LP$_{SR}$2/1 Pipeline

LP$_{SR}$2/1 is our final dynamic pipeline. This design can be thought of as a derivative of LP2/1, or LP3/1, adapted to a single-rail bundled-datapath.

Pipeline Structure. Figure 9 shows the structure of the pipeline. Each stage has a function block and a control block identical to those of the first single-rail design. However, a stage receives control inputs not only from the subsequent stage (PC), but also from its successor (EVAL). Much like LP2/1 and LP3/1, the second control input is used for "early evaluation."

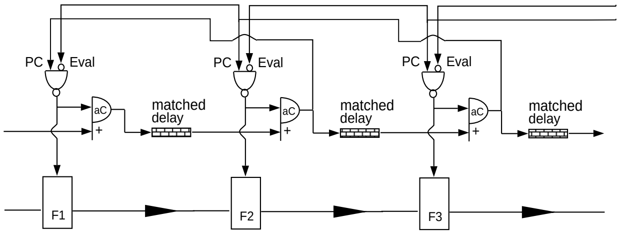

Figure 9. Block diagram of an LP$_{SR}$2/1 pipeline

Pipeline Protocol and Performance. The sequencing of control is very similar to that in LP2/1 or LP3/1. A complete cycle of events, from one evaluation of stage 1 to the next, consists of three events: (i) Stage 1 evaluates, (ii) Stage 2 evaluates, and finally, (iii) stage 3 evaluates, triggering "early evaluation" of stage 1. Thus, the cycle time is:

$$T_{\mathrm{LP_{SR}}2/1} = 2 \cdot t_{Eval} + t_{gC} + t_{\mathrm{NAND}}$$

The analytical cycle time is somewhat better than that of LP$_{SR}$2/2, because $t_{\mathrm{NAND}} < t_{gC}$.

Once again, forward latency $L_{\mathrm{LP_{SR}}2/1} = t_{Eval}$.

4. Comparison of LP3/1 with Williams' PA0 Pipelines

In [16], Williams introduces another pipeline design, called PA0. Although we have so far only referred to Williams' PS0 pipeline, his PA0 pipeline is also directly relevant to LP3/1: both designs effectively use control inputs from two subsequent stages, instead of one. Below is an in-depth comparison between the two.

The structure of a PA0 pipeline is shown in Figure 10. As in LP3/1, each pipeline stage receives two control inputs, PC and EVAL. The PC input of stage N is the completion signal

[5] More precisely, completion of evaluation can be safely indicated a time t_{hold} after the start of evaluation (cf. Section 2.7).

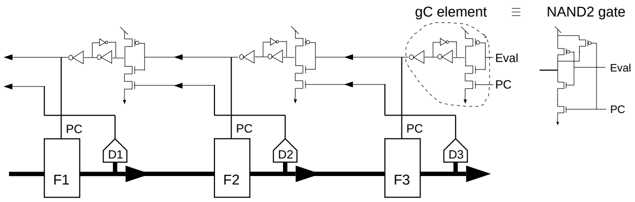

Figure 10. Block diagram of a PA0 pipeline

from stage $N+1$. The EVAL input of N is derived from the completion detector of stage $N+2$.

A PA0 pipeline operates as follows. Stage N is driven into evaluation as soon as stage $N+1$ *starts* to precharge; thus, the pipeline allows early evaluation, much like LP3/1. The "trigger signal" which causes the start of is EVAL=low. Stage N is precharged when $N+1$ is done evaluating (PC=high) and $N+2$ is done precharging (EVAL=high). This stage's control is implemented by a generalized C-element, shown in Figure 10.

There is an important difference in the controls of PA0 and LP3/1, although they seem to be functionally quite similar: the PA0 control uses a generalized C-element, whereas the LP3/1 control uses a NAND2 gate. The net difference is not only a change of function in our design, but the removal of two inverters in series from the critical path.[6]

We now justify our use of a NAND2 gate instead of the generalized C-element. A simple timing assumption must be made on the arrival of inputs to the NAND2 gate. In each design, an early evaluation of stage N is enabled by the trigger signal, EVAL=low, which is an input to the control. In PA0, the C-element holds this value, and evaluation persists, until the desired precharged phase begins. In contrast, in LP3/1, while EVAL=low (input to the NAND2) correctly enables an early evaluation of stage N, this trigger signal is non-persistent: the control output could incorrectly get de-asserted. Therefore, for correct operation, a takeover signal (PC=low) is required to arrive at the gate input, *before* EVAL is de-asserted (see Section 2.7).

Once this timing assumption on the arrival of PC is satisfied, the C-element can safely be replaced by the combinational gate. As shown in Figure 10, the NAND2 gate is identical to the logic portion of the generalized C-element, but with one extra parallel PMOS transistor, controlled by PC. This modification makes the gate fully complementary, hence the two output inverters can be deleted.

The net effect of eliminating two inverters from the critical path is the elimination of *four inverter delays* from the cycle time of the pipeline, because the PA0 critical path for stage N goes through two of these C-elements: the C-element of stage $N+1$, and that of stage $N+2$.

Interestingly, while Williams points out that the throughput of PA0 is likely to be worse than that of PS0 [16], we have observed a significant throughput improvement in LP3/1 (see Section 8). We believe this improvement is brought about partially because of the elimination of the extra inverters.

5. Interface with the Environment

This section presents modifications to the pipeline designs in order to handle two special types of environments. The first type is an environment that can only absorb one control input, whereas some of the pipeline designs use two control inputs per stage. The second type is an unusually slow environment that cannot meet certain timing requirements for correct operation.

5.1. Handling Environments that Cannot Absorb Two Control Inputs

For LP3/1, LP2/1 and LP$_{SR}$2/1, the presence of two control inputs per stage implies that the input and the output environments must have the capability to adequately handle both PC as well as EVAL control signals. However, if the environment can only handle one control signal, the simple scheme of Figure 11 is proposed to address the situation. At the input interface, the environment simply uses a NAND gate to combine the two control signals, PC and EVAL, *exactly as each of the pipeline stages does so internally* (cf. Figure 4). At the output interface, the last pipeline stage derives its EVAL input as a delayed version of the PC provided by the environment; the delay should correspond to the precharge delay of the last pipeline stage. Clearly, this scheme adds no extra delays to either the cycle time or the latency.

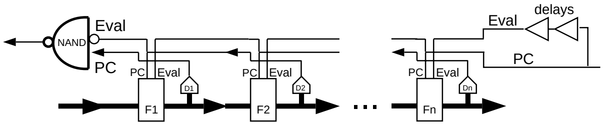

Figure 11. Interfacing an LP3/1 pipeline with the environment

5.2. Handling Unusually Slow Environments

Normally, the environment is expected to be reasonably fast. In the event that it is unusually slow, the whole range of designs in this paper, including Williams' PS0, will malfunction. In the following, a generic and modular solution is proposed, to allow all of these designs to robustly handle such environments.

[6] Theoretically, for an inverting output from the C-element, the internal node of the C-element could be tapped. However, in our experiments, electrical simulations indicated this not to be very reliable.

The problem arises if the left environment is very slow in precharging.[7] Suppose the leftmost pipeline stage has signaled the environment to precharge, but the environment takes too long to precharge. Then, two potential problems could arise: (i) the leftmost stage could subsequently de-assert the precharge signal to the environment before the environment has completed precharge, or (ii) the leftmost stage could get precharge-released before the stale data inputs from the environment have been reset. In either case, an invalid data token might appear on the datapath, causing the pipeline to malfunction. (Interestingly, the root of this problem is an assumption made in Williams' PS0 [16, pp. 33–34], that "precharges are fast enough.")

Our solution is simply to add additional synchronization between the environment and the leftmost pipeline stage. Until the environment has actually completed precharging, two critical events will thus be delayed: precharge-release of the environment, and precharge-release of the leftmost pipeline stage. This mechanism ensures that the environment resets properly, and that the leftmost stage does not evaluate prematurely.

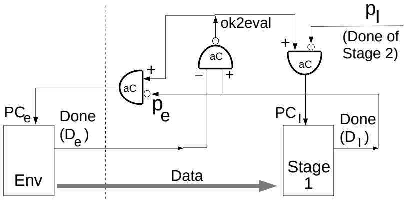

Figure 12. Handling Slow Environments

An implementation of this solution requires modification of the control of the environment as well as the leftmost pipeline stage (Figure 12). A state variable, *ok2eval*, is introduced to keep track of whether the precharge of the environment is complete. The following production rules [8] describe the behavior of the circuit:

$$\begin{array}{rcl}
\overline{De} & \longrightarrow & ok2eval\uparrow \\
[De\wedge]\, D_I & \longrightarrow & ok2eval\downarrow \\
p_e & \longrightarrow & PCe\downarrow \\
p_I & \longrightarrow & PC_I\downarrow \\
\overline{p_e} \wedge ok2eval & \longrightarrow & PCe\uparrow \\
\overline{p_I} \wedge ok2eval & \longrightarrow & PC_I\uparrow
\end{array}$$

While the pipeline cycle time does increase by one gate delay due to the added circuitry at the interface, in practice this is not a serious overhead because the real bottleneck to high throughput is actually the slow environment.

6. On the Practicality of Gate-Level Pipelining

While it may appear that the overhead of completion detection may severely limit the performance of fine-grain pipelines, in practice this overhead can be kept very low.

So far, for simplicity of exposition, this paper has only shown *linear pipelines:* each stage appears to have a single completion detector for the *entire width* of the datapath. Such a scheme would clearly suffer from a significant throughput overhead if the datapath were wide.

In fact, in many cases, this is not the appropriate scheme. As an example, in a pipelined 32-bit ripple-carry adder, each stage only depends on three inputs: two data bits and one carry-in. Therefore, in a gate-level pipeline implementation, each stage would simply have three inputs, two outputs, and a completion detector on *only* those outputs (and acknowledging only the corresponding input sources). As a different example, an array multiplier, where each stage can now be 32 bits wide, a similar scheme can still be used, because the dependence of each output bit is localized: based on only a small number of inputs. Therefore, in such a pipeline, completion detection is inexpensive: involving one, or at most a few, bits. Thus, the overhead of completion detection in gate-level pipelining, in practice, is often quite low.

7. Pipeline Initialization

Initialization of our pipelines is achieved very simply by making use of an additional global "reset" input to every pipeline stage. The *reset* input forces a precharge of every stage's function logic, and in parallel, it forces the completion signal generators (completion detectors in dual-rail designs, and bundling signal generators in single-rail designs) to go low. This resetting is effected by simply adding an extra pull-up transistor to every logic gate and completion signal generator. Once the pipeline is thus initialized, *reset* is de-asserted. The pipeline is now ready for operation.

8. Results

This section presents the results of HSPICE simulations of our new pipeline designs. Results of simulations of Williams' PS0 pipeline are also presented, to serve as the base case for comparison.

Experimental Setup. We chose a 4-bit wide FIFO as our test vehicle. We simulated 10 stages of the FIFO in HSPICE using a $0.6\mu\mathrm{m}$ HP CMOS process. The operating conditions were 3.3V power supply and $300°\mathrm{K}$.

A significant effort was made to fine-tune the transistor sizing of each of the designs. In our view, a comparison of the throughputs of various pipeline designs is most fair only when each of them is fine-tuned for optimal throughput. Therefore, a detailed analysis of capacitive loading on each node of the control circuit was done, to arrive at the optimum transistor sizes. Further, identical datapaths were

[7] Note that other situations, such as those in which environments are slow in computing, or environments are extremely fast, do not pose any problems; our protocols clearly handle those situations.

used in all of the designs, with the following sizes for the transistors in the dynamic gate: the W/L of the precharge PMOS transistor was $24\lambda/2\lambda$, and the W/L of the two series NMOS transistors was $18\lambda/2\lambda$. Further, for each of our designs, the simulations indicated that the timing constraints of Section 2.7 were met: there was at least a 0.24ns safety margin for precharge pulse-width (almost 100% margin), at least a 0.40ns safety margin for safe takeover, and at least a 0.55ns safety margin for input hold time.

8.1. Simulation Results for Dual-Rail Pipelines

The operation of the 4-bit FIFO was simulated for each of the three new dual-rail pipeline designs—LP3/1, LP2/2 and LP2/1—as well as Williams' PS0 design.

Table 1 summarizes the results of the simulation. For each of the four pipeline styles, the table lists the overall pipeline cycle time T, as well as a breakdown of the cycle time into the following components:

t_{Eval} time for a stage evaluation

t_{Prech} time for a stage precharge

t_{CD} delay through the completion detector (average of the up and down transitions). This includes the delay through the buffers that amplify this signal to provide sufficient drive.

t_{NAND} For LP3/1 and LP2/1, this is the delay through the NAND2 gate that combines the two control inputs into one (see Figure 4).

Finally, the throughput of each pipeline, in million data items per second is expressed as a percentage improvement over the throughput of PS0.

The throughput of each of our designs is significantly higher than that of PS0. In agreement with our analysis of Sections 2.2–2.6, the throughput increases in the following order: PS0, LP3/1, LP2/2 and LP2/1. As expected, LP2/1 delivers the highest throughput of all four designs, 860 million data items per second: this rate is an improvement by more than a factor of 2 over PS0 (420 million data items per second). Our other two designs, LP3/1 and LP2/2, also posted higher throughputs: 590 and 760 million data items per second respectively, which are 40% and 79% higher than PS0.

The improvement in throughput is principally due to two factors: (i) improved protocols, and (ii) faster completion detectors. The table confirms the earlier analytical results on the new protocols. In each new design, since there are fewer component delays, overall cycle time is reduced. The second factor is faster completion detection. Column t_{CD} indicates that in two of our designs—LP2/2 and LP2/1—the completion detector delay is significantly lower. The reason is that these two designs use an asymmetric C-element with a very short pull-up stack (Figure 6). In contrast, the completion detectors of PS0 and LP3/1 use a symmetric C-element which is slower.

Finally, note that the latencies of our stages (t_{Eval} and t_{Prech}) are essentially the same as in PS0. Hence, our throughput improvements are obtained without degrading latency.

8.2. Simulation Results for Single-Rail Pipelines

The operation of the 4-bit FIFO was simulated for both of the new single-rail bundled-datapath designs—LP$_{SR}$2/2 and LP$_{SR}$2/1.

Table 2 summarizes the results of our simulation. For each of the pipelines, the overall pipeline cycle time T is shown, as well as the delays of individual components: stage evaluation time (t_{Eval}), stage precharge time (t_{Prech}), the delay though the control block (t_{gC}), and in the case of LP$_{SR}$2/1, the delay through the extra NAND gate (t_{NAND}).

The two new designs, LP$_{SR}$2/2 and LP$_{SR}$2/1, deliver very high throughputs: 1050 million and 1208 million data items per second, respectively. As expected, the throughput of LP$_{SR}$2/1, which combines both early evaluation and early done protocols, is better than the throughput of LP$_{SR}$2/2.

Comparison with the Pipelines of Molnar *et al.* By far the most competitive pipelines presented in literature are the two FIFO designs by Molnar *et al.* [10]. We now compare the performance of our pipelines with that of Molnar's with respect to throughput, latency, area and robustness.

Throughput and Latency. The throughputs of our FIFO designs (1.05 and 1.2 Giga per second) compare quite favorably with that of Molnar's first FIFO design (1.1 Giga per second). Molnar's second FIFO design has a throughput rate of 1.7 Giga per second. However, this design in fact uses *two* parallel FIFO's—one for the odd numbered data items, and another one for the even numbered items—in order to achieve this performance; the throughput of this design can not directly be compared with the throughput of our FIFO's, which use only a single data stream.

For a processing pipeline, as opposed to a FIFO, the performance of our designs is likely to be even better, relative to that of Molnar's designs. When logic processing is desired in Molnar's pipelines, extra logic gates must be inserted *between* the latches of the FIFO. This insertion of gates directly adds to the pipeline cycle time, as well as to the latency. In contrast, our designs use dynamic stages which provide latching as well as logic functionality in the *same* dynamic gate. Adding functionality to a dynamic stage merely involves adding transistors to the evaluate stack of the dynamic gate, which should only marginally adds to the cycle time and latency (*e.g.,* see Figure 2).

Chip Area. Our new designs should occupy significantly less chip area than Molnar's FIFO's. In particular, our designs do not use explicit latches; latching is provided implicitly by the dynamic function blocks. In contrast, Molnar's asp* FIFO uses explicit pass transistor latches which are significantly more area-expensive. Molnar's second design is even more area-expensive: it uses two parallel latches for each stage, and each one is a large capture-pass latch.

					Cycle Time, T		Throughput	
Pipeline Design	t_{Eval} (ns)	t_{Prech} (ns)	t_{CD} (ns)	t_{NAND} (ns)	Analytical Formula	(ns)	10^6 items per sec.	% increase over PS0
LP3/1	0.24	0.26	0.72	0.26	$3 \cdot t_{Eval} + t_{CD} + t_{\text{NAND}}$	1.70	590	**40%**
LP2/2	0.22	0.26	0.45	-	$2 \cdot t_{Eval} + 2 \cdot t_{CD}$	1.33	760	**79%**
LP2/1	0.22	0.25	0.38	0.36	$2 \cdot t_{Eval} + t_{CD} + t_{\text{NAND}}$	1.18	860	**102%**
PS0	0.25	0.25	0.68	-	$3 \cdot t_{Eval} + 2 \cdot t_{CD} + t_{Prech}$	2.38	420	**Base**

Table 1. A comparison of the performance of dual-rail LP pipelines vs. Williams' PS0.

					Cycle Time, T		Throughput
Pipeline Design	t_{Eval} (ns)	t_{Prech} (ns)	t_{gC} (ns)	t_{NAND} (ns)	Analytical Formula	(ns)	10^6 items per sec.
LP$_{\text{SR}}$2/2	0.19	0.21	0.29	-	$2 \cdot t_{Eval} + 2 \cdot t_{gC}$	0.95	1050
LP$_{\text{SR}}$2/1	0.19	0.21	0.26	0.19	$2 \cdot t_{Eval} + t_{gC} + t_{\text{NAND}}$	0.83	1208

Table 2. The performance of single-rail LP$_{\text{SR}}$2/2 and LP$_{\text{SR}}$2/1.

Timing Constraints. Molnar's first design, based on an asp* protocol, has complex timing assumptions which are not explicitly formalized; in fact, an early version was unstable due to timing issues. In contrast, our designs have simple one-sided timing constraints, which were very easily satisfied in our example FIFO designs.

In summary, our new pipelines, LP$_{\text{SR}}$2/2 and LP$_{\text{SR}}$2/1, offer significant advantages over Molnar's designs, for each of the following parameters: throughput, latency, area and ease of satisfiability of timing constraints.

9. Conclusions

This paper presents several new asynchronous pipeline control structures for dynamic datapaths. These structures are especially suitable for very fine-grained, or gate-level, pipelines where each stage consists of a single gate. The dual-rail designs offer significantly higher throughput than a well-known design style by Williams, while maintaining the same forward latency. The single-rail designs are competitive with the leading recent approaches by Molnar *et al.*, while offering significant advantages in area, latency and robustness.

Acknowledgments. The authors gratefully acknowledge Naeem Abbasi and Prof. Ken Shepard of Columbia University for their help with simulations and modeling issues. We also benefited from helpful discussions with Michael Theobald and Tiberiu Chelcea. We also thank the anonymous reviewers for insightful comments, especially for suggesting that we address the problem due to slow environments.

References

[1] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Taubin, and A. Yakovlev. Lazy transition systems: application to timing optimization of asynchronous circuits. In *ICCAD*, 1998.

[2] P. Day and J. V. Woods. Investigation into micropipeline latch design styles. *IEEE TVLSI*, 3(2):264–272, June 1995.

[3] A.E. Dooply and K.Y. Yun. Optimal clocking and enhanced testability for high-performance self-resetting domino pipelines. In *ARVLSI'99*.

[4] C. Farnsworth, D. Edwards, and S. Sikand. Utilizing dynamic logic for low power consumption in asynchronous circuits. In *Proc. Intl. Symp. Adv. Res. Async. Circ. Syst. (ASYNC)*, 1994.

[5] S. B. Furber and J. Liu. Dynamic logic in four-phase micropipelines. In *Proc. Intl. Symp. Adv. Res. Async. Circ. Syst. (ASYNC)*, 1996.

[6] D. Harris and M.A. Horowitz. Skew-tolerant domino circuits. *IEEE JSSC*, 32(11):1702–1711, November 1997.

[7] R. Kol and R. Ginosar. A doubly-latched asynchronous pipeline. In *Proc. ICCD*, 1996.

[8] A. J. Martin, A. Lines, R. Manohar, M. Nystroem, P. Penzes, R. Southworth, and U. Cummings. The design of an asynchronous MIPS R3000 microprocessor. In *Proc. ARVLSI*, September 1997.

[9] G. Matsubara and N. Ide. A low power zero-overhead self-timed division and square root unit combining a single-rail static circuit with a dual-rail dynamic circuit. In *ASYNC97*.

[10] C.E. Molnar, I.W. Jones, W.S. Coates, J.K. Lexau, S.M. Fairbanks, and I.E. Sutherland. Two FIFO ring performance experiments. *Proceedings of the IEEE*, 87(2):297–307, February 1999.

[11] V. Natayanan, B.A. Chappell, and B.M. Fleischer. Static timing analysis for self resetting circuits. In *Proc. ICCAD*, 1996.

[12] A. M. G. Peeters. *Single-Rail Handshake Circuits*. PhD thesis, Eindhoven University of Tech., June 1996.

[13] M. Renaudin, B. Hassan, and A. Guyot. New asynchronous pipeline scheme: Application to the design of a self-timed ring divider. *IEEE JSSC*, 31(7):1001–1013, July 1996.

[14] K. S. Stevens, S. Rotem, and R. Ginosar. Relative timing. In *Proc. Intl. Symp. Adv. Res. Async. Circ. Syst. (ASYNC)*, April 1999.

[15] I. E. Sutherland. Micropipelines. *Communications of the ACM*, 32(6):720–738, June 1989.

[16] T.E. Williams. *Self-Timed Rings and their Application to Division*. PhD thesis, Stanford University, June 1991.

[17] D.C. Wong, G. De Micheli, and M. Flynn. Designing high-performance digital circuits using wave-pipelining. *IEEE TCAD*, 12(1):24–46, January 1993.

[18] K.Y. Yun, P.A. Beerel, and J. Arceo. High-performance asynchronous pipeline circuits. In *Proc. Intl. Symp. Adv. Res. Async. Circ. Syst. (ASYNC)*, 1996.