

A Counterflow Pipeline Experiment

Bill Coates, Jo Ebergen, Jon Lexau, Scott Fairbanks, Ian Jones,
Alex Ridgway, David Harris, Ivan Sutherland
Sun Microsystems Laboratories, Palo Alto, USA

Abstract

The counterflow pipeline architecture [12] consists of two interacting pipelines in which data items flow in opposite directions. Interactions occur between two items when they meet in a stage. We present the design decisions for, and test measurements from, an asynchronous chip that explores the basic ideas of such an architecture. We built the chip in order to confirm proper operation of the arbiters required to ensure that each and every item flowing in one direction interacts with each and every item flowing in the other direction.

Our chip, named “Zeke,” was built in 0.6 μm CMOS through the MOSIS fabrication facility. The maximum total throughput of the chip, which is the sum of the throughputs of the two pipelines, varies between 491 MDI/s (Mega Data Items per second) and 699 MDI/s, depending on the amount of interaction that takes place. Under average data and operating conditions the performance of our chip was roughly halfway between these throughput values.

*“...their construction being as it were
a wheel within a wheel.”
Ezekiel, Chapter 1, verse 16.*

1. Introduction

In [12] Sproull, Sutherland, and Molnar proposed a counterflow pipeline architecture consisting of two interacting pipelines in which data items move in opposite directions. Such an architecture can be used to implement a processor, where one pipeline carries instructions and the other carries results. An essential feature in a counterflow pipeline processor is that each instruction will meet each counterflowing result in some stage, where the instruction and result will interact. The interaction includes the checking of whether an instruction’s operand address matches a counterflowing “result” address and, if so, copying a result value into an operand field of the instruction, an operation we call “garnering.”

The counterflow pipeline architecture was a new architecture that offered many research opportunities. For the past several years this architecture served as a research focus of our Asynchronous Systems Group at Sun Microsystems Laboratories. The initial years were spent investigating the general architecture, while more recently we have concentrated on the design of fast basic circuitry for pipeline control [1, 7, 8].

When we reached the point at which we were prepared to implement and test the basic ideas of the counterflow pipeline, we undertook the building of a counterflow pipeline test chip as “the ultimate verification,” as Charles Molnar would have said.

During the design process we faced several challenges. First, how to design a counterflow pipeline for which we can test correct operation. This was not an easy task, given the high degree of parallelism and arbitration present in the architecture. In particular, we wished to verify correct functionality of the design: that each instruction meets each counterflowing result, that address matching is correctly carried out, and that result garnering occurs whenever there is an address match.

A second challenge was speed. How fast could we make the counterflow pipeline operate, and how would we measure its performance? We have experience with several design styles suitable to implement pipeline control circuits. We selected the asynchronous, symmetric, persistent, pulse protocol (asP*), because it provided high performance without too much complexity and easy interfacing for testing the chip. Furthermore, because the on-chip circuitry could operate at a much higher speed than our off-chip measurement equipment, we designed the chip to permit simple measurement of the on-chip speed with off-chip equipment.

The third challenge was to optimize our learning experience. Could we design the chip such that we could learn some general lessons about counterflow pipeline architectures? For example, we wanted to know how the total throughput of the counterflow pipeline varied as a function of the number of items in each pipeline and as a function of the number of items with matching

addresses. As a side benefit of this exercise we expected to gain more experience and confidence in building fast asynchronous circuits.

These challenges influenced several design decisions, most of which are reported in Section 2. In Section 3 we explain the basic operation of the control circuits and the data path. Section 4 describes the implementation of the arbiters, while Section 5 reports the performance results measured from the fabricated chips. Section 6 explains how we tested the arbiters on our chip. We conclude with some lessons that we have learned from this design.

2. General structure of the chip

In order to keep the chip relatively simple and fully testable we chose the following design. Our test chip consists of two counterflowing rings of 28 stages, one ring for the so-called northbound items and the other ring for the so-called southbound items, as shown in Figure 1. We dispensed with the distinction between instructions and results, treating the data in each pipeline simply as “items.” We chose rings as opposed to straight pipelines so that we could easily run the system continuously at its maximum speed, thus allowing us to measure throughput as a function of the number of full cells in a way similar to that used in previous ring designs [1, 7, 8, 14].

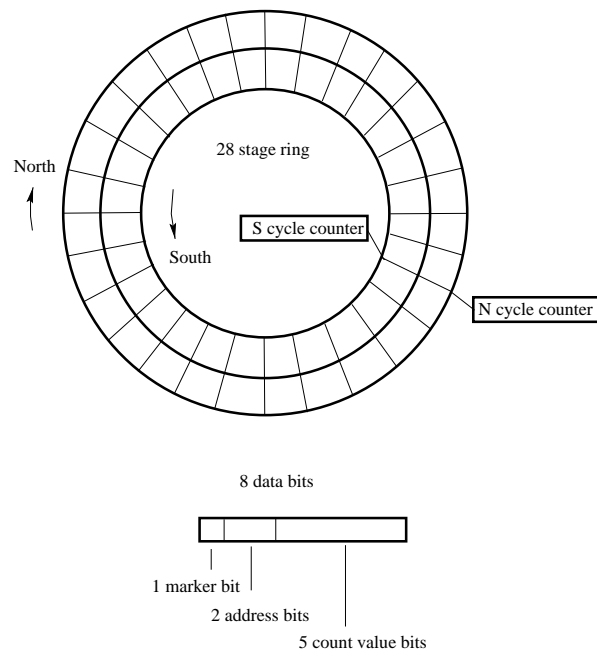


Figure 1. A 28-stage counterflow ring with 8-bit data items

For the correct functionality of the counterflow pipeline, it is essential that a north and southbound item do not miss each other when they try to cross the same stage boundary in opposite directions. In other words, the chip design must prevent simultaneous crossings of a stage boundary by a north and southbound item. We used arbiters to prevent simultaneous crossings of a stage boundary by a northbound and a southbound item. The main purpose of the chip was basically to test this mutual exclusion behavior. We wanted a design in which we could detect even a single arbitration failure in billions of trials. However, we also wanted simplicity in the design.

We devised the following test structure for the counterflow pipeline. Each data item consists of a two-bit address, a five-bit count value, and one marker bit. Whenever a north and southbound item meet in a stage, their two-bit addresses are compared. If the addresses match, the count value in each of the meeting items is incremented modulo $2^5 - 1 = 31$; otherwise both count values remain unchanged. The reason for incrementing modulo 31, and not modulo 32, is that we use a Linear Feedback Shift Register (LFSR) counter. The increment process models the “garnering” operation of the general counterflow pipeline architecture.

In addition to the count value associated with each north and southbound item, there is also a cycle counter associated with one of the stage boundaries in each ring. In our tests we initialize the data items in the rings such that only one marker bit is set in each ring. Each cycle counter is incremented when an item with a set marker bit passes its stage boundary. Each cycle counter holds 48 bits, which is sufficient to allow the rings to run for at least two days without cycle counter overflow.

Additional circuitry is used to initialize the North and South sides of each stage “full” or “empty,” and we can read and write the values of the address, count value, and marker bits. We can also read and write the values of the cycle counters and start and cleanly stop the rings at any time by means of an external signal. Thus we are able to initialize the two rings and the two cycle counters with known values, let them run for a while, stop them, and then read the contents of each stage and the cycle counters.

From the start and end positions of all the items and the cycle counter values, we can calculate what each item’s count value should be, modulo 31. Any discrepancies between the calculated and actual count values indicates that items missed each other, e.g., due to arbitration failures. There is a very small probability that errors are masked due to the modulo 31 counters.

Although our test chip has a structure similar to the original counterflow pipeline proposal, its data opera-

tions are much simplified. The address comparison between a result and an operand is represented by a simple two-bit comparison, and the garnering action is replaced by independent increments to five-bit count values. However, with this stripped-down version of the structure, we are able to thoroughly investigate all of the critical components in the architecture, run at full speed, and detect failures in the system.

Additional performance measurements of our chip was carried out via special output pads. A couple of pads monitored the low-order bits of the cycle counters, enabling us to measure the throughput of the pipeline rings. Because we can initialize the rings to any state, we can measure the throughput of the rings as a function of the number of items in each ring and the number of matching items.

Having chosen the chip architecture and the test procedure, the next challenge was finding a fast implementation. Speed is very important to us, more so than low power consumption or modularity. We had several control and data circuits styles to choose from. The original proposal [12] gives an implementation based on transition signaling, transition arbiters, and single-rail data paths. We also considered the pipeline implementations given in [1] and [7]. In addition to these pipeline implementations, we looked into various schemes to implement the arbitration required at boundary crossings.

For reasons of speed, simplicity, and testability, we chose our asynchronous, symmetric, persistent, pulse protocol (asP*) implementation [7] and single-rail data paths for the pipelines. Although pipeline implementations based on transition-signaling can be faster than asP* pipelines [8], the additional requirement of the arbitration makes asP* pipelines preferable. We implemented the arbitration with a single mutual exclusion element (MUTEX) between stages.

Our experiences indicate that using a return-to-zero protocol, such as asP*, is easier to manage for testing purposes, and it simplifies the task of starting, stopping, loading, and unloading the data items by means of external signals. Also, when the chip or test program do not behave as expected, it is easier to diagnose the problem. For our test procedure to work, it was important that the start and stop mechanisms avoid corrupting the data in the rings. The circuit design for synchronized stopping of pipelines, which we used earlier in unidirectional pipelines [7], also turned out to work well in our counterflow pipeline design and added only a slight performance penalty.

We heavily optimized the circuit design using the theory of “Logical Effort” [13] combined with SPICE simulations to select transistor sizes.

Finally we made some design decisions to simplify

the testing. We chose to make each stage accessible through an externally addressable bus for loading and unloading its contents. As a consequence we did not need a special interface stage, and all stages could be made identical. Thus, in this design, there was no single bottleneck stage in the pipelines, as we have had in all of our previous chips [1, 7, 8]. Making all stages the same also simplified the layout.

3. Basic Control Circuit

The control circuit of our counterflow test chip is based on the asP* protocol. We briefly review the basic principles of the asP* protocol in this section, and in the next sections we expand the basic control circuit to arrive at the control circuit for the counterflow pipeline.

The control circuit shown in Figure 2 controls the movement of data items in a unidirectional pipeline. Each stage in the pipeline is implemented by an SR latch, and each boundary between stages is implemented by an AND gate. Throughout this paper, the SR latches are implemented with NOR gates, so when one of the inputs S or R is high, the latch will be *set* or *reset* respectively –this is to simplify the description of the circuit operation. In our chip we implemented the latches with NAND gates to obtain slightly better performance.

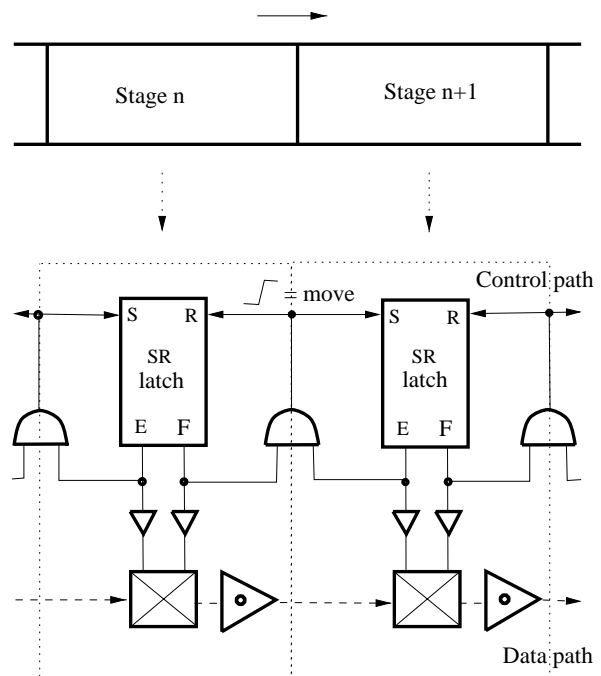


Figure 2. The basics of an asP* control circuit

The data path shown consists of standard data latches, represented here as pass gates (boxes with a cross in Figure 2) followed by a sticky buffer, i.e., buffers with a weak positive feedback (triangles with a dot in Figure 2). The SR latches control the pass gates in the data path. If the control latch is reset, the pass gate it controls is made transparent, and we say that the stage is empty. If the SR latch is set, the pass gate it controls is made opaque, and we say that the stage is full. The outputs E and F indicate whether the stage is empty or full respectively. Although normally these outputs are complementary, during operation there may be short periods when this is not the case, due to the internal delays of the SR latch.

Whenever a stage is full and its righthand neighbor is empty, the AND gate between the two SR latches generates a rising transition. This transition initiates a *move* pulse. The rising transition sets the right SR latch and resets the left SR latch, thereby filling the right stage and emptying the left stage. Shortly after the right SR latch is set, its E output falls, and concurrently shortly after the left SR latch is reset, its F output falls. The first of these falling outputs causes the AND output to fall, ending the move pulse.

For the asP* protocol to operate correctly, delay constraints need to be satisfied. A sufficient condition for correct operation is that an SR latch is never simultaneously set and reset. This condition can be translated into the requirement that move pulses on the S and R inputs of an SR latch must not overlap. A simple way to realize this condition for the implementation above is to ensure that falling transitions propagate at least as fast as rising transitions from the F output to the R input and from the E output to the S input of the latches. When this delay constraint for the asP* protocol is satisfied, move operations into and out of a stage strictly alternate.

Data movement from one stage to the next is directly controlled by the outputs of the SR latches. When a latch is in the empty state, the pass gate that it controls is transparent, and when the latch is full, the pass gate is opaque.

In order to implement data movement correctly, some delay constraints between the control circuits and the data path must also be met. In particular, the set-up and hold times of the data latches must be satisfied. These set-up and hold times are closely related to the “bundling constraints” of a bundled data path.

4. Mutual Exclusion and Interaction

Early in the project, Charles Molnar identified that there were two critical issues that made the counterflow pipeline interesting: arbitration and ensuring that when

two items meet in a stage that they interact. It was these design requirements that appeared to prevent a counterflow pipeline stage being synthesized by the various tools available circa 1994 and that led to the design challenge SCPP-A [6].

Molnar illustrated the two issues in what has now become known as the “Molnar five-state diagram,” shown in Figure 3. The southbound moves in and out of a stage

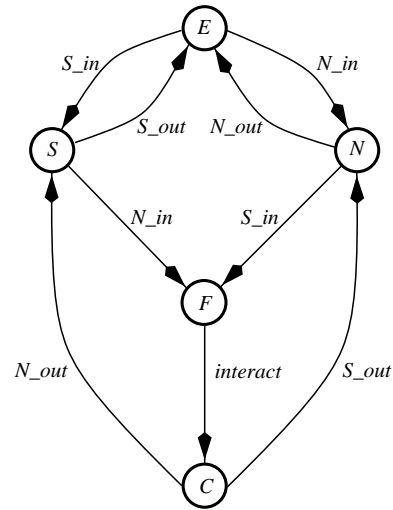


Figure 3. Molnar's five-state diagram

are represented by S_{in} and S_{out} respectively. A similar meaning applies to N_{in} and N_{out} for northbound moves. The state labels have the following meaning: E = “the stage is empty”, F = “the stage is full”, N = “the stage contains a northbound item”, S = “the stage contains a southbound item”, and C = “the interaction between the two items has completed”. The five state diagram clearly specifies that when a stage becomes full, a move out of the stage can take place only when the interaction between items has completed. The five state diagram also specifies, but perhaps less clearly, that moves across a boundary must be mutually exclusive.

To arrive at the complete asP* implementation of our counterflow pipeline ring, we extended the basic control circuit. The first extension enforces mutual exclusion between moves across stage boundaries and ensures that, when items meet in a stage, they interact before leaving the stage.

Figure 4 shows the addition of mutual exclusion. For clarity, the data paths are not shown in this figure. Illustrated are two asP* control circuits of counterflowing pipelines. At each stage boundary we have added a MUTEX, shown in the figure as “ME.” A rising transition at an input of the MUTEX denotes a request to move data, while a falling transition denotes a release of a grant,

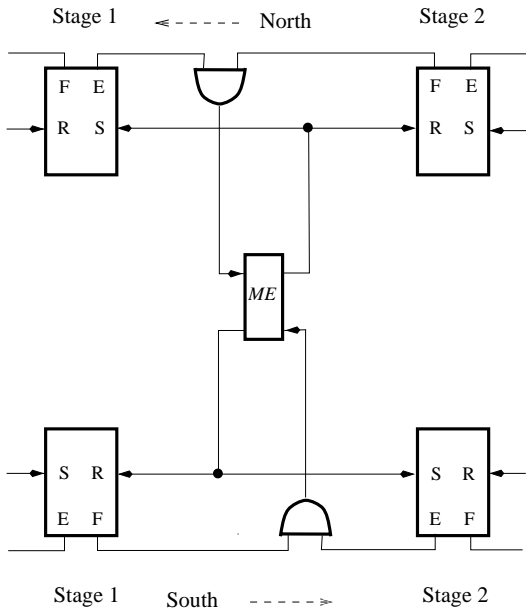


Figure 4. Addition of mutual exclusion

also called the “Done” signal. A MUTEX guarantees that at any time at most one request will be granted. A MUTEX can be built with cross-tied NAND gates and a simple metastability filter, which results in a very fast design, see [3, 11] for example.

Although this implementation prevents concurrent moves across a common boundary, it does not allow enough time for interactions between items to complete, as the following example illustrates. Suppose that a northbound item wants to leave stage 2 and a southbound item wants to leave stage 1. Suppose furthermore that the requests for both moves arrive simultaneously at the MUTEX and that the MUTEX grants the southbound item first. Once the southbound item arrives in stage 2, the falling transitions at the outputs of the latches will release the MUTEX. The MUTEX can then immediately grant the northbound item to leave stage 2. Because of the move of the southbound item into stage 2, however, stage 2 now contains two items, and therefore an address comparison and possibly count value increments must take place before the northbound item may leave stage 2. Unfortunately, the implementation in Figure 4 does not force the northbound item to wait until these actions have completed.

Adding the completion detection circuits is shown in Figure 5. Besides the MUTEX, we have added two AND gates and so-called FULL/EMPTY boxes to the basic asP* control circuits. The MUTEX and the two AND gates are enclosed in a dashed box. We call this part a COP, because it directs the traffic between two

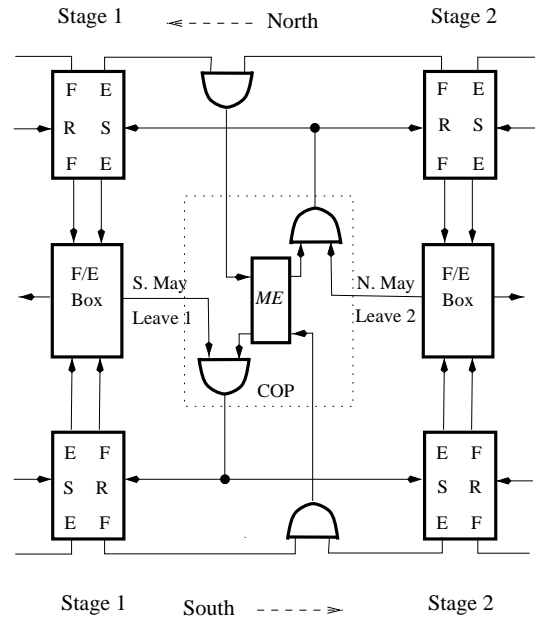


Figure 5. Addition of mutual exclusion and completion detection

adjacent stages. A FULL/EMPTY box detects whether a stage contains two items and, if so, whether the addresses of the items match. It has as inputs copies of the F and E outputs of the control latches and the address wires of the data paths. The address wires are not shown in Figure 5. The FULL/EMPTY box has two outputs: “North May Leave” and “South May Leave,” which indicate whether a northbound or southbound item may leave that stage.

The FULL/EMPTY boxes and the COPs complete the requirements for correct counterflow operation as follows. In order to prevent any northbound item from leaving stage 2 when a southbound item enters stage 2, the FULL/EMPTY box will de-assert the “North May Leave” signal as soon as the southbound item enters stage 2 and before the MUTEX can issue a grant to a northbound item. The “North May Leave” signal is asserted again only when it is safe for a northbound item to leave stage 2, that is, after an address comparison and a possible count value increment have taken place.

From the above example we can see that the COP controls movement of items across a stage boundary in two steps. First, it decides by means of the MUTEX which of the pending moves may proceed and which one will be blocked, if any. Secondly, the COP enables further blocking of a move *out of* a stage until all actions resulting from a move *into* a stage have been completed. The ability to further block a move out of a stage is neces-

sary, because in our implementation each MUTEX will be released a fixed time after it grants a request, irrespective of the actions that may take place as a result of the move. For example, a move can result in a stage becoming full, upon which an address comparison and possible count value increments must take place. In the short time that the MUTEX blocks any pending move out of the stage, the “May Leave” inputs of the AND gates in the COP can be de-asserted to prolong the blocking of the move out of that stage. Once it is safe to do so, the FULL/EMPTY box removes this blocking signal.

There are two important timing constraints that need to be satisfied in this complete counterflow circuit. The first constraint is the same as for the basic asP* control circuit: move pulses at the inputs of a control latch may not overlap. In order to meet this delay constraint, it is sufficient that the COP delays a rising transition at least as much as a falling transition. The MUTEX helps meeting this condition, because its function is exactly to block a rising transition in the case of contention; a falling input transition is never blocked. The AND gate in the COP can also be made to satisfy this condition, provided the “May Leave” input to the AND gate is used to temporarily block the propagation of only rising transitions. The second timing constraint concerns a race between lowering of the “May Leave” signal and releasing the MUTEX. In the case of contention at a boundary, the “May Leave” signal must be lowered before the MUTEX can be released, a delay constraint that is easy to meet.

The performance impact of our implementation of the counterflow pipeline, when compared to a unidirectional pipeline, is about a factor of two. This arises because the minimum cycle time for this counterflow pipeline implementation is about six gate delays longer than the implementation for the unidirectional pipeline. This difference can be explained as follows. For each boundary the extension to the critical cycle consists of a single COP, which adds about three gate delays. A critical cycle contains two boundary crossings, so the minimum cycle time of the basic asP* implementation is increased by about six gate-delays. Given that the minimum cycle time for a unidirectional asP* pipeline is about 6 gate-delays, we obtain a minimum cycle time for a counterflow pipeline of 12 gate-delays. Of course, when address comparisons and count value increments occur, the duration of that particular cycle in that stage becomes longer.

Although the minimum cycle time of our implementation is 12 gate delays, it is important to note that our control circuit implements three useful, basic functions. First, it implements the usual move operations of a single pipeline. Second, it implements the mutual exclusion condition. And third, it implements a data-

dependent extension of the cycle time. Data-dependent extensions of the cycle time can be useful in connection with completion detection, as shown in [9] for example. We were pleased to be able to implement these extra requirements with such little extra delay.

5 The Data Path

The control circuit also manages conditional data processing in the data path. In our case the data processing is a count value increment. Figure 6 shows part of a control and data path of our counterflow test chip, illustrating the conditional control.

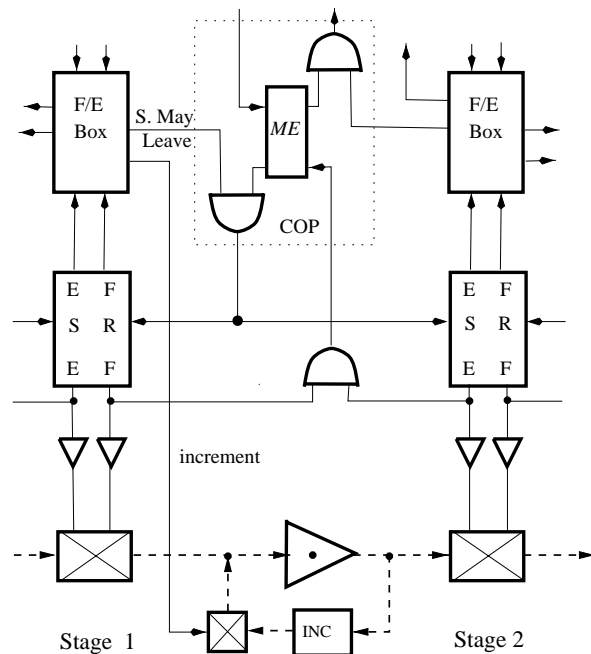


Figure 6. The data path

When the FULL/EMPTY box for stage 1 detects that the stage is full and that the addresses of the items match, the counter value associated with each item must be incremented. For this purpose, the FULL/EMPTY box generates a pulse signal on the “increment” wire, which makes the “increment” pass gate briefly transparent. This pulse has a two-sided delay requirement. The pulse must be long enough to let the incremented counter value pass through the increment gate and overwrite the current counter value; on the other hand the pulse must be short enough so that the counter value will not be incremented twice. Additionally, during the increment pulse the data latch for stage 1 must be opaque.

When the FULL/EMPTY box generates a pulse for the increment pass gate, it also blocks a move signal, if

any, through the COP by de-asserting the “May Leave” signal. The “May Leave” signal remains de-asserted until the increment is complete, after which it is asserted and move signals are enabled again.

The incrementer in the data path is implemented with a Linear Feedback Shift Register (LFSR), because it has about the same increment delay for any value, unlike an adder. Using an LFSR simplified the design of the incrementer, the pulse generator, and the delay analysis.

6. Results

We have built and tested two versions of Zeke. The first version turned out to malfunction in certain circumstances due to a violation of a delay constraint. Although detection of these errors was a good sign that our test procedures worked, the errors prevented us from completing all desired tests.

By means of extensive testing of the chip and simulation of the extracted layout, we traced the cause of the error to a violation of a delay constraint for the COP involving the “May Leave” wire. This error went undetected in the first design, because our SPICE simulations using estimated capacitances showed a safety margin of more than 15% of the total path delay for this particular delay constraint. Experience from previous chips had shown that this margin should be sufficient. Our recent test results, however, indicated that our estimates were in error. Simulations using capacitances carefully extracted from the layout showed that the safety margin for this constraint was close to zero.

Results of the performance measurements for our second chip are given in the figures below. Our second version of the chip turned out to operate about 25% faster than our first version, mostly due to process variations. After extensive testing we have not detected a single error.

Figure 7 shows the total throughput of the counterflow pipeline ring in Mega Data Items per second (MDI/s) under different data conditions, where the number of North items is fixed at 14. We define the total throughput as the sum of the throughput of the northbound ring and the throughput of the southbound ring. The upper curve gives the best-case throughput for our counterflow test chip, which is obtained when no meeting items have matching addresses. The lower curve gives the worst-case throughput, which is obtained when all meeting items have matching addresses. The middle curve gives the average-case throughput taken over many samples, where for each sample the pipelines are filled with random addresses. Because there are only four possible addresses, on average one out of every four meetings has items with matching addresses.

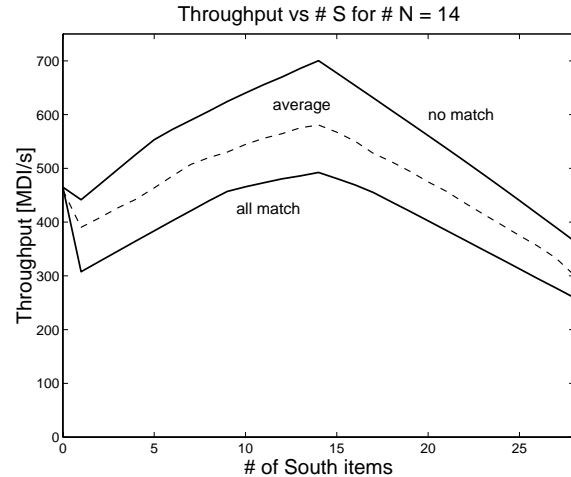


Figure 7. Total throughput versus number of items in the South ring, with 14 items in the North ring

Figure 7 shows that, when we change the number of items in one ring from zero to one, the throughput in the other ring drops markedly. The reason for the drop is that the one item must interact with every item it meets in the other ring. The address matching between items increases the cycle time and, hence, decreases the total throughput. When each ring has at least one item, the total throughput increases almost linearly as the number of items increases. After one of the rings becomes about half full, the total throughput decreases roughly linearly with the number of items in the fuller ring.

Figure 8(a) shows the total throughput as a function of the number of items in the northbound and southbound ring, where all meeting items have different addresses. When one ring is empty, a plot of throughput versus occupancy follows a trapezoid, similar to ones we have seen in other publications [7, 14]. The presence of a short section of a flat top indicates that there is at least one slow stage in the ring, which acts as a bottleneck. At first this may be surprising, given that we had tried to avoid bottlenecks by making all stages identical. The layout of the rings, however, created some asymmetries in the lengths of wires that interconnect the stages. The ring layout is a square with seven stages on each side; the longer wires at the corners result in bottlenecks.

The changes in performance become more pronounced when all the addresses match, as shown in Figure 8(b). When addresses match, the cycle time for that particular interaction increases by the delay to increment a count value, thus reducing the total throughput.

The performance measurements show that the peak

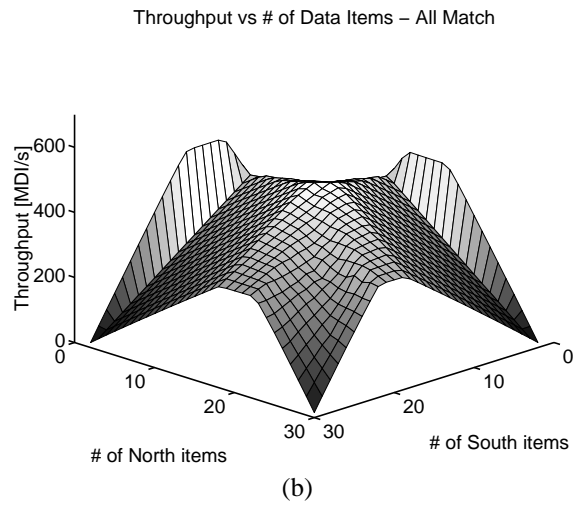
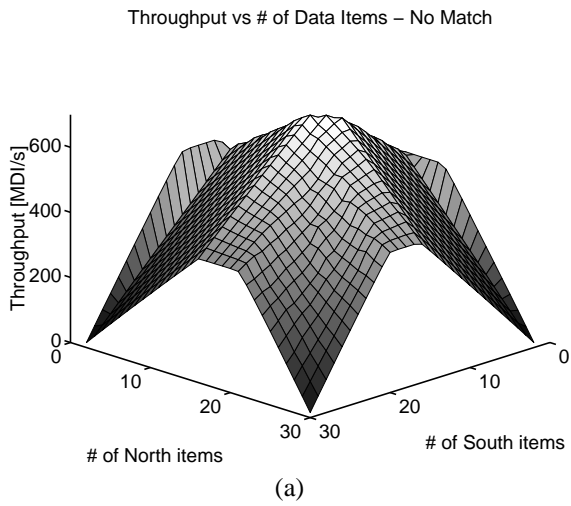
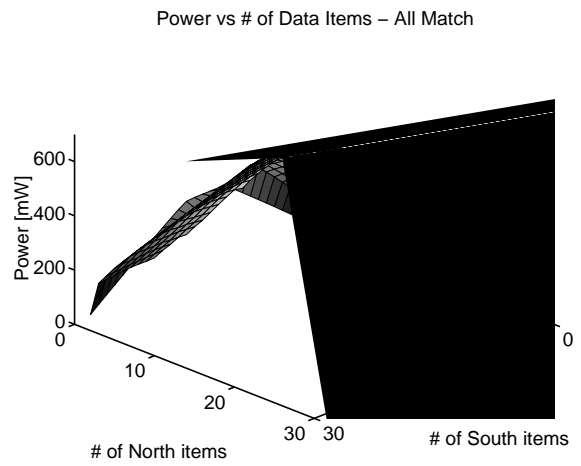
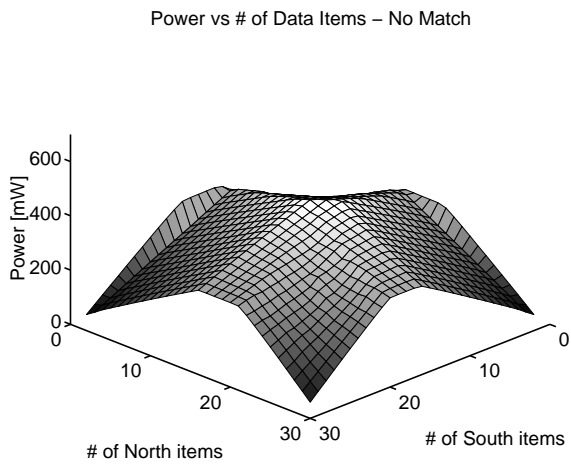


Figure 8. Total throughput versus number of items in the North and South rings when no items match (a) and all items match (b).



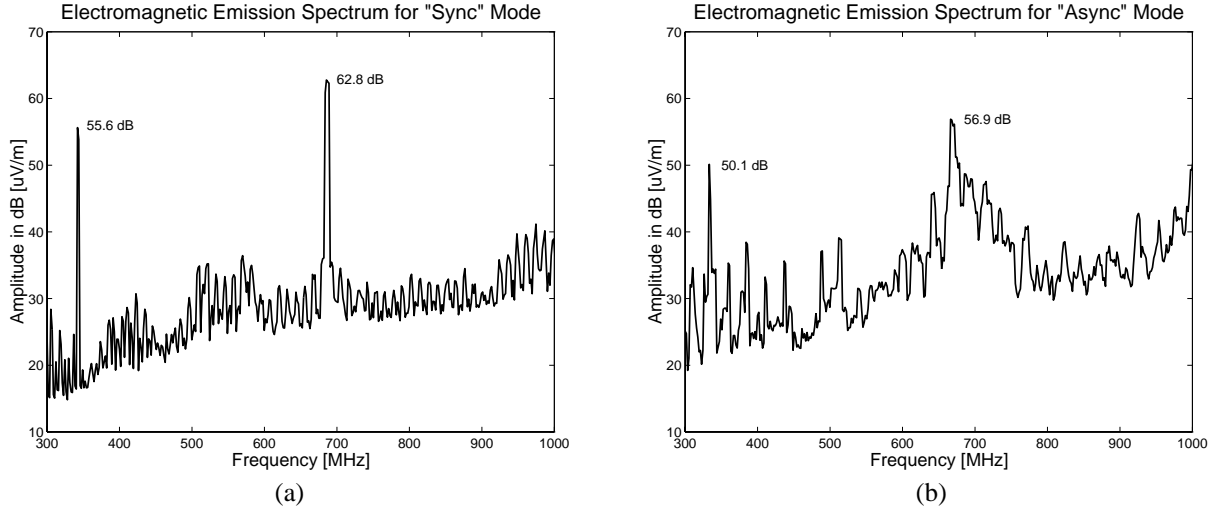


Figure 10. Measured EM Emissions spectrum for “synchronous” mode (a) and “asynchronous” mode (b).

throughput of the North ring is 462 MDI/s and is obtained when the South ring is empty. This throughput corresponds to a minimum cycle time of about 12 gate-delays in 0.6μ technology, where a gate-delay is a fanout-of-4 inverter delay or about 180 ps in this process. In our chip the South ring is the inner ring, and it is slightly faster than the North ring. The peak throughput of the South ring is 498 MDI/s, which corresponds to a minimum cycle time of about 11 gate-delays. These cycle times closely match our predictions from simulations.

The peak total throughput when no addresses match is 699 MDI/s and is obtained when $\#N = 14$, $\#S = 14$ and when $\#N = 15$, $\#S = 13$. The peak total throughput when all addresses match is 491 MDI/s and is also obtained when $\#N = 14$, $\#S = 14$ and when $\#N = 15$, $\#S = 13$. These maximum throughputs also closely match SPICE predictions. SPICE simulations show that the extra delay per boundary crossing for an address match is about 2 gate delays, whereas the extra delay for an address match and an increment is about 5.5 gate delays. For a single ring this gives a cycle time of about $12 + (2 \times 2) = 16$ gate-delays, or about 2.9 ns, when an address match occurs in each stage. This results in a throughput of about 345 MDI/s for a single pipeline, or 690 MDI/s for two perfectly synchronized counterflow pipelines. In case an address match and an increment occurs in each stage, we get for a single ring a cycle time of $12 + (2 \times 5.5) = 23$ gate-delays, or about 4.1 ns. This results in a throughput of about 244 MDI/s for a single pipeline, or 488 MDI/s for two perfectly syn-

chronized counterflow pipelines.

Figure 9 shows the power consumption as a function of the number of items in the North and South ring. When we compare these graphs with the graphs for total throughput, we observe that power consumption is roughly proportional to the total throughput.

We also conducted a number of tests to measure the electromagnetic radiation of the chip in various modes of operation. We carried out these measurements in a 3 meter electromagnetic anechoic chamber. Furthermore, we intentionally enabled the six high-speed output pins of the chip to increase the signal strengths and to ease our measurement comparisons. During the EMI measurements we disconnected the host computer and turned it off, such that the majority of the radiation came from our chip. The upward slope of the base values of the measurements is due to background radiation levels.

Plots of measurements taken for two distinct modes of operation are shown in Figure 10. Figure 10(a) shows the radiation spectrum of the chip when both rings were loaded with 14 items and none of the items in the North ring had a matching address with any of the items in the South ring. Under these circumstances the counterflowing rings lock into a “synchronous” mode of operation, where all items in both rings move one stage forward in lockstep. The synchronous mode of operation is illustrated in Figure 10(a), where the fundamental frequency peak occurs at 342 MHz and the second harmonic at 684 MHz. The fundamental frequency matches the throughput rate we measured from our on-chip cycle counters.

Figure 10(b) shows the counterflowing rings in a more

realistic asynchronous mode of operation that has almost identical throughput to the “synchronous” configuration. In this case the North ring is loaded with 13 items and the South ring with 15 items. Two items in the North ring have addresses that match three items in the South ring. These circumstances give a much more irregular mode of operation than the “synchronous” mode we observed in Figure 10(a). Although the throughput and power consumption for both modes of operation is about the same, the spectrum for the “asynchronous” mode of operation is much more spread out and the heights of the peaks are about 6dB lower. Although these EMI measurements were not conducted with the rigor and precision required for FCC compliance, we found that our measurements were consistent over a number of tests. Our EMI measurements results concur with observations reported in [4, 10], which also show a reduction in emissions and a spreading of the emissions spectrum.

7. Stress Testing The Arbiters

In our counterflow pipeline, arbitration is critical to correct operation, and so we designed a small second experiment on the chip to confirm that the counterflow pipeline would still operate correctly in the presence of severe arbiter contention. Moreover, we wanted to quantify experimentally the additional delay in cycle time due to metastable behavior.

We expected that it would be difficult to force arbiter contention in the 28-stage ring, believing that the ring would lock into modes of behavior in which there is little contention. For this reason we designed and installed on the Zeke chip a separate smaller test circuit that would stress its arbiters.

Our arbiter test circuit consisted of a three-stage counterflow ring, which we placed in the center of our main 28-stage counterflow ring. Figure 11 illustrates this three-stage counterflow ring. The bars in the Figure represent the stage boundaries. In the circuit these boundaries correspond to the COPs. At each request input to the COP we inserted an adjustable delay, and each delay value is controlled by a current source external to the chip. We chose current-controlled delays for this experiment, because of their accuracy and fine time resolution. The range of these delays is from about 550ps to 20ns, with a resolution of about five picoseconds around the metastable point. Except for the adjustable delays, the three stages were identical to the stages in the 28-stage ring. The 48-bit cycle counters used by the main experiment could also be connected to the three-stage ring.

The arbiter stress test is carried out in the following manner. Each ring is initially loaded with one item in stage 1, and both items have the same address and their

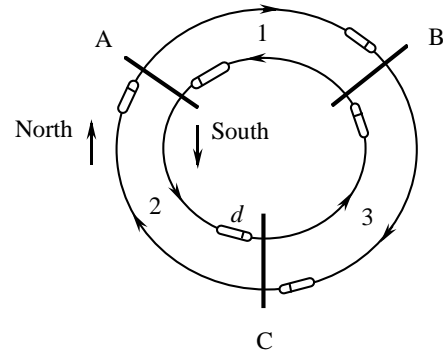


Figure 11. The arbiter stress test topology

marker bit set. The items are released almost simultaneously, and their next meeting will either be in stage 2 or in stage 3, depending on the values of the adjustable delays and the small differences in the delays in the stages. If the northbound item incurs a smaller delay than the southbound item, the items will next meet in stage 3, while if the southbound item incurs less delay than the northbound item, then the items will next meet in stage 2. If both items incur exactly the same delay, the stage in which they will meet depends on the arbiter decision in COP C. By careful setting the adjustable delays at the inputs of COPs A and B, we can force the meetings between the two items into only one of two patterns: either the meetings alternate between stages 1 and 2 or the meetings alternate between stages 1 and 3. By adjusting the delays at the inputs of COP C, we can make the circuit change from one pattern to the other. The change of pattern indicates that we have passed through the point at which maximum contention occurs at COP C: the point at which the two items arrive at COP C at about the same time. In fact, over a very small range of delays, which is of the order of a couple of picoseconds, we have even observed a mix of the two patterns. Furthermore, as expected, when the arbiter in COP C receives its two input requests at about the same time, the arbiter takes longer to decide which request to grant. This arbitration delay is significant and increases the cycle time of the items running around the ring. In all experiments that stressed the arbiters the counting circuits indicate error-free operation over billions of cycles.

Figure 12 quantifies the additional arbiter delay due to metastable arbiter behavior as a function of the extra delay d at the input of COP C. When the adjustable delay is at its lowest value, the meeting pattern of the items is an alternation of meetings in stages 1 and 3. The figure shows that as the adjustable delay value increases, the additional arbiter delay increases, because the ar-

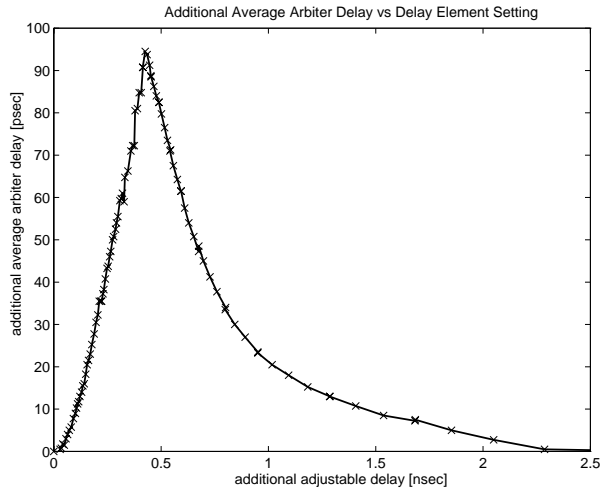


Figure 12. Additional arbiter delay as a function of delay d

biter in the COP then operates closer to the metastable point and takes longer to issue a grant. The measured additional arbiter delay increases roughly as the negative logarithm of the distance to the metastable point. Increasing the adjustable delay further causes the arbiter to grant the other side, and the additional arbiter delay then decreases as the negative logarithm. On this side of the curve the meetings occur in stages 1 and 2. Our observations concur with the theoretical and experimental results on metastable behavior reported in [2, 5].

Figure 12 shows that we observed additional delays in average cycle time of up to 95 ps. We estimate the error in each measurement to be at most 3 ps. Delays larger than 95 picoseconds were not obtainable, due to the finite resolution of our adjustable delays. For comparison, the delay through the arbiter for an uncontested request was measured to be 375 ps.

8. Concluding Remarks

We learned a great deal from our experiences in designing and measuring this test chip. We start with summarizing some advantages and disadvantages of the asP* protocol.

A big advantage of using the asP* protocol is that it can be implemented with standard logic elements, like Boolean gates and latches. Such elements are familiar to any circuit designer and can be found in any standard cell library. Another advantage of using the asP* protocol is that we have found it easier to work with for testing purposes than designs that use transition signalling.

We have found that use of the asP* protocol resulted in a clean implementation of the basic functions of a counterflow pipeline. In addition to the usual “move” operations of a pipeline, the counterflow pipeline must be able to achieve mutual exclusion between moves across the same boundary and must be able to block a move temporarily until match and garnering actions have completed. These requirements were implemented easily by the addition of a MUTEX and an AND gate.

A disadvantage of the asP* protocol is that it has more local delay constraints than transition-signaling protocol implementations. Without a systematic approach to the design and verification of a design, these delay constraints easily become overwhelming. The timing error in our first counterflow pipeline chip made us realize that we have to be more careful in setting the margins for our delay constraints and also demonstrated the shortcomings of the SPICE models that we used during the design. For aggressive designs, careful and complete extraction and simulation of layout is required.

The price we paid for the extensions to the asP* protocol is an increase in minimum cycle time from 6 gate-delays for the asP* implementation for the unidirectional pipeline to 12 gate-delays for the counterflow pipeline implementation. A gate-delay is a fanout-of-4 inverter delay or about 180 ps in our process. This minimum cycle time is achieved when single items move through a stage and do not encounter an item flowing in the opposite direction. The delay of a stage increases with 2 gate delays when two items meet in a single stage, but their addresses do not match, and with 5.5 gate-delays, when the addresses do match. For average-case circumstances, however, our experiments show that the counterflow pipeline operates at a throughput that is roughly halfway between the worst-case and best-case throughput. This is a good demonstration that asynchronous circuits can achieve average-case performance, whereas clocked circuits must be designed to operate for the worst case.

Our experiences gained over several years studying the counterflow pipeline architecture indicate that it is unlikely to be speed competitive with other existing or conventional architectures for general-purpose processing. The reason is that the counterflow architecture forces orderings upon the data that result in local congestion when there are large data-dependent variations in the processing times. However, this counterflow pipeline experiment illustrates that, with small data-dependent variations in processing times, the system adapts quickly to the changing workloads and exhibits performance that is better than if the system was fixed to always operate with the worst-case conditions.

Our experiments with these chips continue, and we

have found some interesting, and as yet not fully explained, modes of operation. We expect that as our understanding deepens of the behavior modes of this chip, we will be better prepared for future chip designs. We are learning to appreciate more and more the value of building and measuring the behavior of real chips, despite the large effort that this requires.

9. Acknowledgments

John Will provided the opportunity and help to conduct the EMI tests. His cooperation is most gratefully acknowledged. We thank Wes Clark for his numerous and valuable comments on an earlier draft of this paper.

References

- [1] W. S. Coates, J. K. Lexau, I. W. Jones, S. M. Fairbanks, and I. E. Sutherland. A FIFO data switch design experiment. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 4–17, 1998.
- [2] G. R. Couranz and D. F. Wann. Theoretical and experimental behavior of synchronizers operating in the metastable region. *IEEE Transactions on Computers*, 24(6):604–616, June 1975.
- [3] S. Furber. Computing without clocks: Micropipelining the ARM processor. In G. Birtwistle and A. Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pages 211–262. Springer-Verlag, 1995.
- [4] S. Furber, J. Garside, P. Riocreux, S. Temple, P. Day, J. Liu, and N. Paver. Amulet2e: An asynchronous embedded controller. *Proceedings of the IEEE*, 87(2):243–256, 1999.
- [5] L. R. Marino. General theory of metastable operation. *IEEE Transactions on Computers*, C-30(2):107–115, Feb. 1981.
- [6] C. Molnar and H. Schols. The design problem SCPP-A. Technical Report SML # 97:0538, Sun Microsystems Laboratories, Oct. 1997.
- [7] C. E. Molnar, I. W. Jones, B. Coates, and J. Lexau. A FIFO ring performance experiment. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 279–289. IEEE Computer Society Press, Apr. 1997.
- [8] C. E. Molnar, I. W. Jones, W. S. Coates, J. K. Lexau, S. M. Fairbanks, and I. E. Sutherland. Two FIFO ring performance experiments. *Proc. IEEE*, 87(2):297–307, 1999.
- [9] S. M. Nowick, K. Y. Yun, and P. A. Beerel. Speculative completion for the design of high-performance asynchronous dynamic adders. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 210–223. IEEE Computer Society Press, Apr. 1997.
- [10] N. C. Paver, P. Day, C. Farnsworth, D. L. Jackson, W. A. Lien, and J. Liu. A low-power, low-noise configurable self-timed DSP. In *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 32–42, 1998.
- [11] C. L. Seitz. Ideas about arbiters. *Lambda*, 1(1, First Quarter):10–14, 1980.
- [12] R. F. Sproull, I. E. Sutherland, and C. E. Molnar. The counterflow pipeline processor architecture. *IEEE Design & Test of Computers*, 11(3):48–59, Fall 1994.
- [13] I. E. Sutherland, R. F. Sproull, and D. L. Harris. *Logical Effort, Designing Fast CMOS Circuits*. Morgan Kaufmann Publishers, Inc, 1999.
- [14] T. E. Williams. Analyzing and improving the latency and throughput performance of self-timed pipelines and rings. In *Proc. International Symposium on Circuits and Systems*, May 1992.