MOUSETRAP: Ultra-High-Speed Transition-Signaling Asynchronous Pipelines*

Montek Singh and Steven M. Nowick
Department of Computer Science
Columbia University, New York, NY 10027
{montek,nowick}@cs.columbia.edu

Abstract

A new asynchronous pipeline design is introduced for high-speed applications. The pipeline uses simple transparent latches in its datapath, and small latch controllers consisting of only a single gate per pipeline stage. This simple stage structure is combined with an efficient transition-signaling protocol between stages.

Initial pre-layout HSPICE simulations of a 10-stage FIFO on a 16-bit wide datapath indicate throughput of 3.51 GigaHertz in 0.25μ CMOS, using a conservative process. This performance is competitive even with that of wave pipelines *[25, 11, 14], without the accompanying problems of complex timing and much design effort. Additionally, the new pipeline gracefully and robustly adapts to variable-speed environments. The stage implementations are extended to fork and join structures, to handle more complex system architectures.*

Keywords: Asynchronous pipelines, high throughput, transition signaling, transparent latches, gate-level pipelines, fine-grain, clocked CMOS.

1. Introduction

A new asynchronous pipeline, called MOUSETRAP, is introduced for high-speed applications. The pipeline uses blocks of static logic for processing data, and simple transparent latches to separate data items.

An asynchronous, or *clockless*, circuit style [23] was chosen for several reasons. First, while synchronous designers are currently capable of achieving multi-GigaHertz clock distributions, the task involves the ever-increasing challenges of design time, verification effort, clock skew and clock power management, and interfacing with different timing domains. Second, since an asynchronous pipeline has no global clock, it has a natural *elasticity:* the number of data items in the pipeline is allowed to vary, and the speeds at each interface can vary. As a result, the pipeline can interface with varied environments operating at different rates, thus facilitating modular and reusable design. Finally, the localized control of asynchronous pipelines is an excellent match for very high throughput fine-grain datapaths.

*This work was supported by NSF ITR Award Nos. 0086036 and 0086007, NSF Award No. CCR-97-34803, and a gift from Sun Microsystems, Inc.

The new pipeline is characterized by simplicity of its structure and operation, as well as by ease of design. The datapath uses standard transparent latches which are small and fast, and the control consists of only a single gate per pipeline stage. Pipeline stages communicate only with immediate neighbors, and the timing constraints are all local, simple and one-sided.

While the pipelining scheme is quite general, a special focus of this work is to target extremely high throughput. In particular, fine-grain, or "gate-level," pipelines are proposed, where each stage is only one gate deep. At this granularity, the shortest cycle times are obtained: the critical cycle consists of a single logic gate plus a small amount of control logic (*e.g.*, 2–3 component delays). As an additional optimization, the critical cycle is further shortened by merging logic and storage elements, using a circuit style called *clocked-logic*, or *clocked-CMOS* (C^2MOS) [1]. This technique has the benefit of reduced critical delays, smaller chip area and lower power consumption. In each case, a highly concurrent protocol is used; as a result, a basic MOUSETRAP FIFO has a cycle time of only 5–6 CMOS gate delays (3–4 components).

The pipeline builds on, and extends, the more conservative approaches proposed in [13, 22, 3]. The MOUSETRAP pipeline generates an earlier completion signal, and also we propose new extensions to handle complex pipelining (forks/joins), as well as a "waveform shaping" strategy, elimination of critical inverters through dual-rail control, and use of a clocked-CMOS style.

The name MOUSETRAP stands for *M*inimal-*O*verhead *U*ltra-high-*S*p*E*ed *TR*ansition-signaling *A*synchronous *P*ipeline. There is another reason why our pipelines are so called: the latching action is somewhat analogous to that of a mousetrap. When a pipeline stage is waiting for data, its latch remains transparent; as soon as data enters the stage, it is captured by closing the latch behind it. While there have been other asynchronous pipelines that have used this kind of latching action [22, 3], each has its own limitations. In effect, our goal in this work has been to build a "better mousetrap." Initial pre-layout simulations using HSPICE are quite encouraging: a 3.51 GigaHertz throughput using a conservative 0.25μm process.

The paper is organized as follows. Section 2 covers some related work. Section 3 introduces the new pipeline, some performance-oriented optimizations, and extensions to handle forks and joins. Section 4 gives a more in-depth comparison with related techniques. Preliminary simulation results are presented in Section 5, and finally, Section 6 gives conclusions.

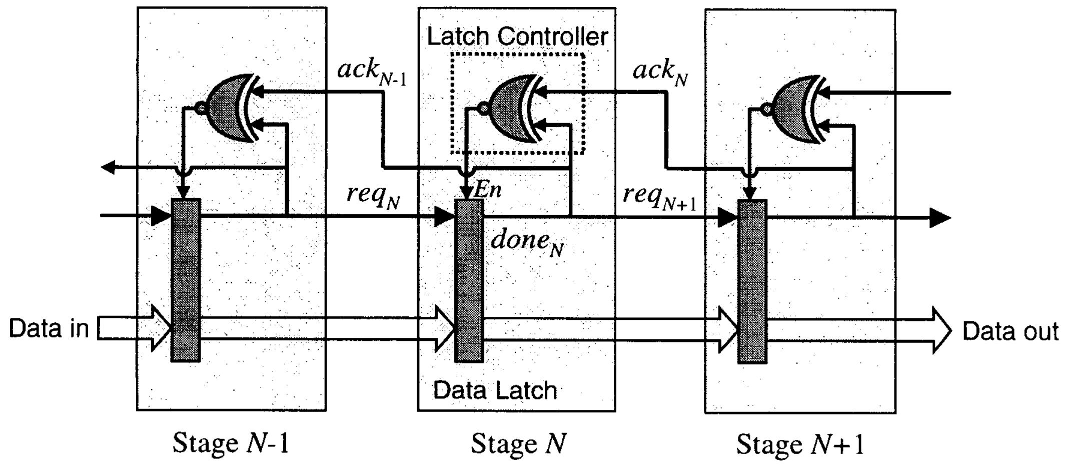

Figure 1. Basic MOUSETRAP pipeline without logic processing

2. Related Work

Synchronous Pipelines. Several synchronous pipelines have been proposed for high-throughput applications. In *wave pipelining*, multiple waves of data are propagated between two latches [25, 11, 14]. However, this approach requires much design effort, from the architectural level down to the layout level, for accurate balancing of path delays (including data-dependent delays), and remains highly vulnerable to process, temperature and voltage variations. Other aggressive approaches include *clock-delayed domino* [26], *skew-tolerant domino* [9, 4] and *self-resetting circuits* [15, 4]. These all require complex timing constraints which are difficult to verify; they also lack elasticity and still require high-speed global clock distribution.

Asynchronous Pipelines. The classic asynchronous pipelines introduced by Sutherland are called *micro-pipelines* [22]. This style uses elegant control, but has slow and complex capture-pass latches which hinder performance. A number of variants using alternative control and latch structures have been proposed [3, 27, 12], but in each case the performance is limited due either to excessive control delays or to sizable latch delays. Very recently, a new style, GasP, has been proposed which obtains even higher throughputs [21, 5]. However, their approach aims for fine-grain transistor sizing to achieve delay equalization for all gates in the control circuitry, and the protocol has more complex timing constraints. In contrast, MOUSETRAP pipelines do not require delay equalization and have simpler one-sided timing constraints.

There have been a number of alternative asynchronous approaches that dispense with explicit latches altogether [24, 20, 19]. These designs typically use dynamic logic for the datapaths, and rely instead on the inherent latching properties of dynamic gates. Throughputs up to 1.2 GHz in 0.6μm technology have been reported.

The fastest designs reported so far are the IPCMOS pipelines [17], with throughputs of 3.3 GHz for normal process.[1] The published results were all for fabricated IPCMOS control circuits which are capable of driving paths typical of 64-bit multiplier stages. A definitive quantitative comparison of IPCMOS with MOUSETRAP is not yet possible for a number of reasons. First, the IPCMOS results were obtained with an IBM high-performance 0.18μm process, while MOUSETRAP results used a 0.25μm TSMC process. The TSMC process is significantly slower than the IBM one not only because of a larger feature size, but also due to an inherently slower silicon technology. In addition, we currently only provide pre-layout simulations, while the IPCMOS results are post-fabrication. Finally, our simulations do not yet include logic processing, while IPCMOS simulations do.

[1]Throughputs of up to 4.5 GHz have been reported, but these are only for extreme process cases ($L_{eff} = -2\sigma$ and low V_t).

However, MOUSETRAP has several inherent advantages over IPCMOS which we expect will be realized in fabricated chips. First, IPCMOS uses large and complex control circuits which have significant delays; in contrast, our designs use much simpler control which should have better performance than IPCMOS in a comparable process. In particular, IPCMOS has 12 levels of CMOS logic plus one pass-gate in its critical cycle. A MOUSETRAP cycle, on the other hand, has only 5 levels of CMOS logic plus one pass-gate (two latches and one XNOR). Even with the fork and join extensions of Section 3.7, MOUSETRAP at most has only 8 levels of CMOS logic. Second, IPCMOS makes use of extremely aggressive circuit techniques which require a significant effort of design and verification. For example, one of the gates in their "strobe circuit" may potentially have a short circuit through its pull-up and pull-down, depending on the relative arrival times of inputs from multiple data sources. Their approach relies on a ratioing of the transistors to ensure correct output. Finally, MOUSETRAP has much simpler timing constraints. In fact, in addition to setup and "datapath bundling" constraints [8], there is only a single one-sided constraint to be satisfied.

3. The MOUSETRAP Pipeline

This section first introduces the basic structure and operation of the MOUSETRAP pipeline (Sections 3.1–3.2). Then, several implementation issues are discussed in detail, and performance and tim-

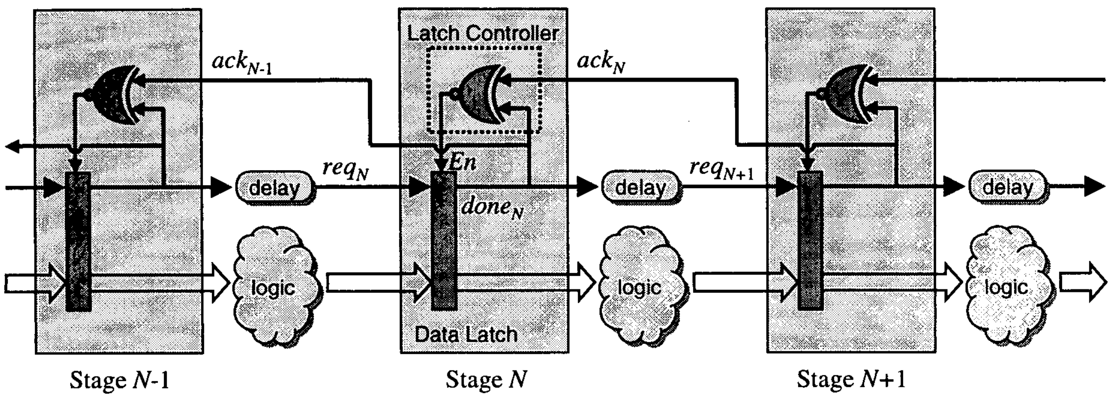

Figure 2. MOUSETRAP pipeline with logic processing

ing constraints are derived (Sections 3.3–3.5). In addition, an optimization is introduced that increases pipeline performance under steady-state operation by carefully "shaping" the controller output so as to reduce critical delays (Section 3.6). Finally, the basic linear pipelines are extended to handle forks and joins (Section 3.7).

Initially, to simplify discussion, Sections 3.1 and 3.2 focus on a basic pipeline without logic processing, *i.e.* a simple FIFO. Later, Section 3.3 shows how logic processing is added to the pipeline.

3.1. Basic Pipeline Structure: A Simple FIFO

Fig. 1 shows the structure of the basic pipeline without logic processing. Three pipeline stages are shown. Each stage consists of a *data latch* and a *latch controller.* The stages communicate with each other using "requests" (*req*'s) and "acknowledgments" (*ack*'s).

The *data latch* is a simple transparent latch. The latch is normally transparent (*i.e.,* enabled), allowing new data to pass through quickly.

A commonly-used asynchronous scheme, called *bundled data* [18, 2], is used to encode the datapath: a control signal, req_N, indicates arrival of new data at stage N's inputs. For correct operation, a simple requirement must be satisfied: req_N must arrive *after* the data inputs to stage N have stabilized.[2] Once stage N has latched the new data, $done_N$ is produced, which is sent to its latch controller, as well as to stages $N - 1$ and $N + 1$.

The *latch controller* enables and disables the data latch. It consists of only a single XNOR gate with two inputs: the *done* from the current stage, stage N, and the *ack* from stage $N + 1$.

3.2. Pipeline Operation

Overview. The operation of the pipeline of Fig. 1 is quite simple. Initially, when the pipeline is empty, all its latches are transparent and all the *done, req* and *ack* signals are low. When the first data item flows through successive stages of the pipeline, it flips the values of all these signals *exactly once* (high). Subsequently, the second data item flips all these signals once again (low). This method of signaling is called *transition signaling.* Each transition, whether up or down, represents a distinct event: the arrival of a new data item.

Once a data item passes through stage N's latch, three actions take place *in parallel:* (i) the data is passed forward to the next stage for further processing, along with the corresponding request, req_{N+1}; (ii) an acknowledgment, ack_{N-1}, is sent to the previous stage, freeing it up to process the next data item; and finally (iii) stage N's latch itself is quickly made opaque to protect the current data from being overwritten by new data produced by stage $N - 1$. Subsequently, when an acknowledgment, ack_N, is received from stage $N + 1$, the latch in stage N is re-enabled (*i.e.,* made transparent).

Note that while transition signaling is used to signal the flow of data (one transition on each *req/done/ack* per data item), as shown above, the latches themselves require *two transitions* per data item: one to capture (make opaque), and one to release (make transparent). The first transition takes place when data passes through stage N's latch ($done_N$ changes value); the second when the same data passes through stage N + 1's latch (ack_N changes value). Thus, the XNOR gate acts like a *phase converter:* it converts the transition signaling *done*'s and *ack*'s into level control for the transparent latches.

There is another interpretation of the behavior of the latch controller, which is useful for understanding the pipeline operation: the XNOR gate is simply an "equality tester." When stages N and $N + 1$ have the same data item, stage N is effectively "empty," and its latch is enabled (made transparent). When the stages have distinct data items, stage N is effectively "full," and its latch is disabled (made opaque).

The latching action by a pipeline stage is analogous to the operation of a household mousetrap: latches remain transparent *before* data arrives; they are closed (*i.e.,* made opaque) as soon as data passes through. It is important to note that this behavior is very different from that of most synchronous, and many asynchronous, pipelines in which latches are opened only *after* new data arrives.

Detailed Operation. A local timing constraint must be satisfied for correct operation. Since a transition on $done_N$ is also a transition on ack_{N-1}, there is a race condition between the disabling of stage N's latch and the reception of new data from $N - 1$. To ensure that the contents of stage N are not corrupted, stage N's latch

[2] If the data has to go through a logic block before arriving at stage N, req_N must be appropriately delayed. This is discussed in more detail in Section 3.3.

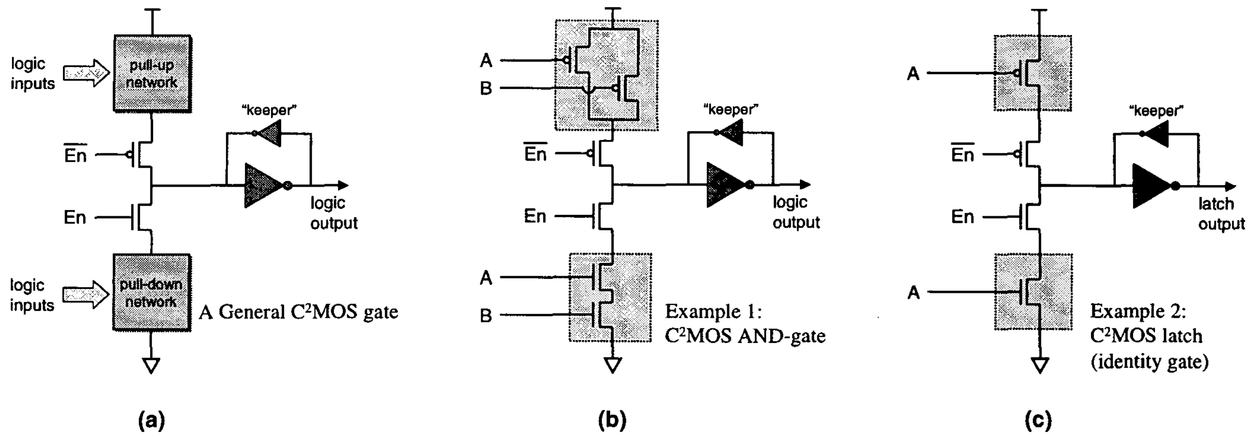

Figure 3. Clocked-CMOS (C^{2}MOS) logic

must be disabled fast enough, *i.e.*, before the stage $N - 1$ can provide new data. This is a simple one-sided timing constraint that can easily be satisfied in practice. (For a more detailed analysis, see Section 3.4.2.)

Note that the choice of a hybrid protocol—*transition signaling* for the handshake signals, and *level signaling* for the latch enable signal—combines the advantages of both the signaling schemes: (i) much less handshaking overhead since there is no wasteful "return-to-zero" phase, and (ii) small and fast transparent latches, since they are level-controlled. While several transition signaling schemes have been proposed—some with phase conversion [22, 3] and others without [27]—the pipeline presented here has much less overhead. (Refer to Section 4 for a detailed comparison.)

In summary, the new pipeline protocol is very simple, and the operation quite fast. The forward latency of an empty pipeline is low because all the latches are initially transparent. The cycle time of the pipeline is short because the pipeline is highly concurrent: as soon as data enters stage N, stage $N - 1$ is freed up for its entire next cycle.

3.3. Pipeline Implementation: Adding Logic Processing

So far, only pipelines without logic processing, *i.e.*, simple FIFO's, have been considered. It is now shown how logic processing can easily be added to the pipeline. First, a basic implementation with explicit latches is presented. Then as a special case, *gate-level pipelines* are considered: each stage is only a single gate deep, with no explicit latches.

3.3.1. General Pipeline Implementation

Fig. 2 shows how logic processing can be added to the pipeline. Blocks of combinational logic and matching delay elements are simply inserted between pipeline stages. The standard asynchronous *bundled data* scheme is again used: req_N must arrive at stage N *after* the data inputs to that stage have stabilized. Therefore, the latency of the delay element must match the worst-case delay through the combinational block. An advantage of this approach is that the datapath itself can use standard single-rail (synchronous style) blocks, which are allowed to be hazardous, as long as the *req* arrives after data has stabilized.

There are several common ways to implement a matched delay. One technique is to simply use an inverter chain, or a chain of transmission gates; the number of gates and their transistor sizing determines the total delay. A more accurate technique duplicates the worst-case critical path of the logic block, and uses that as a delay line ([6] and, in a different context, [16]). If the duplicated critical path is placed in close proximity to the logic block, it can provide good delay tracking even for a wide variation in environmental and process variations. However, this technique is more area-expensive than using a chain of inverters or other standard gates. Bundled data has been widely used, including in a commercial Philips 80C51 asynchronous microcontroller [8].

3.3.2. Special Case: Gate-Level Pipelines Using Clocked-CMOS (C^{2}MOS)

As a special case, our goal of extremely high throughput is best achieved by *gate-level pipelines:* the datapath is sectioned into the finest-grained stages, each consisting of only a single level of logic with no explicit latches. As an additional benefit, the absence of latches also translates into savings of chip area and power consumption.

Clocked-logic, also known as *clocked-CMOS* (C^{2}MOS), is a particularly attractive approach to gate-level pipelining [1]. In this scheme, the latches are eliminated altogether; instead, a clock is applied directly to the logic gate. Fig. 3 shows the structure of several C^{2}MOS gates. The clock input, *En*, directly controls the gate through two transistors, one each in the pull-up and the pull-down network. When *En* is asserted, the gate is enabled and a new output is produced. When *En* is de-asserted, the gate holds its output value. Typically, an inverter pair providing weak feedback is attached at the gate output to provide a more robust hold operation. While C^{2}MOS has been proposed as a synchronous technique [1], it can easily be adapted to very high-speed asynchronous pipelines using handshaking signals to replace the clock.

Fig. 4 shows a C^{2}MOS implementation of the MOUSETRAP pipeline. The explicit data latches of Fig. 2 have been eliminated;

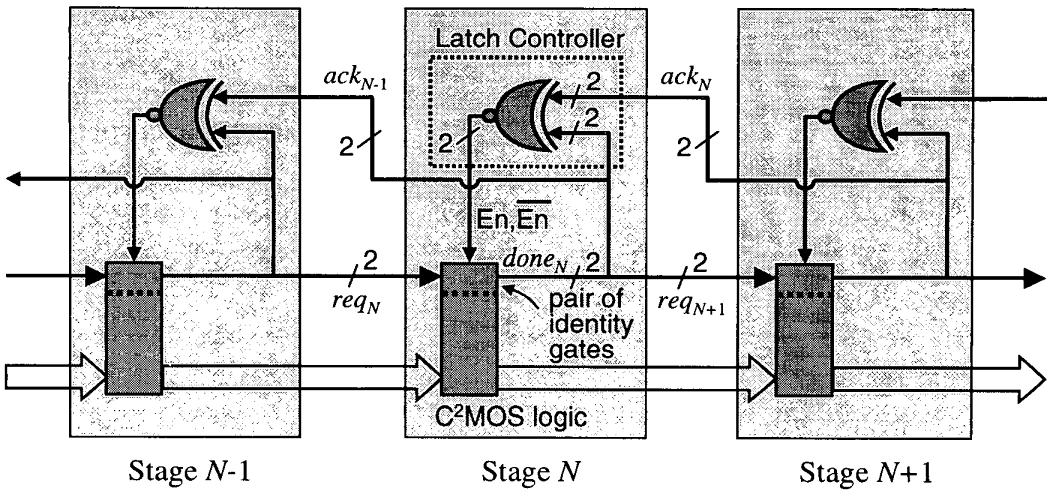

Figure 4. C²MOS implementation of gate-level MOUSETRAP pipeline

instead, C^2MOS gates provide both logic as well as latching functionality. Note that the "clock" input is actually the locally generated *En* signal. Both *En* and $\overline{En}$ are needed for the control of C^2MOS gates. This suggests the use of a *dual-rail* XNOR gate, which is the topic of the next subsection.

3.3.3. XNOR Optimization: Dual-Rail Implementation

Since many transparent latches as well as C^2MOS gates require both true and complemented enables, a useful optimization for both of our pipeline schemes (Figs. 2 and 4) is to implement the XNOR as a monotonic *dual-rail* gate, providing both XOR and XNOR outputs. As highlighted in Fig. 4, the XNOR now has two dual-rail inputs—(*done*, $\overline{done}$) and (*ack*, $\overline{ack}$)—and a dual-rail output (*En*, $\overline{En}$). While this approach increases the overall control area, it directly improves the performance: two inverters are eliminated from the critical cycle (from XNOR inputs and its output).

3.4. Pipeline Performance and Timing Constraints

This section presents an analytical evaluation of both pipeline performance and timing constraints.

3.4.1. Performance

Two key measures of the performance of the pipeline are discussed: *forward latency* and *cycle time.*

Forward latency is the time it takes a data item to pass through an initially empty pipeline. Since, in an empty pipeline, all the latches are transparent, the pipeline latency per stage, L, is simply the stage's latch delay plus logic delay:

$$L = t_{Lt} + t_{\text{logic}} \qquad (1)$$

Cycle time is the time interval between successive data items emerging from the pipeline when the pipeline is operating at maximum speed. A cycle of stage N, from one enabling of its latch to the next, consists of three events: (i) new data passes through the latch and the stage's logic block, (ii) the data passes through stage $N+1$'s latch, producing ack_N, and (iii) ack_N causes stage N's latch controller to re-enable stage N's latch. Therefore, the analytical cycle time T is:

$$\begin{aligned} T &= t_{Lt} + t_{\text{logic}} + t_{Lt} + t_{\text{XNOR}\uparrow} && (2)\\ &= 2 \cdot t_{Lt} + t_{\text{logic}} + t_{\text{XNOR}\uparrow} && (3) \end{aligned}$$

where t_{logic} is the delay through the logic block, and $t_{\text{XNOR}\uparrow}$ is the time it takes the XNOR gate to enable the latch.

For the special case of C^2MOS pipelines, there are no explicit latches. If the delay through a C^2MOS gate is denoted by $t_{C^2\text{MOS}}$, the latency and cycle time are given by:

$$\begin{aligned} L_{C^2\text{MOS}} &= t_{C^2\text{MOS}} && (4)\\ T_{C^2\text{MOS}} &= 2 \cdot t_{C^2\text{MOS}} + t_{\text{XNOR}\uparrow} && (5) \end{aligned}$$

The cycle times of Equations 3 and 5 are quite good, and would be difficult to surpass with synchronous schemes. For example, a standard synchronous pipeline, with alternating latches controlled by complementary clocks, and with logic between every adjacent latch pair, will have a cycle time of at least $2 \cdot t_{Lt} + 2 \cdot t_{\text{logic}}$, plus adequate margins to compensate for clock skew and jitter.

3.4.2. Timing Constraints

Two simple one-sided timing constraints must be satisfied for the correct operation of the pipeline: *setup time* and *data overrun.*

Setup time. Once a latch is enabled and receives new data at its inputs (along with a *req* signal), it must remain transparent long enough for data to pass through. Thus, the path from req_N to *En* de-asserted (XNOR switching low) must be longer than the setup time, t_{su}:

$$t_{req_N \to done_N} + t_{\text{XNOR}_N\downarrow} > t_{su} \qquad (6)$$

This constraint is easily satisfied because the delay from req_N to $done_N$ typically exceeds the setup time.

Data overrun. Once data enters a stage, it should be securely captured before new data is produced by the previous stage. If this condition is violated, stage N's data will be overwritten by new data. Therefore, since ack_{N-1} and $done_N$ are generated in parallel, the path from ack_{N-1} to stage N's data inputs must be longer than the time to close N's latch, plus a hold time, t_{hold}:

$$t_{\text{XNOR}_{N-1}\uparrow} + t_{Lt_{N-1}} + t_{\text{logic}_{N-1}} > t_{\text{XNOR}_N\downarrow} + t_{\text{hold}} \quad (7)$$

The left terms represent the shortest path through the XNOR to the arrival of new input from stage $N-1$. The right terms represent the path to disabling stage N's latch. The equation can be rewritten to simplify the constraint:

$$t_{Lt_{N-1}} + t_{\text{logic}_{N-1}} > (t_{\text{XNOR}_N\downarrow} - t_{\text{XNOR}_{N-1}\uparrow}) + t_{\text{hold}} \quad (8)$$

Assuming $t_{\text{XNOR}_{N-1}\uparrow} \cong t_{\text{XNOR}_N\downarrow}$, the right expression in parentheses is cancelled. The result is a simple hold time constraint, which is easily satisfied because the latch and logic delays through stage $N-1$ usually exceed the hold time.

3.5. Handling Wide Datapaths

An important issue is the handling of very wide datapaths, where control signals must be broadcast across many latches. This control distribution may introduce sizable delays in the critical path, slowing down the operation of the pipeline. There are two practical solutions proposed for efficient pipelining of wide datapaths: (i) *datapath partitioning,* and (ii) *control kiting.*

In the first approach, datapath partitioning, a wide datapath is divided into several smaller independent streams. The pipeline control is replicated for each stream, and each stream has its own sequence of completion generators and matched delays. As a result, the requirement of buffering is significantly reduced: in each stage, the latch controller generates a latch enable signal which is broadcast to only a small number of bits, *i.e.* to only those bits that lie inside that partition. This approach is typically applicable to bit-slice datapaths, such as plain FIFO's and logic function units (e.g. bitwise AND, OR, etc.).

The second approach to handling wide datapaths, called control kiting, allows the datapath to be skewed with respect to the control [27, 12]. No partitioning is used; instead, buffers are inserted to adequately amplify the latch enable signals which drive the datapath latches. However, the latch enables for the completion generators do not need this amplification; they are simply tapped off from *before* the buffers. As a result, much of the overhead of broadcasting the latch enable to the datapath is hidden, occuring in parallel with other pipeline operations.

Figure 5 shows how the second approach is implemented, for example, for a MOUSETRAP FIFO. Since the insertion of buffers only delays the latching (and unlatching) of the datapath, the completion signal of each stage, *req*, is actually produced a buffer delay earlier than the data outputs. The reader can verify that, assuming uniform buffer delays across all stages, the pipeline of Figure 5 operates correctly, and has exactly the same cycle time and timing constraints as those derived in Section 3.4 for narrower datapaths.

3.6. Pipeline Speedup: Optimized Control Generation by "Shaping" XNOR Output

This subsection focuses on a low-level circuit optimization to further improve the pipeline's performance *under steady-state operation.* The main bottleneck to pipeline performance is that the

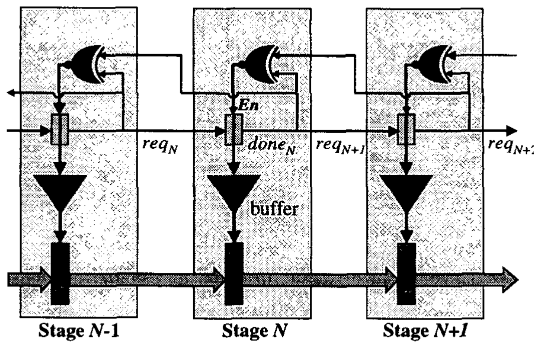

Figure 5. Handling wide datapaths in MOUSETRAP pipelines

XNOR's must switch twice for every data item flowing through the pipeline, causing the latches to repeatedly close and open. The proposed solution is to prevent the XNOR's output from falling completely to a "0" value, and thus to avoid closing the latches fully, in steady state operation. This effect is achieved by slowing down the fall time of the XNOR, through transistor sizing. As a result, in steady-state, both the critical up-transition of the XNOR, and the re-enabling of the latch have shorter delays due to reduced voltage swing.

Interestingly, this optimization is analogous to the behavior of a sliding door at a building entrance: the closing action of the door is deliberately slowed down, so that, when there is a steady stream of people, the door never closes fully, allowing speedier passage. It is interesting to note how slowing down one action speeds up the overall operation of the pipeline.

There is one caveat, though: this circuit optimization may make the timing constraint of Equation 8 (*data overrun*) more difficult to satisfy. In particular, slowing down the latch disable and speeding up the latch enable means higher $t_{\text{XNOR}_N\downarrow}$ and lower $t_{\text{XNOR}_{N-1}\uparrow}$, making the term $(t_{\text{XNOR}_N\downarrow} - t_{\text{XNOR}_{N-1}\uparrow})$ now a non-zero positive quantity. As a result, the margin available to satisfy the inequality is somewhat reduced. In practice, though, experiments indicate that this constraint can still be satisfied safely.

The net impact is that the steady-state performance is as fast as that of a *wave pipeline* [25, 10], and yet the new pipeline provides much greater robustness and require much simpler timing requirements. Consider the interface of a MOUSETRAP pipeline with its right environment. Whether the right environment suddenly stalls or speeds up, the pipeline gracefully handles these variations. If the right environment is slow and cannot respond with an *ack*, the rightmost pipeline stage quickly makes its latch opaque (since no *ack* is received by its XNOR), thus preventing an overrun from the left stage. If the right environment is very fast, it is correctly stalled until the rightmost stage can deliver it data, since the environment is waiting for the stage's *req* signal. The same reasoning also applies to the internal stages in the pipeline, making the pipeline robust to internal delay variations as well.

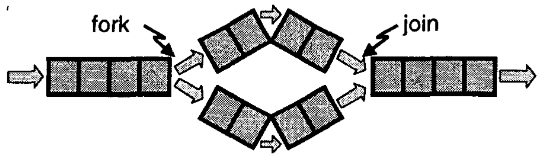

Figure 6. Non-linear pipelining

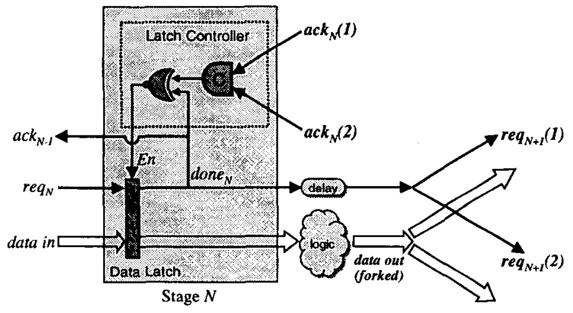

Figure 7. MOUSETRAP fork stage

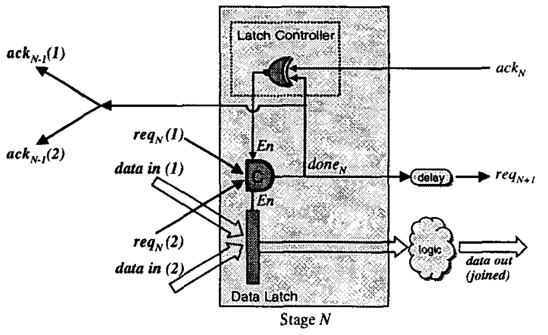

Figure 8. MOUSETRAP join stage

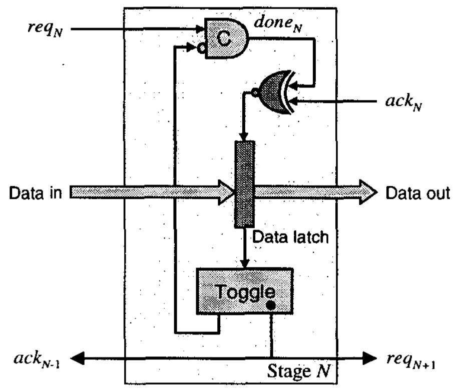

Figure 9. Sutherland's and Day/Woods' micropipeline stage

3.7. Non-Linear Pipelining

The previous section has focused on linear pipelines, which have many practical applications. However, in complex system architectures, there is often a need for non-linear pipelining as well (Figure 6). We now propose two simple primitives —fork and join—which extend the applicability of MOUSETRAP pipelines.

Figures 7 and 8 show simple structures for fork and join components, respectively. In the fork, the data output and corresponding "req" (matched done output) are both simply forked to the two or more destination stages. In turn, the two or more "ack" signals are combined through a Müller *C-element*. A C-element is an "event ANDer": its output makes a transition when all of its inputs change exactly once [22].

In the join, the "ack" is simply a forked wire, communicating with all sender stages. The "req's" and their accompanying data inputs are combined as follows. The various data inputs are simply merged into one stream and latched together. The "req's" (matched done inputs) are merged using an enabled (i.e. "asymmetric") C-element [7]. Whenever the "latch enable" is asserted, the component's output is 1 when all of the merged "req's" are 1, and is 0 when all of the merged "req's" are 0. At all other times (when the "latch enable" is de-asserted, or if the req's are not all equal), the component simply holds its value. At the transistor level, the pulldown network is a single series stack with one transistor for each req, as well as a transistor for the "latch enable". Similarly, the pullup network is a single series stack with one transistor for each req, and with a transistor for the complemented "latch enable".[3]

[3]We are currently developing optimized versions which obtain higher performance using logic decomposition.

4. Comparison with Other Transition-Signaling Asynchronous Pipelines

This section compares the operation of MOUSETRAP in more detail with that of several other asynchronous pipelines. These pipeline designs fall into two categories: those that use phase conversion [22, 3, 13], and those that do not [7, 27].

The pipelines of [22, Fig. 14] and [3, Fig. 10], called *micropipelines*, both use phase conversion. Similar to our design, a micropipeline stage uses transition signaling and transparent latches (see Fig. 9). However, both of these approaches have significantly more complex control. Each has two extra components per stage: a C-element and a Toggle element. The Toggle element routes transitions received on its input to one of two outputs alternately, starting with the output labeled with a dot. The critical paths are also much longer than MOUSETRAP: from req_N to req_{N+1} there are 4 component delays (the C-element, XNOR, latch, and Toggle), and from ack_N to the input of the C-element (to half-enable it) there are 3 component delays (the XNOR, latch and Toggle). In contrast, our pipeline has only a single latch delay for the first path, and only an XNOR delay for the second path.

Closer to MOUSETRAP, the "Charlie boxes" [13] include simpler designs, such as the *S style*. Our MOUSETRAP pipeline can be regarded as a more optimized—and less robust—version of some Charlie boxes. MOUSETRAP generates an earlier completion signal, and also we have proposed new extensions to handle complex pipelining (forks/joins), as well as a "waveform shaping" strategy, elimination of critical inverters through dual-rail control, and use of a clocked-CMOS style, none of which appeared in [13].

There are several alternative approaches to using phase conversion. In [7], Furber and Day propose three distinct 4-phase protocols for asynchronous pipelines: fully-decoupled, long-hold and semi-decoupled. In the first two, pipeline control is significantly more complex than in MOUSETRAP. The best of their designs, semi-decoupled, introduces a highly-concurrent protocol, but still has a minimum of 4 components on the critical cycle. These components are all C-elements, two of which have stack depth of 3, and additional inverters are actually implied for correcting polarity. In contrast, MOUSETRAP only has 3 components on the critical cycle (2 D-latches and an XNOR), no stack depths of 3, no implied inverters, and avoids the extra switching activity of 4-phase communication.

A final alternative approach is to retain transition-signaling control, but replace the transparent latches with dual-edge-triggered D-flip-flops (DETDFF's) [27]: data is latched each time the latch control is toggled. While this approach avoids the overhead of phase conversion, it incurs a heavy performance penalty because DETDFF's are significantly slower than transparent latches, and are also much larger.

5. Preliminary Results

This section presents initial pre-layout simulations, using HSPICE, for a basic MOUSETRAP pipeline. A simple 10-stage FIFO was simulated (with no logic processing) on a 16-bit wide datapath. The FIFO was designed and simulated in two different CMOS technologies: (i) a 0.25μm TSMC process, and (ii) a 0.6μm HP process. For the first technology, only the unoptimized pipeline style was used: we did not include the "waveform shaping" optimization of Section 3.6. For the second technology, both the optimized and the unoptimized versions of the pipeline were simulated. In each case, careful transistor sizing was used to improve performance. (These initial simulations do not include parasitics; we are currently working on layouts and post-layout simulation.)

The first simulation, using the 0.25μm TSMC process, was performed assuming a 2.5V power supply, 300K temperature, and a normal process corner. Simple custom cells were designed: a pass-gate implementation of an XNOR/XOR pair, and a standard 6 transistor pass-gate dynamic D-latch.

Table 1 summarizes the results of pre-layout simulation. The overall pipeline cycle time T, is given, as well as a breakdown of a cycle into latch delay, t_{Lt}, and controller gate delays, $t_{\text{XNOR}\uparrow}$ and $t_{\text{XNOR}\downarrow}$. The initial results are quite encouraging: a 3.51 GigaHertz throughput. Post-layout simulations, currently underway, are needed to truly evaluate the performance and overheads of parasitics and layout issues.

These numbers compare favorably to the IPCMOS style of Schuster *et al.* [17]. Their reported results of 3.3 GHz are for a high-performance IBM 0.18μm process, which in practice is significantly faster than the 0.25μm TSMC process we used. Although our designs do not include logic processing, we anticipate competitive performance with IPCMOS using a comparable process when logic is included. As indicated earlier, the IPCMOS critical path is made up of 12 levels of CMOS logic, plus a pass-gate. In contrast, MOUSETRAP only uses 5–6 levels of CMOS logic on its critical path (plus 2 if there are forks and joins in the datapath). In addition, MOUSETRAP has the benefit of much simpler circuit components and timing constraints.

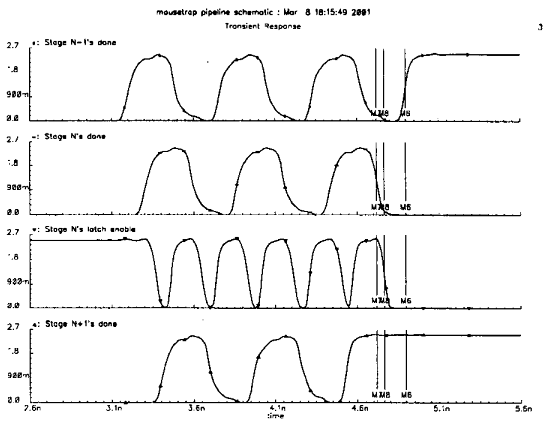

Figure 10. Waveforms

The second simulation was performed to evaluate the waveform shaping optimization of Section 3.6. Currently, the simulation has only been performed in 0.6μm (HP CMOS technology, 3.3V, 300K, normal corner). We plan to carry these simulations to the 0.25μm technology in the near future. The simulations indicate the benefit of the wave shaping approach. A plain FIFO was evaluated both with and without the optimization. The *XNOR/XOR* pair was once again designed with pass gates, but the latch was designed in the clocked-CMOS style (Figure 3(c)).

Table 2 shows the results of the second simulation. The unoptimized FIFO has a throughput of 1.67 GHz, and the optimized one has a throughput of 1.92 GHz, for a performace improvement of 15%. In addition, the timing constraint of Equation 8 (*data overrun*) is easily met: t_{Lt} = 0.20ns, t_{logic} = 0ns, and $t_{\text{XNOR}\downarrow} - t_{\text{XNOR}\uparrow} = 0.06$ns.

Figure 10 gives waveforms for 3 adjacent stages ($N-1$, N, and $N+1$) for a single simulation of the FIFO, in 0.25μm TSMC technology. The simulation includes the *done* signals for each stage, as well as the "latch enable" for stage N (i.e., output of the stage's controller). This latch enable waveform indicates that stage N's latch is disabled (enable=0) soon after stage N indicates it is done (alternating 0 and 1 values), for every data item in the simulation. Similarly, the latch enable for stage N is re-enabled (enable=1) soon after *stage* $N+1$ indicates it is done, for every data item as well. Observe that the one-sided timing constraint between stage N and $N-1$ is clearly satisfied, even though no processing logic is present (such logic would improve margins): as shown at the right-hand side of the simulation, stage N's latch enable is disabled approximately 45 picoseconds before stage $N-1$ produces a new data token (*done* asserted).

Table 1. Performance of MOUSETRAP FIFO (0.25μm TSMC technology)

Pipeline Design	latch delay t_{Lt} (ps)	XNOR delay $t_{\text{XNOR}\uparrow}$ (ps)	$t_{\text{XNOR}\downarrow}$ (ps)	Cycle Time, T Analytical Formula	(ps)	Throughput (GigaHertz)
MOUSETRAP	110	65	63	$2 \cdot t_{Lt} + t_{\text{XNOR}\uparrow}$	285	3.51

Table 2. Performance of MOUSETRAP FIFO's using clocked-CMOS logic (0.6μm HP technology)

Pipeline Design	C^2MOS logic delay t_{C^2MOS} (ns)	XNOR delay $t_{\text{XNOR}\uparrow}$ (ns)	$t_{\text{XNOR}\downarrow}$ (ns)	Cycle Time, T Analytical Formula	(ns)	Throughput (GigaHertz)
MOUSETRAP	0.22	0.16	0.13	$2 \cdot t_{C^2MOS} + t_{\text{XNOR}\uparrow}$	0.60	1.67
MOUSETRAP$_{opt}$	0.20	0.12	0.18	$2 \cdot t_{C^2MOS} + t_{\text{XNOR}\uparrow}$	0.52	1.92

6. Conclusions

A new pipeline design style was introduced for high-throughput applications. The pipeline uses simple structures for both latches and control, and an efficient and highly-concurrent event-driven protocol. In steady-state operation, the pipeline performance is comparable to that of wave pipelines, and yet the new pipelines are more robust and require much less design effort. In the future, we also plan to lay out and simulate pipelines with logic processing, *e.g.* pipelined adders and multipliers, and also evaluate our "wave shaping" optimization in greater detail.

7. Acknowledgments

The authors gratefully acknowledge Sucheta Sharma for suggesting the name "MOUSETRAP" for our pipelines, and Michael Theobald for contributing the "sliding door" analogy.

References

[1] M. Borah, R. M. Owens, and M. J. Irwin. High-throughput and low-power DSP using clocked-CMOS circuitry. In *Proc. Intl. Symp. on Low-Power Design*, pages 139–144, 1995.

[2] A. Davis and S. M. Nowick. Asynchronous circuit design: Motivation, background, and methods. In G. Birtwistle and A. Davis, editors, *Asynchronous Digital Circuit Design*, Workshops in Computing, pages 1–49. Springer-Verlag, 1995.

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

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

[5] J. Ebergen. Squaring the FIFO in GasP. In *Proc. Intl. Symp. Adv. Res. Async. Circ. Syst. (ASYNC)*, pages 194–205. IEEE Computer Society Press, Mar. 2001.

[6] 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.

[7] S. B. Furber and P. Day. Four-phase micropipeline latch control circuits. *IEEE TVLSI*, 4(2):247–253, June 1996.

[8] H. v. Gageldonk, D. Baumann, K. van Berkel, D. Gloor, A. Peeters, and G. Stegmann. An asynchronous low-power 80C51 microcontroller. In *Proc. Intl. Symp. Adv. Res. Async. Circ. Syst. (ASYNC)*, pages 96–107, 1998.

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

[10] O. Hauck, M. Garg, and S. A. Huss. Two-phase asynchronous wave-pipelines and their application to a 2D-DCT. In *Proc. Intl. Symp. Adv. Res. Async. Circ. Syst. (ASYNC)*, Apr. 1999.

[11] W. Liu, C. T. Gray, D. Fan, W. J. Farlow, T. A. Hughes, and R. K. Cavin. A 250-MHz wave pipelined adder in 2-μm CMOS. *IEEE JSSC*, 29(9):1117–1128, Sept. 1994.

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

[13] C. E. Molnar and I. W. Jones. Simple circuits that work for complicated reasons. In *Proc. Intl. Symp. Adv. Res. Async. Circ. Syst. (ASYNC)*, pages 138–149. IEEE Computer Society Press, Apr. 2000.

[14] A. Mukherjee, R. Sudhakar, M. Marek-Sadowska, and S. Long. Wave steering in YADDs: a novel non-iterative synthesis and layout technique. In *Proc. DAC*, 1999.

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

[16] S. M. Nowick, K. Y. Yun, and P. A. Beerel. Speculative completion for the design of high-performance asynchronous dynamic adders. In *Proc. Intl. Symp. Adv. Res. Async. Circ. Syst. (ASYNC)*, pages 210–223. IEEE Computer Society Press, Apr. 1997.

[17] S. Schuster, W. Reohr, P. Cook, D. Heidel, M. Immediato, and K. Jenkins. Asynchronous interlocked pipelined CMOS circuits operating at 3.3-4.5 GHz. In *Proc. ISSCC*, Feb. 2000.

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

[19] M. Singh and S. M. Nowick. Fine-grain pipelined asynchronous adders for high-speed DSP applications. In *IEEE Computer Society Annual Workshop on VLSI*.

[20] M. Singh and S. M. Nowick. High-throughput asynchronous pipelines for fine-grain dynamic datapaths. In *Proc. Intl. Symp. Adv. Res. Async. Circ. Syst. (ASYNC)*, pages 198–209.

[21] I. Sutherland and S. Fairbanks. GasP: A minimal FIFO control. In *Proc. Intl. Symp. Adv. Res. Async. Circ. Syst. (ASYNC)*, pages 46–53. IEEE Computer Society Press, Mar. 2001.

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

[23] C. van Berkel, M. Josephs, and S. Nowick. Scanning the technology: Applications of asynchronous circuits. *Proceedings of the IEEE*, 87(2):223–233, Feb. 1999.

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

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

[26] G. Yee and C. Sechen. Clock-delayed domino for adder and combinational logic design. In *Proc. ICCD*, Oct. 1996.

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