Self-Resetting Latches for Asynchronous Micro-Pipelines

Tiberiu Chelcea   Girish Venkataramani   Seth C. Goldstein
Carnegie Mellon University
Pittsburgh, PA, USA
{tibi,girish,seth}@cs.cmu.edu

ABSTRACT

Asynchronous circuits are increasingly attractive as low power or high-performance replacements to synchronous designs. A key part of these circuits are asynchronous micropipelines; unfortunatelly, the existing micropipeline styles either improve performance or decrease power consumption, but not both. Very often, the pipeline register plays a crucial role in these cost metrics. In this paper we introduce a new register design, called self-resetting latches, for asynchronous micropipelines which bridges the gap between fast, but power hungry, latch-based designs and slow, but low power, flip-flop designs. The energy-delay metric for large asynchronous systems implemented with self-resetting latches is, on average, 41% better than latch-based designs and 15% better than flip-flop designs.

Categories and Subject Descriptors

B.5.1 [**Register-Transfer-Level Implementation**]: Design

General Terms

Design,Performance,Measurement

Keywords

self-resetting latches, micropipelines, asynchronous, low power

1. INTRODUCTION

As technology shrinks and problems of clock distribution and timing closure become increasingly difficult, asynchronous circuits become more attractive, since they offer a modular design paradigm where different sub-circuits of a design can be easily integrated, without the need for retiming. To achieve this effect, asynchronous designs are essentially constructed from a series of pipeline stages which perform synchronization through local handshaking.

The importance of finding energy and performance efficient pipeline structures and seamlessly integrating them with standard toolflows is reflected in a wide body of literature on the subject. One integral aspect of these pipeline styles is the choice of the pipeline data storage unit, which can dramatically influence both performance and energy efficiency. Whereas synchronous registers contain only data storage elements, an asynchronous storage unit (ASU) consists of both storage- (latches, flip-flops) and control elements.

Latch-based ASUs are very common in asynchronous micropipelines [12, 4, 3, 13] since they are fast: the latches are normally open, and thus eliminate the control overheads associated with opening them. However, precisely because of this, glitches are allowed to pass through the latch, leading to large energy consumption overheads. Using edge-triggered D flip-flop (ETDFF) ASUs [14, 10] eliminates glitches, but introduces additional latency overheads on the control paths. As a result circuit performance suffers and the energy-delay metric may be even worse than a corresponding latch-based design.

In this paper we propose self-resetting (SR) latches, a novel ASU, which combines the performance of latch-based designs (by allowing for early data-passing) and the low-energy consumption of ETDFF-based designs (by filtering out glitches). The proposed ASU implementation can easily be integrated with a number of existing pipeline styles.

SR-latches have been integrated into an existing synthesis flow [14, 1] which synthesizes high-level specifications into pipelined asynchronous circuits. The results of simulating Mediabench kernels indicate that the self-resetting latch implementations have an energy-delay metric which is, on average, 41% better than that of the latch-based designs and 15% better than that of ETDFF designs. Furthermore, the SR-latch implementations are, on average, only 14% slower than the corresponding latch-based implementations but 13% faster than ETDFF designs.

In the next section we present some background information on asynchronous pipelines and ASU styles, as well as compare our proposed design with existing designs. Section 3 discusses the implementation of the self-resetting latches, and its timing constraints. Section 4 shows a characterization of the proposed designs, both for performance and for energy. Finally, Section 5 concludes the paper.

2. BACKGROUND

Asynchronous modules communicate on channels, using a handshaking communication protocol; a very common protocol is the 4-phase bundled-data handshaking [10]. A bundle-data channel connects a producer and a consumer, and consists of a data bus, used for transferring data items, encoded with one wire per bit, and two control signals, Req and Ack. The producer starts the hanshake by placing a new data item on the $Data$ bus and raising Req; the consumer acknowledges using the data item by raising the Ack signal. Then, the two control signals are reset to zero in the same order.

A typical asynchronous pipeline stage is shown in Figure 1. It consists of two main blocks: the functional unit (FU) and the con-

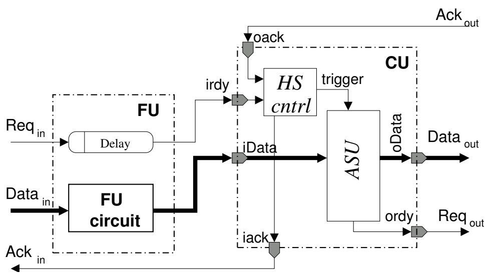

Figure 1: *A typical pipeline stage*

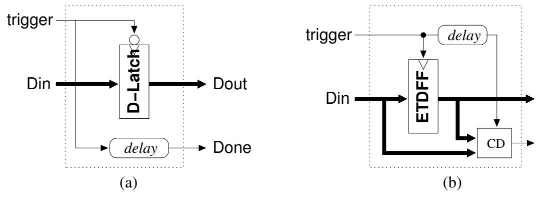

Figure 2: *Typical storage unit implementations for asynchronous pipelines: (a) Latch-based and (b) ETDFF-based.*

trol unit (CU). The FU computes the result of the stage; it consists of a combinational block, which implements the actual computation (e.g. an adder), and a matched delay (usually a chain of inverters) that signals to the CU when the result is ready. The CU controls the communication with the successor and predecessor stages. When the FU has produced a new data item, the CU waits until the successor stages are ready to receive the new data item, stores it in the register, and signals to the consumers its availability.

Latch-based (Figure 2a) and an ETDFF-base implementation (Figure 2b) are two common implementations of the ASU. The latch-based ASUs are fast, but result in increased energy consumption. The critical path in the ASU is from $trigger\uparrow$ to $Done\uparrow$ (closing the latches) and goes through a single delay matching a D-latch. As the latches are normally open, data items are available to the consumers as soon as they are computed; however, glitches are also passed this way, which results in increased power consumption.

The ETDFF ASU's filter out all glitches, but are slow. The critical path ($trigger\uparrow$ to $Done\uparrow$) goes through a completion detection circuit, which checks for equality between the the data item before the storage elements and the data item after the storage elements, and synchronizes this comparison with a delayed version of the "trigger" signal. ETDFF's are normally closed, and data is passed to the consumers only on the last moment (hence the completion detection); thus, all glitches are filtered out, and energy consumption is decreased.

In contrast, our proposed self-resetting latches exhibit low-latency as in the latch-based implementations and filter out glitches as in the ETDFF-based implementations.

2.1 Related Work

Since the seminal work of Sutherland [13], an impressive number of asynchronous pipeline elements have been proposed. They can be categorized as either coarse-grained (each stage operates at the level of "functional units") or fine-grained (stages operate at the bit level).

Fine-grained pipeline structures were introduced by Williams [15], and improved on in [11, 8, 9, 6, 7, 16, 17]. These pipeline templates are usually implemented using dynamic logic, which provides intrinsic storage capability, but are not amenable to automatic synthesis with existing commercial standard-cell CAD tools.

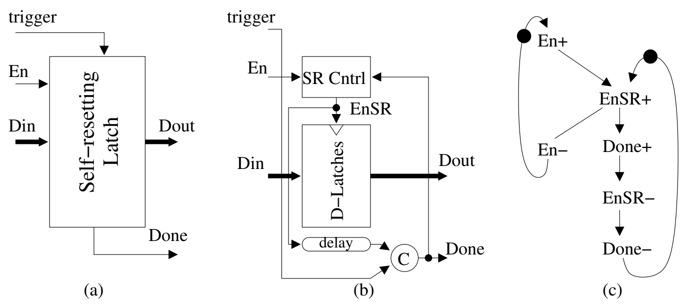

Figure 3: *The implementation of the Self-Resetting ASU.*

Coarse-grained pipeline structures are implemented with standard gates [13, 12, 4, 3], and are thus amenable for manipulation with standard CAD tools. Of these, [13, 12] communicate using 2-phase handshaking. The ASUs in [13] are implemented using slow pass-capture logic. Interestingly, the ASU's in [12] communicates *internally* using 4-phase handshaking; however, the design uses normally open latches, and is thus energy hungry. Finally, the ASUs in [4, 3] are latch-based. Starting from the extremely simple handshake controller of [13] (just a C-element) the authors of [4, 3] present several variants of the handshake controllers based on the pipeline occupancy; the protocol of these controllers controls the data validity on the pipeline latches. This approach results in slow controllers, and thus degraded performance. In contrast, the proposed self-resetting latches do not require any modifications in the pipeline handshake controllers, and they can be used with any 4-phase pipelining style.

3. SELF-RESETTING LATCHES

This section presents the implementation of self-resetting latches and discusses how a pipeline stage needs to be modified to work with SR latches.

3.1 Storage Unit Implementation

To understand the behavior of self-resetting latches, we have to analyze when the input data item to an ASU is passed to its output and when the data item is kept stable (i.e., stored). Latch-based ASUs are open by default and thus continuously pass data from inputs to outputs; they will store the data item (i.e., close) only when the "trigger" control input to the storage unit is raised. ETDFF-based ASUs are always closed and data is passed and stored only on the positive edge of the "trigger" input.

In contrast, self-resetting latches are controlled by two separate events: one for opening the latches to pass data items and one for closing them to store the data items. This separation achieves two goals. First, by controlling when the ASU becomes open, is is possible to filter out glitches, while still passing data items early. Second, by having a separate event for closing the storage elements, all the control overheads associated with ETDFF style can be avoided, since data items are already present at the output.

Figure 3a shows the new interface of the storage unit. Notice that in comparison with the implementations in Figure 2, the self-resetting ASUs have an extra input, "En". This input opens the latches, whereas the "trigger" input now only closes them. The other interfaces are the same: "Din" and "Dout" are data buses, and "Done" indicates when the data item on "Dout" is safe to use.

Figure 3b shows the architecture of an ASU implemented with self-resetting latches. The storage elements are D-latches. In addition, the proposed architecture also has a small controller which controls the D-latches, a matched delay for the storage elements, and a C element.

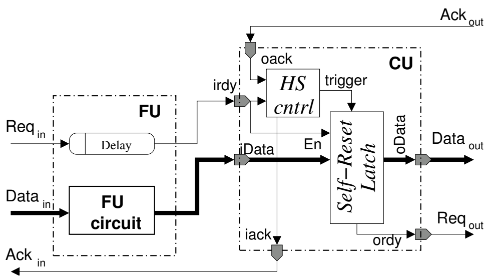

Figure 4: *A pipeline stage with SR Latches.*

The SR-latch works as follows. On the rising edge of the "En" signal the SR controller opens the latches by raising the "EnSR" signal. This signal goes through a delay which matches the delay through the latches and becomes an input to the C element. After a while, the positive edge of the "trigger" signal is synchronized in the C element with the delayed "EnSR" to produce the "Done" signal transition. Once this occurs, the SR controller immediately lowers the "EnSR" signal, thus closing the latches. As will be shown in Section 4, it is possible, though not desirable, to have the "trigger" event come earlier than the "En" event: the ASU will still function correctly, but its latency will increase.

Figure 3c shows the signal-transition graph (STG [2]) specification for the SR controller. The specification follows the behavior described above. This specification can be synthesized with Petrify [2] to obtain the final implementation. In STMicro 180nm technology, the controller can be implemented with 8 gates. The controller occupies 46% of the area of a 1-bit ASU, but for a 32-bit ASU, the control overhead is only 6.5%. In the synthesized Mediabench kernels (Section 4), the total area overhead is only 7% over latch-based implementations.

The design has two one-sided timing constraints: the matched delay has to match the delay through the latches, and the controller must meet the minimum pulse width (i.e. EnSR == 1) constraints for the latches. Any timing violations can be eliminated by increasing the matched delay, at the price of decreased performance.

3.2 Pipeline Implementation

Figure 4 shows the modified generic pipeline stage that uses an SR-based ASU. Notice that the only modification to the architecture of the pipeline is that the output of the matched delay is routed to the "En" input of the self-resetting latch.

A pipeline stage with the SR-latch works exactly as a pipeline stage with latch- or ETDFF-based implementations. Since the output of the matched delay is the "En" input to the ASU, the latches in the SR-based ASU are now open only after the result has reached its final value – and thus all the glitches present in a latch-based implementation are eliminated. In addition, since the latches are opened early, this implementation avoids the overheads of ETDFF implementations.

When using SR-based ASU's in a pipeline, there is an additional constraint that needs to be met. The SR controller has an unacknowledged transition from $En \downarrow$ to $En \uparrow$, which means that the time separation between these events needs to be larger than the internal state changes in the controller. In practice, these events are separated by an entire handshaking cycle, which is typically larger than the delay through the controller (2 gates). This constraint can be met by increasing the matched delay of the functional unit.

4. RESULTS

This section presents several experimental results which characterize the performance and energy efficiency of self-resetting latch-based implementations. All experiments were performed post-layout using the 180nm STMicro Electronics library. We will first characterize the latency of SR latches, and then show the results of integrating SR latches into complex systems.

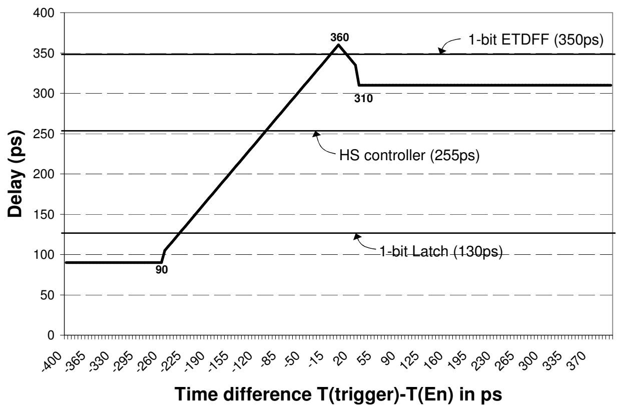

Figure 5: *The latency of a 1-bit Self-Resetting Latch as a function of the time difference (in ps) between "En" and "Trigger" inputs. If negative, "En" arrives before "Trigger".*

Latency Characterization. As mentioned in Section 3, the delay through the SR latch depends on the timing difference between the positive edge of "En" (to open latches) and the positive edge of "trigger" (to close the latches). We have performed an experiment in which we have varied the arrival of "trigger" and "En" inputs to a stand-alone single-bit SR latch. Figure 5 displays the latency through a 1-bit ASU as a function of this difference. For comparison, Figure 5 shows the latency of a 1-bit latch-based ASU (130ps), of a 1-bit ETDFF-based ASU (350ps), and the delay of the SR latch (255ps) when the difference is that of the handshake controller.

The SR latch works correctly in either case ("trigger" before "En", or vice-versa), but its latency varies dramatically. If "En" arrives at least 260ps before "trigger", then the ASU has a minimum latency (90ps – just the C element). In this case, the latency of the SR latch is even lower than that of a 1-bit latch ASU (130ps). After this, the delay increases steadily, until the difference between the two is zero. Once the delay has reached its peak (360ps), it decreases slightly and stabilizes at 310ps when "trigger" arrives at least 30ps after "En". In this case, the SR latch will incur all the penalties of an ETDFF implementation (opening the latches on the critical path) – and, in fact, the latency of the SR latch is close to that of an ETDFF-based ASU (350 ps).

This experiment shows the possibility of some interesting optimizations involving SR latches. First, to increase performance, the SR latches can be opened *before* the FU starts computing; this comes at the expense of increased power consumption. Second, to slow down pipeline stages (for example, to avoid complex arbitration to a shared resource, one of the callers can be slowed down; thus, the other caller can win arbitration faster) the "En" signal can be tapped from "trigger", which would produce maximum latency through the ASU.

System-Level Impact. To fully evaluate the impact of the SR latches on performance and energy consumption, we have integrated them into the synthesis flow of [1]. We have synthesized and simulated several kernels from the Mediabench [5] suite.

Figure 6 shows the speed ratio of the ETDFF-based and SR-based implementations vs. the latch-based implementations for several kernels from Mediabench benchmarks. The SR-based implementations are between 7.1% and 21.8% slower than the latch-

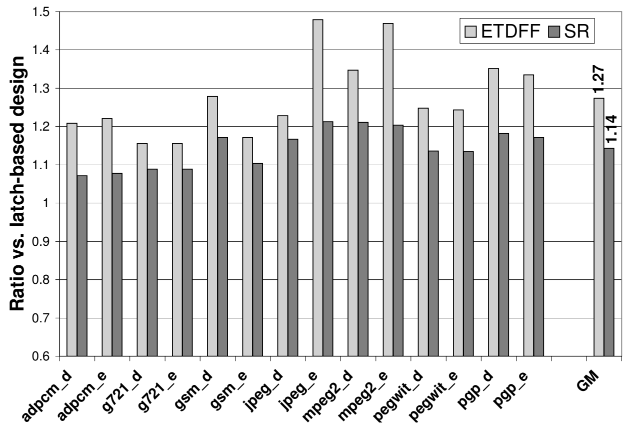

Figure 6: *Speed ratio of ETDFF-based and SR-based implementations vs. latch-based implementations. A value greater than 1 means the design is slower.*

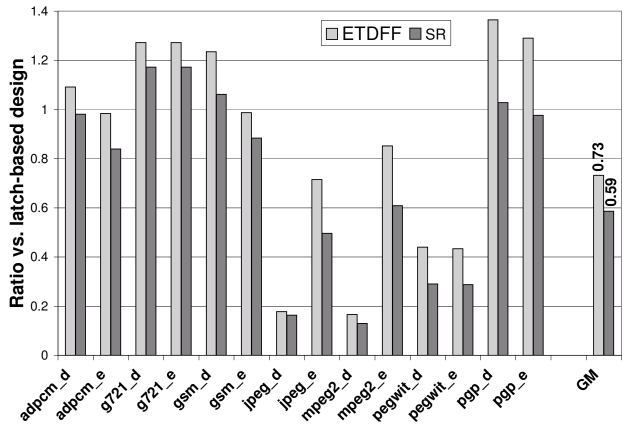

Figure 7: *Energy-delay ratio of ETDFF-based and SR-based implementations vs. latch-based implementations. A value greater than 1 means that the design is less efficient.*

based designs (14.3% on average). In contrast, the ETDFF-based designs are between 15.5% and 47.8% slower, with an average of 27.3%. This experiment confirms that, if speed is the most important cost metric in the design, then the latch-based pipeline implementations should be used.

Figure 7 shows the energy-delay ratio between the ETDFF-based and SR-based implementations vs. the latch-based implementations for the same kernels from Mediabench. The SR-based implementations are on average 41.4% better than the latch-based implementations, and 26.8% better than the ETDFF-based implementations. These results show several interesting characteristics.

First, note that the SR-based design has very good energy-delay for the six largest benchmarks in the set (jpeg, mpeg2, and pegwit, both encoding and decoding for each): the energy-delay is between 0.15 and 0.6 of the latch-based energy delay, even if their speed is 13% to 21% smaller. The power savings for these benchmarks are substantial (11x for mpeg2_d).

Second, for four benchmarks (g721_d, g721_e, gsm_e, and pgp_d), the latch-based design is more energy–delay efficient. The first two benchmarks do not have loops and the the SR implementations do not reduce power, but are slower. For the other two, the power savings are not large enough to offset the decrease in performance.

Finally, these experiments show that the SR implementations are more energy–delay efficient than the ETDFF implementations. In turn, the ETDFF implementations are also *on average* more energy efficient than the latch-based implementations, but for a large number of benchmarks, they perform poorly. This suggests that the SR implementations are a more suitable energy–delay alternative to latch-based implementations than ETDFF-based ones.

5. CONCLUSIONS

Existing latch implemenations for asynchronous pipeline systems have several shortcomings: either they are optimized for speed or for power. This paper introduces a novel storage unit implementation style, called self-resetting latches, which decreases power consumption, while avoiding performance penalties. Our experiments have shown that, in large systems, the benefits of using self-resetting latches can yield an average of 41.4% energy-delay improvement over corresponding latch-based implementations. These results are encouraging and can be further improved by investigating the proposed pipeline optimizations which take advantage of the variable latency of SR-latches.

6. REFERENCES

[1] M. Budiu, G. Venkataramani, T. Chelcea, and S. C. Goldstein. Spatial computation. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 14–26, Boston, MA, October 2004.

[2] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on Information and Systems*, E80-D:315–325, 1997.

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

[4] S. Furber and P. Day. Four-phase micropipeline latch control circuits. *IEEE Transactions on Very Large Scale Integration Systems*, 4-2:247–253, 1996.

[5] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 330–335, 1997.

[6] A. Lines. *Pipelined Asynchronous Circuits*. PhD thesis, Caltech, 1995.

[7] A. J. Martin, A. Lines, R. Manohar, M. Nyström, P. Pénzes, R. Southworth, and U. Cummings. The design of an asynchronous MIPS R3000 microprocessor. In *Advanced Research in VLSI*, pages 164–181, Sept. 1997.

[8] R. Ozdag and P. Beerel. High-speed qdi asynchronous pipelines. In *Proceedings of the Eighth International Symposium on Asynchronous Circuits and Systems, 2002*, pages 13–22, April 8-11 2002.

[9] R. Ozdag, M. Singh, P. Beerel, and S. Nowick. High-speed non-linear asynchronous pipelines. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition, 2002*, pages 1000–1007, March 4-8 2002.

[10] A. M. Peeters. *Single-rail handshake circuits*. PhD thesis, Technische Universiteit Eindhoven, 1996.

[11] M. Singh and S. Nowick. High-throughput asynchronous pipelines for fine-grain dynamic datapaths. In *(ASYNC 2000) Proceedings of the Sixth International Symposium on Advanced Research in Asynchronous Circuits and Systems, 2000*, pages 198–209, April 2-6 2000.

[12] M. Singh and S. M. Nowick. MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines. In *Proc. International Conf. Computer Design (ICCD)*, pages 9–17, Nov. 2001.

[13] I. E. Sutherland. Micropipelines. *Commun. ACM*, 32(6):720–738, 1989.

[14] G. Venkataramani, M. Budiu, T. Chelcea, and S. C. Goldstein. C to asynchronous dataflow circuits: An end-to-end toolflow. In *International Workshop on Logic Syntheiss*, Temecula, CA, June 2004.

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

[16] C. G. Wong and A. J. Martin. High-level synthesis of asynchronous systems by data-driven decomposition. In *40th Design Automation Conference*, pages 539 – 546, June 2003.

[17] K. Yun, P. Beerel, and J. Arceo. High-performance asynchronous pipeline circuits. In *Proceedings of the Second International Symposium on Advanced Research in Asynchronous Circuits and Systems, 1996*, pages 17–28, March 8-21 1996.