

Asynchronous Computing in Sense Amplifier-based Pass Transistor Logic

Tsung-Te Liu, Louis P. Alarcón, Matthew D. Pierson, and Jan M. Rabaey
Berkeley Wireless Research Center, University of California, Berkeley CA 94704
{tliu, lalarcon, mpierson, jan}@eecs.berkeley.edu

Abstract

This paper presents the design and implementation of a low energy asynchronous logic architecture using sense amplifier-based pass transistor logic (SAPTL). The SAPTL structure can realize very low energy computation by using low leakage pass transistors and low supply voltage. The introduction of asynchronous operation in SAPTL further improves energy-delay performance and reliability without increasing hardware complexity. We show two different self-timed approaches using a bundled-data and a dual-rail handshaking protocol, respectively. The proposed self-timed SAPTL architectures provide robust and efficient asynchronous computation using a glitch-free protocol to avoid possible dynamic timing hazards. Simulation results show that the self-timed SAPTL with dual-rail protocol exhibits energy-delay characteristics better than synchronous and bundled-data self-timed approaches.

1. Introduction

As CMOS technology continues to scale, both supply voltage and device threshold voltage must scale down together to achieve the required performance. Lower supply voltage effectively reduces dynamic energy consumption, however, the leakage energy increases dramatically because of lower device threshold voltage [1]. As a result, for low energy applications, the leakage energy the system can tolerate ultimately limits the minimum device threshold voltage. Speed, therefore, benefits only little from technology scaling. The sense amplifier-based pass transistor logic (SAPTL) [2] is a novel circuit topology that breaks this tradeoff to realize very low energy logic without sacrificing speed. SAPTL modules were initially designed to operate synchronously.

The most important motivation for research on asynchronous computing in SAPTL is reliability. Process variability increases dramatically as

technology scales. Therefore, it is hard to design reliable timing schemes using the traditional synchronous approach. To meet a certain reliability requirement, a synchronous approach must use a very conservative design slow enough for the needs of the statistically slowest circuit elements, and thus will fail to exercise the whole capacity of statistically faster parts of the circuit. The asynchronous approach, on the other hand, can exploit local timing information to achieve both speed and reliability. An asynchronous design can get the best performance out of all components independent of statistical variations in local speed while still guaranteeing circuit reliability.

Asynchronous operation is also attractive to the low power designer. The absence of a clock distribution network can significantly reduce power overhead. Furthermore, an idle asynchronous system avoids consuming any active power.

Despite all the advantages of asynchronous operation, the circuit complexity and performance overhead required to implement a handshaking protocol may not be trivial. The overhead cost might offset all benefits and make the asynchronous approach impractical. The SAPTL, however, offers a relatively easy way to realize asynchronous operation. Because it uses differential signaling, we can easily determine when a logical operation completes. Therefore, a self-timed SAPTL topology is a promising candidate for reducing power consumption and improving speed.

This paper presents the design and implementation of self-timed SAPTL and is organized as follows. Section 2 introduces the basic operation and circuit architecture of SAPTL. Section 3 defines the handshaking protocol between two SAPTL modules to realize self-timed operation. Section 4 presents a self-timed SAPTL architecture with bundled-data protocol including the circuit architecture and a detailed performance analysis. Section 5 introduces a glitch-free adaption to the handshaking protocol between self-timed SAPTL modules that can further improve reliability and performance. Section 6 presents the

design and implementation of self-timed SAPTL with a dual-rail protocol. Section 7 discusses the design technique and optimization of C-element circuits for self-timed SAPTL. Section 8 shows simulated performance for energy, delay, and leakage of synchronous SAPTL and self-timed versions with different handshake protocols. Section 9 concludes this paper and presents future research directions.

2. SAPTL Architecture

Fig. 1 shows the basic architecture of a SAPTL circuit, which is composed of a *stack*, a *driver*, and a *sense amplifier* [2].

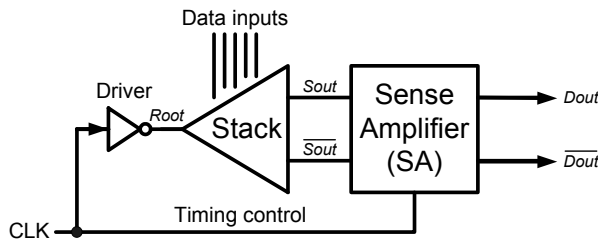


Fig. 1. Architecture of SAPTL module with synchronous timing control.

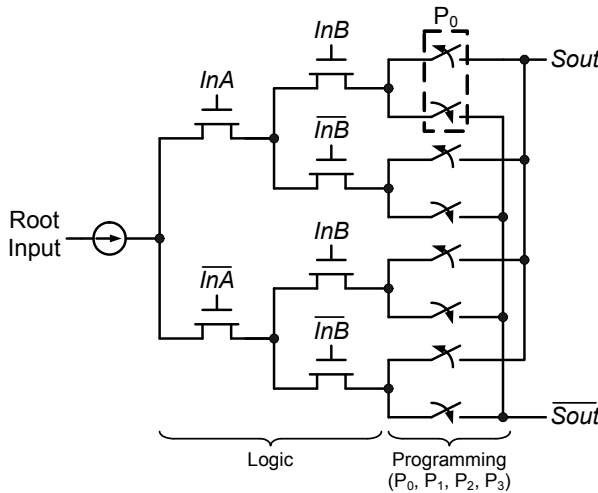


Fig. 2. Schematic of two-input stack.

2.1. Stack and Driver

The *stack* consists of an NMOS-only pass transistor tree with full-swing inputs and low-swing pseudo-differential outputs to perform the required logic function, as shown in Fig. 2. The stack can implement a given Boolean expression by connecting the minterm branches of the tree to one output, and the maxterm branches to the other as illustrated by the programming

switches in the diagram. In our current implementation, the logic function of a SAPTL stack is determined and permanently fixed at fabrication by replacing the programming switches with vias. Because the stack has no supply rail connections, a *driver*, which is a simple inverter in this example, is placed at the root input of the stack to inject the evaluation current. In operation, either *Sout* or \overline{Sout} , but not both, is charged toward the supply rail when the driver acts. After each computation, both differential outputs must be reset to ground. The alternate charging and resetting of *Sout* or \overline{Sout} realizes a standard dual-rail encoding [3].

The speed of the SAPTL module depends strongly on the depth of the stack, N_{stack} , which is defined as the number of transistors in series from the root node to the differential outputs. Because the stack contributes no subthreshold leakage current, the stack transistors can have a very low threshold voltage and still operate in the superthreshold region even with a very low supply voltage. Therefore, SAPTL is a promising candidate to realize ultra low energy computation without soliciting subthreshold operation [2].

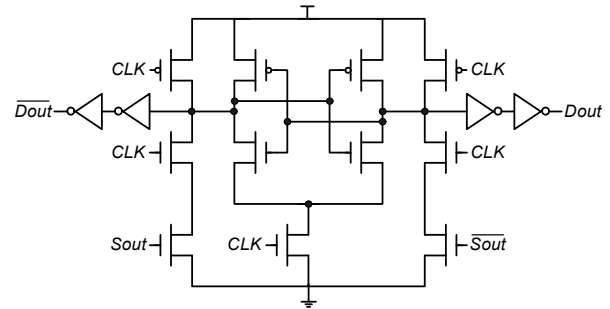


Fig. 3. Sense amplifier circuit.

2.2. Sense Amplifier

The *sense amplifier* shown in Fig. 3 serves two purposes. First, it amplifies the low voltage stack output, restoring the signal to full voltage. Second, it serves as a buffer stage to store the output of the stack, so as to improve overall speed. The sense amplifier consists of two stages; the first stage acts as a pre-amplifier to reduce the impact of mismatch in the actual technology environment, and the second stage acts as a cross-coupled latch which retains the processed data even after the stack is reset. The leakage of the sense amplifier accounts for most of the leakage energy of the SAPTL module. It can be directly traded off against the input sensitivity of the sense amplifier to width, length, and threshold voltage mismatch. The complete analysis and design of the sense amplifier circuit for SAPTL can be found in [2].

2.3. Comparison to DCVSL and Dynamic Logic

The NMOS-only pass transistor tree of SAPTL is similar to the NMOS pull-down network of differential cascade voltage switch logic (DCVSL) [4]. However, SAPTL injects the current at the root of the transistor tree, whereas the root node of the pull-down network in DCVSL is connected to ground. Therefore, the NMOS transistors in the SAPTL stack operate as “source follower”, pulling up either S_{out} or $\overline{S_{out}}$ toward the supply rail, as opposed to the inverting pull-down NMOS transistors in a DCVSL network. As a result, the data inputs to the SAPTL stack do not see the Miller-multiplied gate-to-drain capacitances.

The two-phase evaluation-reset operation of SAPTL is also used in conventional dynamic logic [4]. In dynamic logic, the output nodes are charged to a preset state in the reset cycle, and then conditionally discharged for data evaluation. SAPTL, on the other hands, uses the reset state to discharge the stack and remove any built-up charge from the previous operation. Moreover, in SAPTL, data are stored statically in the sense amplifier, but not on capacitors.

2.4. Synchronous Timing

In the synchronous SAPTL design of Fig. 1, the global clock signal, CLK , controls the timing of the two-phase evaluation-reset operation of the SAPTL module. Alternate SAPTL stages in a pipeline of synchronous SAPTL modules operate in different clock phases; while one evaluates, the other resets the stack. This is similar to the operation of latched-based pipelines in synchronous static CMOS design [4], and requires alternate two-phase non-overlapping clock signals [2]. The following sections will present the design and implementation of self-timed SAPTL modules and pipelines.

3. Self-Timed SAPTL Protocol

Fig. 4 illustrates the communication between two self-timed SAPTL modules which is based on a request-acknowledge handshaking protocol. The operation of self-timed SAPTL involves two parts, data evaluation and reset, as illustrated in Fig. 5. We can think of the handshaking protocol of self-timed SAPTL as similar to the two-phase protocol in a Micropipeline [5]; the self-timed SAPTL module acts in each cycle alternately doing data evaluation and data reset. However, because it consumes new data values only every other cycle, it may be easier to think of this SAPTL protocol as a four-phase handshake [3]. Note that in the reset cycle, both data inputs must be reset to

logical high rather than the commonly used logical low in [3]. When either one, but not both, of the data input signals falls to the logical low, there are new valid data.

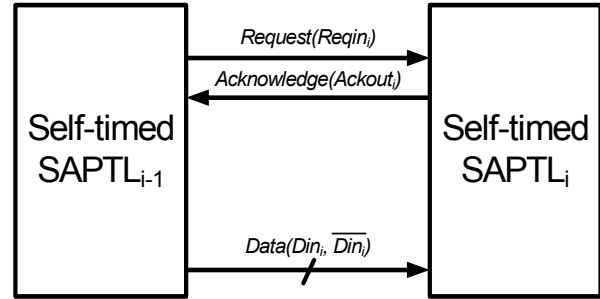


Fig. 4. Communication between two self-timed SAPTL modules.

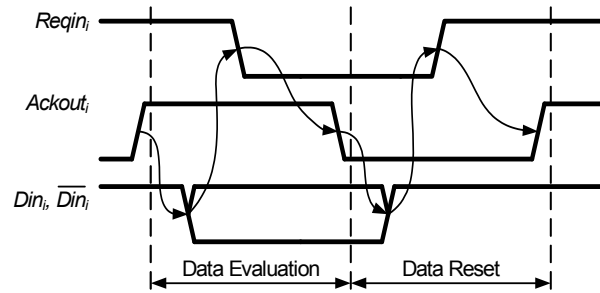


Fig. 5. Two-cycle evaluation-reset operation for the self-timed SAPTL stage i in Fig. 4.

4. Bundled-Data Self-Timed SAPTL Design

Fig. 6 shows the circuit implementation of a self-timed SAPTL module using a bundled-data protocol. The main data path, composed of a driver and a stack, evaluates data or resets after receiving request signal Req_{in} and data input signals Din and \overline{Din} from a previous SAPTL stage. The control path, which consists of a delay line and a C-element, produces the local clock signal $Enable$ to trigger the sense amplifier. The delay line mimics the stack to generate the control signal $Ready$ to indicate when the stack has finished operation. The C-element then produces $Enable$ by collecting $Ready$ and the acknowledge signal Ack_{in} from the next SAPTL stage. When triggered by $Enable$, the sense amplifier latches the stack output data or resets depending on the logical state of $Enable$. The full-swing data output signals D_{out} and \overline{D}_{out} are made available at the outputs of the sense amplifier. The AND gate serves as a completion detection circuit to generate the handshaking signals Ack_{out} and Req_{out} that indicate completion of the current cycle.

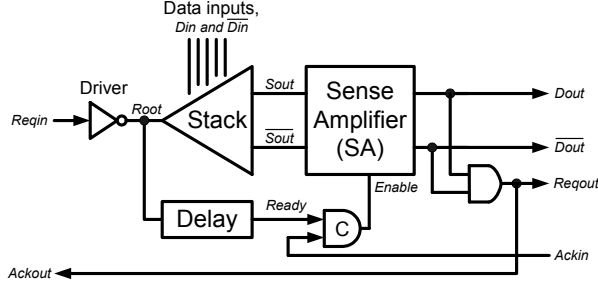


Fig. 6. Architecture of self-timed SAPTL module with bundled-data protocol.

We can summarize the relationship between the input and output signals of a SAPTL stage i as

$$\begin{aligned} D_{out,i}, \overline{D_{out,i}} &= f(Enable_i, S_{out,i}, \overline{S_{out,i}}) \\ &= ff(Req_{in,i}, Ack_{in,i}, Din_i, \overline{Din_i}) \end{aligned} \quad (1)$$

$$\begin{aligned} Ack_{out,i} &= Req_{out,i} \\ &= g(D_{out,i}, \overline{D_{out,i}}) \end{aligned} \quad (2)$$

$$Enable_i = h(Req_{in,i}, Ack_{in,i}) \quad (3)$$

where for the subsequent SAPTL stage $i+1$

$$\begin{aligned} Din_{i+1} &= D_{out,i} \\ \overline{Din}_{i+1} &= \overline{D_{out,i}} \\ Req_{in,i+1} &= Req_{out,i} \\ Ack_{out,i+1} &= Ack_{in,i} \end{aligned}$$

4.1. Data Evaluation Cycle

When a self-timed SAPTL stage finishes a data reset cycle, it raises the acknowledge signal Ack_{out} , $Ack_{out}\uparrow$, and waits for Din , \overline{Din} , and a falling Req_{in} , $Req_{in}\downarrow$, before performing a data evaluation cycle. The delay between $Ack_{out}\uparrow$ and $Req_{in}\downarrow$ is the reverse latency for the data reset cycle $T_{Lr,reset}$ [6], and given by

$$T_{Lr,reset} = T_{C-element} + T_{SA,data} + T_{AND} \quad (4)$$

Before the driver applies the evaluation current, which is controlled by $Req_{in}\downarrow$ the data input signals Din and \overline{Din} must be set to correct logic states. Therefore, we can express the first relative timing assumption (RTA) [7] of SAPTL in the data evaluation cycle by

$$DIN\downarrow < Req_{in}\downarrow \quad (RTA1)$$

where $DIN\downarrow$ indicates that a voltage difference has been developed between the two data input signals Din

and \overline{Din} , which means that one of the two data input signals has been pulled down to logical low, while the other stays at logical high. In most cases, RTA1 can easily be satisfied if the fan-in of the SAPTL is small and the wire delay is insignificant compared to the gate delay. However, if the SAPTL has a large fan-in or the wiring capacitance is significant, RTA1 must be verified carefully after physical layout.

After the stack starts to perform data evaluation, either S_{out} or $\overline{S_{out}}$ will be pulled towards the voltage level $V_{dd}-V_t$. The sense amplifier must be triggered after the outputs of the stack have developed a “sufficiently large” voltage difference. How large the voltage difference must be depends on the design of the sense amplifier; the difference is set to $V_{dd}/3$ in this design. For a valid bundled-data protocol, the following relative timing assumption between data path and control path therefore must be satisfied to ensure correct operation of the sense amplifier

$$SOUT\uparrow < Enable\uparrow \quad (RTA2)$$

or equivalently,

$$T_{Stack,data} < T_{Delay-line,data} + T_{C-element} \quad (RTA3)$$

where $SOUT\uparrow$ represents development of a sufficient voltage difference between the two stack output signals S_{out} and $\overline{S_{out}}$, meaning that one of the output signals has been charged high enough, while the other stays at logic low. Unlike RTA1, which is easily satisfied, it is hard to guarantee RTA2 in actual implementation. Process, voltage, and temperature variations make delay matching difficult between a data path and a separate controlling path. Moreover, the delay characteristic of the stack also depends on input signal statistics, while the delay line has a fixed delay. As a result, the design of the delay line not only must track the “worst case delay” of the stack, but also requires extra margin to compensate for possible process, voltage, and temperature variations. Because of the delay line, the speed performance of a single SAPTL stage is fixed and limited to the slowest possible case even if the computation finished earlier. We can thus determine the forward latency for the data evaluation cycle $T_{Lf,data}$ [6] as the delay between the incoming SAPTL request going down, $Req_{in}\downarrow$, and the outgoing request respectively acknowledge going down, $Req_{out}\downarrow$ and $Ack_{out}\downarrow$, i.e.

$$\begin{aligned} T_{Lf,data} &= T_{Driver} + T_{Delay-line,data} + T_{C-element} + T_{SA,data} \\ &\quad + T_{AND} \end{aligned} \quad (5)$$

4.2. Data Reset Cycle

In the data reset cycle, we can derive similar timing assumptions and forward and reverse latencies for the self-timed SAPTL module, giving

$$DIN\uparrow < Reqin\uparrow \quad (RTA4)$$

$$SOUT\downarrow < Enable\downarrow \quad (RTA5)$$

$$T_{Lf,reset} = T_{Driver} + T_{Delay-line,reset} + T_{C-element} + T_{SA,reset} + T_{AND} \quad (6)$$

$$T_{Lr,data} = T_{C-element} + T_{SA,reset} + T_{AND} \quad (7)$$

where $DIN\uparrow$ indicates that both data input signals have been reset to logical high and $SOUT\downarrow$ means that both stack outputs have been reset to logical low.

In contrast to the Micropipeline two-phase handshaking protocol [5] which is able to exploit both phases for data transmission to achieve maximum throughput, the self-timed SAPTL architecture can transmit data only in alternate phases because the stack must reset after every data evaluation cycle. So, one handshake phase in the self-timed protocol is always dedicated to data reset. As a result, the minimum cycle time T_P to transmit valid data in a self-timed SAPTL stage is given by

$$T_P = T_{Lf,data} + T_{Lr,data} + T_{Lf,reset} + T_{Lr,reset} \quad (8)$$

4.3. Speed Enhancement by Signal Restructuring

It is possible to improve the speed performance of self-timed SAPTL by signal restructuring. By carefully inspecting (1)-(3), we can observe that the signal $Reqout$ is actually a delayed version of signal $Enable$, although they are not logically equivalent. Therefore, a more aggressive timing strategy is to use $Enable$ as acknowledge signal $Ackout$ for the current SAPTL stage, as shown in Fig. 7. This new SAPTL design can achieve higher speed performance with the same functionality under certain relative timing assumptions. It is safe to generate the acknowledge signal from $Enable$ rather than from the output of the AND gate, provided the conversion time of the sense amplifier $T_{SA,data}$ is smaller than the sum of the reverse latency $T_{Lr,reset}$ and the time to reset the stack $T_{Stack,reset}$ in the next data reset cycle, which can be written as

$$T_{SA,data} < T_{Lr,reset} + T_{Stack,reset} \quad (RTA6)$$

As a result, the SAPTL module can begin the handshaking process for the data reset operation of its next cycle immediately after the stack completes data evaluation, and concurrently with the sense amplifier starting to latch the evaluated data. This scheme shortens the reverse latency of a SAPTL stage, and thus yields higher throughput than the original handshaking scheme. The new reverse latencies $T_{Lr,data,new}$ and $T_{Lr,reset,new}$, and the new minimum cycle time $T_{P,new}$ can be given by

$$T_{Lr,data,new} = T_{Lr,data} - T_{SA,data} - T_{AND} \quad (9)$$

$$T_{Lr,reset,new} = T_{Lr,reset} - T_{SA,reset} - T_{AND} \quad (10)$$

$$T_{P,new} = T_P - T_{SA,data} - T_{SA,reset} - 2T_{AND} \quad (11)$$

The new minimum cycle time $T_{P,new}$ can easily be verified by observing the loop behavior starting at the output of the C-element, and remembering that one has to do this loop twice: once for data evaluation, and once for reset.

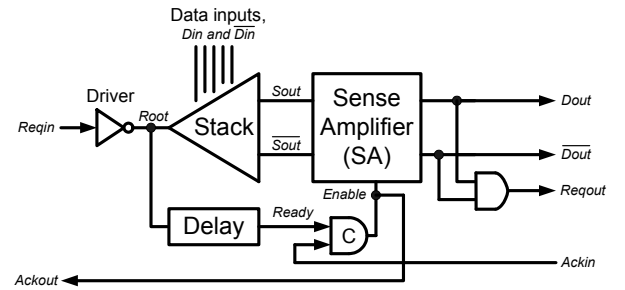


Fig. 7. Alternative self-timed SAPTL structure with speed enhancement.

4.4. Glitch Problem

Fig. 8 shows the timing diagram of the self-timed SAPTL modules in Fig. 6 and Fig. 7 for two successive cycles. The solid arrows represent control flow for normal operation, while the light-gray arrows represent timing sequence governed by a relative timing assumption. The dotted arrows indicate the unwanted events that may introduce glitches in the data reset cycle. Such a glitch can be harmless if its amplitude is less than the trigger threshold of the sense amplifier, or in other words, if the glitch takes a relatively long settling time to develop the voltage level in excess of the trigger threshold. The glitch will always happen in the interval between the two events $DIN\uparrow$ and $Reqin\uparrow$, and its duration is approximately equal to the sum of the delays contributed by the driver

T_{Driver} and AND gate T_{AND} . Therefore, the relative timing constraint to guarantee the system functionality in the presence of a glitch can be expressed as

$$T_{Driver} + T_{AND} < T_{Stack,data} \quad (RTA7)$$

If the design of SAPTL fails to meet RTA7, both outputs of the stack may simultaneously be in the logical state “1”, which violates the basic assumption of SAPTL operation. The overall system may thus enter a metastable state and the operation of SAPTL will be ill-defined. Typically, the sum of T_{Driver} and T_{AND} is much smaller than the time required by the stack to pull up its output nodes, $T_{Stack,data}$. Therefore, RTA7 is usually satisfied and the glitch cannot cause problems. However, the generation of a glitch increases the total energy dissipation of self-timed SAPTL. The unwanted stray signal coupling induced by the glitch may also affect overall reliability. In the next section, we will discuss the design and implementation of glitch-free handshaking protocol for self-timed SAPTL to prevent this glitch problem.

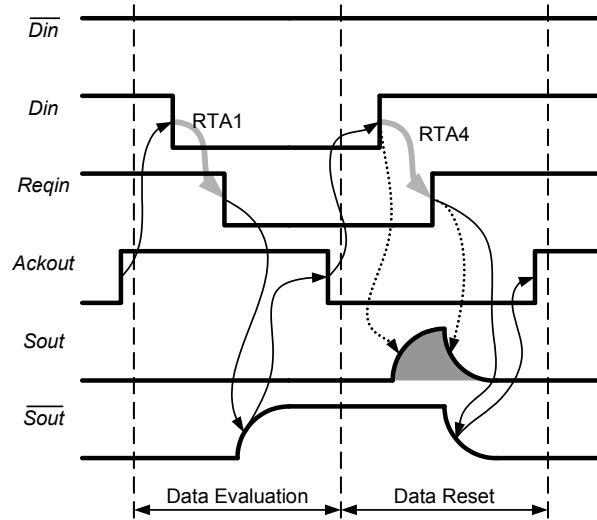


Fig. 8. Timing diagram of self-timed SAPTL.

5. Glitch-Free Handshaking Protocol

It is interesting to note that the glitch in Fig. 8 can occur only in a data reset cycle. The primary reason is the adoption of RTA4. While RTA4 is perfect for initializing the next data evaluation cycle in a Micropipeline architecture, it is not appropriate for performing the data reset operation in a self-timed SAPTL architecture. In the data reset cycle, both the SAPTL data inputs, Din and \overline{Din} , are charged to the supply voltage as “reset signals”, rather than as “data

valid signals” for data evaluation. Consequently, if the $Reqin$ signal triggering the driver is still logical low, the stack will perform a false data evaluation and produce a glitch. Therefore, the revised relative timing constraint to avoid the glitch in the data reset cycle can be given by

$$Reqin\uparrow < Din\uparrow \quad (RTA8)$$

5.1. Protocol Design

Fig. 9 shows two approaches to realize RTA8, which we call the *early reset* and the *late reset* protocol, respectively. In the *early reset* protocol, we introduce another event $Reqin^*\uparrow$ between the original $Ackout\downarrow$ and $Din\uparrow$ events, as shown in Fig. 9(a). We can use the $Reqin^*$ signal, instead of $Reqin$, as a triggering signal to start the data reset operation earlier and avoid generating a glitch. The other way to implement glitch-free operation, the *late reset* protocol, is shown in Fig. 9(b). The stack employs signals Din^* and \overline{Din}^* , which are the replica delayed versions of Din and \overline{Din} , as reset input signals in the data reset cycle. The only requirement for Din^* and \overline{Din}^* is that both signals be triggered later than $Reqin$.

From Fig. 9, we can observe that the timing slack to implement the early reset protocol is smaller than for the late reset protocol, or in other words, the implementation of the early reset protocol requires stricter relative timing assumptions. However, employing the early reset protocol will not affect the original latency of the data reset operation. Therefore, the early reset protocol can achieve higher speed performance than the late reset protocol. Moreover, the early reset protocol can also minimize the leakage current consumption per handshaking operation of the SAPTL. This advantage of low-leakage operation makes the early reset protocol the preferred option.

5.2. Circuit Implementation

Fig. 10 shows the circuit implementation of a self-timed SAPTL module with the early reset glitch-free handshaking protocol. Fig. 11 shows the corresponding timing diagram for two successive evaluation and reset cycles. The additional OR gate and extra input $Reqin$ to the C-element do not change the functional behavior of self-timed SAPTL in the data evaluation cycle. In a data reset cycle, however, $Reqin^*$ will be pulled up to logical high and will start resetting the internal nodes of the stack immediately after the SAPTL module lowers the acknowledge signal $Ackout$. No glitch will be generated during the data reset cycle if $Reqin^*$ is charged to logical high and

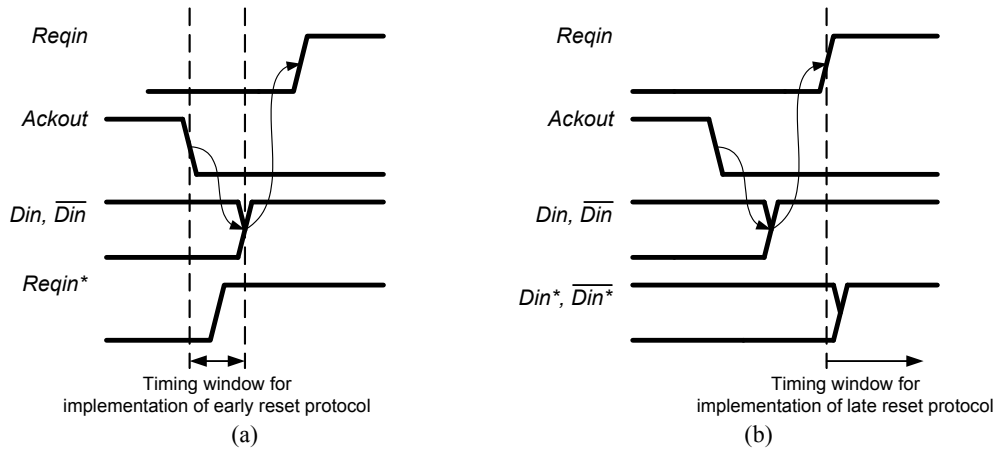


Fig. 9. (a) Early reset and (b) late reset glitch-free handshaking protocol.

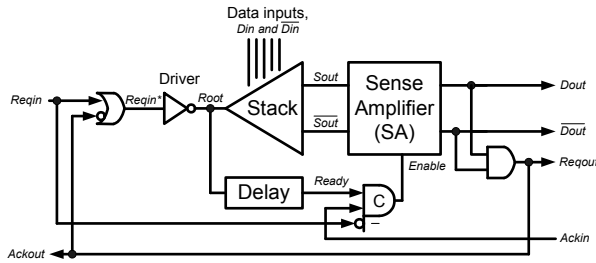


Fig. 10. Self-timed SAPTL structure with early reset glitch-free protocol.

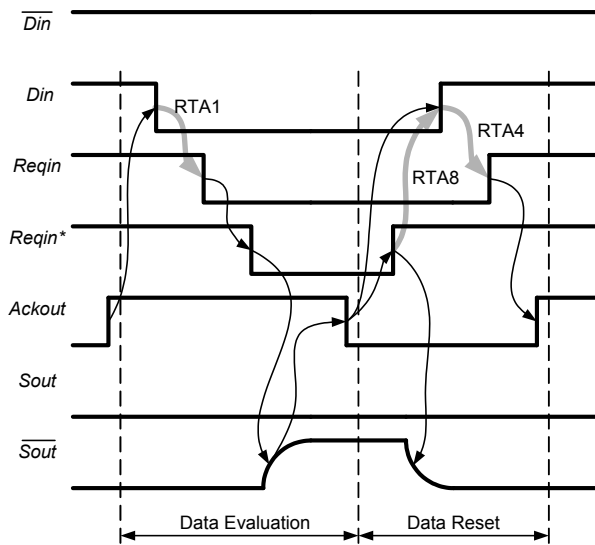


Fig. 11. Timing diagram of glitch-free self-timed SAPTL.

drives the stack root low before Din and \overline{Din} go high, which can be described by the following relative timing constraint

$$T_{Driver} + T_{OR} < T_{C-element} + T_{SA,reset} \quad (RTA9)$$

Because the OR gate and driver may be merged into one NOR gate, the total delay in the left hand side of RTA9 is really small and RTA9 can thus easily be met.

The extra input $Reqin$ to the C-element is essential to maintain the reset state of the current SAPTL stage until the previous SAPTL raises $Reqin$ for the next data evaluation cycle. This explains the three-input C-element. Note that the third C-element input signal $Reqin$ is necessary only in the data reset cycle, but not in the data evaluation cycle. Therefore, we are able to use an asymmetric C-element circuit in this glitch-free self-timed SAPTL for minimum delay and energy consumption [3]. In Section 7, we will discuss the circuit implementation and optimizations for this three-input asymmetric C-element in more detail.

By employing an additional NOR gate and a higher fan-in asymmetric C-element, the self-timed SAPTL architecture can achieve higher reliability and lower energy consumption and avoid the glitch problem. In addition, there are several advantages to resetting the stack immediately after the SAPTL module sends the acknowledge signal. First, $Sout$ and \overline{Sout} will stay above ground for only the short period required by \overline{Ackout} for the sense amplifier to latch their data. Once $Dout$ or \overline{Dout} has reached full swing, every internal node of the stack, as well as $Sout$ and \overline{Sout} , will be reset to ground. Therefore, during the remainder of the cycle, the stack will stay in the data reset mode and consume minimum leakage power. Finally, because the handshaking

events with the previous SAPTL stage and the data reset events within the current SAPTL stack operate in parallel, the SAPTL will have a lower latency data reset cycle and thus achieve higher speed performance.

This leaves the employment of a delay line as the major performance limitation in the self-timed SAPTL design with bundled-data protocol. In the next section, we will introduce a self-timed SAPTL version with dual-rail protocol to address this issue.

6. Dual-Rail Self-Timed SAPTL Design

In a self-timed SAPTL structure with bundled-data protocol, RTA2 is the most critical design constraint. In order to guarantee reliability under process, voltage, and temperature variations, the latency of the delay line may become very large, and severely limit overall performance. Because SAPTL uses dual-rail coding to represent data, we can use the output signals of the stack $Sout$ and \overline{Sout} , instead of $Ready$ from the delay line, to trigger the C-element. As a result, we can eliminate the delay line, and the C-element will respond immediately after the stack finishes operation, without being limited by RTA2. Furthermore, we can combine the sense amplifier and the C-element circuits into a composite block through gate-level optimization, which yields a more energy-efficient architecture, as shown in Fig. 12. The optimized architecture with dual-rail protocol eliminates the sense amplifier circuit completely and directly employs two C-element circuits as a gain stage at the outputs of the stack. The overall conversion speed, however, may be slower than the design with a sense amplifier due to the absence of a differential amplification and the loss of a positive-feedback mechanism between the two data paths.

Fig. 13 shows the implementation of a glitch-free self-timed SAPTL architecture without the delay line. The design and performance of the C-element circuits are especially important in this architecture because the C-element not only plays the role of the gain stage but also serves as the handshaking element. The self-timed SAPTL with dual-rail protocol has latency and cycle time expressions similar to (4)-(8). Note that the speed enhancement in Fig. 7 of section 4.3 does not apply to the dual-rail design in Fig. 13, because we removed the internal $Enable$ signal. However, the single self-timed SAPTL stage is now elastic and able to achieve best performance across process, voltage, and temperature variations and different input characteristics. The self-timed SAPTL can thus exercise the full potential of asynchronous computation without the limitations of the delay line.

It is interesting to note that the optimized self-timed SAPTL architecture in Fig. 13 has almost as little hardware complexity as the original synchronous SAPTL design in Fig. 1. This means that with almost “zero cost” SAPTL is able to achieve both better performance and better reliability from asynchronous operation.

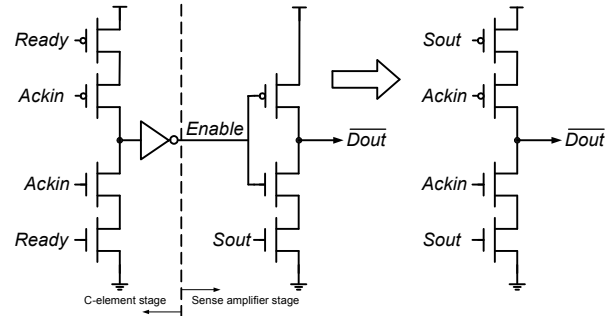


Fig. 12. Logic combination of two-input C-element and sense amplifier circuits.

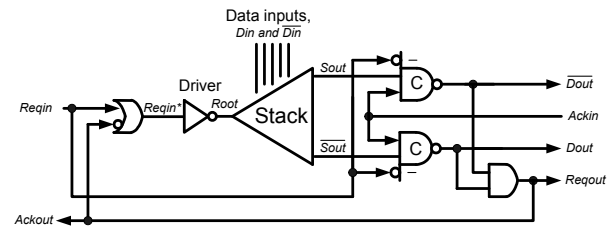


Fig. 13. Architecture of glitch-free self-timed SAPTL module with dual-rail protocol.

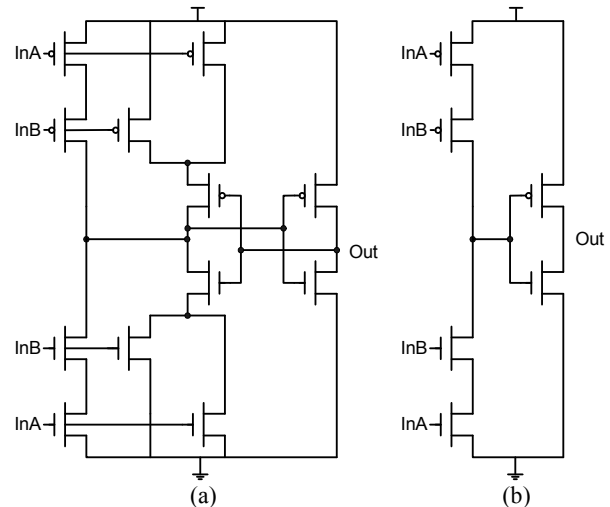


Fig. 14. (a) Static and (b) dynamic two-input symmetric C-element circuits.

7. C-Element Circuit Design

Various kinds of static and dynamic C-element circuits can serve different applications [8]. Fig 14 shows typical static and dynamic CMOS implementations of symmetric two-input C-element circuits [3] [5]. Higher fan-in C-elements can easily be realized by using similar circuit structures.

Fig. 15 shows a three-input asymmetric C-element circuit [3] to implement the glitch-free handshaking protocol used in Fig 10 and Fig. 13. In the data evaluation cycle, the three-input asymmetric C-element circuit achieves speed performance similar to a two-input C-element. This is especially favorable if the C-element circuit is also used to replace a sense amplifier circuit to restore the signal level in the outputs of the stack. To further speed up the data amplification process, the C-element circuit can use skewed gate sizing. In a self-timed SAPTL architecture with dual-rail protocol, the additional keeper circuit reduces the leakage consumption of the C-element circuit by restoring the input swing to full voltage after the stack finishes data evaluation. In the reset cycle, on the other hand, the keeper circuit is completely turned off by the signal $Reqin^*$ without affecting the stack reset.

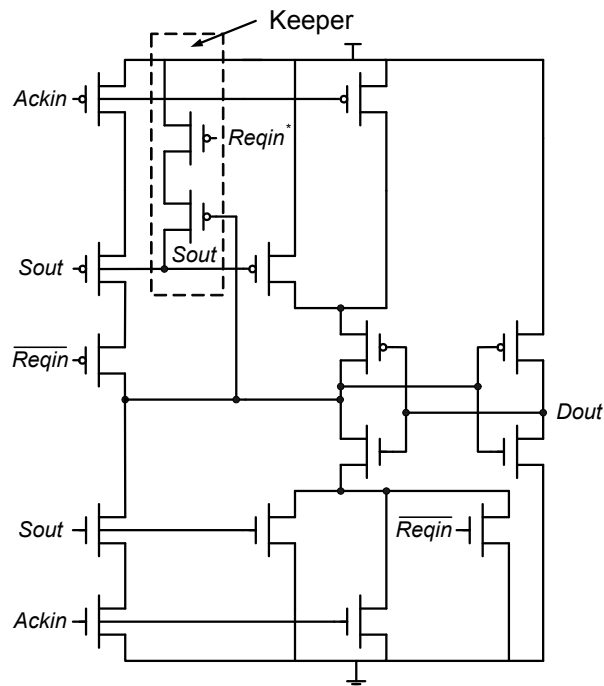


Fig. 15. Three-input asymmetric C-element circuit for glitch-free self-timed SAPTL.

An alternative C-element design for the glitch-free handshaking protocol is illustrated in Fig. 16. In this

circuit, two NMOS pass transistors, controlled by signals $Ackout$ and $Reqin$, respectively, serve as the decision-making logic before the signal $Sout$ can trigger the following two-input C-element. $Sout$ can normally trigger the C-element without impediment except for the period when $Ackout$ and $Reqin$ are both in the logical low state. This happens during a data reset cycle, as depicted in Fig. 5, when the C-element is driven by the two back-to-back inverters in its feedback circuitry. Therefore, the state of the current SAPTL stage is maintained until the previous SAPTL stage raises the request signal $Reqin$ for the next data evaluation cycle. Compared to the C-element circuit in Fig. 15, the circuit configuration shown in Fig. 16 has slightly slower speed performance during data evaluation, but is able to achieve lower leakage current consumption because it has fewer leakage paths. Moreover, the signal swing at the stack outputs can remain at a low voltage because the keeper circuit in Fig. 16 restores only the signal after the decision-making logic stage to full voltage.

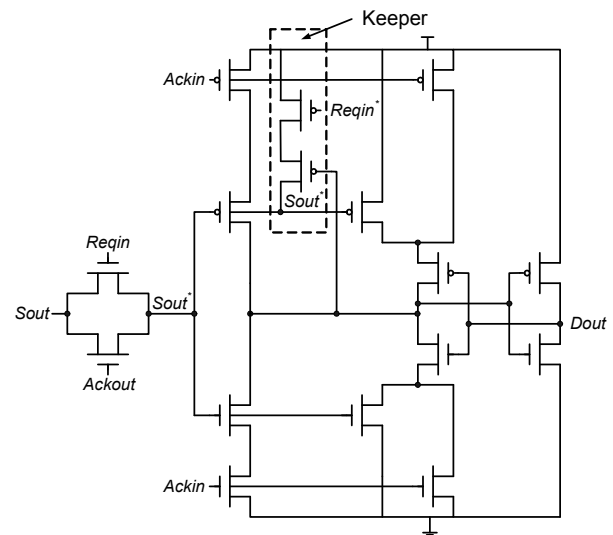


Fig. 16. Two-input C-element circuit with additional decision-making logic for glitch-free self-timed SAPTL.

8. Simulation Results

We evaluated and compared the performance of the synchronous SAPTL and the self-timed SAPTL circuits with bundled-data and dual-rail protocol in CMOS 90-nm technology using the Spectre circuit simulator. In order to reflect the actual technology environment, we also perform Monte Carlo simulations so as to ensure the reliability of SAPTL circuits even with $6\text{-}\sigma$ process variations. This section presents the energy-delay and leakage behaviors of synchronous versus self-timed SAPTL. The

simulations exclude the parasitic contributions from the interconnect wires and the clock network. Performance comparisons between SAPTL and other logic styles can be found in [2].

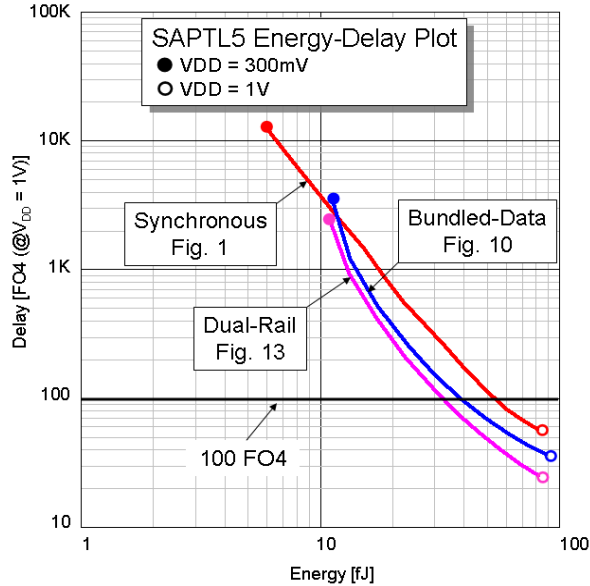


Fig. 17. Energy-delay plots of different implementations of SAPTL as the supply voltage varies from 300mV to 1V.

8.1. Energy-Delay Simulations

Fig. 17 shows the energy and delay behavior of a synchronous SAPTL module and two versions of self-timed SAPTL modules as the supply voltage varies from 300mV to 1V. Each module has five data inputs ($N_{stack}=5$) as indicated by the term “SAPTL5” in Fig. 17. The energy results for the synchronous SAPTL exclude the clock distribution energy. We can observe that both self-timed SAPTL configurations have better energy-delay characteristics than the synchronous design. This is because the sense amplifiers in self-timed SAPTL are able to respond earlier as the output swing of the stack starts to build up. Moreover, employing the early reset glitch-free protocol also reduces the delay and the energy consumption from leakage. Although self-timed SAPTL with the bundled-data protocol can theoretically achieve its highest speed performance by overlapping the data evaluation and reset cycles, as shown in (9)-(11), the delay line will require extra padding to compensate for process variations and create both delay and energy overhead. As a result, the self-timed SAPTL with dual-rail protocol emerges as the most efficient design. With a delay of 100 fanout-of-four (“100 FO4” in Fig. 17) the self-timed SAPTL with dual-rail protocol can achieve

more than 15% and 30% energy saving compared against the bundled-data and synchronous SAPTL designs, respectively. The leakage of the C-element circuit accounts for most of the overall energy consumption when the self-timed SAPTL operates with low supply voltage, and thus prevents us from efficiently achieving any further energy reduction by scaling down the supply voltage. The synchronous SAPTL architecture appears to be capable of lower energy operation at low supply voltage. But this is misleading, because the energy simulation of the synchronous SAPTL omits the energy contribution from the clock network, which corresponds to the energy cost of the C-elements in the self-timed designs.

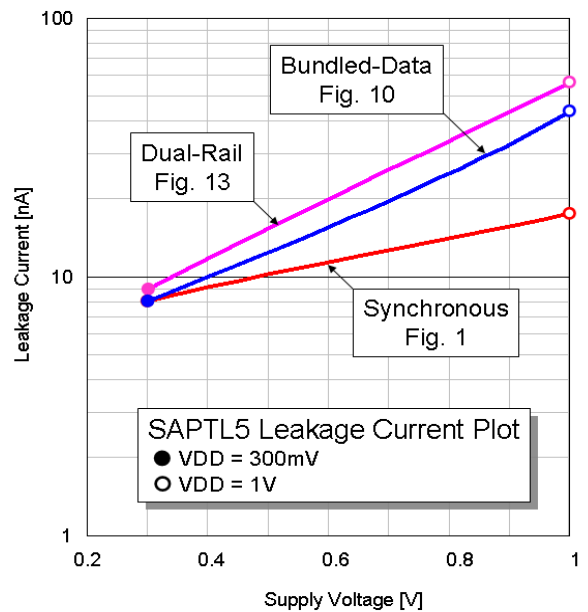


Fig. 18. Leakage current of different implementations of SAPTL as the supply voltage is varied from 300mV to 1V.

8.2. Leakage Simulations

Fig. 18 shows the leakage behavior of different SAPTL designs as a function of supply voltage. The self-timed SAPTL with dual-rail protocol has slightly higher leakage current consumption because the C-element circuit is employed as the gain stage as well as the handshaking element. Compared to the design with both C-element and sense amplifier circuits, its conversion speed is slower and hence more leakage prone. The synchronous SAPTL design will have leakage performance similar to the self-timed SAPTL design with bundled-data protocol if we include the leakage from a clock distribution network associated with synchronous SAPTL.

9. Conclusions and Future Work

Asynchronous operation in SAPTL provides reliability and performance advantages over synchronous operation. While self-timed SAPTL with a bundled-data protocol can potentially achieve higher speed performance by overlapping the data evaluation and reset cycle, the self-timed design based on a dual-rail protocol has less rigid relative timing constraints and provides higher reliability, which leads to better energy and speed performance in technologies with process variations. The early reset operation of self-timed SAPTL not only prevents dynamic timing hazards from glitches, but also improves both energy and speed performance. The low implementation cost of the asynchronous operation makes the self-timed SAPTL family a very promising candidate to realize robust and low-energy computations.

The self-timed SAPTL architecture shown in Fig. 13 represents one of the extreme designs to realize asynchronous computations with the least hardware complexity. However, due to the absence of the differential sense amplifier circuit, Fig. 13 is not necessarily the optimum self-timed SAPTL architecture to achieve the best energy-delay characteristic. We expect that the optimum self-timed SAPTL design will be some combined approach of the two self-timed structures shown in Fig. 10 and Fig. 13. We are investigating this as part of our current research.

We are also developing a design methodology and tool flow to support large-scale system design based on SAPTL. Two CMOS 90-nm test chips have been designed and are being tested to verify our ideas.

10. Acknowledgment

The authors would like to acknowledge Marly Roncken and Ivan Sutherland for valuable discussions and reviews of this paper. The authors also wish to acknowledge the contributions of the students, faculty and sponsors of the Berkeley Wireless Research Center, the National Science Foundation Infrastructure Grant No. 0403427, the wafer fabrication donation of STMicroelectronics, and the Gigascale Systems Research Center (GSRC) for their support of this research.

11. References

- [1] T. Sakurai, "Perspectives on Power-Aware Electronics," in *ISSCC Digest of Technical Papers*, vol. 1, pp. 26-29, 2003.
- [2] L. Alarcón, T.-T. Liu, M. Pierson, and J. Rabaey, "Exploring Very Low-energy Logic: A Case Study," *Journal of Low Power Electronics*, vol. 3, no. 3, pp. 223-233, Dec. 2007.
- [3] J. Sparsø and S. Furber, *Principles of Asynchronous Circuit Design*, Kluwer Academic Publishers, 2001.
- [4] J. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits*, Prentice Hall, 2003.
- [5] I. Sutherland, "Micropipelines," *Communications of the ACM*, pp. 720-738, June 1989.
- [6] T. Williams, "Performance of Iterative Computation in Self-Timed Rings," *Journal of VLSI Signal Processing*, 7, pp. 17-31, Oct. 1993.
- [7] K. Stevens, R. Ginosar, and S. Rotem, "Relative Timing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, pp. 129-140, Feb. 2003.
- [8] M. Shams, J. Ebergen, and M. Elmasry, "Optimizing CMOS Implementation of the C-Element," *Proceedings of 1997 International Conference on Computer Design*, pp. 700-705, Oct. 1997.