Efficient Failure Detection in Pipelined Asynchronous Circuits

Song Peng and Rajit Manohar
Computer Systems Laboratory
Cornell University
Ithaca, NY 14853, USA
{speng,rajit}@csl.cornell.edu

Abstract

This paper presents an efficient concurrent failure detection method for pipelined asynchronous circuits. We first validate permanent and transient fault modeling in clockless systems. By augmenting the rails to each data channel and adding extra logic to each circuit module, we make pipelined asynchronous circuits achieve fail-stop with respect to hard or soft errors. The experimental evaluations show this method incurs both reasonable hardware cost and low performance overhead.

1 Introduction

According to the definitions in [6], a *fault* is a physical defect, imperfection, or flaw that occurs within some hardware components, an *error* is the manifestion of a fault, and a *failure* is the non-performance of some action that is due or expected. Although a failure must be caused by some fault(s), a fault does not necessarily cause a failure. Thus, we can say a system is *fault tolerant* if that system can correct its abnormal behavior in the presence of failures (not necessarily all faults). The first step to achieve the fault tolerance is to enable failure detection. In other words, observable abnormal behavior must show up when any failure occurs. One way to implement such observable abnormal behavior is to force system stop in the presence of failures, preventing further error propagation (fail-safe [6]).

With higher clock frequency, decreased feature sizes and increased transistor counts, clock distribution and wire delays present a growing challenge to the designers of singly-clocked, globally synchronous systems. It is becoming more and more difficult and expensive to distribute a global clock signal with low skew throughout a processor die. On the other hand, asynchronous circuits do not suffer such problem due to no global clock. This fact makes it more attractive for researchers to eventually abandon singly-clocked globally synchronous systems in favor of asynchrony [17], making asynchronous circuits be an important topic in future digital VLSI designs.

Quasi delay-insensitive (QDI) circuits, which are considered in this paper, are an important class of asynchronous circuits that operate correctly regardless of gate delays. These circuits do not use any clock signal for sequencing, as well as no logic gate timing assumption is required. They are designed as a collection of concurrent hardware modules (called *processes*) that communicate atomic data items (called *tokens*) with each other through one-to-one message-passing channels. The message-passing channels usually consist of data and acknowledge rails. The notion of causality and event-ordering is implemented in terms of handshake protocols on those channels [10].

While there is a wealth of literature that examines concurrent failure detection in clocked logic, little attention has been paid to QDI circuits. The absence of clock signals means that a faulty asynchronous circuit might exhibit problems that would not normally arise in a clocked system, making existing failure detection techniques for synchronous systems ineffective or inefficient [7]. For instance, it is non-trivial to apply duplication-based concurrent failure detection to QDI logic

without significant timing assumptions [4, 18]. Unlike clocked systems where outputs from both replicas can be sampled at the same time and compared against each other, the lack of a global synchronization in QDI circuits makes it unclear when those outputs are expected to match. Moreover, faults in QDI logic may prevent the result from appearing on the output, blocking comparison procedure. Thus, efficient failure detection methods have to be explored for asynchronous circuits.

The goal of this paper is to develop an efficient concurrent failure detection method for pipelined QDI circuits through achieving fail-stop with respect to permanent or transient errors. Specifically, the following contributions are made in this paper: (1) We validate the modeling of a transient fault as a momentary bit-flip in clockless circuits through transistor-level simulations (Section 2). (2) We propose an effective method to force pipelined QDI circuits to deadlock in the presence of failures by permanent or transient faults, with reasonable hardware cost and small performance overhead (Section 3). (3) We present a systematic way of deadlock detection with the minimum cost in an asynchronous system (Section 3). Section 4 shows the experimental results. We review the related work in Section 5 and draw conclusions in Section 6.

2 Fault Modeling in Asynchronous Circuits

This section describes the fault modeling in asynchronous circuits. According to [6], faults experienced by semiconductor devices can be either permanent or transient[1].

2.1 Permanent faults

One standard fault model for permanent faults is single stuck-at fault (SSAF) model [6]. According to SSAF model, a circuit line is stuck-at one or zero if it is disconnected from any other circuits' wires and connected to the power supply or ground respectively. Although SSAF model is simple and cannot represent all possible fault cases, it can cover at least 70% of fabrication defects [1] as well as many permanent faults and unrecoverable failures [12]. Consequently, techniques to detect single stuck-at faults are expected to discover most permanent faults, making circuit achieve high lifetime reliability. Since SSAF model has nothing to do with clocks, it can be applied to asynchronous circuits without any change. Thus, we choose SSAF as hard error modeling for the fail-stop asynchronous circuit construction.

2.2 Transient faults

The analog behavior of transient faults by particle strikes is very close to a double exponential current injection [16]. A discrete model used to approximate the same behavior can be a *glitch*: a voltage pulse with the chosen duration and amplitude for the transient current strength [16]. Like in RAM cells, the glitch is usually further simplified to be *single event upset* (a momentary bit-flip) in synchronous logic, due to the fact that a transient change in the value of a logic circuit will not affect the computation results unless it is sampled by a memory circuit such as a flip-flop [3]. However, the absence of clock signals in asynchronous circuits changes the impact of a glitch. Although there are equivalent memory circuits (state-holding elements) in asynchronous systems, which are expected to convert a glitch into a bit-flip at the logic-level, it is possible for the glitch to propagate in reality: an output pulse will be generated if the input glitch is weak enough not to accumulate or remove sufficient charge at the output node. Thus, to model transient faults in asynchronous circuits as single bit-flips has to be validated. Specifically, we should investigate: (1) What is the probability for a glitch to propagate through state-holding element(s) in an asynchronous circuit; (2) How can we reduce that probability.

Without considering logic masking effect (i.e., the gate output is completely decided by the glitch-affected input), a state-holding element with different circuit topologies, can be modeled

[1] An intermittent fault is taken as either a permanent or a transient fault, depending on its behavior.

as a state-holding (dynamic) gate (shown as Figure 1) with a staticizer at the output (shown in the dashed box). R_1-R_4 of Figure 1 represent the on-resistances of other transistors. Glitch propagation is investigated as follows. Initially, the state-holding output S_o is set to logic low, and both pull-up (N_1) and pull-down (N_2) transistors are turned off. A downward voltage pulse with a specific duration (width) W and amplitude (height) H is generated on input in_1, and the outputs S_o and out are evaluated accordingly. For the input pulse with a given W, it does not change the output (i.e., inhibited) if the pulse is too weak (height $\leq H_l(W)$), and it will be converted into an output bit-flip if the pulse is strong enough (height $\geq H_h(W)$). However, if the pulse is weak but not too weak ($H_l(W) <$ height $< H_h(W)$), a pulse will be generated at the state-holding output. The region ($H_l(W)$, $H_h(W)$) is called *glitch sensitive window with respect to input pulse width W*, which is represented by $gsw(W)$. Let glitch sensitive window size $|gsw(W)|$ be $H_h(W) - H_l(W)$. The probability of glitch propagation at input pulse width W, $gp(W)$, can be estimated as $|gsw(W)|/Vdd$, assuming the input pulse heights are uniformly distributed.

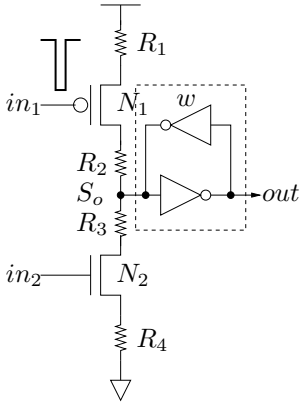

Figure 1. State holding element.

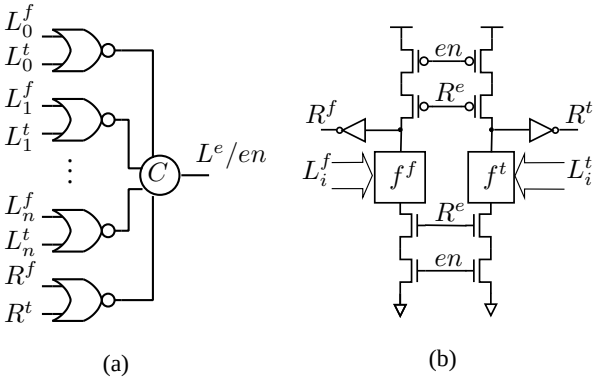

Figure 2. Precharge half buffer.

It should be clear that $gsw(W)$ is decided by input pulse width and circuit properties such as transistor sizing, supply voltage, fabrication technology, etc. In the remaining of this section, we examine the effects of all those factors respectively. The investigation of transistor sizing and supply voltage effects help us find a systematic method to reduce the probability of glitch propagation, while the study on feature size scaling allows us to sepculate the impact of future technology. We used HSPICE to simulate the state-holding element in Figure 1. To make glitch to be transient compared with circuit normal behavior, the maximum width of input voltage pulse is set to 100ns. Given a circuit, pulses with different amplitudes and durations from 0V to Vdd and from 10ps to 100ns are generated at input in_1 to derive the curves of $gsw(W)$. The step sizes of input pulse width and height are 10ps and 1mV respectively. Due to the symmetry of PMOS and NMOS transistors, the simulation results for upward voltage pulses at input in_2 are similar.

Transistor sizing. Since the feedback inverter (the inverter labeled with *'w'* in Figure 1) of output staticizer inhibits weak changes of output charge or maintains the flipped output voltage level, the $H_l(W)$ and $H_h(W)$ of $gsw(W)$ are mostly decided by the relative driving strength of the feedback inverter of staticizer to forward transistors N_1 and N_2. Thus, the impact of transistor sizing is evaluated by changing the transistor sizing ratio of the feedback inverter to N_1 and N_2 transistors in Figure 1. HSPICE simulations used TSMC 0.18um CMOS technology with 1.8V supply voltage. Figure 3(a) shows the curves of $H_l(W)$ with respect to a feedback inverter with different relative sizing. The ratio number represents the ratio of N_1 (or N_2) transistor size to NMOS (or PMOS) transistor size of the feedback inverter (a smaller number indicates stronger feedback inverter). Figure 3(b) shows the probabilities of different glitch sensitive window sizes ($|H_h(W) - H_l(W)|$) over all input pulse widths, given the transistor sizing ratio.

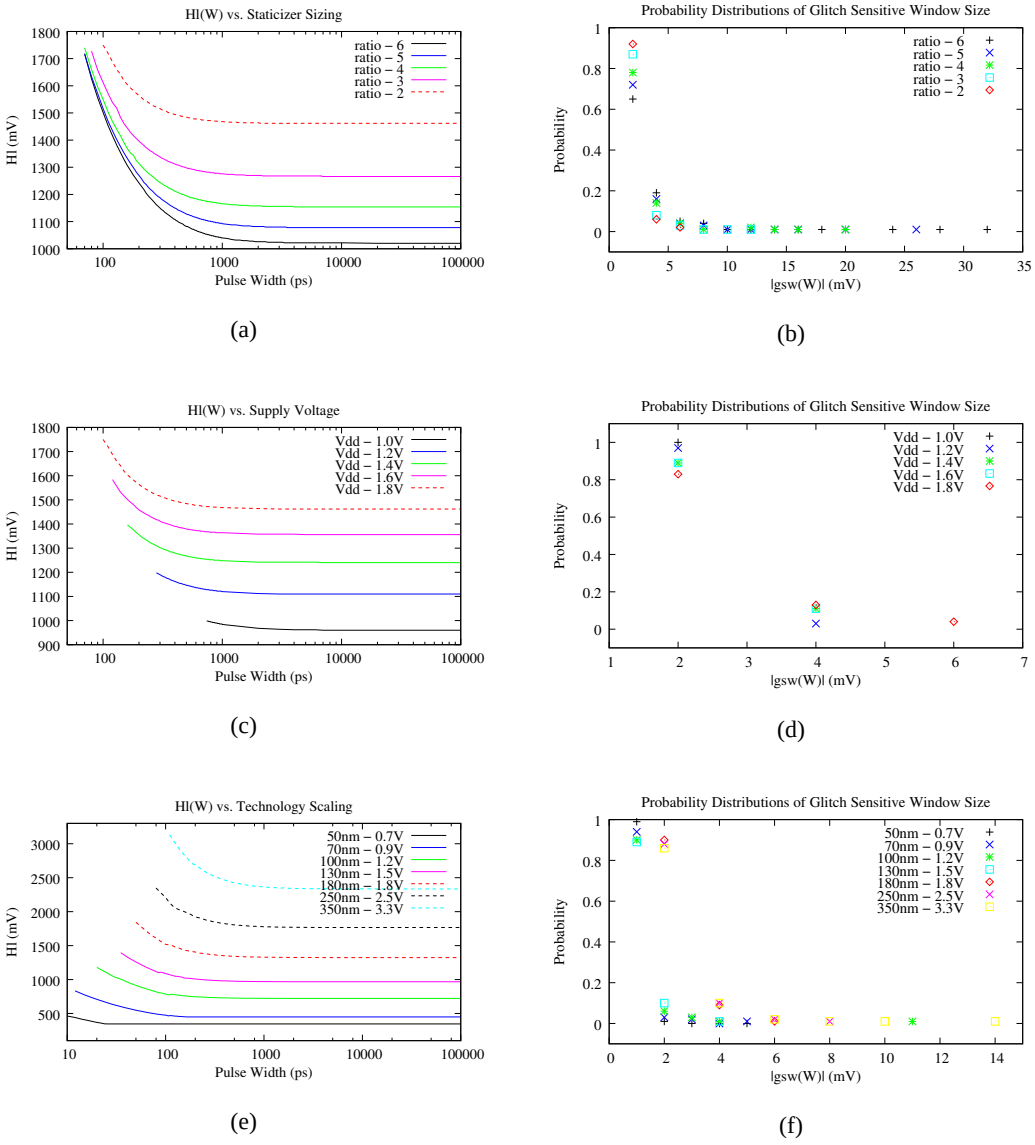

Figure 3. Glitch sensitive window vs. staticizer, Vdd and technology.

Several observations can be made from Figure 3(a) and 3(b): (1) Input pulses with too short durations cannot affect the output no matter what its amplitude is. (2) Larger pulse duration is likely to decrease H_l, as there is more time to change the output charge. (3) $gp(W)$ is small at most pulse widths for all feedback inverters, due to the small $|gsw(W)|$ (less than 10mV with at least 90% probablity) compared with the full voltage swing (1.8V). (4) Staticizer with stronger feeback inverter tends to increase H_l and decrease the average $gp(W)$ to be negligible (0.13%) by reducing the glitch sensitive window size (i.e., smaller $|gsw(W)|$ with higher probability), while slowing down the circuit. We futher exhaustively examined the shapes of the generated output pulses by setting input pulses into glitch sensitive windows, to check the possibility that those pulses propagate through the following state-holding elements. The experiments showed that all those output pulses escaped glitch sensitive windows. In other words, they are either inhibited (20.4% on average) or turned into bit-flips (79.6% on average) at the next state-holding element.

Supply voltage. With the strongest feedback inverter in Figure 3(a), we investigated the impact of supply voltage on glitch sensitive windows. HSPICE simulations used the same technolody (TSMC 0.18um) with varying supply voltages. Because the device threshold voltage is around 0.4V, the minimum supply voltage is set to be 1.0V to guarantee that both PMOS and NMOS transistors can be turned on with some noise margins. Figure 3(c) shows the curves of $H_l(W)$ with respect to different supply voltages. Figure 3(d) shows the corresponding probability distributions of glitch sensitive window size.

The following observations can be made from Figure 3(c) and 3(d): (1) Lower supply voltage tends to inbibit input glitches with longer duration no matter what its amplitude is. (2) Lower supply voltage reduces both $H_l(W)$ and $|gsw(W)|$ of glitch sensitive windows, but only slightly increases the average $gp(W)$ (e.g., 0.23% at 1.0V). We also exhaustively examined the shapes of the generated output pulses by setting input pulses into glitch sensitive windows, and the experiments showed that all those output pulses either are inhibited (31.2% on average) or become bit-flipped (68.8% on average) at the next state-holding element.

Technology scaling. We constructed a set of SPICE Level 3 technology models corresponding to the technology generations from the Semiconductor Industry Assocation Technology Roadmap [1]. Values for drawn gate length, typical supply voltage and oxide thickness are taken directly from the roadmap. The remaining parameters are obtained using a scaling methodology developed by McFarland [13] with calibration to TSMC 0.18um CMOS device parameters. The circuits simulated also used the strongest staticizer in Figure 3(a). Figure 3(e) shows the curves of $H_l(W)$ with respect to different technologies and their typical supply voltages. Figure 3(f) shows the corresonding probability distributions of glitch sensitive window size.

From Figure 3(e) and 3(f), we can draw several conclusions: (1) The glitch sensitive window exists for short input pulses at smaller feature sizes. (2) Smaller feature size coupled with lower supply voltage tends to reduce both $H_l(W)$ and $|gsw(W)|$, and $gp(W)$ may slightly increase due to lower supply voltage but is still negligible (e.g., 0.15% at 50nm technology). We exhaustively examined the shapes of the generated output pulses by setting input pulses into glitch sensitive windows, and the experiments showed that all those output pulses either are inhibited or become bit-flipped at the next state-holding element.

Summary. It is safe to draw the following conclusions. First, it is possible for a glitch to propagate through a state-holding element (to produce an output pulse) but the probability ($gp(W)$) can be neglected if we make the feedback inverter of staticizer strong. Second, $gp(W)$ tends to increase with lower supply voltage and smaller device feature size, but the increment is small with the strong feedback inverter so that $gp(W)$ is still negligible. Third, all of the output pulses escape the glitch sensitive windows, making them unlikely to propagate through the next state-holding element. Therefore, it is reasonable to model a transient fault as a momentary bit-flip (single event upset) in asynchronous circuits as long as there are state-holding elements. It should be noted that this statement still holds even if the input pulse heights are not uniformly distributed, because the third conclusion does not depend on that distribution assumption.

3 Failure Detection in Pipelined Asynchronous Circuits

In this section, we propose an efficient method of concurrent failure detection in a commonly-used pipelined QDI circuit template. By augmenting the rails to each channel and adding extra logic to that circuit, the asynchronous pipeline will deadlock in the presence of failures with respect to the fault model. Failure detection is then achieved by deadlock recognition logic. Unlike previous researches [2, 4, 14, 15, 18, 19], both permanent and transient faults can be detected by this method, while no significant timing assumption is required. We first explain that widely-used QDI circuit template, and then present the fault model based on the conclusions in Section 2. After that, we describe the fail-stop design and show how to recognize deadlock with the minimum cost in an asynchronous system.

3.1 Baseline Pipelined Asynchronous Circuits

An important pipelined QDI circuit template is half-buffer based computational block (called *Precharge Half Buffers (PCHB)*) [8], and can be used to construct most of asynchronous logic. Due to its high speed, this template has been widely used in many asynchronous designs [11]. Thus, to achieve failure detection in PCHB circuits takes an important step toward fault tolerance in general asynchronous systems.

In a PCHB circuit, an input/output variable X is usually implemented in terms of dual-rail encoded channel (X^f, X^t) with an explicit active-low acknowledge (X^e) [10]. Figure 2 shows the general template for a PCHB circuit. Validity and neutrality of the inputs and the output are checked in Figure 2(a), generating the acknowledge of all input channels as well as the enable signal to the block of Figure 2(a), which computes (evaluates) or resets (precharges) the data outputs. The muller C-element in Figure 2(a) is a state-holding element that waits for the inputs to be equal, and then changes the output to be the same as the inputs. The behavior of a PCHB circuit can be described with the Handshaking Expansion (HSE) language [10] as follows. The channel signals in a HSE statement are called *handshake signals*.

$$*[[R^e \wedge L]; R \Uparrow; L^e \downarrow; [\neg R^e]; R \Downarrow; [\neg L]; L^e \uparrow] \tag{1}$$

In this notation, $[X]$ means wait for an event X caused by the outside environment (either an assertion X, or a de-assertion $\neg X$). $Y \uparrow$ or $Y \downarrow$ represents the up or down transition initiated by the circuit on signal Y respectively. If the signal is one data rail of a channel Z with 1-of-N encoding, $Y \uparrow$ or $Y \downarrow$ can be further denoted as $Z \Uparrow$ (data valid) or $Z \Downarrow$ (data reset) respectively. A semicolon is used to indicate a sequence of events, and a comma is used to denote concurrent events. The $*$ character implies that the process is repeated indefinitely.

3.2 Fail-Stop Pipelined Asynchronous Circuits

Since there are state-holding elements in a PCHB circuit, it is reasonable to model transient faults as single upset events, according to the conclusions in Section 2. To consider both hard and soft errors in failure detection, the asynchronous circuit should halt in the presence of any failure by single stuck-at faults or single event upsets. Since the probability of multiple errors in a small region is orders of magnitude lower than that of a single error [6], we assume that each PCHB circuit can have either a single stuck-at fault or a single event upset (but not both). Although this fault model cannot represent all fault cases, fail-stop circuit with respect to this model still has the potential to achieve high reliability.

The authors in [2] and [19] proved that any single stuck-at fault in a PCHB-like circuit either deadlocks the circuit or throws an illegally dual-rail encoded output ('11'). By checking unacceptable data token on each output channel before receiving the incoming acknowledge, we can easily make the circuit also deadlock in the presence of any illegally encoded output from current process. However, a single event upset in a PCHB circuit may generate premature firings while the system does not halt, causing wrong computation and illegal event-ordering [7]. In order to make the PCHB circuit halt in the presence of any failure by a single event upset, extra rails and logic are added (shown as Figure 4). We call the augmented PCHB circuit *FS-PCHB*.

Like the baseline PCHB circuit, a FS-PCHB circuit consists of two parts: control module (shown as Figure 4(a)) and computational module (shown as Figure 4(b)). An extra rail X^v is added to each input/output channel X to check the data validity, while the acknowledge rail is duplicated as $(X^a)_1$ and $(X^a)_2$. An I-element is used to permanently block (unless system is reset) the incoming acknowledges ($(R^a)_1$, $(R^a)_2$) if any illegally encoded data output shows up. Since the I-element is close to the data outputs ((R^f), R^t) and much faster than other PCHB circuits, it is trivial to guarantee that the incoming acknowledges become blocked before they precharges the circuit. The following HSE statement describes the behavior of a FS-PCHB circuit.

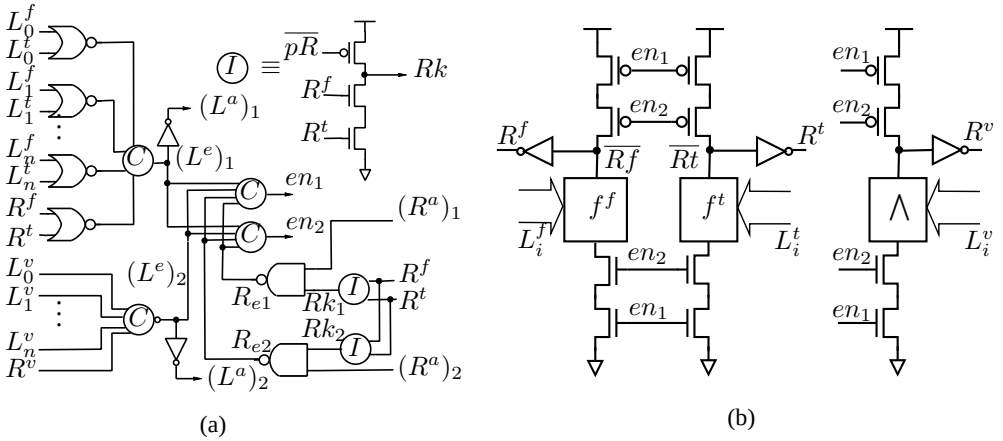

Figure 4. Fail-stop Precharge Half Buffer (FS-PCHB).

$$*[([\neg(R^a)_1 \wedge \neg(R^a)_2 \wedge L]; R\Uparrow; (L^a)_1\uparrow), ([\neg(R^a)_1 \wedge \neg(R^a)_2 \wedge L^v]; R^v\uparrow; (L^a)_2\uparrow); \\ [(R^a)_1 \wedge (R^a)_2]; (R\Downarrow, R^v\downarrow); ([\neg L]; (L^a)_1\downarrow), ([\neg L^v]; (L^a)_2\downarrow)] \quad (2)$$

Claim 1 *A FS-PCHB circuit achieves fail-stop with respect to a single stuck-at fault (SSAF).*

Proof: According to the HSE statement (2), all signals in a FS-PCHB circuit other than those of pull-down stacks in computational module and the inputs/outputs of I-elements, must enable both up and down transitions in order to complete the handshake sequence. If any of such signals get stuck, the communication between current and neighbor pipeline stages cannot move forward and the whole pipeline deadlocks.

If the computational module is in precharge state (i.e., data outputs are reset), any stuck-at fault of the pull-down stacks (f^f, f^t, or $\wedge$), is masked (i.e., no faulty handshake signal); If the computational module is in evaluation state (i.e., data outputs are valid), a single stuck-at fault of the pull-down stacks will cause at most one data output (R^f, R^t or R^v) to be wrong: (1) If R^v is wrong, the next pipeline stage will not consume the token due to the crosscheck with R^f and R^t and the pipeline deadlocks. (2) If R^f or R^t is wrong, the resulting data-rail encoding of $R^f R^t$ must be either '00', which stalls pipeline due to no valid token sent to the next pipeline stage, or '11', which blocks the incoming acknowledges ($(R^a)_1$, $(R^a)_2$) through the I-element and makes the circuit deadlock in evaluation state.

If the output of an I-element (Rk_1 or Rk_2) becomes stuck, the fault either is masked (if stuck-at-1), or deadlocks the pipeline by blocking an incoming acknowledge (if stuck-at-0); If an input of an I-element becomes stuck, the fault either is masked (if stuck-at-0), or makes the pipeline deadlock when the other input becomes high (if stuck-at-1).

Thereafter, a SSAF deadlocks a FS-PCHB circuit if any failure occurs. ∎

For a single event upset (SEU), the flipped signal may be changed back to its original logic value when the transient source disappears. Therefore, we say a circuit *fail-stop with respect to a SEU* if the circuit either deadlocks when failure occurs or stalls temporarily but resumes correctly after the transient source disappears.

Claim 2 *A FS-PCHB circuit achieves fail-stop with respect to a single event upset (SEU).*

Proof: For each input channel (X) in Figure 4(a), X^v crosschecks (X^f, X^t), and $(X^a)_1$ crosschecks $(X^a)_2$. A counterpart can be found for each of the rest signals in Figure 4(a) (e.g., (R_{e1}, R_{e2}), (en_1, en_2)). Consequently, a flipped signal in the control module can change at most one

control output ($(L^a)_{1/2}$ or $en_{1/2}$) due to the full crosschecking, and the computational module still holds current state (precharge or evaluation state).

A flipped signal in a pull-down stack of computational module causes at most one data output (R^f, R^t or R^v) to be wrong, only when the module is in evaluation state. A single event upset is always masked if it flips one enable signal (en_1 or en_2). Thus, a flipped signal in computational module can change at most one data output of R^f, R^t or R^v.

With the above analysis, it should be clear that a single event upset in a FS-PCHB circuit cannot switch precharge/evaluation state, and only causes at most one data output (R^f, R^t or R^v) and/or at most one acknowledge output ($(L^a)_1$ or $(L^a)_2$) to be faulty. However, one faulty acknowledge output $(L^a)_1$ (or $(L^a)_2$) cannot change the handshake state due to the crosschecking with its counterpart $(L^a)_2$ (or $(L^a)_1$). As long as no illegally dual-rail encoded data output ($R^f R^t = \text{`}11'$) is generated, one faulty data output will not change the handshake state either, because R^v crosschecks with (R^f, R^t). The FS-PCHB circuit only stalls temporarily when those faults occurs, but resumes correctly after the transient source disappears. If any illegally encoded data output is generated, it will be immediately captured by the local I-element, which deadlocks the FS-PCHB circuit.

Thereafter, a SEU deadlocks a FS-PCHB circuit in the presence of any failure. ∎

With Claim 1 and 2, it is safe to conclude that a FS-PCHB circuit achieves fail-stop with respect to a SSAF or SEU. Compared with the duplication-based failure detection method [4, 18], the FS-PCHB circuit does not require any significant timing assumption, no component has to be fault-free, and smaller hardware cost as well as less power overhead is expected. Since most of the extra logic in the FS-PCHB circuit works in parallel, small performance overhead is also anticipated.

3.3 Deadlock Detection

System deadlock is recognized by deadlock monitor, which checks handshake activity. A deadlock monitor works in the following way. At any time, if a transition occurs on the data channel, a timer is started (implemented as a delay line [9]). The deadlock detector waits for the next valid protocol state to occur. If it does not occur for a large amount of time (in terms of microseconds or milliseconds), it assumes that the circuit has deadlocked. Note that there are some states where the circuit can wait for its environment. In these states, a timer is not set.

It is not necessary to apply a deadlock detector to every channel between different hardware processes. Say channel X *dominates* channel Y when any switching activity occurs on Y only if X is switching. Channel X *reciprocally dominates* channel Y if X both dominates and is dominated by Y. An *equivalent dominator set* is a set of channels and they reciprocally dominate each other. A *top equivalent dominator set* is an equivalent dominator set and each channel in this set is not dominated by any channel other than the channels in this set. It should be clear that an asynchronous system must deadlock if and only if a channel of one top equivalent dominator set deadlocks. Therefore, system deadlock can be detected if a deadlock detector is assigned to one channel of each top equivalent dominator set. For an asynchronous system without independent modules, there is only one top equivalent dominator set and thus one deadlock detector is required.

4 Experimental Evaluation

We evaluated the FS-PCHB circuit in terms of hardware cost, performance and power consumption. Power overhead should be almost the same as the hardware overhead, because most of the extra transistors are switching. The amount of hardware is estimated in terms of transistor count.

Since full adder is the most common datapath operator, we applied FS-PCHB design to an asynchronous full adder of different sizes, and evaluated the corresponding overheads. Strong feedback inverters are applied to these FS-PCHB circuits to reduce glitch sensitive window size. We used

HSPICE to simulate the circuits with TSMC 0.18um technology at 1.8V supply voltage to get the throughput and power consumption numbers. Table 1 shows the normalized results.

Experimental data in the table shows that the fail-stop augmentation logic increases hardware cost by a constant of 92%, compared with the baseline adder. Throughput is reduced by around 30% on average and power consumption is increased by around 80% on average. Since our fail-stop design does not duplicate everything of baseline PCHB circuit, the incurred hardware overhead is usually less than 100%. Compared with the previous failure detection methods in asynchronous circuits [4, 18] which can also detect both hard and soft errors but requires full duplication and conservative time window, our design incurs small performance penalty, reasonable hardware/power overhead and good scalability for different circuit sizes.

Table 1. Normalized metrics of fail-stop asynchronous adder

Adder Size	HW Cost	Throughput	Power
8-bit	1.92	0.61	1.79
16-bit	1.92	0.67	1.81
32-bit	1.92	0.70	1.80
64-bit	1.92	0.69	1.81

Because less transistors are required for data validity (R^v) generation than those for computational data outputs (R^f and R^t), the hardware overhead of FS-PCHB circuits decreases with more complicated computational logic (i.e., more transistors in pull-down stacks of Figure 2(b)). Consequently, the hardware overhead of our fail-stop design depends on the logic function implemented, and those numbers reported by full adder are pessimistic for many computational modules, due to the simplicity of add function. As an example, the hardware overhead of a 4-input (2 data-inputs + 2 control-inputs) FS-PCHB function bock is only 71% with throughput reduction of 32% and power increase of 65%, while that overhead of a 6-input (4 data-inputs + 2 control-inputs) FS-PCHB function block is further reduced to 54% with throughput reduction of 35% and power increase of 50%. Thus, we can merge the computations of neighbor pipeline stages, resulting in coarser-grained pipeline and more complicated computational logic in one FS-PCHB circuit, which significantly reduces the hardware overhead as well as extra power consumption.

5 Related Work

Although there is a lot of research conducted on concurrent failure detection in synchronous logic [6], only a little work has been done for asynchronous circuits. Rennels et.al. [15] designed a checker circuit to compare signal pairs from self-timed Differential Cascode Voltage Switch Logic to detect errors with high probability. Piestrak et.al. [14] developed a self-testing code-disjoint completion checker for QDI circuits to report error by internal single stuck-at faults. David et.al. [2] and Yang [19] investigated stuck-at fault detection in dynamic asynchronous circuits, and proposed the self-checking clockless designs with respect to hard errors. Hyde et.al. [4] presented concurrent error detection in a micropipeline-based asynchronous RISC processor by applying both duplication-based method and Dong's code, while with significant gate delay assumptions. Verdel et.al. [18] used duplications to achieve concurrent error detection in asynchronous circuits with synchronized comparison and monitor functions, which incurs both conservative time window and large hardware overhead. LaFrieda et.al. [7] investigated some fault detection and isolation techniques to improve the reliability of QDI circuits. Jang et.al. [5] proposed several SEU-tolerant QDI circuit designs which cause the circuits three times larger and twice slower. However, none of the aforementioned methods can be applied to QDI circuits to guarantee fail-stop with respect to both permanent and transient faults, while at reasonable hardware cost and with small performance overhead.

6 Conclusion

In this paper, we developed an efficient concurrent failure detection method for QDI circuits, providing the basis for the construction of reconfigurable fault-tolerant QDI systems. Compared with previous researches, this method can achieve fail-stop with respect to both hard and soft errors in the asynchronous system, no significant timing assumption is required and no hardware has to be fault-free. The experimental evaluations showed that this method incurs both reasonable hardware cost and low performance overhead. Moreover, we investigated the transient pulse propagation in asynchronous logic, and validated the transient fault modeling in clockless circuits through transistor-level simulations. In addition, we presented a systematic way of deadlock detection with the minimum cost in an asynchronous system.

References

[1] *International Technology Roadmap for Semiconductors*. Semiconductor Industry Association, 2004.

[2] I. David, R. Ginosar, and M. Yoeli. Self-timed is self-checking. *Journal of Electronic Testing: Theory and Applications*, 6(2), 1995.

[3] P. E. Dodd and L. W. Massengill. Basic mechanisms and modeling of single-event upset in digital microelectronics. *IEEE Transactions on Nuclear Science*, 50(3), 2003.

[4] P. D. Hyde and G. Russell. A comparative study of the design of synchronous and asynchronous self-checking RISC processors. In *Proceedings of IEEE International On-Line Testing Symposium*, 2004.

[5] W. Jang and A. J. Martin. SEU-tolerant QDI circuits. In *Proceedings of International Symposium on Asynchronous Circuits and Systems*, 2005.

[6] B. W. Johnson. *Design and Analysis of Fault Tolerant Digital Systems*. Addison Wesley, 1989.

[7] C. LaFrieda and R. Manohar. Robust fault detection and tolerance in quasi delay-insensitive circuits. In *Proceedings of International Conference on Dependable Systems and Networks*, 2004.

[8] A. Lines. Pipelined asynchronous circuits. Master's thesis, 1995.

[9] N. R. Mahapatra, A. Tareen, and S.V. Garimella. Comparison and analysis of delay elements. In *Proceedings of the 45th Midwest Symposium on Circuits and Systems*, 2002.

[10] A. J. Martin. Synthesis of asynchronous VLSI circuits. Technical Report CS-TR-93-28, 1993.

[11] A. J. Martin, M. Nyström, and C. G. Wong. Three generations of asynchronous microprocessors. *IEEE Design and Test of Computers*, 20(6), 2003.

[12] E. J. McCluskey and C. W. Tseng. Stuck-fault tests vs. actual defects. In *Proceedings of International Test Conference*, 2000.

[13] G. McFarland. *CMOS Technology Scaling and Its Impact on Cache Delay*. PhD thesis, Stanford University, August 1997.

[14] S. J. Piestrak and T. Nanya. Towards totally self-checking delay-insensitive systems. In *Proceedings of International Symposium on Fault-Tolerant Computing*, 1995.

[15] D. A. Rennels and H. Kim. Concurrent error detection in self-timed VLSI. In *Proceedings of International Symposium on Fault-Tolerant Computing*, 1994.

[16] G. L. Ries, G. S. Choi, and R. K. Iyer. Device-level transient fault modeling. In *Proceedings of International Symposium on Fault-Tolerant Computing*, 1994.

[17] I. E. Sutherland and J. Ebergen. Computers without clocks. *Scientific American*, 2002.

[18] T. Verdel and Y. Makris. Duplication-based concurrent error detection in asynchronous circuits: Shortcomings and remedies. In *Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2002.

[19] J. L. Yang, C. S. Choy, C. F. Chan, and K. P. Pun. Design for self-checking and self-timed datapath. In *Proceedings of IEEE VLSI Test Symposium*, 2003.