Design of Asynchronous Circuits for High Soft Error Tolerance in Deep Submicron CMOS Circuits

Weidong Kuang, *Member IEEE*, Peiyi Zhao, *Member IEEE*, J.S. Yuan, *Senior Member, IEEE*, and R. F. DeMara, *Senior Member, IEEE*

Abstract – As the devices are scaling down, the combinational logic will become susceptible to soft errors. The conventional soft error tolerant methods for soft errors on combinational logic do not provide enough high soft error tolerant capability with reasonably small performance penalty. This paper investigates the feasibility of designing quasi-delay insensitive (QDI) asynchronous circuits for high soft error tolerance. We analyze the behavior of Null Convention Logic circuits in the presence of particle strikes, and propose an asynchronous pipeline for soft-error correction and a novel technique to improve the robustness of threshold gates, which are basic components in NCL, against particle strikes by using Schmitt trigger circuit and resizing the feedback transistor. Experimental results show that the proposed threshold gates do not generate soft errors under the strike of a particle within a certain energy range if a proper transistor size is applied. The penalties, such as delay and power consumption, are also presented.

Index Terms – Soft error, asynchronous circuit, Null Convention Logic

I.INRODUCTION

Semiconductor devices are becoming susceptible to particle strikes as they shrink to nano-scale. Soft errors to be addressed in this paper are radiation-induced transient errors caused by neutrons from cosmic rays or alpha particles from packaging materials [1]. Specifically, when these particles with sufficient energy hit the silicon substrate of a Complementary Metal Oxide Semiconductor (CMOS) chip, a large number of electron-hole pairs are generated and an undesired short-duration current may be formed to change the output of a logic gate. A soft error occurs when this corrupt output is captured by a memory cell, register, latch, or flip-flop.

Soft error protection is very important for enterprise computing and communication applications since the system-level soft error rate (SER) has been rising with technology scaling and increasing system complexity [2] [3]. Several designs today exploit extensive error correction codes (ECC) mainly for on-chip SRAMs and register files [4]. However, soft errors in combinational logic circuits are significant contributors to the system-level SER [5]. The lack of efficient soft error protection for combinational logic poses a major challenge to robust computing and networking system designs.

Quasi delay insensitive (QDI) asynchronous circuits have a strong potential for soft error protection. First, the dual-rail encoding scheme theoretically provides the QDI circuits with an ability to detect soft errors. For instance, a soft error can be identified when code "11" is detected. Secondly, asynchronous handshaking communications allow the QDI circuits to correct soft errors by re-computing. Upon the detection of soft errors, the QDI circuit has a chance to stop the corrupted DATA from propagating, and to re-compute the result through modified handshaking circuitry. Besides soft errors, particle strikes may cause other malfunctions on a chip, e.g., charges induced by particle strikes may slowly accumulate in the substrate of a chip. Those long-term dose effects usually cause parameter shifts, in particular threshold voltages, which affect the timing of the system. QDI circuits are very robust to timing variation. Monnet et.al proposed a metric, *sensitive time*, to evaluate the sensitivity of asynchronous circuits to transient faults [6] [7], and developed several harderning techniques for QDI circuits with full duplication of circuit parts and synchronization of replicated results through C-elements [8]. By using doubled-up production rules, Jang et.al [9] proposed several SEU-tolerant QDI circuit designs without any requirement of significant timing assumptions. However this approach usually results in large hardware cost and significant performance overhead. Peng et.al [10] developed an efficient concurrent failure detection method for pipelined asynchronous circuits so that the asynchronous circuits halt in the presence of failure by single stuck-at faults or single event upsets, and [11] then they proposed a framework for constructing a self-healing asynchronous array based on reconfiguration logic and deadlock detection. With several assumptions, Gardiner et.al [12] proposed a new latch to stop the propagation of faults through asynchronous pipeline. The penalty associated with this method is the long latency of the latch, which makes this latch less suitable for high speed applications.

This paper presents a built-in soft error correction (BISEC) technique by exploiting the handshaking protocol and dual-rail encoding in QDI circuits. As shown later in this paper, all soft errors at the output of computational blocks can be fully detected and corrected with very small area and speed overhead. Although the BISEC technique is developed through Null Convention Logic (NCL) design paradigm [13], it can be applied in other QDI design paradigms. It is assumed that the NCL registers and completion detection circuitry are soft error free since they are relatively small compared to computational blocks and therefore less likely to be struck by particles. The discussion on the effects of soft error in register and completion detection circuitry is beyond the scope of this paper. The major contributions of this paper include:

1) Understanding soft error generation and propagation in QDI circuits.
2) Soft error blocking technique that utilizes an inserted self-feedback register to block most of soft errors in computational blocks.

3) Soft error correction technique that detects the rest soft errors (not blocked in 2)), and performs re-computation without affecting the pipeline handshake timing, under reasonable delay assumptions.

4) Optimization of threshold gates for soft error tolerance.

The rest of paper is organized as follows. Section II presents an overview of asynchronous logic, focusing on NCL. Section III describes the generation and propagation of soft error in NCL. In Section IV, we describe the principle of the built-in soft error correction. The simulation results are presented in Section V. Section VI concludes the paper.

II. OVERVIEW OF NULL CONVENTION LOGIC

A. Architecture and Dual-rail Encoding

Asynchronous circuits can be grouped into two main categories: bundled data and delay insensitive models [14]. The bundled data model uses normal Boolean levels to encode data information, but requires matching delay elements for handshaking protocols. This leads to extensive timing efforts to ensure correct circuit operation. On the other hand, delay insensitive model uses dual-rail or quad-rail logic to encode data information. The delay insensitive circuits only assume delays in both elements and interconnects to be unbounded, and wire forks within a component to be isochronic [15]. Completion detection of the output signal allows for handshaking to control input wavefronts. Delay insensitive design paradigms therefore require very little, if any, timing effort to ensure correct operation.

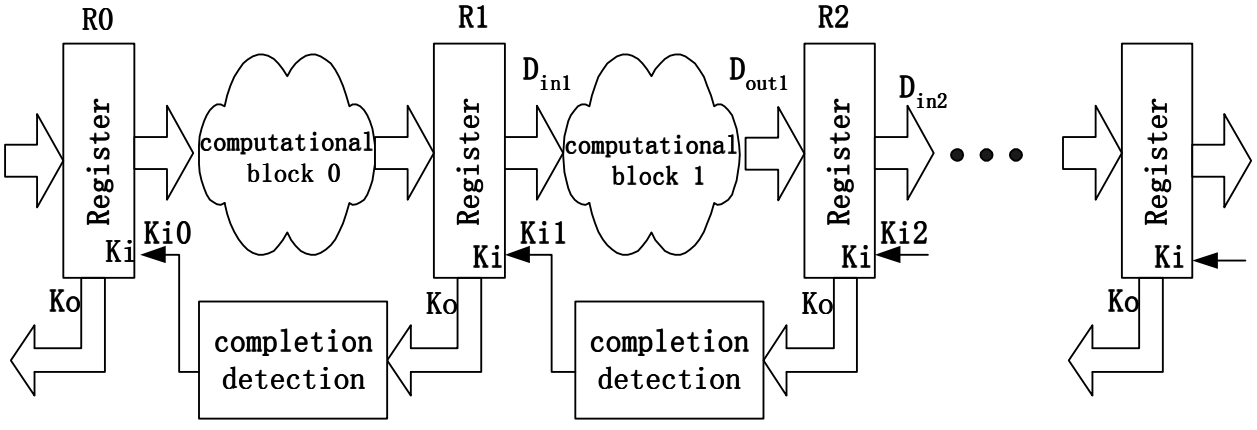


Fig. 1. NCL pipeline architecture

NCL is a quasi delay insensitive asynchronous paradigm since wires connecting components have to adhere to the isochronic fork assumption. A typical NCL pipeline architecture consists of computational blocks, registers and completion

detection circuits, as shown in Fig. 1. Two adjacent register stages interact through their request and acknowledge signals, K_i and K_o, respectively, to prevent the current DATA wavefront from overwriting the previous DATA wavefront, by ensuring that the two DATA wavefronts are always separated by a NULL wavefront. When a register (e.g. R2) detects a complete set of DATA at the output of computational block, it will inform the previous register (e.g. R1) that the current computation is done and a NULL wavefront is allowed to come into the computational block, by setting Ki_1 low.

NCL circuits utilize dual-rail or quad-rail or quad-rail encoding technique to achieve delay insensitivity [13]. A dual-rail signal D is encoded by two wires, D^0 and D^1, as shown in Table 1. The signal D may assume any value from the set {DATA0, DATA1, NULL}. The DATA0 state (D^0=1, D^1=0) corresponds to a Boolean logic 0, the DATA1 state (D^0=0, D^1=1) corresponds to a Boolean logic 1, and the NULL state (D^0=0, D^1=0) corresponds to the empty state meaning that the value of D is not yet available. The two rails are mutually exclusive, such that both rails can never be asserted simultaneously; otherwise the state is defined as an illegal state. One of the erroneous consequences of the particle radiation is the occurrence of this illegal state, as shown later. It is this illegal state that provides a fundamental for soft error detection in NCL designs.

TABLE 1. DUAL-RAIL ENCODING

Dual-rail encoding (D^1, D^0)	**Logic value**
(0,0)	**NULL**
(0,1)	**DATA0**
(1,0)	**DATA1**
(1,1)	**Invalid**

B. Threshold Gates with Hysteresis

NCL circuits are comprised of a family of threshold gates with hysteresis. The primary type of threshold gate is TH*mn* gate where *n* is the number of inputs, *m* is the threshold, and $1 \le m \le n$. A THmn gate will set its output high when any *m* inputs have gone high and it will reset its output low only when all *n* inputs are low. A more general type of threshold gate with hysteresis is referred to as a weighted threshold gate, denoted as TH*mn*W$w_1w_2...w_R$, where *n* is the number of inputs, *m* is the threshold, $w_1, w_2, \ldots w_R$ ($1 < w_i \le m$, $1 \le R < n$) are the integer weights of *input 1*, *input 2*, … *input R*, respectively. For example, TH*34* has 4 inputs (A, B, C, D) and a threshold of 3, as shown in Fig. 2 (a). When any three inputs go high, its output will be asserted to high. Only when all inputs are low, the output will be reset to low. For all other input patterns,

the output will remain unchanged. A weighted gate TH*34*W*22* has the same number of inputs (4) and threshold (3) as TH34 gate, but there is a weight 2 applied to each of the first two inputs (A and B), as shown in Fig. 2 (b). For the gate TH*34*W*22,* the output is asserted only when either input A is high along with any other input, or input B is high along with any other input. The output is deasserted only when all inputs are low. NCL threshold gates may also include a reset input to initialize the output. Either a *d* or an *n* is attached at the end of the gate name to designate these gates, such as TH*22n* shown in Fig. 2 (c). *d* denotes the gate as being reset to high while *n* to low. These resettable gates are used in the design of registers. The principle of transistor-level threshold gate design can be found in [16]. A bubble attached at the output denotes an inverter connected at the output, as shown in Fig. 2 (d). As an example, the schematic of TH*23* is shown in Fig. 3.

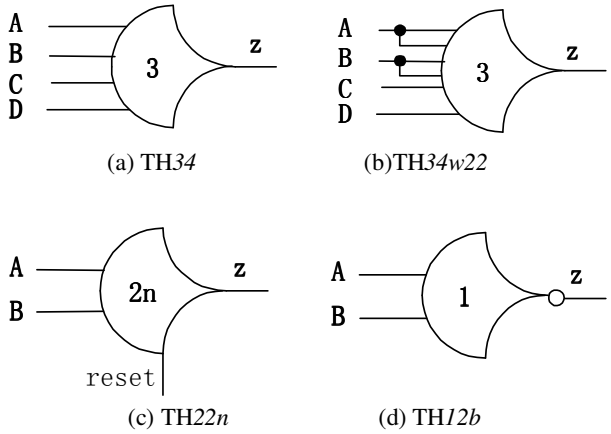


Fig. 2. Symbol examples of threshold gates

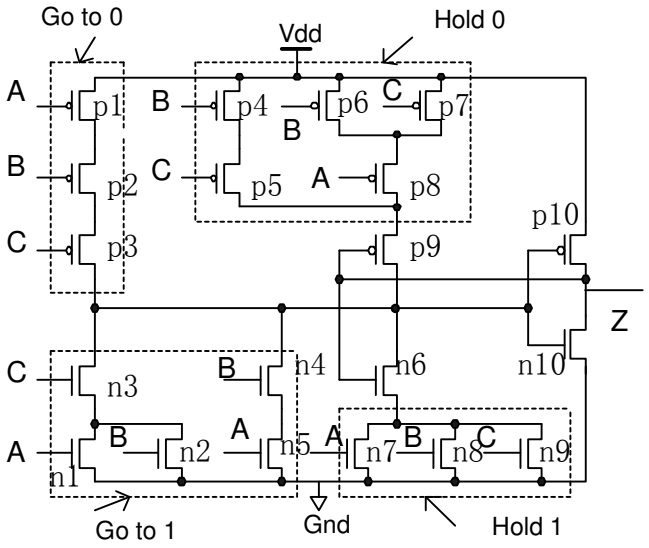


Fig. 3. Schematic of TH*23*

C. Computational Block

It is noticed that any threshold gate with hysteresis, except the TH*1n* gate that is equivalent to an *n*-input OR gate, is a sequential component. Therefore, strictly and generally speaking, there is no combinational logic block in NCL circuits. We refer the NCL counterpart of combinational block in traditional synchronous circuits as to *computational block*. The dual-rail computational block in Fig. 1 consists of various threshold gates described above. The following behavior constraints on the computational block must be satisfied for quasi delay insensitivity: 1) its outputs may not transition from all NULL to a complete set of DATA until the input values are completely DATA; and 2) its outputs may not transition from a complete set of DATA to all NULL values until the input values are completed NULL. These constraints are equivalent to the "weak conditions" [17], illustrated in Fig. 4. The orderings labeled in Fig. 4 are explained hereafter.

(1) Some inputs become DATA, and then some, not all, outputs become DATA.

(2) All inputs become DATA, and then all outputs become DATA.

(3) All outputs become DATA, and then some, not all, inputs becomes NULL.

(4) Some inputs become NULL, and then some, not all, outputs become NULL.

(5) All inputs become NULL, and then all outputs become NULL.

(6) All outputs become NULL, and then some, not all, inputs become DATA. And then repeat (1) through (6).

For example, a dual-rail full adder can be implemented in Fig. 5 with three dual-rail input signals (c_i^1, c_i^0), (x^1, x^0) and (y^1, y^0), and two dual-rail outputs (c_o^1, c_o^0) and (s^1, s^0). The schematic of TH*34W2* used in the full adder is illustrated in Fig. 6. However, one can not design a dual-rail circuit with only carry output (c_o^1, c_o^0) by simply deleting two TH*34W2* gates in the full adder, because the resultant design violates the weak conditions (1) and (4). A systematic method for the synthesis of computational blocks can be found in [18].

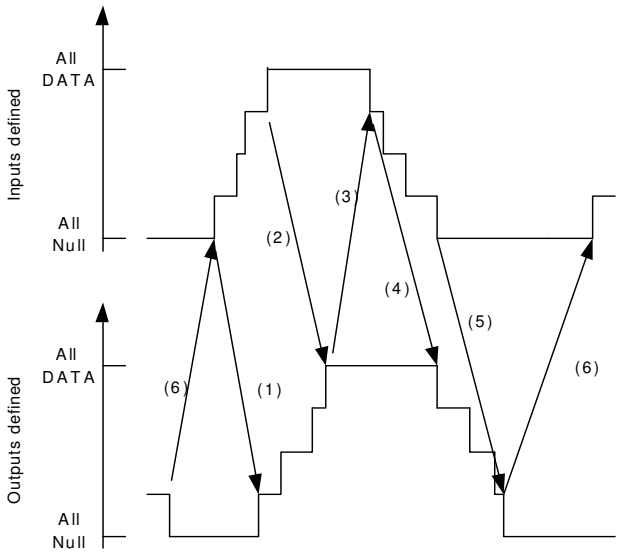


Fig. 4. Weak conditions for NCL

Due to the hysteresis of threshold gates, the signal transition in computation blocks possesses a monotonic property, which does not exist in traditional Boolean combinational logic. Specifically, during the computation, i. e. transition from NULL to complete DATA, the number of asserted gate-level nodes monotonically increases. On the other hand, during returning to all NULL from complete DATA, the number of asserted gate-level nodes monotonically decreases to zero. This monotonicity will be exploited to analyze the generation and propagation of soft errors in NCL circuits.

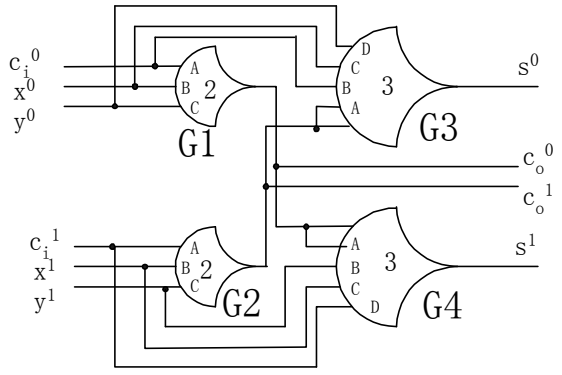


Fig. 5. An optimized NCL fulladder

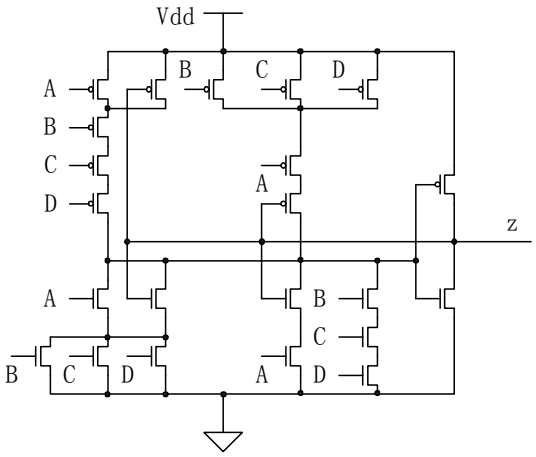


Fig. 6. Schematic of TH*34W2* (input A with weight 2)

D. Register and Completion Detection

Registers and completion detection circuitry are required to coordinate the adjacent computational blocks to ensure the correct data communications. A single-bit dual-rail register consists of two TH*22n* gates and one NOR gate, depicted in Fig. 7. The TH*22n* gates pass a DATA value from input to output only when K_i is *request for data* (rfd) (i.e. high) and likewise pass NULL only when K_i is *request for null* (rfn) (i.e. low). The request signal K_i comes from the output of the completion detection circuit of the following stage. The NOR gate generates K_o, which is *rfn* when the register output is DATA and *rfd* when the register output is NULL. The register shown is reset to NULL for initialization since all TH*22n* gates can be initialized to low. However, register could be instead reset to a DATA value by replacing exactly one of the TH*22n* gates with a TH*22d* gate.

An N-bit register is comprised of N single-bit dual-rail register in parallel. These single-bit registers share a request signal K_i and a reset signal, and generate N completion signals, one for each bit. The completion detection circuitry, shown in Fig. 8, uses these N completion signals to detect complete DATA and NULL sets at the output of every register stage and generate a total acknowledge signal K_o to request the next NULL and DATA set, respectively. This K_o is connected to K_i of the previous register stage.

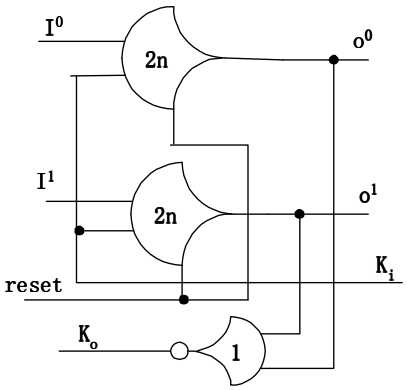


Fig. 7. Single-bit dual-rail register

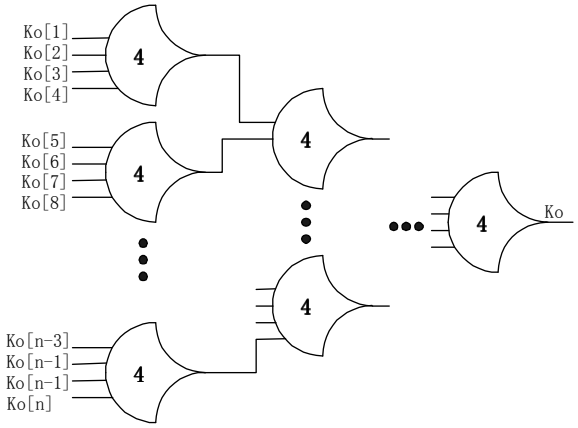


Fig. 8. N-bit completion detection circuitry

III. SOFT ERROR IN NULL CONVENTION LOGIC

A. Modeling Soft Error at Device Level

Since the NCL circuits are implemented in standard CMOS technology, modeling soft error for an individual transistor in NCL circuits should be the same as what has been done for general CMOS technology [19]. Fig. 9 shows the mechanism of soft errors in a Metal Oxide Semiconductor Field Effect Transistor (MOSFET). Electron-hole pairs with a very high carrier concentration are generated as the particle loses energy in silicon when a particle hits the drain of the MOSFET, and the resulting charges can be rapidly collected by the electric field to create a large transient current at that node. The transient current can be modeled as

$$I(t) = \frac{2Q}{T\sqrt{\pi}}\sqrt{\frac{t}{T}} \cdot \exp(-\frac{t}{T}) \tag{1}$$

where Q is the amount of collected charge, and T is a process technology-dependent time constant. The detailed discussion about this model and related parameters can be found in [19]. For sake of simplicity, we use a trapezoid current model for our sufficient circuit behavioral simulation.

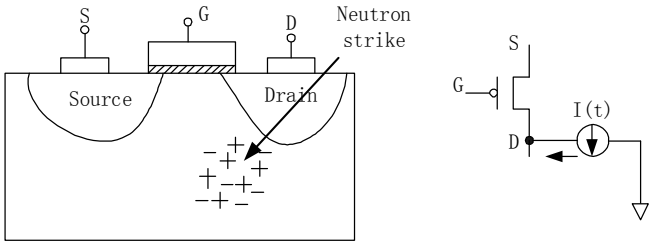


Fig.9. Mechanism of soft error in MOSFET and equivalent circuit

Whether the current is injected into or drawn from the node depends on the type of victim drain. For example, a current is injected into the node if a particle hit occurs at a p-type drain, therefore momentarily increasing the node voltage. If the logic value of the node is 0 and the current is injected to the node, a transient positive glitch (0-1-0) may occur at the node. Similarly, a transient negative glitch (1-0-1) may be generated if an n-type drain is hit.

These transient glitches are either killed by three masks (logic mask, electrical mask, latching window mask) during the propagation [20], or transformed into static errors when these transient glitches are captured by feedback logic circuits, such as threshold gates in NCL circuits, D flip-flops and memory elements.

B. Soft Errors at the Output of Threshold Gates

It is well known that the basic gates in Boolean logic include NOT, NAND, NOR gates. However, threshold gates with hysteresis are the basic gates in NCL circuits. Each threshold gate is a storage element due to the hysteresis behavior. Therefore, it is essential to investigate the soft error generation at the output of an individual threshold gate.

Theoretically, there are four types of soft errors that may be generated at the threshold gate output. These four types include positive glitch (0-1-0), negative glitch (1-0-1), positive static error (0-1), and negative static error (1-0). The type of soft error depends on the input pattern, output state, and the location of the particle strike. For example, let us examine TH*23* in Fig. 3. When all inputs are low and the drain of NMOS transistor *n6* (or *n3* or *n4*) is hit, a positive glitch may be generated, as shown in Fig. 10(a). If the input ABC=001 and the output is zero, the same strike may result in a positive static error, as shown in Fig. 10 (b). If the input ABC=011 and the output is high, a particle strike on the drain of transistor

p3 or *p9* may generate a negative glitch at the output, as shown in Fig. 10 (c). If the input ABC=001 and the output is high, a particle strike on the transistor *p3* or *p9* may generate a negative static error at the output, as shown in Fig. 10 (d).

Fortunately, it will be shown that, among the four types of soft errors, only two of them, named positive glitch and positive static error, potentially jeopardize the functionality of the circuits.

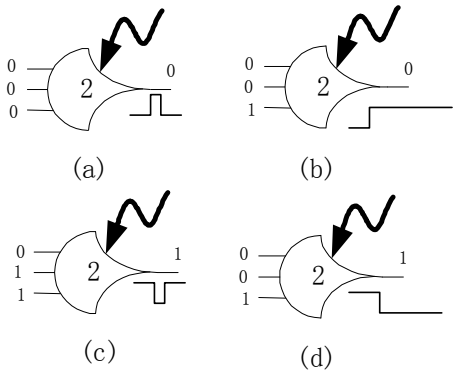


Fig. 10. Soft error generations in threshold gates

C. Soft Errors at the Output of NCL Computational Block

In this section, we will investigate the NCL computational block as a whole in terms of soft error. Three questions will be answered: 1) how do soft errors propagate in computational block? 2) What kind of soft errors at the output of computational block really lead to malfunctions? 3) How sensitive are a specific circuit topology to a random particle strike?

In order to understand the soft error propagation, it is useful to highlight the following characteristics of computational block: 1) No feedback connection at gate-level. Signals flow only forward in computational block; and 2) during a complete computation period, a computational block sequentially experiences four states: complete NULL, transition from NULL to DATA, complete DATA, and transition from DATA to NULL. It is assumed that no two particle strikes occur simultaneously.

It is difficult for glitch soft errors to propagate from a victim gate to the primary output. Due to the hysteresis, a glitch soft error (0-1-0 or 1-0-1) will be either killed or transformed into a static soft error (0-1 or 1-0) by the following gate, as shown in Fig. 11 (a)-(b) and (c)-(d) respectively. The only situation for glitch occurrence at the computation output is that the victim gate delivers a primary output and its all inputs equal to zero, or that the propagation path consists of threshold gates with thresholds equal to one and all inputs equal to zero. These two situations can be ignored since they seldom happen. Furthermore, the generated glitch soft errors can be easily suppressed under a tolerant noise level by introducing a Schmitt trigger at the output stage of threshold gates [21] [22]. The propagation of static soft errors depends on the input-

output states of the gates along the path, as shown in Fig. 11 (e)-(f). Therefore, only static soft errors are considered to occur at the output of computational blocks. Actually, among two possible types of static soft errors at the output of the computational block, only positive static soft error (0-1) may cause error DATA in the NCL pipeline [23].

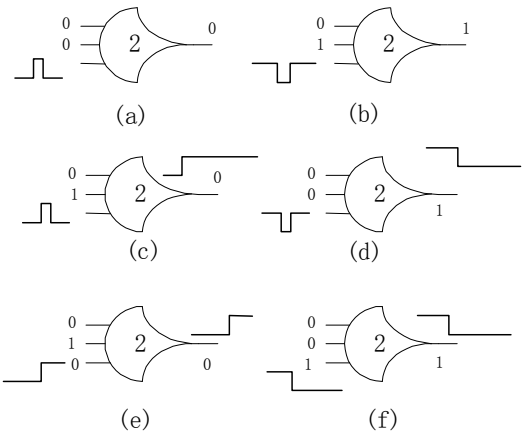


Fig. 11. Soft error propagation through threshold gates

When the computational block is in the complete NULL state, i.e. all nodes at gate-level are low, no static soft error happens at the output.

Fig. 12 is used to illustrate possible soft error effects on the input of the next stage. Due to the particle strike on the computational block, a soft error may be generated at (D^1, D^0), and eventually affect the output of the register depending on K_i. The dotted lines in Fig.12 denote soft error signals.

When the block is in the transition state from NULL to DATA, i.e. the number of nodes with high signals increases monotonically, a particle strike can only generate positive static soft error at the output of computational block because the gate states for negative static soft errors, shown in Fig. 10(d), Fig. 11(d) (f), never occur due to the monotonicity. For the same reason, only positive static errors could occur at the output when the computational block is in a complete DATA state. If the positive static soft error occurs at the rail whose error-free signal is high, but before the rising edge of the error-free signal, as shown in Fig. 12(b), this soft error may lead to an earlier completion of the current computation, and therefore a premature firing without affecting the circuit logic function. If the positive static soft error occurs at the rail whose error-free signal low, it will result in an invalid dual-rail signal (1, 1) at the output of computational block, as shown in Fig. 12 (c)-(f). This invalid dual-rail signal will lead to one of three possible consequences on the input DATA to the next stage, depending on the timing relationship between the invalid dual-rail signal and K_i connected to the output register:

1) No effect, as shown in Fig. 12(c). If the static soft error appears only when the K_i is low to allow NULL wavefront to pass, the positive static soft error will be logically blocked by the register.

2) A wrong DATA value is delivered. For example, as shown in Fig. 12(d), the dual-rail DATA is expected to be DATA0 without soft error. However, A DATA1 will be delivered to the next stage.

3) An invalid dual-rail code (1, 1) is delivered to the next stage if the invalid code (1, 1) appears when K_i is high, as shown in Fig. 12 (e) and (f).

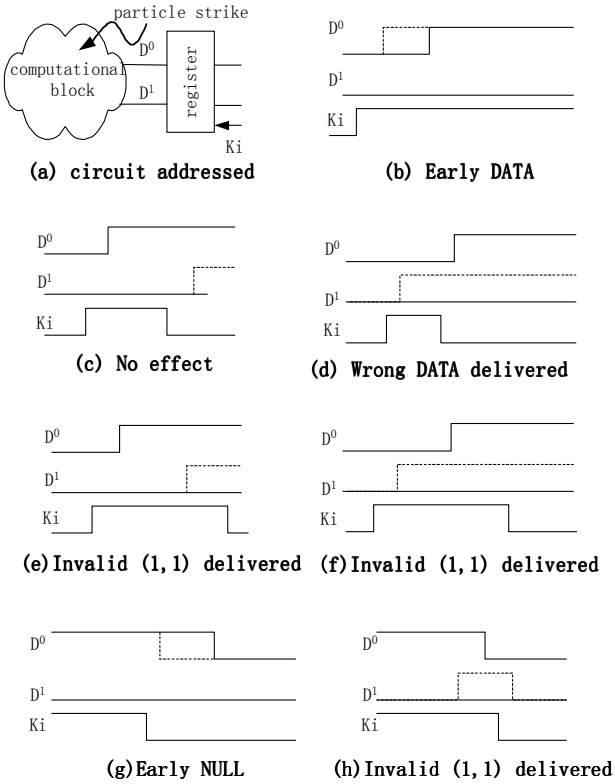


Fig. 12. Soft error effects on the input DATA to the next stage

When the computational block is in the transition state from DATA to NULL, a negative static soft error may only result in faster transition from DATA to NULL, as shown in Fig. 12(g), i.e. no error is delivered to the next stage. A positive static soft error may result in an invalid dual-rail code (1, 1) only when K_i is high during this transition, as shown in Fig. 12(h). This invalid code (1, 1) can be avoided by inserting a self-feedback register to make sure that K_i is low during this transition, as shown in Section IV.

Based on the above analysis, only positive glitch soft errors (if big enough), and positive static soft errors may jeopardize the circuit function. The following question is: how likely do these soft errors occur in a specific computational block? To answer this question, it is assumed that: 1) each transistor is hit by a particle with equal probability; 2) the

particle has enough energy to induce a soft error; 3) the dual-rail signal has equal probabilities 0.5 and 0.5, for "DATA0" and "DATA1" respectively; and 4) the computational block is in a complete DATA state, i.e. both inputs and outputs are complete DATA. This state is the worst case because it is much less likely for soft errors to occur in other three states.

TABLE 2. SOFT ERROR SENSITIVITY OF THE DUAL-RAIL FULL ADDER

Inputs (Ci^1,Ci^0)	(x^1,x^0)	(y^1,y^0)	*Sensitivity of each gate*				*Sub-total*
			G1	G2	G3	G4	Full adder
(0,1)	(0,1)	(0,1)	0	4*	0	8	12
(0,1)	(0,1)	(1,0)	0	6	7	0	13
(0,1)	(1,0)	(0,1)	0	5	6	0	11
(0,1)	(1,0)	(1,0)	4	0	0	7	11
(1,0)	(0,1)	(0,1)	0	4	5	0	9
(1,0)	(0,1)	(1,0)	5	0	0	6	11
(1,0)	(1,0)	(0,1)	6	0	0	5	11
(1,0)	(1,0)	(1,0)	4*	0	8	0	12
Average sensitivity of the full adder							11.25

* Only positive glitch soft errors are generated in these two situations, only positive static soft errors are generated in all other situations.

Let's consider an individual threshold gate first. If a positive soft error occurs at the output of the gate when a certain transistor is hit by a particle, the victim transistor is called a *sensitive transistor*. The soft error sensitivity of the threshold gate is defined as the number of sensitive transistors in the gate associated with a specific state vector $\{x_1, x_2 \ldots x_3; y\}$, where x_i is an input ($i=1, 2 \ldots n$) and y is the output. For example, with input ABC=001 and output y=0, the sensitive transistors in TH23 gate, shown in Fig. 3, include $n_1, n_2, n_3, n_4, n_6,$ *and* p_{10}. So the soft error sensitivity of TH23 with state {0, 0, 1; 0} is six. It is noticed that the soft error sensitivity for output y=1 is zero. Now we consider a computational block. For a specific input pattern, the total number of sensitive transistors in the computational block is the sum of the sensitivities of each threshold gates. The average number of sensitive transistors, N_{avg}, is the statistical expectation of its sensitivity over a specific input DATA probability distribution:

$$N_{avg} = \sum_{i=0}^{2^n-1} \left(\sum_{j=1}^{m} N(i,j) \right) \cdot p_i \qquad (2)$$

where n is the number of dual-rail inputs, m is the number of threshold gates, $N(i,j)$ is the number of sensitive transistors in threshold gate j for input pattern i, p_i is the probability of input pattern i. Table 2 shows the calculation of average sensitivity of the NCL full adder in Fig. 5 over uniform random input DATA. Notice that the total number of transistors in the full adder is 84, and only 11.25 transistors in average are sensitive.

D. Soft Error Propagation in NCL Pipelines

In traditional synchronous circuits, when a soft error propagates to the inputs of storage elements, such as D flip-flops, it may be captured by the storage elements. If the soft error duration overlaps the clock rising edge by t_{setup} before the edge and by t_{hold} after the edge, the soft error will be captured by the D flip-flop. Note that t_{setup} and t_{hold} are the setup time and hold time of the D flip-flop, respectively. Unlike the traditional synchronous circuits, there is no global clock in NCL circuits. The delivery of the computation results from one stage to the next stage is implemented by the handshaking scheme. Therefore, whether a soft error at the output of a computational block introduces error DATA to the next state through register depends on when the soft error appears relative to the handshaking signal K_i.

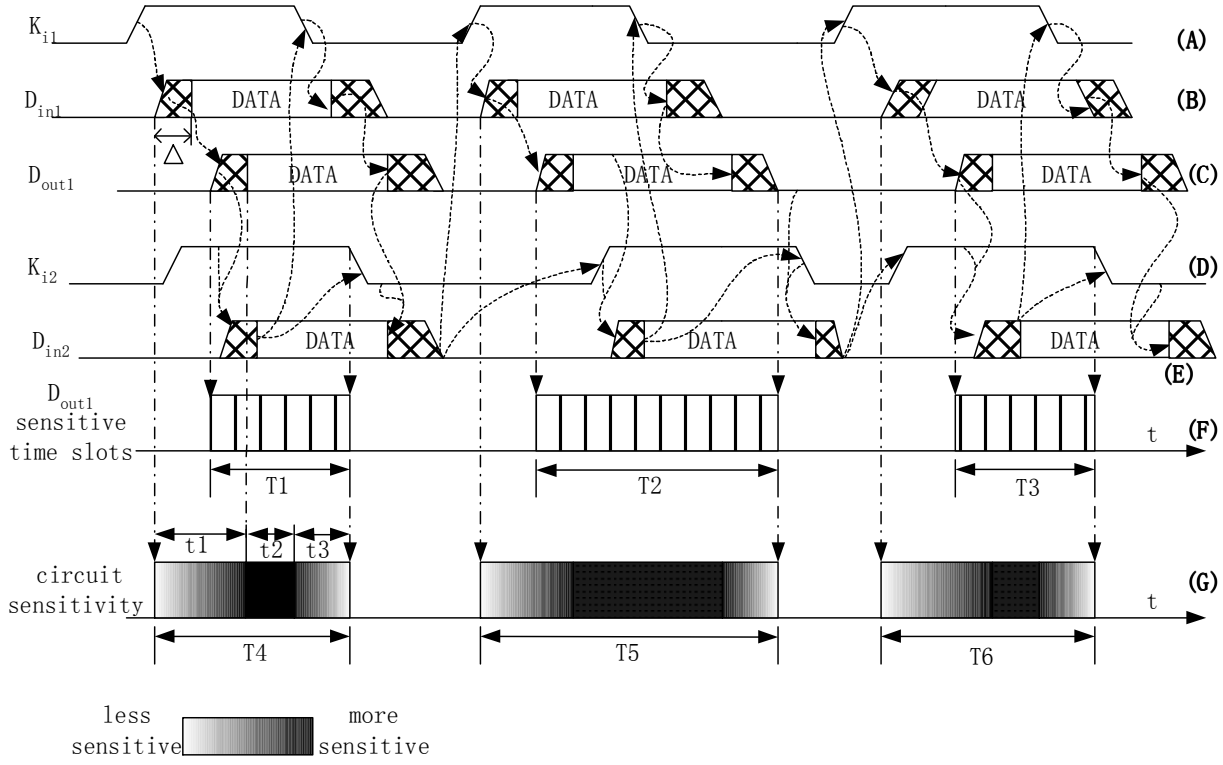


Fig. 13. NCL pipeline timing diagram and soft error sensitive time slots

Fig. 13 shows the NCL pipeline timing diagram and soft error sensitive time slots. Let us focus on computational block1 and its input register R1 and output register R2 in Fig. 1. Register R1 with high K_{i1} passes the DATA from block0 to block1 input $D_{in1.}$ The DATA arrives at block1 output D_{out1} after a propagation delay. the DATA at D_{out1} will be passed by R2 to D_{in2} when K_{i2} is high. The complete DATA at D_{in2} will turn K_{i1} to low immediately with a completion detection delay, and turn K_{i2} to low depending on the completeness of computation in block2. Then Register R1 with low K_{i1} passes the

NULL to D_{in1}. The NULL propagates to D_{out1} with a propagation delay. When K_{i2} is low, the NULL will be passed to the next stage D_{in2}. The NULL at D_{in2} will turn K_{i1} to high to receive the next DATA, and K_{i2} will eventually go to high upon the completeness of nullifying in block3.

It is reasonable to assume that only positive static soft errors should be taken into consideration because positive glitch soft errors can be easily suppressed by Schmitt trigger and transistor sizing for current or near future CMOS technology [21]. Once a static soft error is generated at the output of computational block, it will survive until a complete NULL overwrites the computational block. If there is an overlap between the lifetime of the soft error and high K_{i2}, the soft error will result in an error at the next stage. Fig. 13 (F) sketches soft error sensitive time slots, which means that if a soft error starts to appear at the output of computational block1 during those time slots (e.g. T1, T2, T3), an error DATA will be delivered to the next stage, otherwise the soft error has no effect. It can be seen that any sensitive time slot starts when the block output D_{out1} begins to transition from NULL to DATA, and ends when either K_{i2} changes to low or D_{out1} returns to NULL.

However, the computational block sensitive time slots, as shown in Fig. 13(G), are not the same as sensitive time slots at the output of computational block due to location distribution of particle strikes and soft error propagation delay. The computational block is partially sensitive to particle strike between the time when the input starts to leave all NULL state and the time when the output starts to leave all NULL state, which is denoted by t_1. A particle strike on a sensitive transistor close to inputs during t_1 has a chance to result in a positive static soft error at the output during T_1 after a propagation delay. Similarly, the computational block is also partially sensitive to particle strike during t_3. A particle strike on transistor closer to outputs during t_3 may more likely generate a positive static soft error at the output during the sensitive time slot T_1 after a shorter propagation delay. The computational block is fully sensitive to particle strike during t_2, which implies that as long as a sensitive transistor is hit by a particle, a positive static soft error will propagate to next stage.

To evaluate the circuit sensitivity quantitatively, it is assumed that any threshold gate has equal rising and falling delay, and that all stages in the NCL pipeline have an equal delay, and that the input data rate is less than the allowed maximum input rate limited by component physical delays. The following parameters are defined:

T input DATA-NULL cycle

t_R data forward delay of a register

t_{comp} completion detection delay in a register

t_{comb} the delay of the computational block

Δ the transition time from all NULL to all DATA

The minimum allowed DATA-NULL cycle is given by

$$T_{\min} = 2(2t_R + t_{comb} + t_{comp}) \tag{3}$$

A weight function *w(t)* is defined to describe the time dependency of the sensitivity of NCL pipeline, as shown in Fig. 13(G). *w(t)* is assigned "1" during fully sensitive time t_2 while it is less than one during partially sensitive time t_1 and t_3. Under the above delay assumptions, t_1, t_2 and t_3 can be expressed as

$$t_1 = t_{comb} + \Delta \tag{4}$$

$$t_2 = 2t_R + t_{comp} \tag{5}$$

$$t_3 = t_{comb} \tag{6}$$

The soft error sensitivity of a computational block in NCL pipeline is defined as the product of the average number of sensitive transistors in the computational block and the weight function *w(t)*

$$S(t) = N_{avg} \cdot w(t) \tag{7}$$

The average soft error sensitivity of each stage in NCL pipeline is defined as

$$\bar{S} = \frac{1}{T}\int_T S(t)dt = \frac{N_{avg}}{T}\int_T w(t)dt \tag{8}$$

This equation is very useful for soft error evaluation and circuit design optimization for soft error tolerance. N_{avg} is determined by the computational block while function *w(t)* mainly depends on pipeline timing. An example of applications of equation (8) can be found in [23], where a modified NCL pipeline architecture is evaluated for soft error tolerance based on the equation.

IV. BUILT-IN SOFT ERROR CORRECTION

Based on the analysis of sensitivity in Section III, the asynchronous pipeline shown in Fig. 1 is much more vulnerable to soft errors than its synchronous counterpart. Fortunately, a high soft error tolerance can be achieved by modifying the asynchronous pipeline architecture. Most importantly, a soft error at the output of computational block can be detected and corrected by utilizing the properties of asynchronous circuits. This section describes these techniques.

A. Glitch soft error suppression

In general, a threshold gate consists of four transistor networks, a pair of feedback transistors (Mp, and Mn), and an inverter, as shown Fig. 14(a). These four transistor networks are “Go to NULL”, “Hold NULL”, “Go to DATA” and “Hold DATA”. The first two networks are built with PMOS transistors to generate low output at *Z* while the later two networks using NMOS transistors to generate high output at *Z*. A positive glitch soft error is induced by particle strikes either on the drain of any NMOS transistor connected to node *A* or on the drain of the PMOS in the inverter, when all inputs are zero. Simulation shows that the drain of the PMOS in the inverter is much less sensitive to particle strike than the drain of NMOS connected to node *A*.

In order to suppress glitch soft errors in NCL circuits so that the glitch soft errors (0-1) can be ignored at logic design for soft error tolerance, a Schmitt trigger is introduced to substitute the inverter in threshold gate design, as shown in Fig. 14(b). A double-side trigger can suppress both positive and negative glitches. However, only positive glitches need to be suppressed in NCL circuits, therefore a single-side Schmitt trigger can be used for this need with less delay and power penalties. Furthermore, increasing the size of feedback transistor Mp can enhance the positive glitch suppression capability. However, the Schmitt trigger has very little improvement in static soft error preventions.

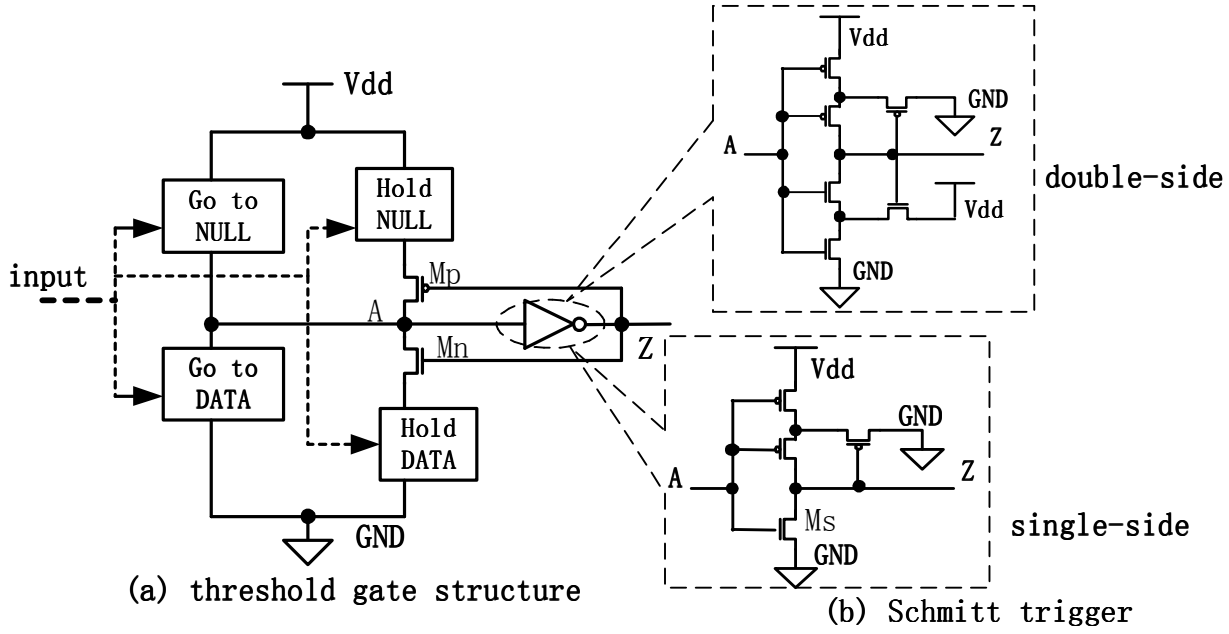


Fig. 14 Glitch soft error suppression using Schmitt trigger

B. Soft error detection and correction

A soft error at the output of computational block can be detected and corrected by utilizing the properties of asynchronous circuits. Fig. 15 shows the proposed scheme for soft error detection and correction in NCL pipeline. Several

blocks, which are assumed to be soft error free, are locally added for each stage while the asynchronous handshaking protocol between stages (in Fig. 1) is maintained. A register is inserted between the computational block *1* and output register R_2. The output of completion detection, *req*, is connected to the K_i of its own. Once the inserted register passes a complete DATA, the *comp detect* block will reset *req* to low so that any positive soft error at *D1* after the complete DATA wavefront can not propagate through the inserted register, as shown in Fig. 12 (c).

When a soft error reaches the inserted register before the DATA wavefront, the "*reset circuit*" will reset *q* to low immediately, thus resetting the whole combinational block1. After this reset is completed, *q* will go back to high, and D_{in1} will come to the combinational block1 again for re-computation. The "*SE detect"* outputs 0 if "11" code occurs in D1. The delay amount of "*delay*" block is around the difference between the propagation delay and contamination delay of the combinational block 1. The function of "*reset circuit*" can be described by Table 3, and its schematic is shown in Fig.16. It is reasonably assumed that D_{in1} would not change until the re-computation is completed. Although a corrupted DATA token may occur at D_{out1}, the signal "*error*" can indicate the timing location of the corrupted DATA token so that the following stage can discard the corrupted DATA based on the "error" signal and handshaking signals. Normally, the signal "*error*" is low. If an erroneous DATA has been delivered to the next stage, a positive pulse "*error*" will be generated and attached to this DATA, as shown in Fig. 17.

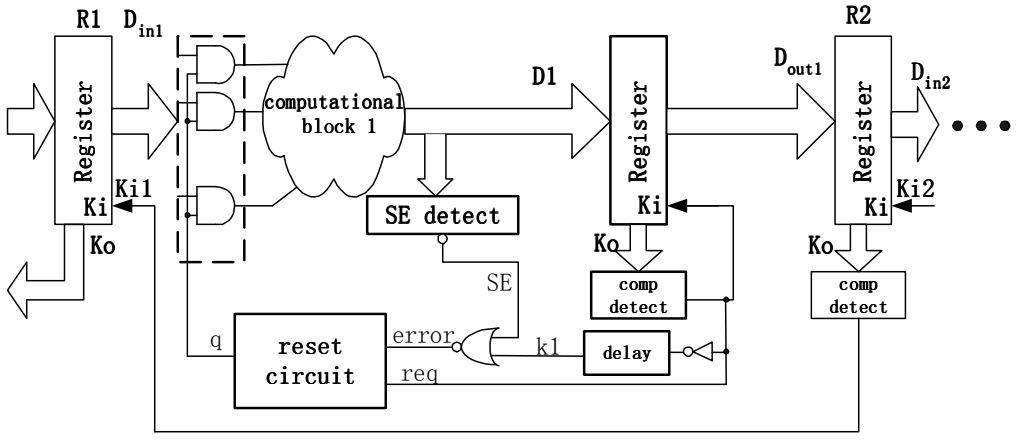


Fig. 15 Built-in soft error detection and correction scheme

req	error	q(n+1)	R	S
0	0	q(n)	1	1
0	1	0	1	0
1	1	q(n)	1	1
1	0	1	0	1

Table 3. Truth table of reset circuit

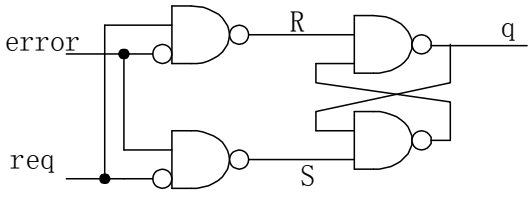


Fig. 16 Schematic of reset circuit

For simplicity, the "computational block 1" in Fig.15 is designed as a dual-rail full adder with delay elements inserted within the adder. The purpose of delay insertion is to generate the various delay distributions between output signals for typical larger designs so that the correction scheme can be sufficiently verified. Besides *error* signal, only D_{out1} is plotted in Fig.17 as (S^1, S^0) and (C^1, C^0). A particle strike at a certain time and location in the full adder results in erroneous output (1, 0) for (S^1, S^0) identified by "*error*" pulse. It should not be difficult to design a circuit to filter out the erroneous DATA based on the "*error"* signal. After resetting, an error-free re-computing result (0, 1) is followed. It should be pointed out that the delays in Fig.17 are dominated by the inserted delay elements, and do not reflect the real delay information of the full adder.

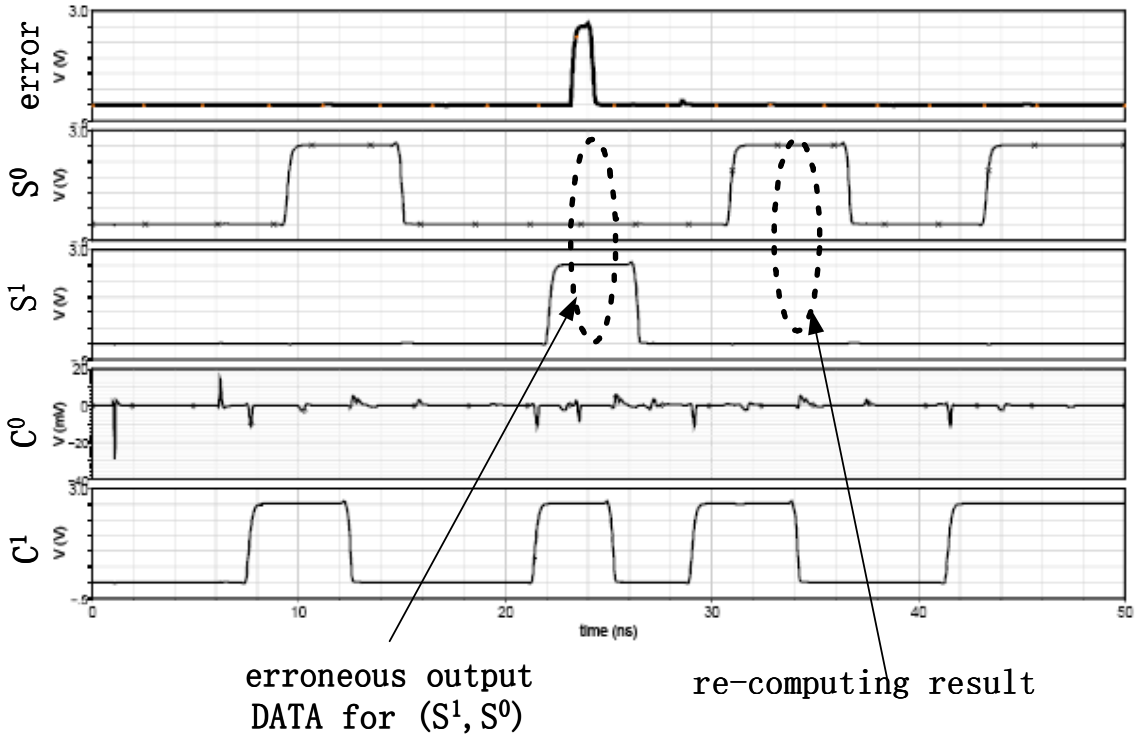
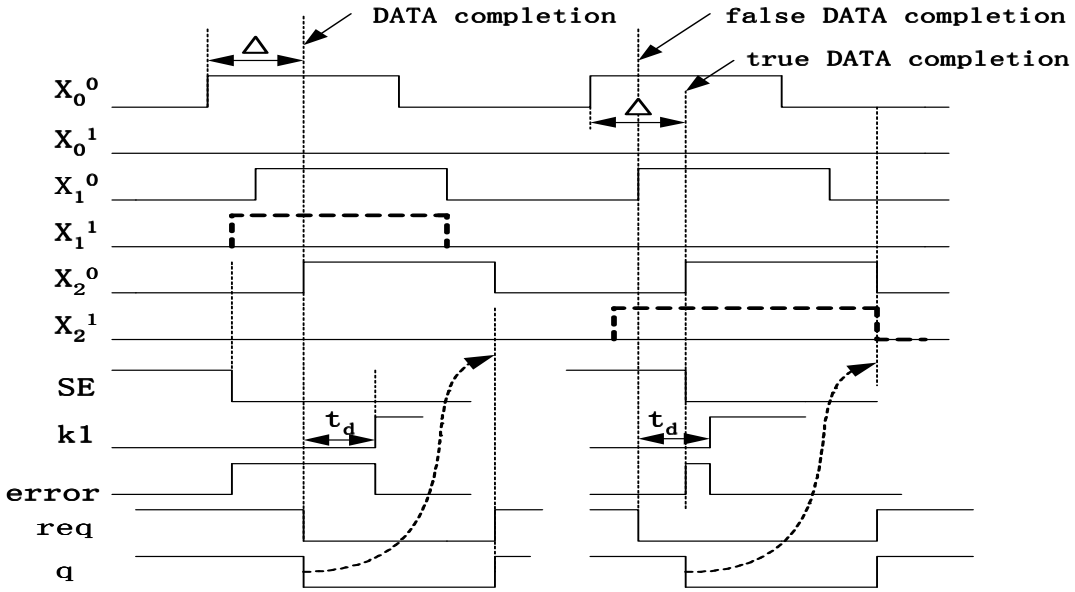


Fig. 17 Behavior of proposed error correction scheme

C. Performance analysis

The traditional NCL pipeline without the proposed correction scheme is very sensitive to soft error, shown in Fig.13, and the sensitivity is timing dependent. When the input data rate is close to the maximal pipeline speed, the sum of t_2 and t_3 (in Fig. 13) will reach the maximum, thus resulting in a maximal sensitivity. The proposed correction scheme eliminates the sensitivity during t_2 and t_3 since the inserted self-feedback register blocks any soft errors after the completion of the data. Fig. 18 is used to analyze the soft error tolerance capability, where t_d is the delay of the delay element. For sake of

simplicity, the data bus D1 in Fig.15 consists of three dual-rail bits (X_1, X_2, and X_3). Due to the multiple-paths, X_3 is assumed to be generated latest, as shown in Fig.18. Based on the analysis of Section III (c), only soft errors at zero-value rail (e.g. X_0^1, X_1^1, and X_2^1) of D1 need to be addressed, and among them those after DATA completion are eliminated by the inserted register. Thus, the re-computation is needed only when a soft error appears at D1 zero-rail during Δ. For example, a soft error at X_1^1 during Δ effectively generates *error* pulse signal and sets *q* to zero to reset the combinational block for re-computation even without the delay element (i.e. t_d=0). However, if a soft error appears at the zero-value rail X_2^1 of the latest bit X_2 during Δ, the DATA completion will falsely be detected before the true DATA completion or "11" code, therefore a delay element is required to generate a positive pulse *error*. Fig. 19 shows the impact of delay t_d on the soft error coverage, assuming that a soft error randomly appears at each zero-value rail during Δ with equal probability. As long as $\Delta \le t_d \le \frac{T}{2}$, where T is the DATA-NULL cycle, all soft errors will be detected, and result will be re-computed. If $t_d \le \Delta$, some soft errors may be missed with the worse case that all soft errors in the latest bit are missed when t_d=0. Therefore, as a trade-off option, the delay element can be deleted with a 1/N decrease of error tolerance if the design and overhead of delay element is a concern.



soft error at non-latest bit X_1 soft error at the latest bit X_2

Fig. 18 analysis of soft error correction scheme

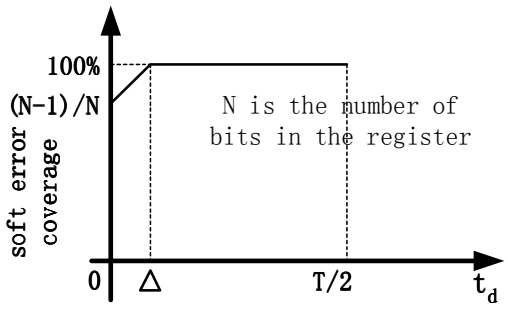


Fig. 19 the effect of delay element on soft error tolerance

Circuit	energy (pJ)/cycle
pipeline without correction circuits	4.875
pipeline with correction circuits (no soft error)	6.00
pipeline with correction circuits (a soft error)	12.375

Table 4. Energy overhead of the correction scheme

The error correction scheme leads to a hardware overhead which depends on the size of the computational block. For example, if the computational block is just a full adder that consists of 80 transistors, 66 transistors are needed for the correction circuitry, and the relative overhead is 82%. If the computational block is a 4x4 unsigned multiplier consuming 2004 transistors [24], the overhead is approximately 7%. The delay overhead due to the AND gate and inserted register is 312 ps based on the pipeline simulation. To investigate the power overhead of the correction circuitry, the energy consumed during one DATA-NULL cycle is measured for the circuit in Fig. 15 where the computational block 1 is implemented as a full adder, under three different conditions: 1) without correction circuits and no soft error; 2) with correction circuits but no soft error; 3) with correction circuits and a soft error. The results are listed in Table 4. The energy overhead of the correction scheme is 1.125 pJ. It is noticed that a soft error doubles energy consumption due to recomputation. As the increase of the complexity of the computational block, the relative energy overhead will drop accordingly.

V. SIMULATION RESULTS FOR HARDENING TECHNIQUE

The built-in soft error correction scheme has been verified by Cadence simulation, illustrated in Fig.17. The following simulation will focus on the soft error hardening technique. In order to evaluate the effectiveness of the proposed soft error hardening technique, we have performed the experiments on TH23 gate, and have compared the results of different implementations. In our experiment, every circuit is designed in a 0.12*um* CMOS technology and simulated by Cadence SPECTRE with supply voltage 1.2V. All transistors (NMOS or PMOS) have a channel width of 160*nm* and a channel length of 120*nm* except stated otherwise. Fig. 20 shows a soft error occurrence model used in our threshold gate with single-side Schmitt trigger. The same model is applied to other implementations of threshold gate in our experiment. The load capacitor is set to 17 fF [25]. A pulse current source **I** is connected to node A to mimic the effect of particle strike. When the output Z is low, a pulse current at A may result in a glitch (0-1-0) or a fault transition (0-1), as shown in Fig.21.

For TH23 gate, when ABC=000 and Z=0, the possible soft error is a glitch; when ABC=001 and Z=0, the possible soft error is a static soft error (0-1).

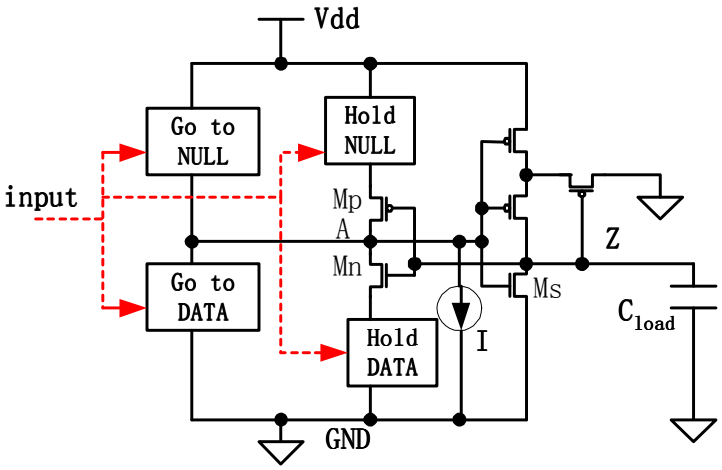


Fig. 20 Simulation setup

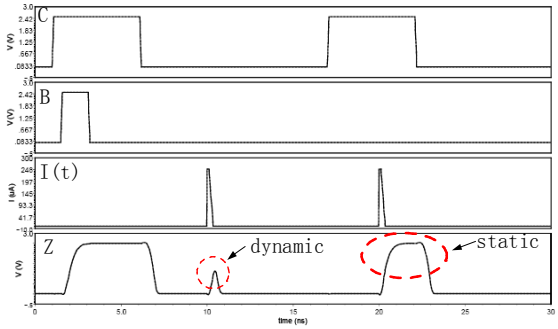


Fig. 21 Soft errors for TH23 with A=0

A. Transient Current Model

A transient current source can be used for soft error simulation, as shown in Fig.20. The current I(t) is modeled by equation (1), where two parameters, T and Q, are needed to determine current I(t). T depends on semiconductor process, and Q is proportional to particle energy. The T for 0.12*um* CMOS is assumed to be 20 ps according to [19], and the typical range of Q is from 20 fC to 120 fC [26]. For simplicity, we use a trapezoid pulse current to approximate I(t), as shown in Fig.22. The basic idea of approximation is that the trapezoid current pulse should generate the same charge as the exponential pulse, i.e. their integrals with time are the same. In our experiments, tr=3 *ps*, tf=57 *ps*, pw=20 *ps*. Imax is linearly proportional to Q, modeled as Imax=20Q *uA*. For example, Imax=400 uA corresponds to Q=20 fC.

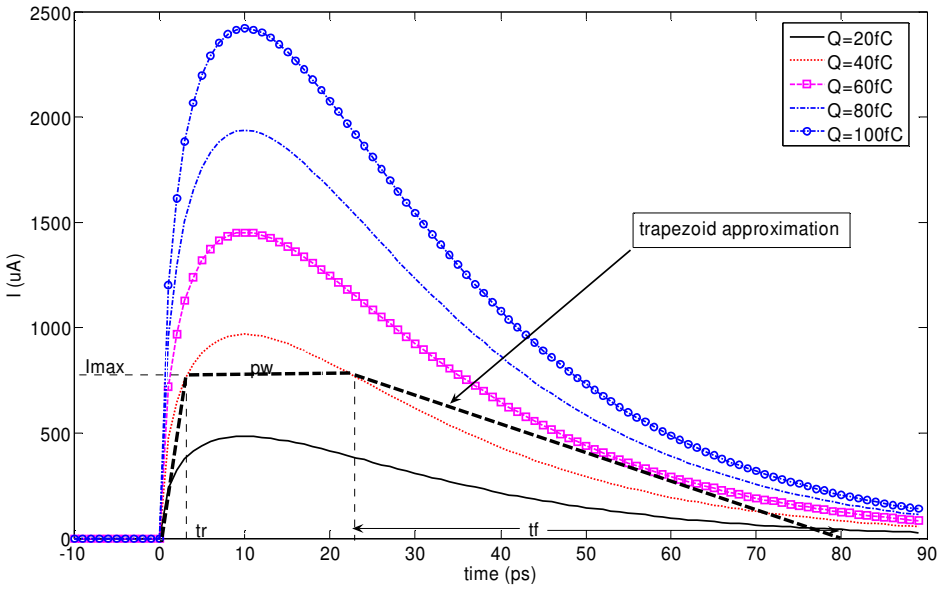


Fig. 22 Trapezoid approximation for pulse current

B. Simulation Results

To demonstrate the performance of the proposed technique, we simulated three different implementations of TH23 gate: basic, single Schmitt and double Schmitt, as shown in Fig.14. For each of them, three different widths (0.32 *um*, 0.64 *um* and 1.28 *um*) for the feedback PMOS transistor Mp are used since a larger Mp is expected to suppress more soft errors. Therefore, nine circuits are simulated and compared. To effectively compare the single Schmitt and the double Schmitt, we set the NMOS transistor Ms in single Schmitt (Fig. 14) as three times big (0.48 *um*) as one NMOS (160 *nm*) in the double Schmitt so that they consume the same area. All other transistors have the same size (W=160 *nm*, L=120 *nm*).

To measure the sensitivity of each circuit to particle strike, two current pulses are generated to mimic particle strike: one at ABC=000, and another one at ABC=001 and Z=0. The former may create a dynamic glitch, and the later may lead to a static soft error, as shown in Fig.21.

Fig.23 plots the dynamic glitch magnitude as a function of Q for different circuits. For example, "*single32*" means the design with single-side Schmitt trigger and 320 *nm* wide feedback PMOS transistor. When Q is higher than 10 fC, the basic designs will generate significant dynamic glitch that may eventually lead to a static soft error at the output of its succeeding gates. The dynamic glitches generated by the designs with Schmitt trigger are much smaller than those generated by basic designs for the same Q below 40 fC. Therefore, the Schmitt trigger does suppress dynamic glitches.

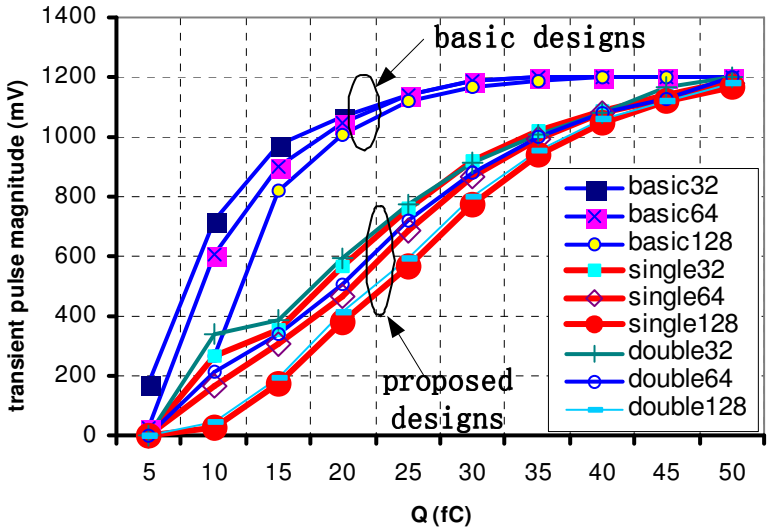


Fig. 23 Dynamic glitch magnitude

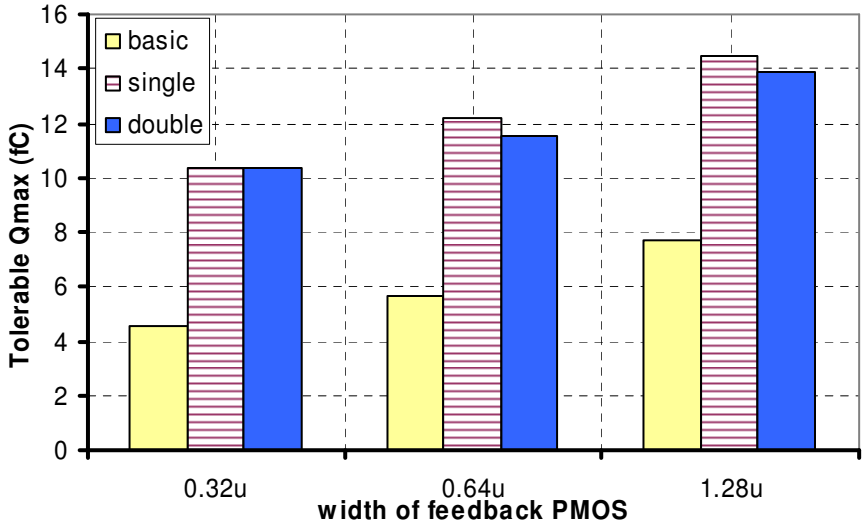


Fig. 24 Particle strike tolerance for different designs in terms of static soft error

A static soft error will occur when Q is more than a threshold Q_{max}, even for the proposed designs. It is obvious that the bigger the Q_{max}, the more robust the circuit. To find Q_{max}, during simulation we increase Imax of the pulse current source until a static soft error occurs. The Q_{max} corresponds to the maximum Imax which does not cause a static soft error. The Q_{max} is plotted in Fig.24 for nine circuits. From Fig.24, two conclusions can be drawn: 1) increasing the feedback PMOS transistor can improve the robustness to particle strike; 2) both single and double Schmitt triggers significantly increase the

maximum allowed Q without static soft error; and 3) single and double Schmitt triggers have very close impacts on the robust improvement, compared to basic design.

The proposed technique increases the insensitivity of threshold gate to particle strike. On the other hand, the penalties of the proposed technique include increased power consumption and increased delay. Fig. 25 shows the energy consumed by TH23 gate during a switch cycle. A switch cycle is defined as the time duration when the output of the gate transitions from 0 to 1, and back to 0. The circuits with single-side Schmitt trigger consume around 30% more energy than basic designs. The double-side Schmitt trigger almost doubles the energy consumption of the basic design.

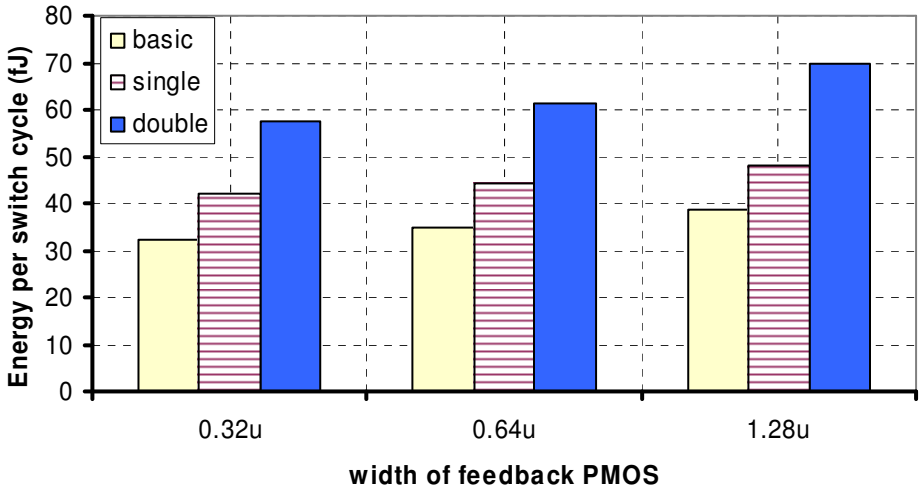


Fig.25 Power consumptions for different designs

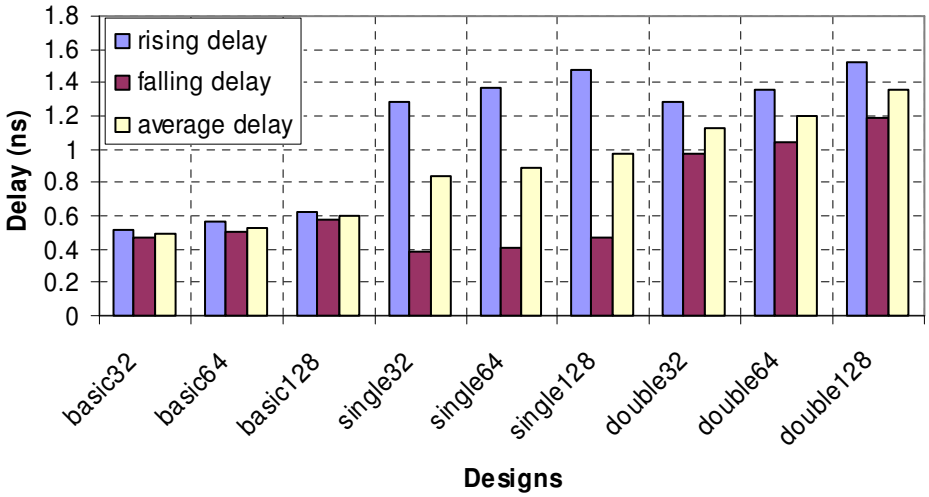


Fig. 26 Delays for different designs

Fig. 26 shows the rising, falling and average delays for each design. The rising and falling delays are measured by Cadence SPECTRE while the average delay is calculated by (rising delay+falling delay)/2. The proposed technique imposes the average delay overhead. The double-side trigger increases the average delay by a larger amount than the single-side trigger. The single-side trigger increases the rising delay and decreases the falling delay, but the average delay is bigger than that of basic design. And also, for each specific design, increasing feedback PMOS transistor size will lead to an increased delay.

Based on the above simulations, the high tolerance of particle strike can be achieved by sacrificing the power and delay performance. To achieve the same particle strike tolerance, the single-side trigger designs sacrifice less than the double-side trigger designs do. For example, *single64* has similar tolerance with *double64,* illustrated by Fig 23 and Fig 24. The *double64* design increases power by 82% and average delay by 120% while the *single64* design increases power by 29.4% and average delay by 50%, compared to the *basic64* design. Therefore, designs with single-side Schmitt trigger are better than designs with double-side Schmitt trigger.

VI. CONCLUSION

Radiation-induced soft errors threaten the reliability of digital systems as devices sizes are shrinking. In this paper, we have investigated the effect of soft errors in asynchronous circuits, and introduced a built-in soft error correction scheme with an appropriate assumption. A framework has been proposed to analyze and develop soft error tolerated digital circuits. Only positive error transitions (from low to high) may generate possible error data in the proposed asynchronous circuit. Therefore, negative error transitions can be ignored in the analysis and design for soft error tolerance at logic level. This greatly simplifies the scheme of soft error detection and correction. As a result, the corrupted data can be identified and the correct data can be obtained by re-computation with a small overhead of logic block.

Another effort of this paper focuses on a technique to suppress soft error generation at gate-level with small area, power and delay overhead. This technique employs single-side Schmitt trigger in threshold gates for Null Convention Logic, and achieves a certain amount of soft error suppression. However, this hardening method will become less effective as transistor scaling down. When considering applying this hardening technique, one should pay attention to the increased power consumption and delay due to Schmitt trigger and the larger feedback PMOS transistor.

Based on the simulations, circuits in 120 *nm* or more advanced CMOS technologies are very sensitive to particle strikes even the Schmitt trigger hardening technique is applied. Fortunately, the proposed asynchronous circuits are able to

detect and correct all soft errors in computational blocks assuming registers and correction circuits be error-free. This makes our soft error detection and correction scheme more attractive. However, from practical point of view, two topics need to be investigated in the future work: 1) a logic design for discarding the wrong data token; and 2) design and analysis of error tolerance for the correction circuits and registers.

ACKNOWLEDGMENT

This work is supported by Missile Defense Agency, the Department of Defense, USA, under the contract HQ0006-07-C-0013.

REFERENCES

[1] R. C. Baumann, "Soft errors in advanced semiconductor devices – Part I: the three radiation sources," *IEEE Trans. Device Mater. Reliab.*, vol.1, no.1, pp.17-22, Mar. 2001.

[2] P. Shivakumar, et al., "Modeling the effect of technology trends on the soft error rate of combinational logic," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2002.

[3] M. Zhang, and N. R. Shanbhag, "Soft-Error-Rate-Analysis (SERA) methodology," *IEEE Trans. CAD of ICs and Syst.*, vol. 25, no. 10, pp.2140-2155, Oct. 2006.

[4] C. L. Chen, and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: a state-of-the-art review," *IBM J. Res. Develop.* vol. 28, no. 2, pp. 124-134, 1984.

[5] S. Mitra, M. Zhang, T. M. Mak, N. Seifert, V. Zia, and K. S. Kim, "logic soft errors: a major barrier to robust platform design," in *Proc. IEEE Int. Test Conf.*, 2005, pp. 687-696.

[6] Y. Monnet, M. Renaudin, R. Leveugle, "Asynchronous circuits transient faults sensitivity evaluation," *DAC 2005*, pp. 863-868, June 13-17, 2005, Anaheim, California, USA.

[7] Y. Monnet, M. Renaudin, R. Leveugle, "Asynchronous circuits sensitivity to fault injection," in *Proc. 10th IEEE International On-Line Testing Symposium*, 2004, pp.121-126.

[8] Y. Monnet, M. Renaudin, and R. Leveugle, "Hardening techniques against transient faults for asynchronous circuits," in *Proc. 11th IEEE International On-Line Testing Symposium*, 2005, pp. 129-134.

[9] W. Jang, and A. J. Martin, "SEU-tolerant QDI circuits," In *Proc. IEEE International Symposium on Asynchronous Circuits and Systems,* 2005, pp. 156-165.

[10] S. Peng, and R. Manohar, "Efficient failure detection in pipelined asynchronous circuits," in *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*, 2005,

[11] S. Peng, and R. Manohar, "Self-healing asynchronous arrays," in *Proc. IEEE International Symposium on Asynchronous Circuits and Systems,* 2006.

[12] K. T. Gardiner, A. Yakovlev, and A. Bystrov, "A C-element latch scheme with increased transient fault tolerance for asynchronous circuits," in *Proc. 13th IEEE International On-Line Testing Symposium*, 2007.

[13] K. M. Fant and S. A. Brandt, "Null Convention Logic: A complete and consistent logic for asynchronous digital circuit synthesis," in *Int. Conf. Application-Specific Systems, Architecture and Processors,* pp. 261-273, 1996.

[14] J. Sparso and S. Furber, "Principles of asynchronous circuit design: A systems perspective," Kluwer academic publishers, 2001.

[15] K. Van Berkel, "Beware the isochronic fork," *Integration, the VLSI Journal*, vol. 13, no. 2, pp. 103-128, 1992.

[16] G. E. Sobelman, and K. Fant, "CMOS circuit design of threshold gates with hysteresis," in *Proc. Int. Symp. Circuits and Systems,* pp. 61-64, 1998.

[17] C. L. Seitz, "system timing," in *Introduction to VLSI Systems*, Addison-Wesley, 1980, pp. 218-262.

[18] W. Kuang, *Iterative ring and power-aware design techniques for self-timed digital circuits*, Ph. D. Dissertation, Department of electrical and computer engineering, University of Central Florida, August, 2003.

[19] P. Hazucha, C. Svensson, "Impact of CMOS technology scaling on the atmospheric neutron soft error rate," *IEEE Trans. on Nuclear Science*, vol. 47, no. 6, pp. 2586-2594, Dec. 2000.

[20] Y. S. Dhillon, et al., "Analysis and optimization of nanometer CMOS circuits for soft-error tolerance," *IEEE Trans. on VLSI syst.*, vol. 14, no.5, pp.514-524, 2006.

[21] Yoichi Sasaki, *et. al*., "Soft error masking circuit and latch using Schmitt trigger circuit," in *Proc. 12th Int. Symp. on Defect and Fault-Tolerance in VLSI Syst.*, 2006

[22] W. Kuang, et al., "Soft error hardening for asynchronous circuits," in *Proc. Int. 22nd IEEE Int. Symp. Defect and Fault-Tolerance in VLSI Syst*., 2007.

[23] W. Kuang, et al., "Design asynchronous circuits for soft error tolerance," in *Proc. IEEE International Conf. on Integrated Circuit Design and Technology,* Austin, Texas, June, 2007.

[24] S. K. Bandapati, S. C. Smith, and M. Choi, "Design and Characterization of NULL Convention Self-Timed Multipliers," *IEEE Design and Test of Computers: Special Issue on Clockless VLSI Design*, Vol. 30/6, pp. 26-36, November-December 2003.

[25] J.M. Rabaey, A. Chandrakasan, B. Nikolic, "Digital integrated circuits, a design perspective," Prentice Hall, 2003.

[26] H. S. Deogun, D. Sylvester, D. Blaauw, "Gate-level mitigation techniques for neutron-induced soft error rate," in *International Symposium on Quality of Electronic Design*, pp. 175-180, March, 2005.

Dear Editor,

I am glad to submit the **second minor revision** of the manuscript TVLSI-00015-2008.R1, titled "Design of Asynchronous Circuits for High Soft Error Tolerance in Deep Submicron CMOS Circuits." I appreciate all the comments from reviewers!!

I am replying the reviewer's questions as follows. (The message from EIC, associate editor, and reviewers is copied and pasted as bold and italic font, and my reply is typed as normal font)

07-Aug-2008
Dear Dr. Kuang,

Your manuscript, "Design of Asynchronous Circuits for High Soft Error Tolerance in Deep Submicron CMOS Circuits TVLSI-00015-2008.R1," has been reviewed by the IEEE Transactions on Very Large Scale Integration Systems editorial review board and we recommend reviewing the comments below, making the minor revisions, and then resubmitting the manuscript. In addition, please check your author center on Manuscript Central for any additional review files that may be available.

Please electronically resubmit your manuscript through IEEE Manuscript Central under "Revised Manuscripts." Your revised submission will now have an .R extension. Please include replies to the reviewers at the end of your revised manuscript. It is expected that you resubmit the revised manuscript within 3 weeks. If you need some additional time, please contact Stacey Weber Jackson at tvlsiadm@princeton.edu stating the reasons for additional time. If the revised manuscript is not submitted within the time allowed, it will be considered as withdrawn from the TVLSI review process. For more information, please visit our website at www.princeton.edu/~tvlsi under "Info. for Author".

Please note, once accepted, your manuscript must be in a format that is in accordance with IEEE final submission requirements. For detailed final submission requirements, kindly refer to the IEEE Information for Authors guidelines contained on the website at www.princeton.edu/~tvlsi.

Sincerely,
Prof. Niraj Jha
Editor-in-Chief
IEEE Transactions on VLSI Systems

EIC's Comments
Please revise and resubmit.

Associate Editor:
Comments to the Author:
Thank you for addressing most of the reviewer comments. One reviewer still has some minor concerns that could be addressed with a quick revision. Please go ahead and make these changes.

Reviewer's Comments
Reviewer: 1
SPECIFIC FEEDBACK TO AUTHORS
None

Reviewer: 2
SPECIFIC FEEDBACK TO AUTHORS
The authors answered all questions and changed the paper accordingly. The paper can be published as either regular or brief.

Reviewer: 3
SPECIFIC FEEDBACK TO AUTHORS
This paper investigated soft error tolerance in QDI (quasi-delay insensitive) circuits. By using Schmitt trigger circuits and feedback transistor resizing, the authors proposed a new asynchronous pipeline with soft error detection and correction. The authors also conducted a comprehensive survey of the related work on fault tolerant asynchronous designs. However, there are some issues the authors still need to address.

[Reply] In Section IV, a sub-section "C. Performance analysis" (on Page 20, 21, 22) is added to address the following issues.

(1) Delay element and recomputation
I have some concerns on using delay element for error tolerance in the asynchronous pipeline (shown in Figure 15). Due to process variation, temperature variation and voltage variation, implementing a delay element of the expected delay number is challenging in today's process technology. A delay element of longer-than-expected delay may cause a problem to the authors' design: say a particle strikes computation block 1 and causes a wrong DATA delivered (as Figure 12(d)). If the delay of delay element is longer than expected, the wrong data item could have been propagated to the following stages before 'error' becomes '1'. Now the question is, how do those pipeline stages discard the wrong data token (which is already consumed) and do recomputation when signal 'error' asserts later? Does the design have to maintain checkpoints at architecture-level and restart the computation from the last checkpoint, if any signal 'error' raises? In short, the authors need to further explain how to do recomputation, instead of simply saying that "... the signal "error" can indicate the timing location of the corrupted DATA token so that the following stage can discard the corrupted DATA ..." (on page 19). In my opinion, this is the key part to achieve error tolerance in their design. From practice point of view, the author also needs to analyze the impacts of delay variance of delay elements on error tolerance capability, because this paper is targeted for asynchronous designs with delay insensitivity.

[Reply] Delay element: a detailed analysis of delay element effect is added as the first paragraph (including Fig. 18 and Fig. 19) in Section IV C. Performance analysis. The main points include: the purpose (function) of delay element, allowed delay range (from 0 to T/2), and soft error tolerance loss when the delay is less than Δ. The results show that the allowed delay range is big enough for typical process, and the delay element can be deleted with a very small reliability loss if the design of the delay element is really a concern.
Recomputation: the statement "Although a corrupted DATA token…" has been revised to "Although a corrupted DATA token may occur at D_{out1}, the signal "*error*" can indicate the timing location of the corrupted DATA token so that the following stage can discard the corrupted DATA based on the "error" signal and handshaking signals. Normally, the signal "*error*" is low. If an erroneous DATA has been delivered to the next stage, a positive pulse "*error*" will be generated and attached to this DATA, as shown in Fig. 17."
The wrong result is wrapped by the "error" positive pulse signal, and after that, the re-computed result (right) is automatically generated and delivered to the next stage. The wrong data token will be processed by the its following stage as normal data token, but the "error" pulse signal accompanies the wrong data token wherever the wrong data token goes. Yes, eventually, a logic circuit is needed to discard the wrong result based on the "error" pulse signal. For example, if the next stage is a memory, the "error" pulse will be used as a "flag" bit to identify the data should be discarded when read. If the next stage is a typical logic circuit, a logic circuit is needed. The specific design of the "discard" logic circuit is not the focus of this paper, and will be investigate in our future work (mentioned in "VI Conclusion").

(2) Hardware overhead
I don't see that the authors reported any hardware overhead in their experiments. In fact, this is a very important metric in evaluating a fault tolerant design: If a "fault tolerant" design requires too much hardware overhead, the overall reliability may get even worse than the "non-fault-tolerant" counterpart.

I don't think the authors' statement that "It is reasonably assumed that soft errors can only be introduced in the computational blocks because other components are relatively small" (on Page 19), is true. Although each component is small compared to the computational block, the aggregate overhead of all such components cannot be neglected at all. Say the computational block 1 (in Figure 15) is an

optimized NCL full adder (as Figure 5), for example, the number of transistors required by the assumed error-free circuits (including AND gates, reset circuit, the extra register, completion detection, delay element, etc) could be almost the same as (or even larger than, if we change the computational block to TH23 gate) that of the computational block. Since the probability of particle strikes increases with the layout area, now the question is: will an "error-tolerant" design with almost twice layout area and almost half of it must be error-free, have higher reliability than the original design? The authors need to investigate this issue by analyzing reliability vs. hardware overheads of different computational block sizes.

[Reply] Yes, for the same process technology, the probability of particle strikes increases almost linearly with the layout area. According to your comments, it is important to analyze the hardware overhead. A full adder and a 4x4 multiplier are respectively used for computational block. The hardware overhead for full adder is 82% while 7% for multiplier. The effect of correction circuits and registers (assumed error-free in the paper) on the overall reliability is a difficult topic and mentioned as a part of future work in "VI Conclusion."

(3) Simulation results and benchmark circuits

It is good to see that the authors reported delay and power numbers for TH23 gate with Schmitter trigger circuits and larger feedback transistor sizing. Besides that, however, the authors should simulate an asynchronous pipeline using their design methodology (shown as Figure 15) and report the delay and power overheads. Specifically, the authors would better show that how much delay is added by inserting the extra AND gates and register in each pipeline stage, how much more power is consumed by those error detection circuits, etc. I believe that the overheads would be worse than the numbers reported in Section V.

To be a good journal paper, I suggest the authors to use a more complicated benchmark circuit (which would be mapped onto an asynchronous pipeline with multiple stages) for evaluation, so that the readers could get a better view of their proposed design methodology.

[Reply] a complete pipeline (shown in Fig.15) has been simulated for three different scenarios: non-fault tolerant design (without correction circuits), error tolerant design (but no soft error happens), error tolerant design (with the presence of particle strike). The energy overhead (1.125 pJ by error detection circuits) is shown in Table 4, and explained in the second paragraph in Section IV C. The delay overhead due to the AND gate and register is about 312 ps in 0.12 cmos process.

(4) Specific comments

-- For Figure 22 on page 24, the authors would better explain why Qmax becomes less for double Schmitt trigger than single trigger.

[Reply] If the feedback PMOS is 0.32u (width), the Qmax is almost the same for double and single. The reason why Qmax becomes less for double than single when the width of feedback PMOS is 0.64u or 1.28u is not clearly known. No further conclusion on Qmax can be made in terms of double and single. Note that the transistor Ms in single is three times big as one in double to make two Schmitt triggers consume the same area. This makes a simple comparison/explanation difficult.

-- The statement of "Fortunately, the proposed asynchronous circuits are able to detect and correct all soft errors" (on Page 26) seems a little overstated to me. Depending on the number of faults and fault locations, not all soft errors can be tolerated by this design. I suggest the authors to make the conclusion in a moderate way.

[Reply] the statement has been changed to "Fortunately, the proposed asynchronous circuits are able to detect and correct all soft errors in computational blocks assuming registers and correction circuits be error-free. This makes our soft error detection and correction scheme more attractive. However, from practical point of view, two topics need to be investigated in the future work: 1) a logic design for discarding the wrong data token; and 2) design and analysis of error tolerance for the correction circuits and registers."

In fact, it works no matter the number of faults.

Thank you for your constructive comments which make this manuscript much better!!

********** The End **********