4-2011

Soft error in FPGA-implemented asynchronous circuits

Weidong Kuang

The University of Texas Rio Grande Valley, weidong.kuang@utrgv.edu

Yu Bai

Recommended Citation

W. Kuang and Y. Bai, "Soft error in FPGA-implemented asynchronous circuits," 2011 VII Southern Conference on Programmable Logic (SPL), 2011, pp. 221-226, doi: 10.1109/SPL.2011.5782652.

SOFT ERROR IN FPGA-IMPLEMENTED ASYNCHRONOUS CIRCUITS

Weidong Kuang, and Yu Bai*

Department of Electrical Engineering, University of Texas – Pan American
1201 W. University Dr., Edinburg, TX 78539
*Email: kuangw@utpa.edu

ABSTRACT

In this paper, we investigate the mechanism of soft error generation and propagation in asynchronous circuits which are implemented on FPGAs. The effects of the soft errors on Quasi-delay-insensitive (QDI) asynchronous circuits are analyzed. The results show that it is much easier to detect the soft error in asynchronous circuits implemented on FPGAs so that FPGAs can be reprogrammed, compared with traditional synchronous circuits.

1 INTRODUCTION

Soft errors may be generated when energetic neutrons originating from cosmic rays and/or alpha particles coming from radioactive contaminants in the package material hit the surface of silicon devices. As semiconductor device scaling down, radiation-induced soft errors are the major reliability threat for digital VLSI systems [1] [2].

SRAM-based FPGAs are widely used in many applications where short time-to-market, low-cost for low-production volumes, and in-the-field-programming ability are important issues. However, FPGA-based designs are more susceptible to soft errors than application-specific integrated circuit (ASIC) implementations [3]. In SRAM-based FPGAs, the content of a configuration memory (configuration bits) specifies the functionality of the circuit mapped into the FPGA, whereas the user bits, such as flip-flops, hold the current state of the circuit. After a design being programmed into an FPGA, the content of configuration bits is supposed to remain unchanged, while the content of user bits may be changed at any clock cycle. The majority of an FPGA chip area is dedicated to memory cells which are the most vulnerable components (compared to combinational logic) to soft errors. The change of a configuration bit due to soft error will modify the functionality of the mapped circuit. Furthermore, the modification is an undetectable and permanent error in the absence of error correction schemes.

Therefore, techniques are needed to mitigate radiation effects in modern FPGAs. Many studies have focused on solutions either at device level or at architecture level. At device level, one possible solution [4] is to use radiation-hardened FPGA devices at prohibitive cost. At architecture level, redundancy designs such as triple module redundancy (TMR) are explored to protect FPGAs from soft errors [5]. TMR-based mitigation techniques impose more than 200% overhead in terms of area and power. Scrubbing, i.e., the periodic refresh of the configuration memory, is another effective approach, especially when used in conjunction with TMR.

In this paper, we investigated the generation and propagation of soft errors in asynchronous circuits implemented in FPGAs. The proposed asynchronous logic, called Null Convention Logic (NCL) [6], employs dual-rail encoding for each bit to achieve the quasi-delay insensitivity of the whole circuit. The preliminary results show the unique behavior of NCL mapped into FPGAs, and thus provide a clue for possible solutions to soft error problem in FPGAs through asynchronous design at circuit level.

The rest of the paper is organized as follows. Section II presents how NCL circuits are implemented on FPGAs. Section III analyzes the behavior of the FPGA-implemented NCL circuits. In Section IV, possible mitigation techniques are discussed, and the paper is concluded.

2 ASYNCHRONOUS CIRCUIT ON FPGAS

2.1 FPGA Architecture

An FPGA is a logic device that contains a two-dimensional array of generic logic elements (LEs) and programmable switches. A logic element can be configured (i.e., programmed) to perform a simple function, and a programmable switch can be customized to provide interconnections among the logic elements. A custom design can be implemented by specifying the function of each logic element and selectively setting the connection of each programmable switch. A logic element usually contains a programmable look-up table (LUT), programmable interconnects, and flip-flops (FF). An *n-input* look-up table is typically implemented by a static random access memory (SRAM), and is used to implement any *n-input* combinational function. The flip-flops can be selectively used to implement sequential circuits. Most FPGA devices also embed certain macro cells, such as BlockRAMs, dedicated multipliers, clock managers, and I/O interface circuits. Logic elements are usually grouped into logic array blocks (LABs).
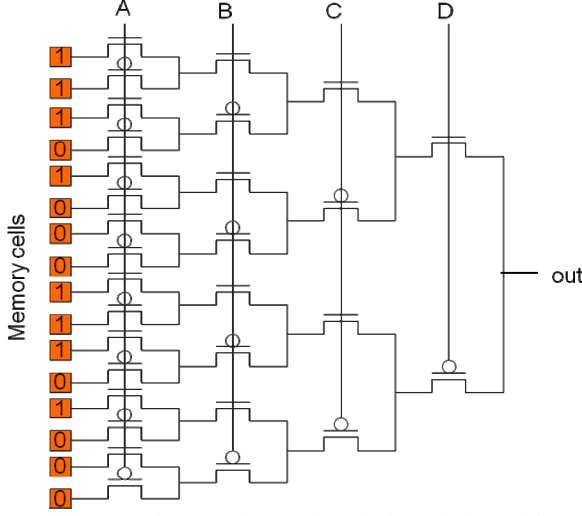
Fig.1 Architecture of SRAM-based 4-input look-up table (LUT)

The architecture of an SRAM-based 4-input look-up table, implementing a logic function $f(D, C, B, A)$, is illustrated by Fig.1. The truth table of $f(D, C, B, A)$ is stored in the memory cells. For instance, binary data 1110_1000_1110_1000 is stored in the memory cells as shown in Fig.1, to implement logic function is $f(D, C, B, A) = AB + BC + AC$.

2.2 Null Convention Logic

Asynchronous circuits can be grouped into two main categories: bundled data and delay insensitive models. 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. Delay insensitive design paradigms therefore require very little, if any, timing effort to ensure correct operation.

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.2. 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.
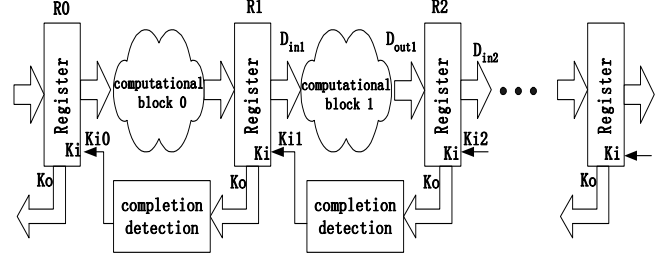

Fig. 2. NCL pipeline architecture

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

NCL circuits utilize dual-rail or quad-rail or quad-rail encoding technique to achieve delay insensitivity [6]. 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 set 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; this state is defined as an illegal state.

NCL circuits are comprised of a family of threshold gates with hysteresis. The primary type of threshold gate is THmn gate where n is the number of inputs, m is the threshold, and $1 \leq m \leq n$. A THmn gate will set its output high when any m inputs have gone high and it will reset its output low 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 THmnW$w_1w_2...w_R$, where n is the number of inputs, m is the threshold, $w_1, w_2, ...w_R$ ($1 < w_i \leq m$, $1 \leq R < n$) are the integer weights of input 1, input 2, … input R, respectively. For example, TH34 has 4 inputs (A, B, C, D) and a threshold of 3, as shown in Fig. 3 (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 TH34W22 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. 3 (b). For the gate TH34W$22,$ 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. 3 (c). *d* denotes the gate as being reset to high while *n* to low. These resettable gates are used in the design of registers. A bubble attached at the output denotes an inverter connected at the output, as shown in Fig. 3 (d). The principle of transistor-level threshold gate design can be found in [7]. The design of computational blocks, registers and completion detection blocks using threshold gates is available in [8]. For example, a dual-rail NCL full adder can be optimized as Fig. 4.
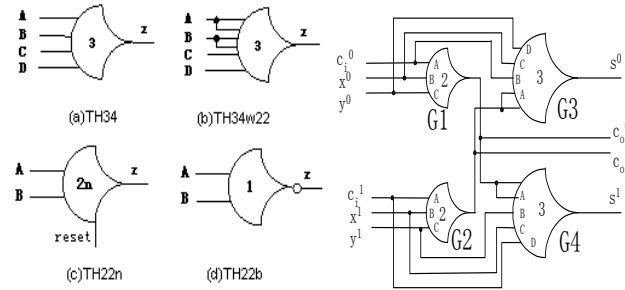
Fig. 3. Symbol examples of threshold gates   Fig.4. An NCL full adder

2.3 FPGA-Based Implementation of NCL Circuits

Generally, a threshold gate is synthesized into three logic elements in FPGAs, as shown in Fig. 5. Each logic element, utilizing an SRAM LUT, performs *Set, Reset, and Hold* respectively. The function of *Set LUT* is defined as this: its output t1 is "1" when the number of inputs equal to "1" reaches (or more than) the threshold *m*; otherwise the output is "0". For example, the *Set* function of TH23 is $t1 = AB + BC + AC$, and the *Set* function of TH34w2 (A has a weight of 2) is $t1 = A(B + C + D) + BCD$. The function of *Reset LUT* for all threshold gates is an OR gate delivering "0" when all inputs are "0". The *Hold LUT* delivers the final output with hysteresis using a feedback. It is easy to find out that its function is $z = t2 \cdot (t1 + z)$ for all threshold gates. The concept in Fig.5 has been verified by implementing TH23 and TH34w2 gates on Altera Cyclone II EP2C35F672C6N chip.
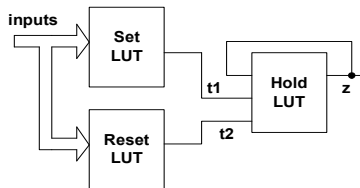
Fig. 5 Threshold gate implemented on FPGAs

3 SOFT ERROR IN FPGA-BASED ASYNCHRONOUS CIRCUITS

There are two major types of memory resources in FPGAs: user bits and configuration bits. A single event upset (SEU) induced by a particle strike in a user bit causes a transient error, whereas an SEU in a configuration bit would lead to a permanent error which remains in the FPGA until the next reconfiguration of a new design. This permanent error may result in a logic error or routing error depending on which part of the configuration memory is affected. A logic error may lead to complement one of the entries of the LUT modifying the functionality of the mapped logic function. A routing error may lead to a signal getting misrouted or disconnected [9].

We focus on the permanent logic error due to bit-flips in LUT configuration bits. It is assumed that a particle strike lead to the change of a configuration bit in LUT. Under this assumption, the implemented circuit such as a threshold gate has been changed into an undesired circuit

3.1 Soft Error Generation in a Single Threshold Gate

The TH34w2 gate is used as an example to show how and what kind of soft errors may be generated at the output of a threshold gate. TH34w2 is mapped into a 4-input *Set LUT*, a 4-input *Reset LUT* and a 3-input *Hold LUT* with feedback, as shown in Fig.6. Each entry in LUT is associated with a cell in SRAM. A particle strike may randomly lead to the data flip in a cell.

TABLE 2 lists all possible soft errors of TH34w2 associated with different data-flip (SEU) locations. The index of each box in K-map (for example, Fig.6) is used to represent the location of LUT SRAM cell. Some SEUs, such as Set LUT 0000 and Hold LUT 100 or 101, will not lead to any error. Other SEUs result in four types of soft errors: 1) premature fire, i.e., the output is erroneously equal to 1 when the inputs do not reach the threshold), 2) no fire, i.e., the output is equal to 0 when inputs reach the threshold, 3) no return to 0, i.e., the output is still 1 even when all inputs are 0, and 4) oscillating. "*Early return to 0*" usually does not lead to malfunction under reasonable delay-timing assumption, therefore being ignored here.
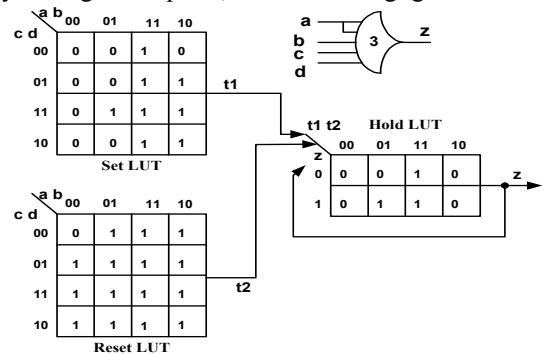
Fig. 6 TH34w2 implemented on FPGA

Other threshold gates can be implemented on FPGAs in the same way illustrated in Fig.6, except that different threshold gates have different content in Set LUT. Therefore, the above statement about four types of soft errors is generally true for all threshold gates.

3.2 Soft Error Propagation in NCL Pipelines

The behavior of any computational block in Fig.2 has a monotonic property that does not exist in traditional 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.

It is assumed that an SEU occur only in a computational block in Fig.2 because computational blocks consume the major resource of FPGAs. The behavior of an NCL pipeline in the presence of SEU can be delivered by considering: 1) dual-rail encoding, 2) monotonic property, 3) weak conditions, and 4) handshake protocol of pipeline. TABLE 3 lists all possible soft errors at the output of computational block and pipeline behavior originating from SEU in the computational block. *Premature fire* will eventually generate an invalid dual-rail code "11". *No fire* makes the computation process everlasting long while *No return to 0* makes the reset process unlimited long. *Oscillating* may lead to an invalid dual-rail code "11".

TABLE 2 Soft error of TH34w2

SEU location	Soft error at Z (worst case)
Set LUT:0000 (0→1)	No error
Set LUT: 0001,0010,0011,0100, 0101,0110,1000 (0→1)	0→1 (premature fire)
Set LUT: others (1→0)	1→0 (No fire)
Reset LUT: 0000 (0→1)	0→1 (No return to 0)
Reset LUT: 0001,0010,0011,0100 0101,0110,1000 (1→0)	1→0 (Early return to 0)
Reset LUT: others (1→0)	1→0 (No fire)
Hold LUT: 000 (0→1)	Oscillating (when abcd = 0000)
Hold LUT: 001 (0→1)	0→1 (No return to 0)
Hold LUT: 010 (0→1)	0→1 (premature fire)
Hold LUT: 011 (1→0)	1→0 (Early return to 0)
Hold LUT: 100,101 (0→1)	No error
Hold LUT: 110 (1→0)	1→0 (No fire)
Hold LUT: 111 (1→0)	Oscillating (when abcd • threshold 3)

TABLE 3 Soft errors in NCL pipeline

Soft error at TH gate	Soft error at computational output	Behavior of pipeline
Premature fire	"11"dual-rail code	**Invalid DATA**
No fire	Never to complete DATA	**Deadlock**
No return to 0	Never to complete NULL	**Deadlock**
Oscillating	"11"dual-rail code	**Invalid DATA**

4 SIMULATION RESULTS

The soft errors derived in Section III will be simulated using Quartus II software. A fault in a cell of LUT SRAM is introduced manually through Quartus II tool by choosing a specific bit location. The TH34w2 gate and NCL dual-rail full adder are used as test circuits.

4.1 Fault Injection

Quartus II provides an easy way to change the content of a desired LUT SRAM cell after compiling the VHDL files. This change is equivalent to a fault injection to the cell. After saving the change, one can simulate the behavior of the circuit with a fault injection.

TH34w2 is used an example to explain the procedures of fault injection. From Quartus II tools →Netlist viewers → Technology map viewer, one can find out TH34w2 is implemented on three LUTs: Set LUT, Reset LUT, and Hold LUT, as shown in Fig.6. The K-map, truth table, and logic function are available for each LUT in *Properties* pop-up window. The "Resource property editor" can be used to change the logic function of any LUT so that the K-map and truth table are changed accordingly as desired.

4.2 Simulation Results

First, TH34w2 gate implemented on Altera FPGA Cyclone II EP2C35F672C6N with different soft errors was simulated. The above method of fault injection is used to introduce soft error to a specific location of LUTs. The simulation results are shown in Fig.7 where a, b, c, and d are inputs. Input "a" has weight 2. "z_error_free" is the output of TH34w2 for the given input waveforms without any soft error. For simplicity, we select some soft errors in Table 2 for simulation. "z_no_fire" plots the erroneous output when a soft error (1→0) occurs in Set LUT 1010 cell. When input abcd=1010, the output is wrong "0". If a soft error (0→1) occurs in Set LUT 0001, the output may fire by mistake or fire earlier than normal depending input combinations, as shown by "z_pre_fire". If Reset LUT 0000 cell has a soft error (0→1), the output will never return to "0" after reaching "1" plotted as "z_no_return_0" in Fig.7. The output "z_early_0" returns to "0" earlier than normal when some soft errors occur, such as Reset LUT

0001 cell ($1 \rightarrow 0$). It is interesting to notice that some soft errors will lead to oscillating at the output, shown by "z_oscillate_0" and "z_oscillate_1" in Fig.7. Soft error ($0 \rightarrow 1$) in Hold LUT 000 cell leads to oscillation when all inputs are equal to "0" while soft error ($1 \rightarrow 0$) in Hold LUT 111 cell results in oscillation when the inputs meet the threshold 3. The oscillating frequency depends on the delay of Hold LUT.

Secondly, in order to investigate the soft error at RTL level, instead of TH gate level, a simple pipeline in Fig.8 is simulated with various soft errors injected into the computational block. The Computational block is implemented as an NCL full adder shown in Fig.4. Without the loss of generality, soft errors are injected into the different LUT SRAM cells in TH34w2 gate G3 that delivers the output of sum rail0. The simulation results are shown in Fig.9. From the simulation results, it can be concluded that an erroneous output at computational block may only result in one of the followings at the output of the pipeline: 1) No error, 2) "11" invalid code, or 3) deadlock. This conclusion provides the basics for soft error detection in the future work.
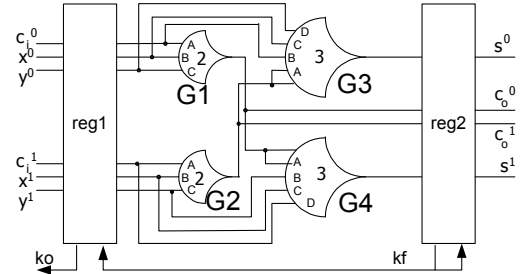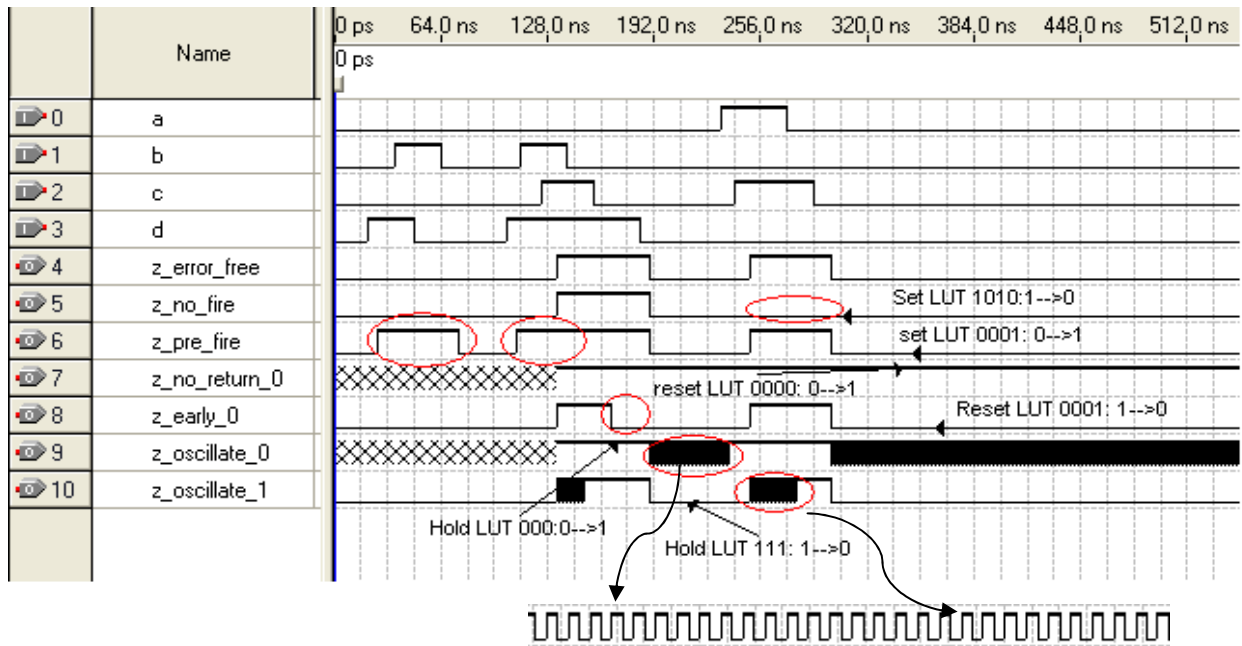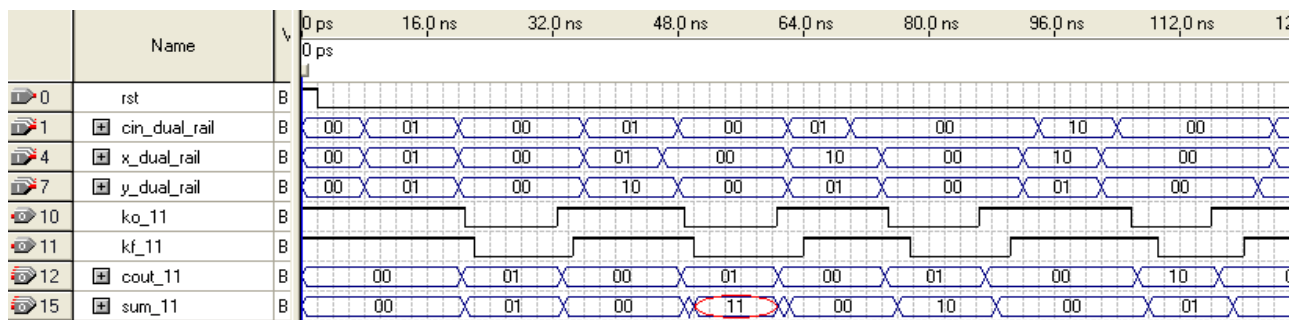

Fig.8 Circuit simulated

Fig.7 Simulation results of TH34w2 gate with different soft errors

(a) Invalid "11" appears at the output sum (soft error setting: Set LUT 0110 ($0 \rightarrow 1$) in G3 gate in full adder)
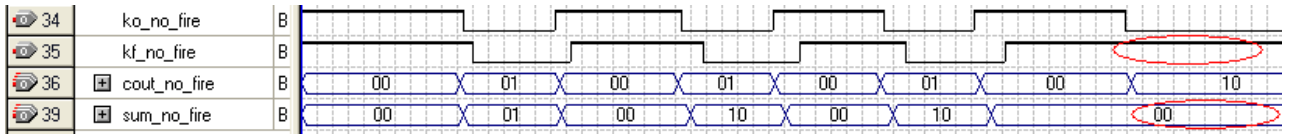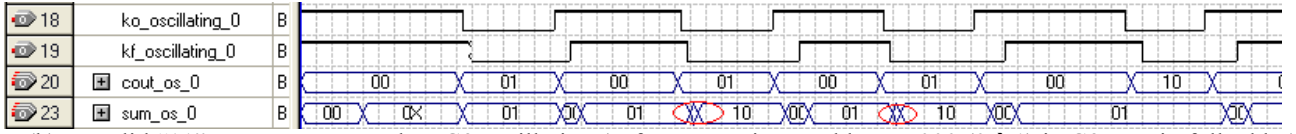
(b) Invalid "11" appears at sum when G3 ocsillating (soft error setting: Hold LUT 000 (0→1) in G3 gate in full adder)

(c) No error appears at the output when G3 ocsillating (soft error setting: Hold LUT 0110 (1→0)) in G3 gate in full adder)

(d) Deadlock in the pipeline when no fire at sum (soft error setting: Set LUT 1001 (1→0) in G3 gate in full adder)

Fig. 9 Simulation results for dual-rail full adder with different soft errors

5 DISCUSSIONS AND CONCLUSION

Null Convention Logic circuits can be implemented through either full-custom design at transistor level or FPGAs, but the effects of SEU on the two implementation styles are different. In the full-custom implementation, the soft errors are transient and no deadlock happens. The corresponding soft error detection and correction scheme was proposed in [10]. Unfortunately, the soft errors in FPGA implementations are permanent. To remove the errors, the FPGA has to be reprogrammed.

TABLE 3 gives us a hint to detect SEU in NCL pipelines. Invalid code "11" can be easily detected by ORing two rails for each bit at output of computational block. Deadlock would never be detected without sacrificing delay-insensitivity. In practice, the circuit delay and data rate are usually known to be in a particular range, therefore a counter can be designed to detect the deadlock according to whether the time of computation or reset elapses the worst case.

This paper investigated FPGA-based implementations of asynchronous circuits and their behavior in the presence of SEU in FPGA SRAM cells. The preliminary results show that dual-rail asynchronous circuits have inherent potential for SEU detection and thus FPGA reprogramming. Future works could include the estimation of soft error rate and specific reprogramming scheme related to asynchronous circuits on FPGAs. It is also essential to investigate the routing error in asynchronous circuits.

REFERENCES

[1] R. C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies," *IEEE Trans. Device and Materials Reliability*, vol. 5, no. 3, pp. 305-316, Sept. 2005.

[2] P. Shivakumar, et al., "Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic," *Proceedings of the International Conference on Dependable Systems and Networks,* 2002.

[3] H. Asadi, and M.B.Tahoori, "Analytical techniques for soft error rate modeling and mitigation of FPGA-Based Designs," *IEEE Trans.VLSI Systems,* vol.15, no.12, pp1320-1331, 2007.

[4] C.Carmichael, M.Caffrey, and A.Salazar, "SEU mitigation techniques for virtex FPGAs in space applications," in *Proc. Military Aerosp. Appl. Program. Logic Devices (MAPLD),* 1999, pp.B2.1-B2.11.

[5] F.Lima, CCarmichael, J.Fabula, R.Padovani, and R.Reis, "A fault injection analysis of virtex FPGA TMR design methodology," in *Proc. Radiation Effects Components Syst. Conf. (RADECS),* 2001, pp.275-282.

[6] K. M. Fant and S. A. Brandt, "Null convention logic: A complete and consistent logic for asynchronous digital circuit synthesis," in *Proc. Int. Conf. Appl.-Specific Syst., Arch. Process.*, 1996, pp. 261–273.

[7] G. E. Sobelman, and K. Fant, "CMOS circuit design of threshold gates with hysteresis," *Proceedings of the International Symposium on Circuits and Systems*, pp. 61-64, 1998.

[8] S. C. Smith, R. F. DeMara, J. S. Yuan, D. Ferguson, and D. Lamb, "Optimization of NULL Convention Self-Timed Circuits," *Elsevier's Integration, The VLSI Journal*, Vol. 37/3, pp. 135-165, August 2004.

[9] H.R.Zarandi, S.G.Miremadi, D.K.Pradhan, J.Mathew, "Soft error mitigation in switch modules of SRAM-based FPGAs," *IEEE Intl. Symp.on Circuits and Systems (ISCAS)* 2007, pp.141-144.

[10] W. Kuang, P. Zhao, J.S. Yuan, R.F. DeMara, "Design of asynchronous circuits for high soft error tolerance in deep submicrometer CMOS circuits," *IEEE Trans. VLSI Systems*, vol.18, no.3, pp.410-422, 2010.