SEU-tolerant QDI Circuits

Wonjin Jang, Alain J. Martin
Computer Science Department
California Institute of Technology
Pasadena, CA 91125

Abstract

This paper addresses the issue of Single-Event Upset (SEU) in quasi delay-insensitive (QDI) asynchronous circuits. We show that an SEU can cause abnormal computations in QDI circuits beside deadlock, and we propose a general method to make QDI circuits SEU-tolerant. We present a simplified SEU-tolerant buffer implementations for CMOS technology. Finally, we present a case study of a one-bit comparator and show SPICE-simulation results.

1. Introduction

1.1. Overview

When a high-energy particle hits the silicon substrate of a CMOS chip, it can generate electron-hole pairs and form a short-duration current that can change the output of a gate. The corrupt output can affect the circuit so that it performs incorrectly or halts. An error that does not damage the circuit but just changes the state of the circuit is called a *soft error*. A soft error due to excess charge (primarily induced by ionizing particles) is called a *Single-Event Upset* (SEU) [1].

With the continuous scaling down of VLSI circuits, the number of nodes in a system keeps increasing, and the charge stored at a node keeps decreasing due to lower capacitance and lower supply voltages. As a result, the probability that high-energy particles affect states of the system increases [2]. Furthermore, while radiations effects were so far mostly a concern for space-based systems, they are now a source of errors for land-based systems as well. Radiation may cause other malfunctions on a chip besides SEUs: Charges induced by radiation may slowly accumulate in the substrate of a chip. Those long-term dose effects (TID) usually cause parameter shifts, in particular threshold voltages, that affect the timing of the system.

Because asynchronous QDI systems are largely insensitive to timing variations, they may be robust to a larger fraction of the radiation effects that affect timing. Those advantages, combined with low power and robustness to environmental changes (supply voltage, temperature, ...) make it interesting to investigate the SEU-tolerance of QDI circuits.

This paper first analyzes the effect of an SEU on a QDI circuit and shows that an SEU can cause either deadlock or an erroneous computation. The paper then proposes a method for making a QDI circuit SEU-resistant. The method is different from, and more robust than, the main approach ("Triple Modular Redundancy") used for clocked circuits. In particular, the method corrects all uncorrelated upsets at once, i.e. it corrects multiple upsets as long as they do not affect the nodes that encode the same bit. Furthermore, no register or node is left corrupted, unlike standard methods that require a clean-up phase.

The paper is organized as follows. In Section 2, the representation of QDI circuits and the modeling of an SEU are outlined; the behavior of QDI circuits in presence of an SEU is analyzed. In Section 3, a method for SEU-tolerant QDI circuits based on doubling-up of nodes is presented, and its correctness is proved. Section 4 describes a case study, including SPICE simulations, and compares the performance figures of the SEU-tolerant QDI circuits with those of regular QDI circuits. Section 5 concludes.

1.2. Prior Work

At the system level, Triple Modular Redundancy (TMR) is a general solution in case of a single error in circuits [3]. It consists in replicating computation cells and implementing a voter to determine the correct value. One problem with TMR is that the system will fail if the voter fails regardless of whether

cells fail or not. A simple solution to this problem is to use three identical copies of the voter. This scheme is also known as triplicated TMR.

Also, if a common clock signal between the computation cells fails, then the TMR system cannot mask the error. A fault-tolerant TMR clock can be designed to address this problem [4]. But more importantly, the majority voting scheme of TMR cannot readily be used to design SEU-tolerant QDI circuits because the scheme may lead to a deadlock or an instability in the circuit. An SEU at boolean variables used to implement communication protocols can cause a computation cell to generate an unexpected output or to skip an expected output. If an output is missed, a deadlock may happen in the voter that waits for the output. It is not trivial to adapt the TMR scheme to make SEU-tolerant QDI circuits.

Another system-level approach to protect circuits from SEUs is Hamming codes. A SEU-hardened synchronous micro-controller using Hamming codes has been designed and tested [5]. But Hamming codes are of no use for SEUs affecting the control variables (for instance the handshake variables) of a QDI circuit.

2. SEU in QDI Circuits

2.1. QDI Circuit Representation

In this paper, QDI circuits are modeled using the HSE and PRS notations developed at Caltech. The results are of course valid independently of the model.

Caltech's methodology for designing asynchronous VLSI circuits [6] involves first writing a high-level language description in the Communicating Hardware Processes (CHP) language. A CHP program consists of one or more concurrent processes communicating via *channels*.

A CHP program is decomposed into concurrent CHP processes that are small enough to be easily compiled into the intermediate Handshaking Expansion (HSE) language. In HSE, the communication actions on channels are replaced with their "handshaking expansions" which are sequences of waits and assignments to the boolean variables implementing a handshaking protocol between sender and receiver. (These variables correspond to nodes in the physical implementation.)

The HSE is subsequently transformed into a Production-Rule Set (PRS) that is the canonical representation of a QDI circuit. A Production Rule (PR) has the form $G \rightarrow S$, where G is a boolean expression called the *guard* of the PR, and S is a *simple assignment*. A simple assignment is $z\uparrow$ or $z\downarrow$, corresponding to $z :=$`true` or $z :=$`false`. An execution of a PR $G \rightarrow S$ is an unbounded sequence of *firings*. A firing of $G \rightarrow S$ with G `true` amounts to the execution of S, and a firing with G `false` amounts to a `skip`. If the firing of a PR does change any variable's value, the firing is called *effective*. From now on if we say that a PR fires, it means that the firing is effective.

A PR $G \rightarrow S$ is said to be *stable* if whenever G becomes `true` it remains `true` until the assignment S is completed. And two PRs $G1 \rightarrow z\uparrow$ and $G2 \rightarrow z\downarrow$ are *non-interfering* if and only if $\neg G1 \vee \neg G2$ always holds. Stability and non-interference guarantees that the execution of a PR set is hazard-free. *Non-self-invalidating* of PRs is necessary to implement a PR Set (PRS) in the CMOS technology because the assignment of nodes are not instantaneous in the physical implementation. A PR $G \rightarrow z\uparrow$ is said to be *self-invalidating* when $z \Rightarrow \neg G$. Likewise $G \rightarrow z\downarrow$ is self-invalidating when $\neg z \Rightarrow G$. From now on, we only consider stable, non-interfering, and self-invalidating-free PR Set (PRS).

The two complementary PRs that set and reset the same variable, such as $G1 \rightarrow z\uparrow$ and $G2 \rightarrow z\downarrow$ form a *gate*. The variables in the guards are *inputs* of the gate and the variable in the assignment is the *output* of the gate. If $G1 \neq \neg G2$ holds, then z is a *state-holding variable*. In CMOS implementation, state-holding variables that are not always driven need staticizers. A *QDI circuit*, which interacts with its environment, is an interconnection of gates. Each input of a gate is either connected to the output of another gate, or to an environment. An input of a gate that is connected to the environment is a *primary input*; an output that is connected to the environment is a *primary output*. The environment sets values of primary inputs by reacting to values of primary outputs of the circuit according to a specification such as a four-phase handshaking protocol. We say that a circuit and its environment form a *system*.

Consider a PRS with boolean variables $x_1, x_2, ..., x_n$. A *state* representation of the PRS is a vector with one element per variable. (For convenience, we use "0" for `false` and "1" for `true` in a state representation.) And $s[x_k]$ is the value of x_k in the state s. $assignment(P)$ is the simple assignment of the PR P. For example, $assignment(x \wedge y \rightarrow z\uparrow) = z\uparrow$. For a PR P and a state s, $enb(s, P)$ is `true` if and only if the guard of P is `true` in the state s. We say that PR P is *enabled* in state s. And for a PR P and a state s, $eff(s, P)$ is `true` if and only if firing of P in state s changes the value of a variable. We call such a PR *effective* in the state s. An *execution path* $< P_1, ..., P_{m-1}, P_m >$ of a PRS is a trace of firings of PRs from an initial state. An *execution-path*

set of a PRS is a set of every possible execution path from an initial state of the PRS.

A *PRS Computation (PRSC)* is defined as follows:

- Two disjoint finite sets Σ_{Env}, called an environment, and $\Sigma_{Circuit}$, called a circuit, whose elements are PRs. ($\Sigma \stackrel{\text{let}}{=} \Sigma_{Env} \cup \Sigma_{Circuit}$.)
- An initial state $s_0 \in \{0,1\}^n$. (n is the number of distinct variables in Σ.)
- An execution-path set EP

An *environment path* of an execution path is a projection of the execution path onto Σ_{Env}. A finite set S_v is called a *valid-state set* if its elements are states reachable from s_0 by firing of PRs in Σ. A directed graph, called a *transition diagram*, is associated with a PRSC as follows. The vertices of the graph correspond to the valid states in S_v. If a PR P is effective in a state s, and it changes the state s into another state s', then there is an edge labeled P from s to s' in the transition diagram. If a PRSC is deadlock-free, for all $s \in S_v$, there exists $P \in \Sigma$ such that *eff*(s, P) is `true`. That is, every vertex has at least one outgoing edge in a deadlock-free PRSC. From now on we consider only deadlock-free PRSC.

An example PRSC is as follows:

- $\Sigma_{Env} = \{\neg Y \rightarrow Z\uparrow, Y \rightarrow Z\downarrow\}$
  $\Sigma_{Circuit} = \{\neg X \rightarrow Y\uparrow, X \rightarrow Y\downarrow, \neg Z \rightarrow X\uparrow, Z \rightarrow X\downarrow\}$
- An initial state $s_0 = (X, Y, Z) = (001)$.
- $EP = \{< Y\uparrow >, < Y\uparrow, Z\downarrow >, < Y\uparrow, Z\downarrow, X\uparrow >, ...\}$

We will use only the assignment of a PR to represent the PR in transition diagrams and execution paths for simplicity if it does not cause ambiguity. The environment-path set of the PRSC is $\{< Z\uparrow >, < Z\uparrow, Z\downarrow >, ...\}$. Figure 1 shows the transition diagram of the PRSC.

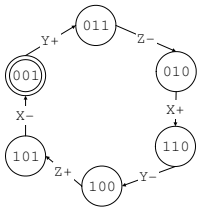

Figure 1. *Example of transition diagram*

2.2. Single-Event Upset (SEU) in PRSC

We consider an SEU as flipping the value of a single variable in a PRSC. We expand the definition of an execution path with a symbol seu_{x_i} to include effects of SEU at x_i. For example, a SEU execution path of a PRS is $< P_1, ..., P_{k-1}, seu_{x_i}, P_{k+1}..., P_{m-1}, P_m >$, which means that an SEU at x_i happens after the firing of the PR P_{k-1}, and the value of x_i is flipped. (We will use the terms 'execution path' and 'SEU execution path' interchangeably.) A *PRS Computation(PRSC) with SEU* at a variable x_i can be defined as follows:

- Two disjoint finite sets Σ_{Env}, called an environment, and $\Sigma_{Circuit}$, called a circuit, whose elements are PRs. ($\Sigma \stackrel{\text{let}}{=} \Sigma_{Env} \cup \Sigma_{Circuit}$)
- An initial state $s_0 \in \{0,1\}^n$.(n is the number of distinct variables in Σ.)
- an SEU execution-path set EP_{SEU}

Elements of the valid-state set S are states reachable from s_0 only by firing of PRs in Σ, and those of the invalid-state set Q are states reachable with SEU at x_i and unreachable without SEU at x_i. The vertices of the transition diagram with SEU correspond to the states in S and Q. If $s[x_i] \neq s'[x_i]$, then there is a two-way edge labeled *seu* between s and s' in the transition diagram.

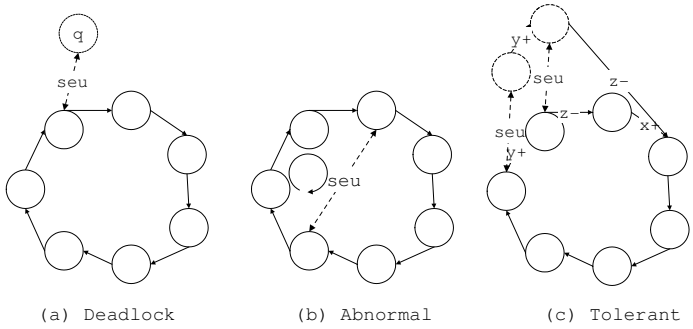

Figure 2. *Examples of PRSC with SEU*

There are three types of possible PRSCs with SEU. (1) An SEU can cause a deadlock if there exists an invalid state q such that there are no effective PRs in the state q, as shown in Figure 2 (a). (2) Some PRs in Σ_{Env} are excluded or added in SEU execution paths, compared with SEU-free execution paths. We look into the situation in the next subsection. Figure 2 (b) illustrates the case. (3) An SEU may lead to an invalid state, but it does not inhibit a firing of other PRs. The invalid state will be eventually restored to a valid state, as shown in Figure 2 (c).

In other words, if the environment-path set of a PRSC and that of the PRSC with SEU are the same, and there is no deadlock, then the PRS is

SEU-tolerant because the environment cannot distinguish them. Otherwise, the PRS is *SEU-vulnerable*. The environment-path set of an SEU-vulnerable PRSC contains deadlock execution paths or *abnormal-computation* execution paths. The environment paths of abnormal computations are not included in the SEU-free environment-path set because some PRs in the paths of abnormal computations are missed or inserted unexpectedly.

2.3. SEU in QDI Buffers

A *buffer* is a basic building block of QDI circuits. There are three common implementations of buffers, which we call PCFB (Pre-Charged Full Buffer), PCHB (Pre-Charged Half Buffer), and WCHB (Weak-Condition Half Buffer) [7]. Let us consider a single-rail PCHB whose specification in CHP is $*[L; R]$. The input channel L of the buffer is encoded with two variables L and Le and the output channel R is encoded with two variables R and Re. The HSE of the PCHB is $*[[Re \wedge L]; R\uparrow; Le\downarrow; [\neg Re]; R\downarrow; [\neg L]; Le\uparrow]$. ($Le$ and Re are inverted-sense acknowledgment variables.) The PRS constructed from the HSE is

$$\begin{array}{ll} Le \wedge Re \wedge L & \rightarrow R\uparrow \\ \neg Le \wedge \neg Re & \rightarrow R\downarrow \\ L \wedge R & \rightarrow Le\downarrow \\ \neg L \wedge \neg R & \rightarrow Le\uparrow. \end{array}$$

And the environment PRS is as follows:

$$\begin{array}{ll} Le & \rightarrow L\uparrow \\ \neg Le & \rightarrow L\downarrow \\ R & \rightarrow Re\downarrow \\ \neg R & \rightarrow Re\uparrow \end{array}$$

If the initial state is $(L, Le, R, Re) = (0101)$, the PRSC for the PCHB system is

- $\Sigma_{Env} = \{Le \rightarrow L\uparrow, \neg Le \rightarrow L\downarrow, R \rightarrow Re\downarrow, \neg R \rightarrow Re\uparrow\}$, $\Sigma_{Circuit} = \{Le \wedge Re \wedge L \rightarrow R\uparrow, \neg Le \wedge \neg Re \rightarrow R\downarrow, L \wedge R \rightarrow Le\downarrow, \neg L \wedge \neg R \rightarrow Le\uparrow\}$
- Initial state $s_0 = (0101)$
- $EP = \{< L\uparrow >, < L\uparrow, R\uparrow >, < L\uparrow, R\uparrow, Re\downarrow >, < L\uparrow, R\uparrow, L\downarrow >, < L\uparrow, R\uparrow, L\downarrow, Re\downarrow >, ...\}$

So the environment-path set of the PRSC is as follows:

$EnvP = \{< L\uparrow >, < L\uparrow, Re\downarrow >, < L\uparrow, L\downarrow >,$
$< L\uparrow, L\downarrow, Re\downarrow >, ...\}$

An SEU may happen at Le, L, Re or R, but let us assume that the environment is free from an SEU and generates inputs such as L and Re correctly. This assumption helps us to isolate effects of an SEU inside the buffer. So we will consider an SEU at only Le or R.

The PRSC with SEU at Le is as follows:

- Σ_{Env} and $\Sigma_{Circuit}$ are the same as before.
- An initial state $s_0 = (0101)$
- $EP_{SEU} = \{< seu_{Le} >, < L\uparrow >, ...,$ $< L\uparrow, seu_{Le}, L\downarrow, Le\uparrow, L\uparrow >, ..., \}$

Figure 3 shows the transition diagram of the PRSC with SEU at Le. The dotted circles indicate invalid states in Q, and the dotted edges indicate the possible transitions between states when an SEU occurs at Le. The environment-path set of the PRSC with SEU at Le is $EnvP_{SEU} = \{< L\uparrow >, ..., < L\uparrow, L\downarrow, L\uparrow >, ...\}$. The last firing $L\uparrow$ in the environment path $< L\uparrow, L\downarrow, L\uparrow >$ is added, compared with the SEU-free environment paths in $EnvP$. From the definition of SEU-vulnerability, we know that the PRSC with SEU at Le is SEU-vulnerable. The corresponding execution path $< L\uparrow, seu_{Le}, L\downarrow, Le\uparrow, L\uparrow >$ is an abnormal-computation path. This abnormal path corresponds to the situation that an input communication on L is acknowledged before an output communication on R is generated. Likewise, we can show that the PRSC with SEU at R is SEU-vulnerable: its abnormal path corresponds to the situation that the n-th output is generated before the n-th input has arrived. We can also show that the same abnormal computation occurs to other buffer implementations.

The acknowledgment of communications with one variable causes problems when an SEU happens. The change of the acknowledgment variable such as Le lets PRs in the environment fire, which results in resetting communications before a computation happens in a process. That is, firings of the PRs that can affect primary outputs are skipped, and some PRs are missed in the environment path. Similar misbehavior happens at data variables. With one-hot encoding, one boolean variable is used per data value. If an SEU occurs at one of the data variables, then the environment may react as if there is a data value. That is, an SEU at a data variable can generate an output even though there are no inputs. Another data encoding scheme for a channel is k-out-of-n encoding in which k variables are set to `true` to transmit a value over the channel. If the Hamming distance between *codewords*, valid states of data variables, is less than 2, an SEU may cause the environment to react as if it got an incorrect data value. For example, (1100) and (0110) in 2-out-of-4 encoding are codewords, and the neutral state (0000) is passing through the state (0100) to reach the state (1100). If an SEU occurs at the third variable in the state (0100), then the environment acknowledges the incorrect codeword (0110). Communication thorough one variable can experience problems under SEU. Thus we need to implement handshaking with more than one variable to avoid erroneous communications.

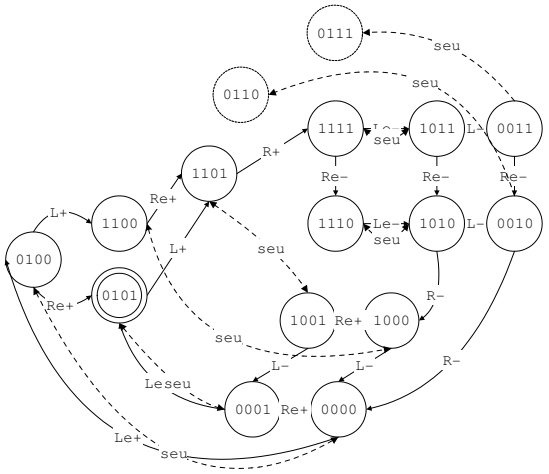

Figure 3. *Transition diagram of PCHB with SEU at Le*

3. Design for SEU-Tolerant QDI Circuit

3.1. Doubled-up Production Rules

We duplicate all variables in PRS to achieve SEU-tolerance. Let us define a *doubled-up PRS*. Every guard in a PR $G \rightarrow S$ can be written in disjunctive normal form as follows:

$$... \vee (...x_i \wedge x_j ...) \vee (...\neg x_k \wedge x_l ...) \vee ... \ \rightarrow \ S.$$

We replace all literals x_i and $\neg x_i$ with $x_{ia} \wedge x_{ib}$ and $\neg x_{ia} \wedge \neg x_{ib}$ and replace all assignments $z\uparrow$ and $z\downarrow$ with $z_a\uparrow, z_b\uparrow$ and $z_a\downarrow, z_b\downarrow$. Then we have a doubled-up PRS.

For example, consider the following PRS, which describes a two-input NAND:

$$\begin{array}{ll} x \wedge y & \rightarrow \ z\downarrow \\ \neg x \vee \neg y & \rightarrow \ z\uparrow \end{array}$$

The doubled-up NAND is

$$\begin{array}{ll} (x_a \wedge x_b) \wedge (y_a \wedge y_b) & \rightarrow \ z_a\downarrow, z_b\downarrow \\ (\neg x_a \wedge \neg x_b) \vee (\neg y_a \wedge \neg y_b) & \rightarrow \ z_a\uparrow, z_b\uparrow. \end{array}$$

From now on, we will use the notation $G \rightarrow z_a\uparrow, z_b\uparrow$ and $G \rightarrow z_a\downarrow, z_b\downarrow$ to represent $G \rightarrow z_a\uparrow$ $G \rightarrow z_b\uparrow$ and $G \rightarrow z_a\downarrow$ $G \rightarrow z_b\downarrow$.

Doubling up, however, is not enough to provide QDI circuits with SEU-tolerance because the doubled-up PRS can experience deadlock under SEU (for detail, see [8]). We need a bit more protection in addition to doubling up. Let us now define *double-checking* of doubled-up variables. We replace doubled-up variables such as x_a, x_b in all assignments with new variables such as x'_a, x'_b and supplement C-elements whose inputs are $x'_a x'_b$ and whose outputs are $x_a x_b$. For example, if we have doubled-up PRs as following:

$$\begin{array}{ll} G_{double} & \rightarrow \ x_a\uparrow, x_b\uparrow \\ G'_{double} & \rightarrow \ x_a\downarrow, x_b\downarrow, \end{array}$$

then we have the following PRs whose variables x_a, x_b are double-checked.

$$\begin{array}{ll} G_{double} & \rightarrow \ x'_a\uparrow, x'_b\uparrow \\ G'_{double} & \rightarrow \ x'_a\downarrow, x'_b\downarrow \\ x'_a \wedge x'_b & \rightarrow \ x_a\uparrow, x_b\uparrow \\ \neg x'_a \wedge \neg x'_b & \rightarrow \ x_a\downarrow, x_b\downarrow \end{array}$$

Figure 4 shows what the double-checked circuit looks like. Every variable in the doubled-up PRS becomes a state-holding variable. We normally need staticizers for these variables in a CMOS-implementation. However, the symmetry that results from doubling up helps doubled-up variables change their values simultaneously. States where nodes x_a and x_b are not driven last only for a very short time. So we can avoid some staticizers for the doubled-up nodes in practical implementations.

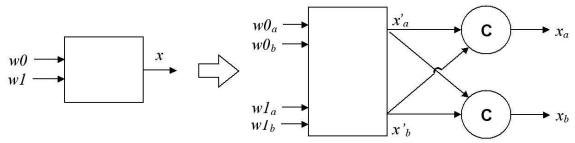

Figure 4. *A gate and a doubled-checked gate*

Theorem 1 *A doubled-up PRSC with all variables double-checked is free from deadlock and abnormal computations caused by SEU. That is, the PRSC is SEU-tolerant.*

Proof: If the original PRS is stable and non-interfering, then the doubled-up PRS is also stable and non-interfering. (The proof is simple and is omitted.) Actually we have a strong stability condition, called *doubled-up stability*, in the doubled-up PRS: the guard of doubled-up PRs $G_{double} \rightarrow S_a, S_b$ will hold until both assignments, such as $x_a\uparrow, x_b\uparrow$ or $x_a\downarrow, x_b\downarrow$, are completed. For all guards can hold only if doubled-up literals such as $x_a \wedge x_b$ or $\neg x_a \wedge \neg x_b$ hold. That is, if $x_a x_b$ are 01 or 10 in a certain state s, it means that G_{double} or G'_{double} still holds, and $x_a x_b$ are in transition from 00 to 11 or from 11 to 00.

Every variable in the doubled-up PRS with double-checking is either of the type $x_a x_b$ from doubled-up PRS or of the type $x'_a x'_b$ introduced by the double-checking scheme, and the two variables of a pair such as (x_a, x_b) are interchangeable in any context. Therefore it is enough to check whether an SEU at x_a and an SEU at x'_a break the SEU-tolerance.

First, let us assume that G_{double} holds, and an SEU at x_a occurs in a transition state $s = (...x_a x_b...) = (...10...)$ that comes from a state $q = (...x_a x_b...) = (...00...)$ where G_{double} holds. Then, s will return to the previous state q because no new PRs are enabled by $x_a x_b = 10$ in the state s.Though the SEU at x_a can make $x_a\uparrow$ fire once more than usual, it cannot cause any unexpected firings of other PRs, and it restarts from the state q. If an SEU at x_a occurs in $s = (...01...)$ from q, s will become the next state $(...11...)$ that is supposed to be reached eventually, because $G_{double} \rightarrow x_a\uparrow$ is already effective in the state s due to the doubled-up stability. Likewise the other possible transition states such as $(...10...)$ and $(...01...)$ that come from $(...11...)$ can be analyzed similarly. Therefore an SEU in transition states does not cause deadlock or abnormal computations We can use the same argument for an SEU in transition states of $x'_a x'_b$ such as $s = (...x'_a x'_b...) = (...10...)$ or $(...10...)$.

Secondly let us look into the states where $x_a x_b$ are 00 or 11. An SEU at x_a occurs in the state $s = (...x_a x_b...) = (...00...)$, which is changed into the state $s_{SEU} = (...10...)$. Some PRs may be disabled, but no new PRs are enabled in s_{SEU} because neither $x_a \wedge x_b$ nor $\neg x_a \wedge \neg x_b$ in guards are valid in s_{SEU}. There are no unexpected firings of PRs, which means there are no abnormal computations. Now we must prove that there is no deadlock because of an SEU at x_a in the state $s = (...00...)$. If we consider double-checking variables $x'_a x'_b$, there are four possible types of states for $s = (...x_a x_b x'_a x'_b...) : (...0011...)$, $(...0010...)$, $(...0001...)$ and $(...0000...)$.

1. An SEU at x_a changes $(...0011...)$ into $(...1011...)$ where there is no deadlock due to the effective double-checking PR $x'_a \wedge x'_b \rightarrow x_b\uparrow$.
2. In transition states $(...0010...)$ and $(...0001...)$ which are changing from $(...0000...)$ to $(...0011...)$, G_{double} holds because of the doubled-up stability. In addition, an SEU at x_a cannot invalidate G_{double} which does not have $\neg x_a \wedge \neg x_b$ because it comes from a non-self-invalidating PR. So one of $G_{double} \rightarrow x'_a\uparrow$ and $G_{double} \rightarrow x'_b\uparrow$ is effective in s_{SEU}.
3. An SEU at x_a changes $(...0000...)$ into $(...1000...)$ where the double-checking PR $\neg x'_a \wedge \neg x'_b \rightarrow x_a\downarrow$ is effective. Eventually the double-checking PR will restore x'_a to the correct value.

Therefore there is no deadlock caused by an SEU at x_a in the state $s = (...00...)$. We can apply the same argument to an SEU case in the state $s = (...11...)$.

Lastly let us consider an SEU at the double-checking variable x'_a in the state $s = (...x_a x_b x'_a x'_b...) = (...x_a x_b 00...)$. Only the following PRs can be affected by the change of x'_a.

$$\begin{aligned} x'_a \wedge x'_b &\rightarrow x_a\uparrow, x_b\uparrow \\ \neg x'_a \wedge \neg x'_b &\rightarrow x_a\downarrow, x_b\downarrow \end{aligned}$$

Other PRs except the double-checking PRs fire regardless of the SEU at x'_a. Eventually the variable x'_a will be restored to the correct value after $G_{double} \rightarrow x'_a\uparrow$ or $G'_{double} \rightarrow x'_a\downarrow$ fires. The same holds in case of the state $s = (...x_a x_b 11...)$. Thus there are no abnormal computations and no deadlock. ■

3.2. Multiple-Event Upset

We call variables related to doubling up and double checking such as x_a, x_b, x'_a and x'_b *correlated variables*. If multiple upsets happen among uncorrelated variables, the doubled-up PRS with double checking still computes correctly, because uncorrelated doubled-up variables such as x_a, y_a are restored by their own double-checking PRs.

Generally, the time interval between one SEU and the next SEU in the system is larger than the cycle time of a computation. If not, there may be an accumulated-SEU problem. For example, an SEU at a double-checking variable x'_a may keep a corrupt value for a long enough time that it may overlap with another SEU at x'_b. Two accumulated SEUs at correlated variables can defeat the SEU-tolerance of the double-checked PRS. In a CMOS implementation, this problem can be resolved by introducing weak C-elements, as shown in Figure 5. The weak C-elements get $x_a x_b$ to restore corrupt $x'_a x'_b$ when the double-checking C-elements are disabled. If an SEU happens at x'_a in the state $s = (...x_a x_b x'_a x'_b...) = (...0000...)$ or $(...1111...)$, then the weak C-elements are enabled to correct x'_a. In other possible states such as $(...0010...)$, $(...0001...)$ and etc., the weak C-elements are not driven, or $x'_a x'_b$ are driven by stronger gates so that the circuit behaves the same as a circuit without weak C-elements. So if necessary, we can add weak C-elements to resolve the accumulated-SEU problem.

3.3. Doubled-up Buffer Reshufflings

If a communication channel is replaced with wait and assignment of one variable, an SEU at the communication variable breaks the communication protocols. Though doubling-up PRS is a direct approach to resolve this, we can adapt the doubled-up idea to the HSE level. That is, we double up all variables used for communication in HSE and have *Doubled-up HSE (DHSE)*. The doubled-up active four-phase protocol, passive four-phase and lazy-active protocol for a channel L are as follows:

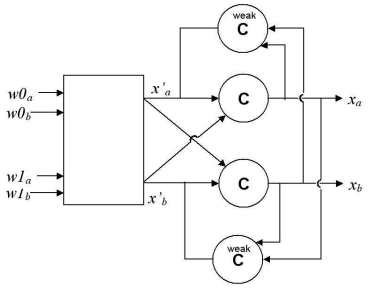

Figure 5. *A doubled-up gate with weak C-elements*

$$La\uparrow, Lb\uparrow; [\neg Lea \wedge \neg Leb]; \quad La\downarrow, Lb\downarrow; [Lea \wedge Leb]$$
$$[La \wedge Lb]; Lea\uparrow, Leb\uparrow; [\neg La \wedge \neg Lb]; Lea\downarrow, Leb\downarrow$$
$$[Lea \wedge Leb]; La\uparrow, Lb\uparrow; [\neg Lea \wedge \neg Leb]; La\downarrow, Lb\downarrow$$

Direct implementation of a system with the DHSE requires a state variable and has too much sequencing to produce efficient circuits. Instead, it is better to re-order some actions of the DHSE to reduce the amount of sequencing and the number of state variables. This transformation, called *reshuffling*, is the source of significant optimization [6]. Every reshuffling for correct DHSE implementation needs to maintain the handshaking protocols on channels. For example, the projection on the L channel in a buffer behaves like $*[[La \wedge Lb]; Lea\downarrow, Leb\downarrow; [\neg La \wedge \neg Lb]; Lea\uparrow, Leb\uparrow]$, and the projection on the R channel in a buffer behaves like $*[[Rea \wedge Reb]; Ra\uparrow, Rb\uparrow; [\neg Rea \wedge \neg Reb]; Ra\downarrow, Rb\downarrow]$.

A simple buffer without data communication would be a good example to which we apply the DHSE. The CHP specification of the buffer is $*[L; R]$, and a DHSE of the buffer without reshuffling is as follows:

$$*[[La \wedge Lb]; Lea\uparrow, Leb\uparrow; [\neg La \wedge \neg Lb]; Lea\downarrow, Leb\downarrow;$$
$$[Rea \wedge Reb]; Ra\uparrow, Rb\uparrow; [\neg Rea \wedge \neg Reb]; \quad Ra\downarrow, Rb\downarrow]$$

There are three requirements for a valid reshuffling, and they are similar to reshuffling requirements of a normal buffer except that every variable is doubled up.

1. The number of inputs is at least the number of outputs. $\#La\uparrow - \#Ra\uparrow \geq 0$ and $\#Lb\uparrow - \#Rb\uparrow \geq 0$.
2. This is a "buffer" and is supposed to acknowledge the input of the channel L independently from the acknowledgment of the channel R. That is, $(Lea\downarrow, Leb\downarrow)$ occurs concurrently with $[\neg Rea \wedge \neg Reb]$, otherwise the reshuffling result will be like a direct connection. This is the constant response time requirement.
3. If $(Lea\downarrow, Leb\downarrow)$ comes before $(Ra\uparrow, Rb\uparrow)$, the input data from the channel L would need to be saved in internal state variables. It makes the circuit larger. So $(Ra\uparrow, Rb\uparrow)$ comes before $(Lea\downarrow, Leb\downarrow)$.

Given these requirements, there are several valid reshufflings. Though each reshuffling has different features, we normally want to have fewer transistors and faster operation. By that metric we can choose three reshufflings, which are similar to the conventional PCFB, PCHB and WCHB. So we call them Doubled-up PCFB (DPCFB), Doubled-up PCHB (DPCHB) and Doubled-up WCHB (DWCHB), and they are as follows:

$DPCFB \equiv$

$$*[([Rea \wedge Reb \wedge La]; Ra\uparrow; [Lb]; Lea\downarrow; ena\downarrow;$$
$$([\neg Rea \wedge \neg Reb]; Ra\downarrow), ([\neg La \wedge \neg Lb]; Lea\uparrow); ena\uparrow),$$
$$([Rea \wedge Reb \wedge Lb]; Rb\uparrow; [La]; Leb\downarrow; enb\downarrow;$$
$$([\neg Rea \wedge \neg Reb]; Rb\downarrow), ([\neg La \wedge \neg Lb]; Leb\uparrow); enb\uparrow)]$$

$DPCHB \equiv$

$$*[([Rea \wedge Reb \wedge La]; Ra\uparrow; [Lb]; Lea\downarrow;$$
$$[\neg Rea \wedge \neg Reb]; Ra\downarrow; [\neg La \wedge \neg Lb]; Lea\uparrow),$$
$$([Rea \wedge Reb \wedge Lb]; Rb\uparrow; [La]; Leb\downarrow;$$
$$[\neg Rea \wedge \neg Reb]; Rb\downarrow; [\neg La \wedge \neg Lb]; Leb\uparrow)]$$

$DWCHB \equiv$

$$*[([Rea \wedge Reb \wedge La]; Ra\uparrow; [Lb]; Lea\downarrow;$$
$$[\neg Rea \wedge \neg Reb \wedge \neg La]; Ra\downarrow; [\neg Lb]; Lea\uparrow),$$
$$([Rea \wedge Reb \wedge Lb]; Rb\uparrow; [La]; Leb\downarrow;$$
$$[\neg Rea \wedge \neg Reb \wedge \neg Lb]; Rb\downarrow; [\neg La]; Leb\uparrow)]$$

Let us look into the handshake of channels. The projection of the DPCHB onto the L channel is as follows:

$$*[([La \wedge Lb]; Lea\downarrow; [\neg La \wedge \neg Lb]; Lea\uparrow),$$
$$([Lb \wedge La]; Leb\downarrow; [\neg La \wedge \neg Lb]; Leb\uparrow)]$$

And the environment of channel L behaves as follows:

$$*[([Lea \wedge Leb]; La\uparrow; [\neg Lea \wedge \neg Leb]; La\downarrow),$$
$$([Leb \wedge Lea]; Lb\uparrow; [\neg Lea \wedge \neg Leb]; Lb\downarrow)]$$

The environment gives the restriction that $[\neg La \wedge \neg Lb]$ does not hold until both $Lea\downarrow$ and $Leb\downarrow$ are completed. Therefore the projection of the DPCHB onto L channel is equivalent to $*[[La \wedge Lb]; Lea\downarrow, Leb\downarrow; [\neg La \wedge \neg Lb]; Lea\uparrow, Leb\uparrow]$. It is easy to check whether the remaining requirements are satisfied, and it is omitted.

The DPCHB has the following PRS:

$$Lea \wedge Rea \wedge Reb \wedge La \rightarrow Ra\uparrow$$
$$Leb \wedge Rea \wedge Reb \wedge Lb \rightarrow Rb\uparrow$$
$$\neg Lea \wedge \neg Rea \wedge \neg Reb \rightarrow Ra\downarrow$$
$$\neg Leb \wedge \neg Rea \wedge \neg Reb \rightarrow Rb\downarrow$$

$$Lb \wedge Ra \rightarrow Lea\downarrow$$
$$La \wedge Rb \rightarrow Leb\downarrow$$
$$\neg La \wedge \neg Lb \wedge \neg Ra \rightarrow Lea\uparrow$$
$$\neg La \wedge \neg Lb \wedge \neg Rb \rightarrow Leb\uparrow$$

We usually modify buffers to compute non-trivial functions. That is, a boolean formula 'La' in $Lea \wedge Rea \wedge Reb \wedge La \rightarrow Ra\uparrow$, called *computation PR*, will be replaced with another formula of input variables. Computation PRs of the DPCFB, DPCHB and DWCHB have just one more literal than those of the PCFB, PCHB and PCWB. For example, if the computation PR in the PCHB for XOR is

$$Le \wedge Re \wedge (X0 \wedge Y0 \vee X1 \wedge Y1) \rightarrow R0\uparrow,$$

then the corresponding PRs in the DPCHB are

$$\begin{array}{l} Lea \wedge Rea \wedge Reb \wedge (X0a \wedge Y0a \vee X1a \wedge Y1a) \rightarrow R0a\uparrow \\ Leb \wedge Rea \wedge Reb \wedge (X0b \wedge Y0b \vee X1b \wedge Y1b) \rightarrow R0b\uparrow. \end{array}$$

PRSCs of the DPCFB, DPCHB and DWCHB with SEU have no abnormal-computation paths. In other words, even though we weaken guards, the environment waits for completion of doubled-up variables like $[Lea \wedge Leb]$, $[\neg Lea \wedge \neg Leb]$, $[Ra \wedge Rb]$ and $[\neg Ra \wedge \neg Rb]$ to keep the environment from computing abnormally.

1. If an SEU at x_a happens in $(...x_a x_b...) = (...00...)$ or $(...11...)$, the state will become $s_{SEU} = (...01...)$ or $(...10...)$, which cannot satisfy the doubled-up waiting conditions of the environment. So there are no unexpected inputs or outputs from the environment.
2. If an SEU at x_a happens in $(...x_a x_b...) = (...10...)$ that started from a state $q = (...00...)$, the SEU may turn the state into another state $q' = (...00...)$. For not all of literals in guards are doubled-up, and a PR in the circuit may fire. It is one of differences between doubled-up PRS and DHSE. But the change of one doubled-up variable cannot cause the environment to react because of doubled-up waiting conditions.
3. If an SEU at x_a happens in $(...x_a x_b...) = (...01...)$ that started from a state $q = (...00...)$, the SEU will change the state into $s_{SEU} = (...11...)$. But it is supposed to be reached even without the SEU because the symmetry of DHSE guarantees that the state $(...10...)$ will become $(...11...)$ eventually. That is, the environment will proceed as we expect.

Therefore there are no unexpected firings of environment PRs caused by an SEU in DHSE.

The PRS of DPCHB has a possibility that deadlock occurs in invalid states caused by SEU. We resolve deadlock situations by double-checking variables as we do it for doubled-up PRS. By the same reasoning, double-checking variables restore doubled-up variables, and an SEU at double-checking variables does not affect doubled-up variables. The PRS of a double-checked DPCHB, which is SEU-tolerant, is as follows:

$$\begin{array}{ll} Lea \wedge Rea \wedge Reb \wedge La & \rightarrow Ra'\uparrow \\ Leb \wedge Rea \wedge Reb \wedge Lb & \rightarrow Rb'\uparrow \\ \neg Lea \wedge \neg Rea \wedge \neg Reb & \rightarrow Ra'\downarrow \\ \neg Leb \wedge \neg Reb \wedge \neg Rea & \rightarrow Rb'\downarrow \\ \\ Ra' \wedge Rb' & \rightarrow Ra\uparrow, Rb\uparrow \\ \neg Ra' \wedge \neg Rb' & \rightarrow Ra\downarrow, Rb\downarrow \\ \\ Lb \wedge Ra & \rightarrow Lea'\downarrow \\ La \wedge Rb & \rightarrow Leb'\downarrow \\ \neg La \wedge \neg Lb \wedge \neg Ra & \rightarrow Lea'\uparrow \\ \neg La \wedge \neg Lb \wedge \neg Rb & \rightarrow Leb'\uparrow \\ \\ \neg Lea' \wedge \neg Leb' & \rightarrow Lea\downarrow, Leb\downarrow \\ Lea' \wedge Leb' & \rightarrow Lea\uparrow, Leb\uparrow \end{array}$$

This PRS has the minimal number of literals. If we remove one of the literals, then the PRS is no longer SEU-tolerant. The other reshufflings can also be turned into SEU-tolerant circuits by the double-checked scheme. However, circuits based on DHSE cannot tolerate multiple upsets at uncorrelated variables. For example, an upset at Lea can cause bit-flipping at Ra', and the upset combined with another upset at Rb' will bring about abnormal computations.

3.4. Decomposition

Long series of transistors have bad effects on a circuit such as charge sharing and slow slew rate. But we can avoid long series of transistors by inserting intermediate variables.

Theorem 2 *Assume one gate in an SEU-tolerant PRS is as follows:*

$$\begin{array}{l} (B_u \wedge G_0) \vee G_1 \vee ... \vee G_n \rightarrow z\uparrow \\ (B_d \wedge G'_0) \vee G'_1 \vee ... \vee G'_m \rightarrow z\downarrow \end{array}$$

If the gate satisfies the following three requirements, we can introduce a new variable without violating the non-interference, stability and SEU-tolerance.

1. $\neg B_u \vee \neg B_d$ *always holds.*
2. $\neg B_u \vee \neg G_1 \vee ... \vee \neg G_n$ *and* $\neg B_d \vee \neg G'_1 \vee ... \vee \neg G'_m$ *always hold.*
3. B_u *holds until* z *becomes* `true`*, and* B_d *holds until* z *becomes* `false`*.*

We can decompose the gate as follows:

$$\begin{array}{ll} B_u & \rightarrow w\uparrow \\ B_d & \rightarrow w\downarrow \\ (w \wedge G_0) \vee G_1 \vee ... \vee G_n & \rightarrow z\uparrow \\ (\neg w \wedge G'_0) \vee G'_1 \vee ... \vee G'_m & \rightarrow z\downarrow \end{array}$$

Proof: We need to make sure the new PRS is SEU-tolerant. Any execution path of the new PRSC with the variable w can be projected onto a path of the original PRSC by removing w in state representation.

If there is an abnormal-computation or deadlock execution path in the new PRSC, then a corresponding abnormal-computation or deadlock execution path exists in the original PRSC. Then it contradicts the SEU-tolerance of the original PRS. Therefore the new PRS is SEU-tolerant.

We have to check that introducing of w does not violate non-interference and stability.

1. From the first condition, there is no interference between $B_u \rightarrow w\uparrow$ and $B_d \rightarrow w\downarrow$.
2. We know that B_u holds until z becomes `true`, which means that B_u holds until $(w \wedge G_0) \vee G_1 \vee ... \vee G_n$ holds. And B_u cannot hold simultaneously with G_1, G_2, ..., and G_n. Therefore B_u holds until w becomes `true`. Likewise B_d holds until w becomes `false`.

Thus the new PRS is stable and non-interfering. ■

For example, we can reduce the number of transistors in series in the following PRS:

$$
\begin{array}{l}
... \\
(L0a \vee L1a) \wedge (R0a \vee R1a) \;\rightarrow\; Lea'\downarrow \\
(L0b \vee L1b) \wedge (R0b \vee R1b) \;\rightarrow\; Leb'\downarrow \\
\\
\neg L0a \wedge \neg L0b \wedge \neg L1a \wedge \neg L1b \wedge \neg R0a \wedge \neg R1a \;\rightarrow\; Lea'\uparrow \\
\neg L0a \wedge \neg L0b \wedge \neg L1a \wedge \neg L1b \wedge \neg R0b \wedge \neg R1b \;\rightarrow\; Leb'\uparrow \\
...
\end{array}
$$

$(L0a \vee L1a) \wedge (R0a \vee R1a)$ can be rearranged like $(L0a \wedge (R0a \vee R1a)) \vee (L1a \wedge (R0a \vee R1a)) \rightarrow Lea'\uparrow$. $B_u \stackrel{\text{let}}{=} L0a$ and $B_u \stackrel{\text{let}}{=} \neg L0a \wedge \neg L0$ satisfy the first and second requirements Let us assume that the third requirement is satisfied. (Actually the PRs are part of a buffer, and they satisfy the assumption.) Then we can introduce a new variable $l0av$ as follows:

$$
\begin{array}{l}
... \\
L0a \;\rightarrow\; l0av\uparrow \\
\neg L0a \wedge \neg L0b \;\rightarrow\; l0av\downarrow \\
\\
(l0av \vee L1a) \wedge (R0a \vee R1a) \;\rightarrow\; Lea'\downarrow \\
(L0b \vee L1b) \wedge (R0b \vee R1b) \;\rightarrow\; Leb'\downarrow \\
\\
\neg l0av \wedge \neg L1a \wedge \neg L1b \wedge \neg R0a \wedge \neg R1a \;\rightarrow\; Lea'\uparrow \\
\neg L0a \wedge \neg L0b \wedge \neg L1a \wedge \neg L1b \wedge \neg R0b \wedge \neg R1b \;\rightarrow\; Leb'\uparrow \\
...
\end{array}
$$

4. Case Study

4.1. SEU in a Single-rail Buffer

An SEU can be modeled by a short-duration current pulse in SPICE simulation. We choose a pulse with a 10-ps rise time and a 250-ps fall time. The current peak value, 2mA is chosen to be able to flip the value of a node. (The chosen values depend on CMOS process, and energy transfer of incident particles.) In Figure 6, we see an effect of SEU in a one-bit wide PCHB whose specification is $*[L?x; R!x]$. The layout was done in the TSMC.SCN 0.18μm CMOS process offered by MOSIS. The channel L is implemented with variables $L0$, $L1$, and Le, and the channel R with variables $R0$, $R1$, and Re. From the specification, there will be the same number of outputs on the variable $R0$ as the number of inputs on the variable $L0$; the number of rising and falling signals representing $L0$ in waveforms is the same as that of $R0$. When an SEU at Le happens at 7.0ns, as shown in the lower waveform of Figure 6, the guard of $Le \wedge Re \wedge L0 \rightarrow R0\uparrow$ in the buffer PRS is invalidated, and we miss the output of $R0$. So there is no output signal between 7ns and 8ns as we see on the upper waveform of Figure 6.

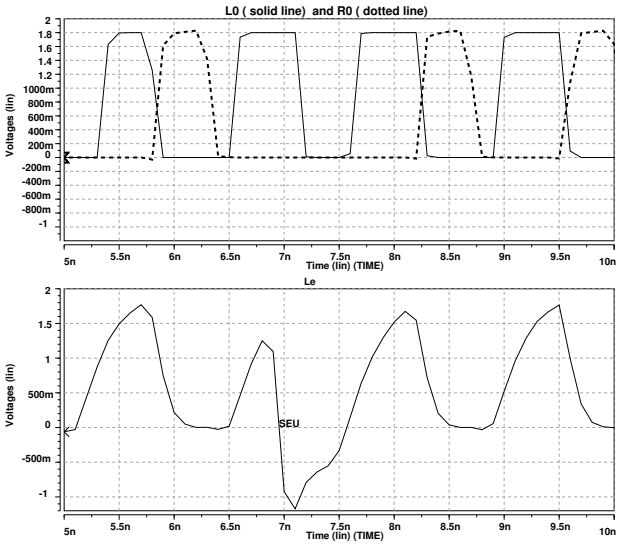

Figure 6. *Input and output waveforms under SEU at Le*

4.2. SEU in a DPCHB Comparator

We construct a one-bit wide DPCHB comparator and compare it with a PCHB comparator. The comparator accepts two 1-out-of-2 inputs and sends one 1-out-of-2 output. Its CHP specification is as follows:

$$Comp \equiv *[A?a, B?b;\;\; [a = b \longrightarrow R!0 [] a \neq b \longrightarrow R!1]]$$

There are at most three transistors in series in the pullup p-series, which is acceptable for CMOS implementation.

As we see in Table 1, the number of nodes in the SEU-tolerant circuit is a bit more than twice that of

	PCHB Comp.	DPCHB Comp.
# of nodes	14	34
Repetition Rate	500 MHz	260 MHz
# of Forward Transitions	4	4
# of Backward Transitions	10	10
Area	$22400\lambda^2$	$75350\lambda^2$

Table 1. Performance Figures

the normal one, and the size of the SEU-tolerant circuit is more than three times that of the normal one. For we double up literals in guards and then double up PRs. We manage to keep the forward and backward latency of the DPCHB comparator the same as that of the PCHB comparator. The DPCHB comparator is slower, because it uses gates that have more inputs in series than gates in the PCHB comparator.

Figure 7 shows the result of an SEU at *Aea* at 9ns. As we expect, the doubled-up nodes rise and fall almost simultaneously in normal conditions, and the SEU makes the signal shape of the node *Aea* different from that of the node *Aeb*. Even though the SEU changes the value of the *Aea* node, it does not affect the circuit behavior, and only the input signals on the node $A0a$ and $A0b$ are sustained longer than usual. The whole system will be restored to a valid state when the node *Aea* has the same value as the node *Aeb* at 12ns.

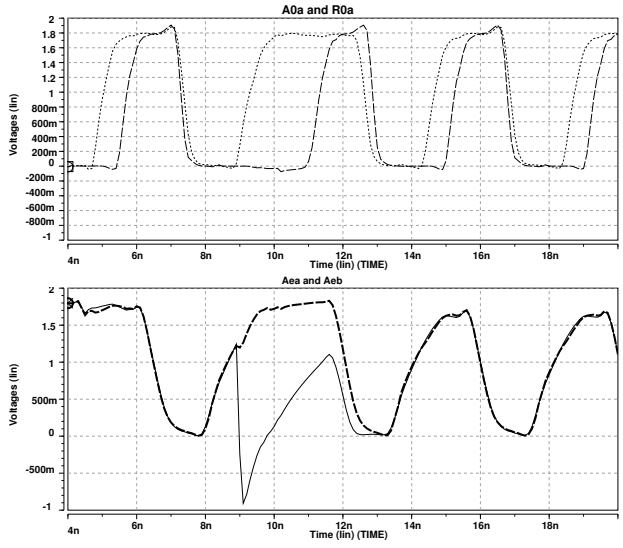

Figure 7. *SEU at Aea in a one-bit DPCHB comparator*

5. Conclusion

QDI circuits using one-variable acknowledgment for synchronization and data communication are vulnerable to SEU. When an SEU happens, the QDI circuit may perform incorrectly or halt. The method of doubling up and double-checking variables provides a useful way to make QDI circuits SEU-tolerant. Though we can use the doubling scheme directly at the PRS level, applying the scheme in a HSE level leads to more efficient PRS. So we have suggested efficient buffer reshufflings for SEU-tolerant circuits such as the DPCFB, DPCHB and DWCHB. The computation PR has only one more variable in series than that of the non-doubled schemes. To reduce the number of transistors in series, we can decompose gates by introducing intermediate variables. However, SEU-tolerant circuits are three times larger and run twice slower than normal circuits.

Acknowledgment

This research was supported by the Defense Advanced Research Project Agency (DARPA), and monitored by the Airforce. Acknowledgment is due to Mika Nyström for his excellent comments and suggestions.

References

[1] Sherra E. Kerns, "Transient-ionization and Single-Event Phenomena", *Ionizing Radiation Effects in MOS Devices and Circuits, ed. T.P. Ma and Paul V. Dressendorfer*, John Wiley & Sons, 1989.

[2] Actel Corporation, "Trends in the ASIC Marketplace", `http://www.actel.com/documents/SERppt.pdf`, June 2002.

[3] J. von Neumann, "Probabilistic logics and synthesis of reliable organisms from unreliable components", *Automata Studies, in Annals of Mathematical Studies, No. 34*, Princeton Univ., 1956, pp. 43-98.

[4] D. Davies and J.F. Wakerly, "Synchronization and matching in redundant systems", *IEEE Transactions on Computer* , June 1978, pp. 531-539.

[5] F.G. Lima, et al., "Designing a Radiation Hardened 8051-like Micro-controller", *SBCCI Conf. Proceedings*, 2000.

[6] Alain J. Martin, "Synthesis of Asynchronous VLSI Circuits", *Formal Methods for VLSI Design, ed. J. Staunstrup,* North-Holland, 1990.

[7] Andrew M. Lines, "Pipelined Asynchronous Circuits", M.S. Thesis, Dept. of Computer Science, California Institute of Technology, 1995.

[8] Wonjin Jang, "SEU-tolerant QDI Circuits", M.S. Thesis, Dept. of Computer Science, California Institute of Technology, 2004.