Soft-error Robustness in QDI Circuits

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

Abstract

This paper addresses the issue of soft errors in quasi delay-insensitive (QDI) asynchronous circuits. We propose a general method to make QDI circuits tolerant of soft errors by duplicating and double-checking variables. Finally, we present a case study of a buffer and show SPICE-simulation results.

1. Introduction

Asynchronous circuits operate without clock. Quasi-delay-insensitive (QDI) circuits are the family of asynchronous circuits that operate with the weakest timing assumption (isochronic fork) [1]. QDI circuits are robust to variable operating conditions, but like any other digital circuits they are susceptible to soft errors. As the circuit feature size decreases, soft error rates increase and become a concern for logic designers [2]. To take advantage of timing robustness and low power consumption of QDI systems, we will investigate soft-error robustness in QDI systems.

2. Design for SEU-Tolerant QDI Circuit

2.1. QDI Circuit Representation

QDI circuits are typically modelled using the Production-Rule Set (PRS) notation. Under the PRS model, a soft error is modelled as changing the value of a single boolean variable in the PRS describing the circuit ("bit-flipping").

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*, i.e., $z\uparrow$ or $z\downarrow$, meaning $z :=$`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 changes the value of any variable, 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. 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. 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. 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 $\neg z \Rightarrow G$. *Non-self-invalidating* of PRs is necessary to implement a PRS in CMOS technology because the assignment of nodes is not instantaneous in the physical implementation. From now on, we only consider stable, non-interfering, and self-invalidating-free PRS.

2.2. Duplicated Double-checking PRS

Let us define a *duplicated double-checking* PRS (DDPRS). To get a DDPRS, we duplicate all PRs in the original PRS and double-check all output variables. Double-checking duplicated output variables z_a, z_b means that we replace z_a, z_b with new variables (e.g., z'_a, z'_b) and introduce two C-elements that share the inputs z'_a, z'_b, called *checked-in (CI) variables*, and whose outputs are z_a and z_b, called *checked-out (CO) variables*. Two variables (e.g., x_a, x_b shown below) that encode the same bit, are called *duplicated variables*. The PRS of a gate are

$$\begin{array}{lcl} G_p(..., x, ...) & \rightarrow & z\uparrow \\ G_n(..., x, ...) & \rightarrow & z\downarrow, \end{array}$$

and the PRS of the corresponding DD gate are

$$\begin{array}{lcl} G_p^a(...,x_a,...) & \rightarrow & z'_a\uparrow \\ G_p^b(...,x_b,...) & \rightarrow & z'_b\uparrow \\ G_n^a(...,x_a,...) & \rightarrow & z'_a\downarrow \\ G_n^b(...,x_b,...) & \rightarrow & z'_b\downarrow \\ z'_a \wedge z'_b & \rightarrow & z_a\uparrow, z_b\uparrow \\ \neg z'_a \wedge \neg z'_b & \rightarrow & z_a\downarrow, z_b\downarrow. \end{array}$$

Figure 1 shows what the DD gate looks like. The DD gate consists of two *plain* gates and two double-checking C-elements.

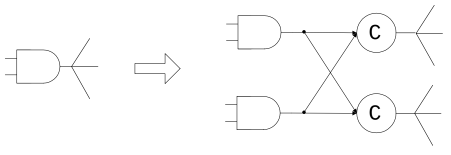

Figure 1. *Gate and DD gate*

If the original PRS is stable and non-interfering, then the DDPRS is also stable and non-interfering. Moreover, the DDPRS has an additional property on its CI variables, called *pseudo doubled-up stability.* That is, assignments of output CI variables z'_a, z'_b can fire only after both input CI variables x'_a, x'_b have the same value, and x'_a, x'_b are reset only after the assignments of z'_a, z'_b are completed. Though DDPRS seems to be much weaker than the more aggressive doubled-up scheme where every literal in guards of duplicated gates is also duplicated [3], the pseudo doubled-up stability makes DDPRS quite robust under a soft error. Figure 2 shows an example of what the resulting DDPRS circuit would look like.

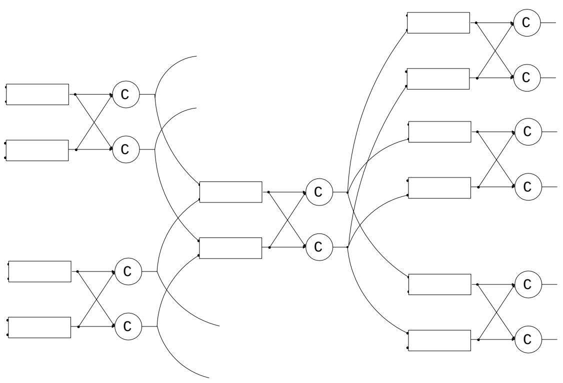

Figure 2. *DD circuit*

Theorem 1 *If there are at least three DD gates in each feedback cycle of gates, then a DDPRS is free from deadlock and abnormal computations caused by a soft error.*

Proof: The part of the DDPRS is as follows:

$$\begin{array}{lcl} ... & & \\ G_p^a(...,x_a,...) & \rightarrow & z'_a\uparrow \\ G_p^b(...,x_b,...) & \rightarrow & z'_b\uparrow \\ G_n^a(...,x_a,...) & \rightarrow & z'_a\downarrow \\ G_n^b(...,x_b,...) & \rightarrow & z'_b\downarrow \\ & & \\ z'_a \wedge z'_b & \rightarrow & z_a\uparrow, z_b\uparrow \\ \neg z'_a \wedge \neg z'_b & \rightarrow & z_a\downarrow, z_b\downarrow \\ & & \\ g_p^a(...,z_a,...) & \rightarrow & w'_a\uparrow \\ g_p^b(...,z_b,...) & \rightarrow & w'_b\uparrow \\ g_n^a(...,z_a,...) & \rightarrow & w'_a\downarrow \\ g_n^b(...,z_b,...) & \rightarrow & w'_b\downarrow \\ & & \\ w'_a \wedge w'_b & \rightarrow & w_a\uparrow, w_b\uparrow \\ \neg w'_a \wedge \neg w'_b & \rightarrow & w_a\downarrow, w_b\downarrow \\ ... & & \end{array}$$

The soft-error-tolerance of DDPRS is based on the fact that at least one duplicated variable of each pair in DDPRS will contain a correct value, and that the double-checking scheme prevents corrupted values from propagating to subsequent gates. We will examine the event scenario. Because of the symmetrical construction, without loss of generality, it is sufficient to consider the effects for soft errors occurring only at z_a and z'_a.

First, let us assume that a soft error at a CI variable z'_a occurs during the state $s = (...z'_a z'_b...) = (...10...)$ that comes from a state $q = (...z'_a z'_b...) = (...00...)$ where G_p^a holds, and s becomes s_{error} due to the soft error. No new PRs are enabled by $z'_a z'_b = 10$ in s, and values of all variables except z'_a in s remain the same as in q. The soft error merely causes s to go back to q and the redundant firing of $z'_a\uparrow$ is not discernable in the environment.

Secondly, a soft error at z'_a occurs in $s = (...01...)$ from q where G_p^b holds. The soft error is equivalent to $z'_a\uparrow$, and the symmetrical construction of DDPRS guarantees that if $G_b^p \rightarrow z'_b\uparrow$ is fired, $z'_a\uparrow$ will fire eventually. While CI variables are assigned correctly, assignments of CO variables such as x_a in G_p^a may not be completed. However the incomplete assignment is confined in the DD gate. After the environment reset the CI variables x'_a and x'_b, the CO variables x_a, x_b will be also reset to normal values. A soft error in the similar states such as (...10...) and (...01...) that come from (...11...) can be analyzed in the same manner.

Thirdly, consider a soft error at z'_a that occurs during the state $s = (...z'_a z'_b...) = (...00...)$. Only the following PRs in DDPRS can be affected by the change of z'_a, and other PRs can fire regardless of the change.

$$\begin{array}{lcl} z'_a \wedge z'_b & \rightarrow & z_a\uparrow, z_b\uparrow \\ \neg z'_a \wedge \neg z'_b & \rightarrow & z_a\downarrow, z_b\downarrow \end{array}$$

If $G^a_n \rightarrow z'_a\downarrow$ fires, the variable z'_a will be restored to a normal value. Or if $G^b_p \rightarrow z'_b\uparrow$ fires, $z'_a\uparrow$ is supposed to fire due to the symmetrical construction of DDPRS, and s_{error} will become a normal state (...11...). The same argument holds in case of a soft error in the state $s = (...11...)$.

Fourthly, consider a soft error that occurs at a CO variable z_a during a state $s = (...z_a z_b...) = (...10...)$ that comes from a state $q = (...z_a z_b...) = (...00...)$ where $z'_a \wedge z'_b \rightarrow z_a\uparrow, z_b\uparrow$ is effective. An assignment of CI variables such as w'_a in a gate can be updated in s, but the change cannot propagate beyond this gate gate due to the double-checking of w'_a and w'_b. Even though the soft error occurs, it will not affect gates beyond w_a and w_b, and input variables such as x_a and x_b remain the same as in s. So $z'_a \wedge z'_b$ still holds, and $z_a\uparrow$ will fire again to turn s_{error} into s. The soft error merely causes $z_a\uparrow$ to fire one more time than usual. There is another case that a soft error at z_a occurs in $s = (...01...)$ reached by the firing of $z_b\uparrow$ first from the state q. The soft error is equivalent to $z_a\uparrow$, which is supposed to be happen because $z'_a \wedge z'_b \rightarrow z_a\uparrow$ is effective in s. A soft error at z_a in the similar states such as (...10...) and (...01...) that come from (...11...) can be analyzed in the same manner.

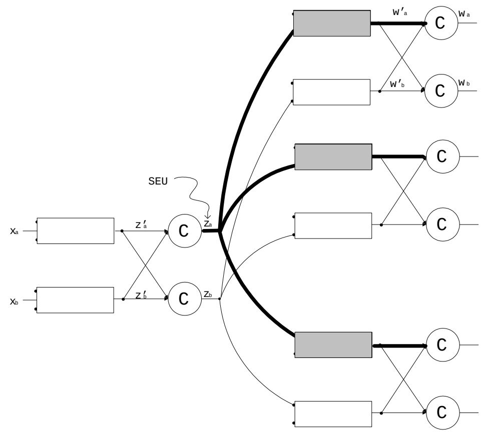

Figure 3. *Soft error at a CO variable*

Lastly, let us look into a soft error at z_a during $s = (...z_a z_b...) = (...00...)$, which is changed into $s_{error} = (...10...)$. Some of the subsequent gates may fire because of s_{error}, as shown in Figure 3. While this can affect the subsequent CI variables (such as w'_a), the corrupted values do not propagate past the double-checking C-elements, and one variable in each duplicated set remains correct. The difference relative to the previous cases is that the input z_a can cause corruption of w'_a the output of the next gate. So if there are no double-checking C-elements from w_a, w_b to x_a, x_b in a feedback cycle, the state $(...w_a w_b...) = (...10...)$ or $(...01...)$ caused by the soft error may bring about $(...z_a z_b...) = (...10...)$ or $(...01...)$, which disables double-checking gates for z and keep z_a from restoring and deadlock can happen. We can avoid this deadlock if there are at least three DD gates in each feedback cycle. In the construction, z_a are restored by the double-checking of z, and then w'_a will be corrected. ■

2.3. Multiple-Event Upset

If multiple errors happen among different set of CI and CO variables, and each error happens in a different DD gate, for example, errors at x'_a and w_a, DDPRS still are executed correctly because each error will be restored by its own double-checking C-elements.

Generally, the time interval between one soft error and the next soft error in the system is larger than the cycle time of a computation, and a corrupted value will be cleaned up before the next error happens. However a soft error at CI variable x'_a may keep a corrupted value for a long enough time that it may overlap with another soft error at x'_b. Two accumulated soft errors at correlated variables can defeat the tolerance of the DDPRS. In a CMOS implementation, this problem can be avoided by introducing weak C-elements, as shown in Figure 4. The weak C-elements use $x_a x_b$ to restore corruption on x'_a, x'_b when the double-checking C-elements are disabled. This construction can be considered as double-checked staticizers.

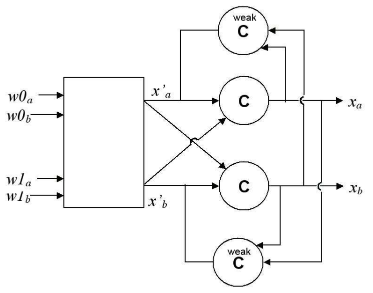

Figure 4. *DD gate with weak C-elements*

3. Case Study

We can construct soft-error-tolerant QDI systems by applying the DD scheme. Figure 5 shows a DD buffer

	e1of4 PCHB	e1of4 DDPCHB
# of nodes	25	52
Repetition Rate	680 MHz	360 MHz
Transitions in Cycle	14	18
Area(λ^2)	36736	79192
Energy per Cycle(pJ)	1.16	2.95

Table 1. Performance Figures

whose inputs and outputs are encoded by duplicated 1-out-of-2 encoding (e.g, L^0_a, L^0_b, L^1_a, L^1_b) with duplicated acknowledgements (e.g., Le_a, Le_b). The construction of the buffer, called *Pre-Charged Half Buffer (PCHB)*, is one of several possible implementation of a buffer. Here we compare DDPCHB with PCHB whose inputs and outputs are encoded by 1-out-of-4 coding. The layout was done in the TSMC.SCN 0.18-μm CMOS process offered by MOSIS.

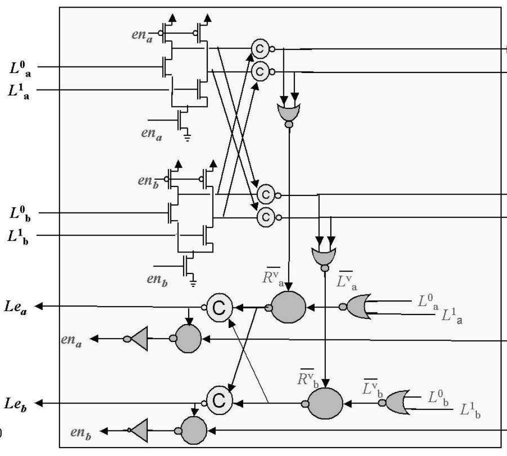

Figure 5. *DDPCHB*

As we see in Table 1, the number of nodes in the soft-error-tolerant circuit and the size of the circuit are a little bit more than twice that of the normal one. This is the result of duplicating all of the gates and then attaching double-checking C-elements. The DDPCHB becomes slower because there are more transitions in a cycle, and it uses gates that have more series transistors than gates in the PCHB.

Figure 6 shows a soft error 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, 1.5mA is chosen to be able to flip the value of a node. In the simulation a bit-flipping at *Lea* occurs at 10ns. As we expect, the doubled-up nodes rise and fall almost simultaneously in normal conditions, and the flipping makes the signal shape of the node *Lea* different from that of the node *Leb*. Even though the value of the *Lea* node is flipped, it does not affect the circuit behavior, and only the input signals arrives later than usual. The whole system will be restored to a valid state when the node *Lea* has the same value as the node *Leb* at 11ns.

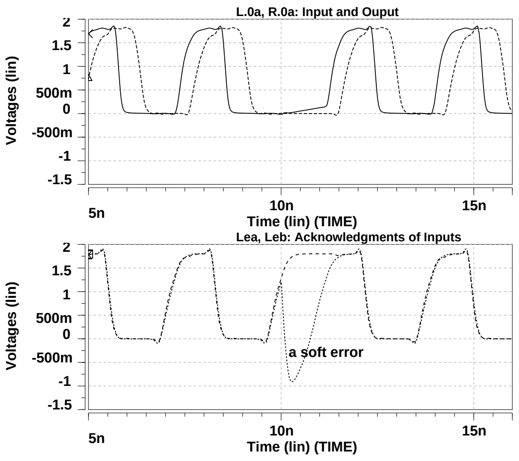

Figure 6. *Bit-flipping at Lea in DDPCHB*

4. Conclusion

When a soft error happens, a QDI system may perform incorrectly or halt. The method of duplicating and double-checking variables provides a useful way of making QDI circuits soft-error-tolerant. Duplicating variables gives the doubled-up stability. Double-checking with C-elements prevents a soft-error effect from propagating to subsequent gates. The stability property permits us to avoid triplicating the logic: the correct data can be reconstructed before it propagates to the next stage. The performance loss due to duplicating of variables and double-checking C-elements is acceptable.

References

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

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

[3] Wonjin Jang and Alain J. Martin. “SEU-tolerant QDI Circuits”. *11th Proc. on Async. Circuits and Systems* Mar, 2005.