XAPP181 (v1.0) March 15, 2000

SEU Mitigation Design Techniques for the XQR4000XL

Author: Phil Brinkley, Avnet and Carl Carmichael

Summary

This Application Note discusses system and FPGA design techniques for applications that operate in space or in other environments exposed to heavy ion or charged particle radiation. Single Event Upset (SEU) detection, correction, and mitigation for the XQR4000XL are demonstrated.

Overview

FPGA design for use in a radiation environment presents new challenges to the traditional digital designer. Often people associate radiation tolerance with the so-called "hardness" of the part. "Hardness" is simply a measure of the total dose of radiation to which an IC can be subjected before critical parameter(s) cross a predefined threshold. An IC is therefore said to be "rad tolerant" to a given total dosage, at which point some critical parameter goes out of specification.

Supply Current (I_{CC}) and Radiation Dosage

For many technologies, the supply current of a device (I_{CC}) is a critical parameter for determining useful life for a device when subjected to ionizing radiation. In some technologies, the gate control voltage at the onset of conduction (the threshold voltage) decreases (or increases) when subjected to radiation. If this threshold voltage gets too low, the integrated circuit can experience an increase on I_{CC} caused by leakage across an "off" transistor. Another cause of an increase in I_{CC} with ionizing radiation is a decrease in the field threshold (the field oxide parasitic transistor in parallel with every active transistor). If the field threshold decreases, the integrated circuit can also experience an increase in I_{CC}. While both of these phenomena limit the useful radiation exposure a device can withstand, they present different aspects to the circuit designer.

A decrease in the threshold voltage will also manifest itself in an increase in the frequency capability of the integrated circuit and an increase in I_{CC}, while a decrease in the field voltage will result only in an increase of I_{CC}. It is this latter effect that has dominates the useful ionizing radiation performance of the XQR4000XL. A plot of I_{CC} versus total dose for the XQR4000XL is shown in Figure 1.

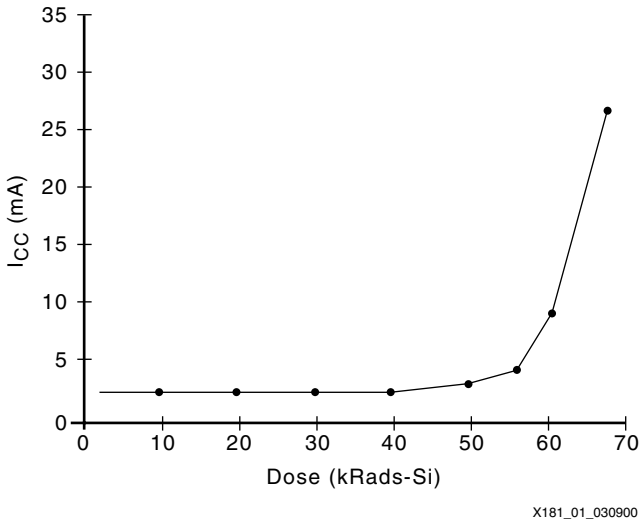

Figure 1: **I_{CC} (mA) vs. Total Dose (KRads)**

Xilinx defines the dose limit of their XCR4000XL FPGAs as the point where I_{CC} has increased to twice the commercial I_{CC} specification, with all AC parameters remaining within specification. The commercial I_{CC} specification is a very conservative value, so twice this number still falls within absolute operating limits. The 0.35µ XC4000XL radiation-tolerant FPGAs are rated as 60 KRad parts.

If the application requires a higher total dosage rating than that specified, shielding may be employed to keep the effective dosage of the FPGA below the maximum specification.

Single-Event Upset (SEU) Logic Errors

In addition to the Total Dose, Single Event Effects (SEE) must be considered. As an IC is bombarded with radiation particles, a temporary logic state change can occur within the IC. This phenomenon is known as a Single Event Upset (SEU). This effect can manifest as a *transient upset* which can last a few nanoseconds, or as a *static upset* which changes the stored charge of a static cell. For simple gates, a transient glitch in the logic is usually not an issue. When an SEU occurs within the latch that makes up a flip-flop or memory cell, however — a *static upset* — the effects on functionality are often problematical. Since a flip-flop is a memory device, the flip-flop can change state and remain in that state until the next occurring clock or reset. In this condition the flip-flop is said to have been "upset" (i.e., its state has changed independently of circuit operation). Likewise, the configuration latches, which define the user's design functionality, can be also susceptible to static upsets.

Before methods to mitigate the risks and effects of an SEU are discussed, it is important to note that the functional effect(s) of an SEU are application specific. For example, if an FPGA is being used as a digital filter and an upset causes the filter to miscalculate, the result is "bad" data for a few clock cycles. This is typically a non-mission-critical function, and as long as the error can be detected and corrected, then it may be fully acceptable. However, a mission-critical function obviously cannot tolerate a functional upset. This application note will demonstrate ways to remove risk from functional upsets. Determining the risks and effects of an SEU in your system should be the first step in deciding upon an SEU mitigation approach.

A conventional design mitigation technique for standard logic is the "majority vote" circuit. The functionality of a single flip-flop is implemented by three flip-flops in parallel. These flip-flops

feed a gating circuit the output of which reflects the state of the majority of the flip-flops. A typical majority vote circuit is shown in Figure 2.

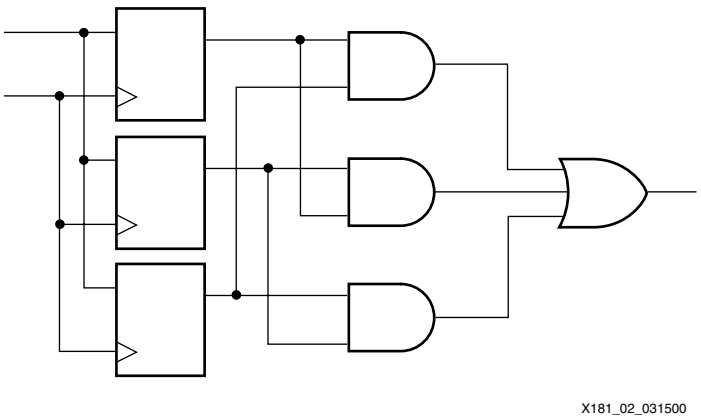

Figure 2: **Majority Vote Circuit**

Inherent in this technique is the assumption that only one SEU occurs within a given time period (i.e., the time it takes for the next clock edge to occur and load the flip-flops with new data). Obviously, if two of the flip-flops suffer contemporaneous upsets, then the majority vote circuit will give the state of the two incorrectly set flip-flops. The chance of this occurring, though, is usually considered statistically negligible, calculated by squaring the "normal" SEU rate (e.g., $[10^{-5}$ bit-upsets/day$]^2 = 10^{-10}$ uncorrected bit-upsets/day).

It is important to acknowledge that FPGA designs for space always come down to a determined acceptable amount of risk. Decreasing risk means increasing design complexity. The cost of the standard majority mitigation technique is obvious: the use of three times as many flip-flops. But with the abundance of resources available in Xilinx's line of rad-tolerant FPGAs, this cost would be tolerable in most cases.

However, there are many more latches in a Xilinx FPGA than those actually design-specified by the user as flip-flops; the majority of latches are in fact used for configuration memory. Because the configuration memory cells are just as susceptible to SEUs as are the design-specified flip-flops, the standard majority mitigation technique alone is not adequate to overcome the effects of SEUs in FPGAs.

Reconfigurability

One of the very notable features of Xilinx FPGAs is that they are reconfigurable, as opposed to *one-time programmable*. If a design change is necessary, then a new configuration can be loaded and the functionality of the FPGA altered without having to remove and discard the IC, as is the case with anti-fuse FPGAs. This also allows upgrades to be made in the field, or even in space. Unfortunately, this increased flexibility results in a more involved design solution for SEU effects. An understanding of how a Xilinx FPGAs configuration works is necessary before we can discuss the next level of SEU design mitigation techniques.

FPGA "First Floor and Basement" Architecture

Before power-up, a Xilinx FPGA is completely unconfigured. In other words, thousands of flip-flops and logic gates are residing in the IC, connected neither to one another nor to the I/O pins. As the power supply voltage rises and crosses a certain threshold, the FPGA begins to load its "brains" (configuration) and all I/O pins are set in a tri-state condition. The internal configuration clock becomes active and begins to clock data from the configuration data storage into the configuration latches. Buried in this configuration data stream are the items that make up a configured FPGA: logic function, I/O pin definition, clock distribution, flip-flops, routing, and so on. Once the configuration data is loaded and the CRC checksum is verified, the FPGA becomes active and the I/O pins begin to function as specified by the design.

Using a "house" analogy, if all the functions that the FPGA is to perform (logic, flip-flops, pins, etc.) are considered to be on the "first floor", then all the configuration latches are in the "basement". See Figure 3.

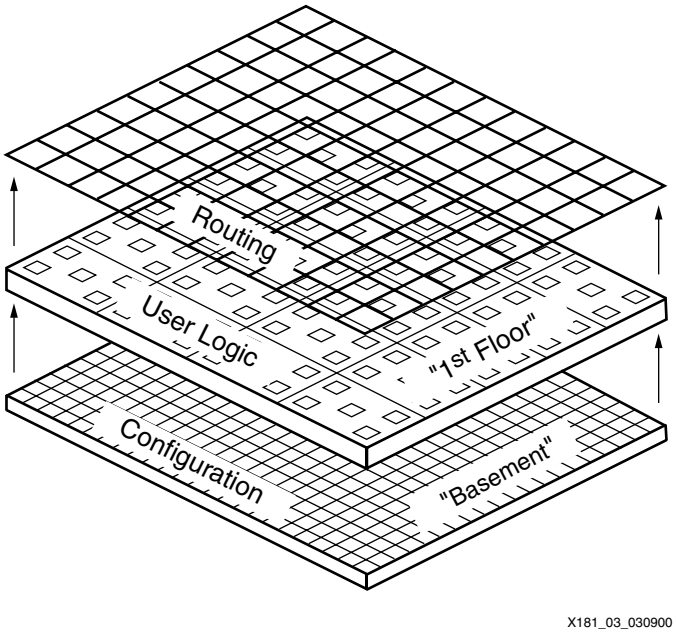

Figure 3: **FPGA Configuration Hierarchy**

As it turns out, the basement is necessarily much larger than the first floor. It takes approximately 30 configuration latches to configure each user-CLB, with each configuration latch controlling some specific property of the CLB or I/O block. The logic implemented in the look-up tables (LUTs) is one of the more important properties held in these latches. If a latch that configures an LUT experiences an upset, then the logic intended in the design may be altered. For example, it could be possible for a design-specified AND gate to become a NAND gate instead.

It should now become apparent that the majority vote circuit shown in Figure 2 is not reliable as an SEU mitigation technique, because the majority vote portion of the circuit can change its function in the event of an SEU occurring on a latch that controls the circuit. Therefore, some new methods of SEU mitigation are required.

Design Mitigation Techniques

FPGA designs are completed with varying degrees of risk based on the mitigation techniques employed. Since the amount of "acceptable risk" varies with the application, the design mitigation strategy employed will also vary. In some cases, it may be acceptable to do very little to accommodate SEUs; in other cases, the techniques may need to be rather sophisticated. The remainder of this Application Note will focus on various techniques for SEU mitigation. These techniques are listed in ascending complexity: auto-reconfiguration; using logic redundancy and an XOR gate for SEU detection; using the Xilinx "Readback" capability for SEU detection; using wired-AND outputs in conjunction with readback; and finally, building an SEU-safe system by combining these techniques.

Auto-Reconfiguration

The simplest approach to SEU mitigation is to reconfigure the FPGA upon detecting a system failure or at specified time intervals. For example, suppose an FPGA used to control a spacecraft heater experiences an SEU, causing the FPGA to improperly turn on the heater. If it can be determined through other spacecraft systems that the heater has been turned on, a command could be sent to restore the heater to its proper state and/or reconfigure the FPGA.

While this strategy may create an annoyance for the system, it might be seen as an acceptable approach in non-mission-critical applications where economy of design is paramount. If an application is of a more critical nature, however, then it may be imperative that the occurrence of an SEU be detected and specifically addressed.

XOR Gate for SEU Detection

One method for accomplishing this, shown in Figure 4, consists of adding a duplicate logic circuit to critical FPGA functions. One output drives whatever function the logic was designed to perform, while the output of the redundant circuit is used in conjunction with the primary output to drive the inputs of an XOR gate. If an SEU occurs which affects either circuit, the outputs of the logic will conflict and the XOR gate will output a "1" indicating that an SEU has occurred.

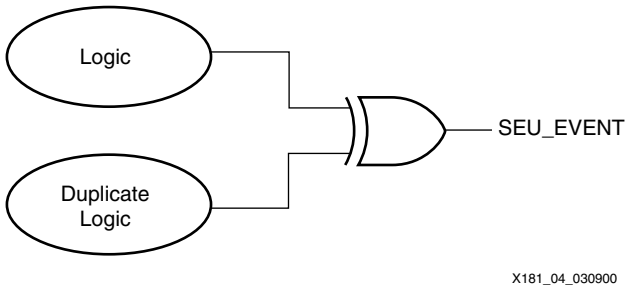

Figure 4: **SEU Detection by Redundancy and XOR Gating**

If there are several places where this method needs to be employed, the XOR outputs can all be ORed together to provide a single SEU status bit. This SEU status bit can also be used to drive the GTS (Global Tri-State) pin of the STARTUP component, causing all outputs to a high-impedance state in the event of an SEU occurrence.

Xilinx "Readback" Capability

Every Xilinx FPGA family incorporates a feature called Readback. Originally designed to facilitate testing during production of the ICs, it provides a non-intrusive method of reading the current state of every flip-flop and configuration memory cell within the FPGA. To make use of this feature, the "Readback" component needs to be instantiated in the design.

This function runs in the background, and in no way affects the performance of the FPGA. The design can run at full speed while simultaneously performing a readback. (See Xilinx Application Note XAPP015 "Using the XC4000 Readback Capability").

A CRC checksum based upon all the bits that have just been read back is generated and inserted at the end of the readback serial stream. This CRC checksum can be compared to the expected checksum for the current configuration; if it does not compare, then an SEU may have occurred.

During Readback, every bit that currently resides in each flip-flop along with every configuration bit is serially shifted out of the readback block. The output of the readback block can drive either an external pin or an internal signal. Readback of the XQR4000XL must be clocked out at a frequency between 1 MHz and 2 MHz. (Virtex™ is two orders of magnitude faster). The amount of time required to read back the FPGA varies on the size of the FPGA. For example, the XC4062XL contains 1,433,812 configuration bits. At a 1.5 MHz rate, it would take 960 ms to read back this FPGA.

There are three different ways to incorporate readback in a design. These are:

- Use a microcontroller or microprocessor to verify the checksum
- Use separate FPGAs to monitor one another
- Self-readback

Each method will be discussed in detail below.

> **NOTE:** If SelectRAM is to be used in the design, then a simple CRC check of the readback data will not work. This is because SelectRAM actually employs the configuration bits as storage elements. Therefore, if a RAM value has been changed, the configuration readback checksum will differ from the default value checksum. When incorporating SelectRAM in the design, therefore, readback should be used to perform a *full bit-for-bit verification* of the readback data (see Application Note XAPP015).

Microcontroller for Readback

The block diagram shown in Figure 5 illustrates a readback CRC compare function easily implemented using a microcontroller. The microcontroller simply extracts the checksum from the readback serial stream and then compares it to the expected value. The output of the circuit, SEU_EVENT, can be used to interrupt to the system's processor signaling the occurrence of an SEU. At the next "convenient" time, the FPGA should be commanded to reconfigure.

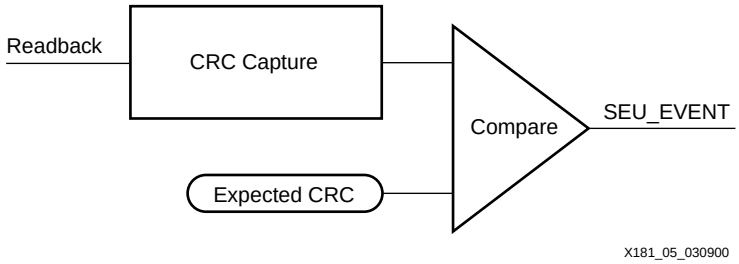

Figure 5: **Readback CRCComparator**

The CRC data is located in the last 11 bits of the readback stream. XAPP015 explains in greater detail the anatomy of the readback data; however, Table 1 summarizes the CRC locations for the XQR4000XL parts. The beginning of the readback stream is identified by a preamble consisting of five dummy "1s" followed by a "0". The amount of data between the preamble and the 11-bit CRC is device-dependent, as shown in Table 1.

Table 1: **Readback Datastream Size**

Device	Preamble	Data Stream	CRC (12 bits)
XQR4013XL	111110	<399,630 bits>	0<11 bits>
XQR4036XL	111110	<841,350 bits>	0<11 bits>
XQR4062XL	111110	<1,445,502 bits>	0<11 bits>

Using Separate FPGAs to Monitor One Another

If a design requires more than one FPGA, or multiple FPGAs are used as redundancy, then each FPGA can be used to monitor the readback serial stream of a neighboring FPGA. The CRC comparator shown in Figure 5 can easily be implemented in an FPGA. If an SEU is detected, one of two possibilities has occurred: Either the *FPGA* being monitored experienced an SEU, or the detection circuit in the monitoring FPGA itself experienced an SEU. The SEU_EVENT signal is used to alert the system that both FPGAs need to be reconfigured at the next opportunity.

Self-readback

Instead of having two or more FPGAs monitor one another's readback CRC, it is possible to use a single FPGA to monitor itself. Design redundancy is required, however, because an SEU can occur in the readback monitor circuit itself, thereby rendering its result invalid. A simple redundancy method involves creating two readback compare circuits in parallel and wire-ANDing the outputs. Simultaneous occurrence of CRC errors in both comparators would indicate an SEU in the configuration logic under test, rather than in one of the readback compare circuits. A block diagram of this technique is shown in Figure 6.

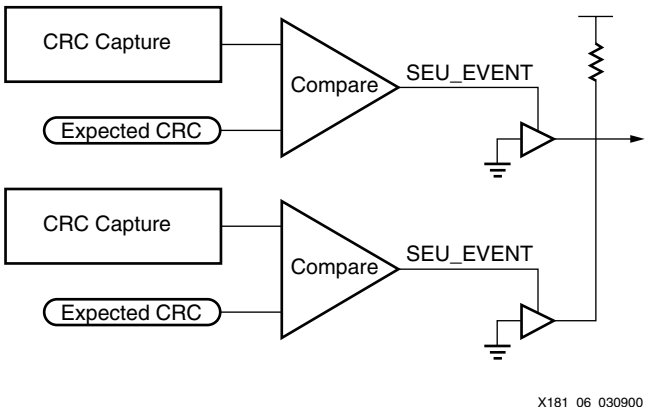

Figure 6: **Redundant CRC Comparator**

Wire-ANDed Outputs

Up to this point, we have focused on methods of detecting when a logic error caused by an SEU has already occurred. Some signals, however, are sufficiently mission-critical that an erroneous logic state on an output cannot be tolerated for any period of time. The technique of wire-ANDing redundant logic outputs can be employed to mitigate the effects of SEUs at this level of criticalness.

For example, suppose that the FPGA is being used to drive a pyrotechnic device that jettisons part of a spacecraft. In this example, it would be unacceptable for the signal output to remain

erroneous for the time required to complete a readback, detect that an SEU has occurred, and remediate the condition. Wire ANDing using redundant design logic only drives a mission-critical output to the active state when the two legs of redundant logic agree.

It is important to understand that this mitigation method does not ensure that a desired signal will be correctly asserted in spite of an SEU which occurs during the assertion function. It does, however, ensure that a signal will not be erroneously asserted due to an SEU.

The technique is shown in Figure 7.

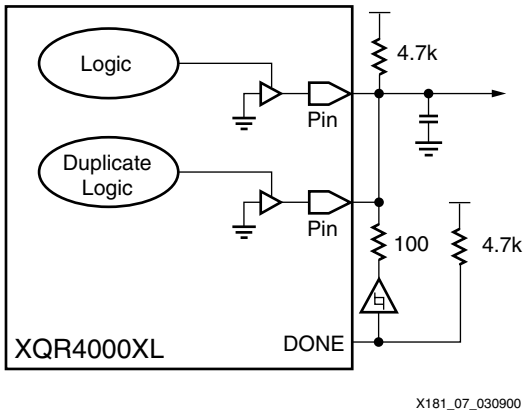

Figure 7: **Wire-ANDing Critical Outputs**

To drive an output High, both the primary and duplicate logic chains must direct their respective output buffers to a high-impedance condition. In this state, both logic outputs are high-impedance (looking back into the output pins), and the external pull-up resistor will pull the output signal High. If the logic chains do not agree, however, one or the other of the output buffers will be enabled, driving the wire-ANDed buffer output signal Low.

This technique is reliable for especially critical control signals, where one output state (logic High) is, by design, more meaningful than the other (logic Low). However, this approach is inappropriate for general data processing applications, where the output logic states are of equal importance and correct data propagation must be ensured.

Using the DONE signal to Control I/O Pins During Configuration

A precaution must be taken to ensure that the output of an unconfigured part is not interpreted as a true logic High. Since the FPGAs I/Os are in a high-impedance state before and during configuration, some other signal must hold the outputs Low during this time. The FPGAs DONE signal can be conveniently used to do this, since it drives Low during configuration. Since DONE will need to transition to High after configuration, an open-drain buffer should be placed between it and the outputs to be protected. (If many outputs are to be controlled in this fashion, additional buffers or relays may be added for each output pair.)

WARNING: *It is imperative that the bit-stream generation (BitGen) software start-up options specify that the I/Os are released before the DONE. Note that this is NOT the default!*

Reliable System Design

To be considered reliable, a system must process and propagate data correctly even in the event of an upset to the configuration and/or user logic. To build a reliable FPGA system, therefore, we must combine the techniques of SEU detection, correction and mitigation.

Whichever method of SEU detection is chosen (full verification or CRC checking), adequate SEU correction requires reconfiguring the FPGA, as the configuration logic and memory cannot be ruled out as being a possible cause of the error detected. The XQR4000XL, therefore, does constrain the designer in one significant respect: a temporary disruption in service must be tolerated when correcting detected upsets. Designers of systems that cannot

tolerate such a disruption should consider using the Virtex FPGA, which can be partially reconfigured without interruption.

Therefore, while an upset is present and being addressed, the logical functionality of the user design must be validated in some way so that incorrect data is not propagated through the system. The classical method for accomplishing this is Triple-Module Redundancy (TMR): that is, three identical FPGAs processing the same data in tandem, with the outputs mediated by an external voting circuit (Figure 2 on page 3).

TMR carries the further advantage that the entire FPGA may be used for the basic design, with no internal SEU mitigation techniques applied. However, since three duplicate FPGAs are required, it also carries the disadvantage of consuming significantly more board space and power. Where full TMR is deemed unsuitable by design economics or other considerations, the number of redundant FPGAs can be reduced from three to two by combining variations of the previously discussed techniques, provided the basic design (including duplicated logic) can be implemented within one FPGA.

Dual-voting Device Redundancy

A dual-voting system incorporates in just two FPGAs a fully redundant, self-mitigating system with built-in SEU detection and correction. The system, shown in Figure 8, is comprised of two FPGAs and a storage PROM.

The basic logic design is duplicated in each FPGA. The two FPGAs configure sequentially and then resynchronize. Corresponding output pairs are XORed, and then all XOR outputs are ORed together to drive the (GTS) pin of the STARTUP component.

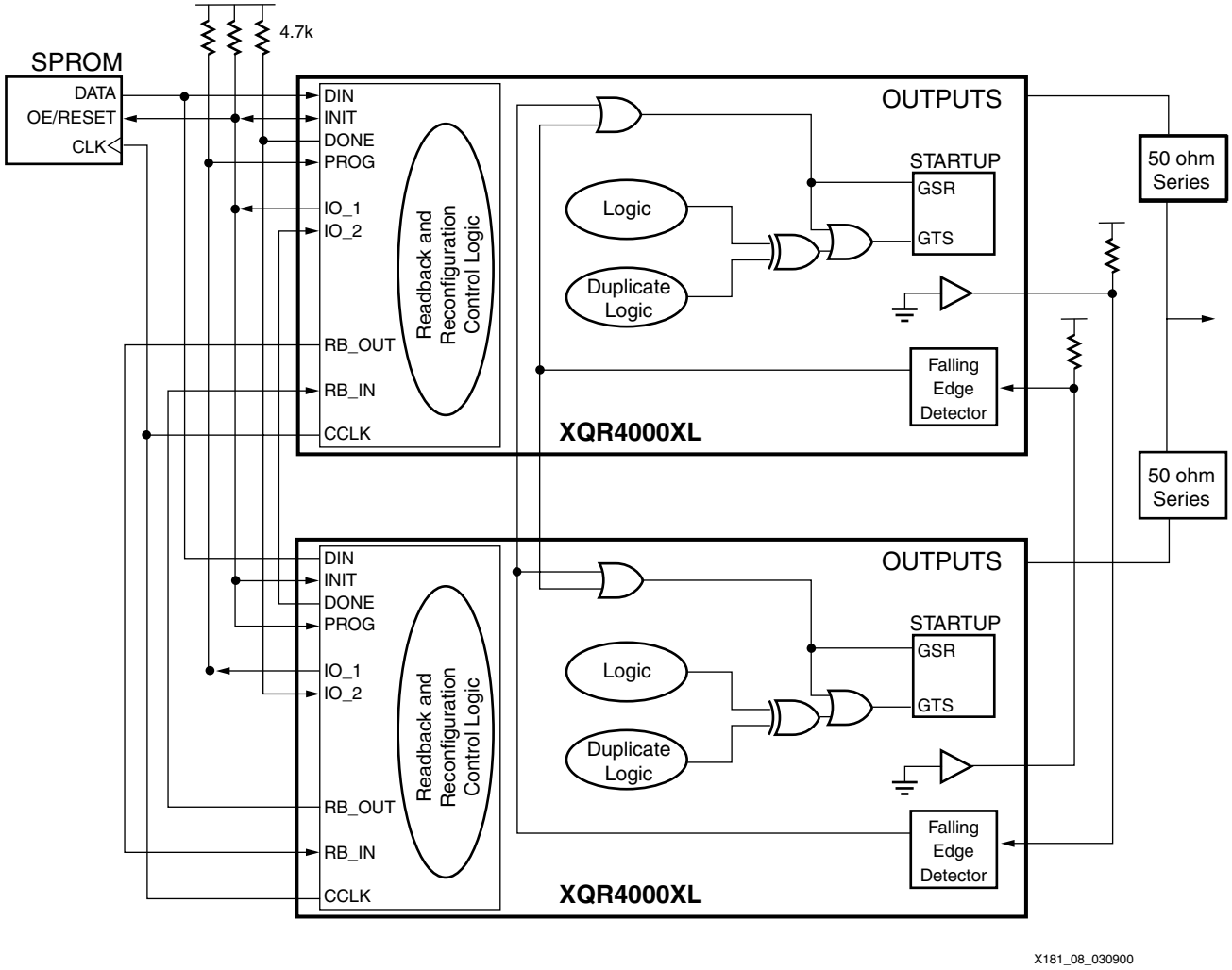

Figure 8: **Dual Redundant Self-mitigating FPGA Design**

If the occurrence of an SEU affects the function of the user logic, the compare circuitry will assert the GTS signal for that device. Asserting GTS causes all the I/O pins of the affected FPGA to a high-impedance state; however, the unaffected FPGA will continue to drive the correct data. If the SEU is merely transient (i.e., no configuration cells are upset), GTS will release when the redundant logic modules are resynchronized. (For complex designs an additional security measure may be added to time-out when one device has been off-line too long and issue a soft reset to both FPGAs to resynchronize the system).

To protect against the effects of an SEU occurring within the configuration memory cells, each FPGA should perform a constant readback on the other. When one FPGA detects that the other has been upset, it will force the upset FPGA to reconfigure. When the upset FPGA is reinitialized and resumes operation, it should notice that the other FPGA is already running, and should assert a soft reset (GSR) to both FPGAs to resynchronize the system. (The soft reset causes an unfortunate disruption of the system, but the interruption is less severe than it would be with less sophisticated SEU mitigation, as the system will still function while an upset FPGA is being reconfigured.)

The following sections describe the different aspects of this system in greater detail.

Power-on Configuration

Both FPGAs (top and bottom) shown in Figure 8 should be set for Master Serial Mode configuration (all mode pins tied Low M[2:0]<000>). The power-on configuration process executes according to the following steps:

1. Upon power-up, both FPGAs will drive their INIT pins Low until they are ready for configuration. Since they are in Master Mode, they will release their INIT pins and commence clocking the configuration data out of the serial PROM once their INIT pins have externally transitioned High. (This process can be delayed by holding INIT Low externally.)
2. The top FPGA will commence configuration first. The DONE pin of each FPGA is driven Low by each device until configuration is complete. Since the DONE pin of the top FPGA is connected to the INIT pin of the bottom FPGA, the bottom FPGA cannot commence configuration until the top FPGA has released its DONE pin upon completion of its own configuration.
3. When the top FPGA has completed configuration and has released its DONE pin, the bottom FPGA will attempt to commence configuration. However, in order for the bottom FPGA to successfully configure, both the PROM and the bottom FPGA must be reset by pulsing Low OE/$\overline{\text{RESET}}$ and PROG, respectively. This is accomplished with the IO_1 pin, which is controlled by user-defined logic and is described in "Auto-Reconfiguration" on page 4.

   NOTE: The IO_1 pin is a user-defined pin that may, if the user so chooses, co-exist on the same pin as INIT, a dual-function pin that becomes a user-programmable I/O (IOB) after configuration is complete. The IO_2 pin is also a user-defined I/O; it must be on a standard programmable I/O pin.
4. Upon configuration and activation, the top FPGA should sense that the DONE of the bottom device is Low on its IO_2 input, and subsequently pulse its IO_1 output Low for at least 300 ns. This will reset the serial PROM and force the bottom FPGA into reconfiguration
5. Upon completion of the bottom FPGAs configuration, the top FPGA's DONE should be observed High on the IO_2 input, and normal system operation will begin.

Top Level Design

As shown in Figure 8, the top level design consists of the user's basic design (logic); a duplicate of the basic design (duplicate logic); The STARTUP component (primitive); a constant Low output; a falling edge detector, and other random logic as shown; and a state machine to control the readback and auto-reconfiguration of the neighboring FPGA.

SEU Correction and Reconfiguration

The user must provide a small circuit within the top level design that will force the neighboring FPGA to reconfigure upon certain conditions. Those conditions should be:

1. The DONE of the neighboring FPGA is observed to be Low (IO_2).
2. A readback of the neighboring FPGA indicates that an upset in the configuration memory is present.
3. (Optional, not illustrated) The neighboring FPGA has held its outputs in a high-impedance state too long.

Condition 1 indicates a failed configuration or "deconfiguration", as well as controlling the Power-on Configuration sequence.

Condition 2 provides SEU correction when an SEU has been detected by readback.

Condition 3 is not illustrated in Figure 8. The basic concept is for each FPGA to be cognizant of the operational status of its neighbor FPGA. If the neighbor FPGA tri-states its pins because of a functional interrupt or effect other than an SEU to the configuration memory, *but does not seem to recover on its own,* then the system should be reset before such time has elapsed that would put the system in danger of both FPGAs being upset simultaneously. See "Optional Watch-Dog" on page 13.

If any of the above conditions occur, the FPGA should pulse the IO_1 output Low for 300 ns (min) to reconfigure the other FPGA.

The constant Low output, shown in Figure 8 as an output buffer (OBUF) tied Low, indicates whether the FPGA is online or off-line.

When an FPGA is configuring, all its outputs are in a high-impedance state. Therefore, the constant Low output will pull High indicating that the FPGA is off-line. When the FPGA is done configuring, the constant Low output will return Low.

The falling-edge detector in the active FPGA generates a pulse when the other FPGA comes back online. This pulse should be used to assert a global reset in the logic of both FPGAs. This will resynchronize all the logic of both FPGAs after one FPGA has been reconfigured, or when one FPGA has been momentarily off-line due to a transient interrupt. This is important, as it protects the hard-wired OUTPUTS from being in a state of contention.

The benefit of this practice is that the system will continue to function on one FPGA while the other is either upset or being reconfigured. However, the basic user's logic must be designed to tolerate unexpected global resets.

Readback and SEU Detection

As described in the section "Design Mitigation Techniques" on page 4, readback provides the method for detection of upsets in the configuration memory. The simplest approach is to capture the 11-bit CRC value at the end of the readback stream. See "Microcontroller for Readback" on page 6.

RB_IN and RB_OUT, shown in Figure 8, are arbitrary bus names for the readback interface and the direction of data flow between the devices.

The RB_OUT port provides external access to the READBACK primitive and consists of three separate pins (two inputs and one output). The two inputs are the readback trigger (RT) and the readback clock (CLK). These must be connected to the RT and CLK pins of the READBACK primitive (see Application Note XAPP015). The output signal is for the readback data which comes from the RD pin of the READBACK primitive.

The RB_IN port interfaces directly to the RB_OUT of the other FPGA, and thus has the same pins but in opposite direction (the clock and trigger are outputs and data is input).

The user must build the control logic for performing and capturing the readback. The process requires execution of the following steps:

1. To begin the readback, assert the RT High, and hold until readback is complete.
2. Clock continuously without interruption from the beginning to the end of readback. The clock signal MUST be between 1 MHz and 2 MHz.
3. Pipe the input readback data through a 6-bit decoder to watch for the "preamble" <111110> as shown in Table 1 on page 7.
4. When the preamble is observed, begin counting the number of clock cycles. When the count reaches the value shown in Table 1, the next bit should be a zero followed by the 11-bit CRC.
5. Compare this CRC to the expected CRC.

WARNING: The CRC of the very first readback after reconfiguration should be ignored. Only the CRC from the second (and subsequent) readback should be used.This is because the value of the expected CRC cannot be known prior to execution of a readback.

The readback control logic must be designed to do three consecutive readbacks in order to perform the first compare: the first to initialize; the second to capture the CRC; and the third to execute the compare. Each subsequent readback then results in an immediate compare. However, if the FPGA being read back is reconfigured, this process must start again from the beginning.

The CRC value captured from the second readback needs to be stored for comparison with succeeding readbacks. This can be done with registers, but should use triple module redundancy so that the wrong value is not used should one of the registers get upset.

In this case, it is acceptable to use LUTs for the voting circuit, because even if LUTs get upset, the system will eventually reconfigure and repair itself.

SEU Mitigation with STARTUP

The primary mitigation technique of this system is for the FPGA to turn off its outputs when a functional upset occurs. This is accomplished by duplicating the user's basic design and XORing output pairs. All XOR outputs should then be ORed together, along with the GSR signal, to drive the GTS. (The GSR is included in case the OR gate driving the GSR gets upset).

As mentioned earlier, the GTS signal, when asserted, will tri-state all FPGA outputs. This will keep incorrect data from propagating out into the system. The GSR and GTS of the STARTUP component are entirely asynchronous and hard-wired. Thus, do not depend on any storage elements or clock sources.

When neither device is upset, both sets of outputs will be driving. The 50 ohm series resistance (actual impedance should be specified by the designer) on each FPGA output provides impedance-matching to board traces to reduce reflections. In addition, the 100 ohm series resistance between output pairs absorbs transient contention caused by output transition skew.

Because the logic is already duplicated in each device, this mitigation approach provides an additional benefit by nicely supporting the wire-AND approach to critical control signals. See "Wire-ANDed Outputs" on page 7. Since the device itself is duplicated as well, a quadruple pin redundancy system actually results.

Combining these techniques creates a reconfigurable system that is reliable for even the most critical functions and applications.

Optional Watch-Dog

It is possible for an SEU to affect the functional operation of the design without upsetting any configuration memory latches (i.e., upsetting the stored value in a CLB flip-flop). Such an upset would not be detected by a readback, and thus would not induce a reconfiguration.

When a functional upset like this occurs, there will most likely be a discrepancy between the "Logic" and "Duplicate Logic" which will cause the FPGA outputs to a high-impedance state. Whether or not the FPGAs' design will eventually resynchronize without a reset depends entirely on the complexity of the design itself.

A simple pipelined arithmetic through-put function, such as a multiplier, will always resynchronize within the number of clock stages present between the upset flip-flop and the output. However, a highly complex state-machine may never recover. It is therefore left to the designer to determine if this is a possibility for the design in question.

If the possibility of a functionally upset design never recovering is of concern, then the designer should include a "watch-dog" timer to reset the system.

For this system the timer would be merely a counter that is clock-enabled by the constant Low output of the neighbor FPGA. When the neighbor FPGA tri-states its pins, the Low output will

pull high and thus cause the timer to start incrementing. When the timer has reached a "terminal count" value, it should pulse the GSR of both FPGAs.

It is left to the designer to determine the appropriate "terminal count" value for the application. For example, one application may require that the timer time-out before the next statistically expected upset. The time interval between upsets depends on the orbit and location. This may be a matter of seconds, minutes, hours, days, or years.

Summary

With the release of Xilinx radiation-tolerant FPGAs, engineers now have a more powerful and flexible option for programmable logic in space applications. While the techniques to mitigate the effects of SEUs are more complicated than those methods employed for older technology radiation-tolerant FPGAs, in many applications the benefits of Xilinx FPGAs are an overwhelming return for the additional design effort. These benefits include: higher density (up to 62K gates); significantly lower cost; in-circuit reprogrammability (ISP), allowing rapid changes with no rework or scrapping; and three densities utilizing the same footprint that adds to cost savings and makes room for design growth.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/15/00	1.0	Initial Xilinx release.