Correcting Single-Event Upsets Through Virtex Partial Configuration

Author: Carl Carmichael
Co-authors: Michael Caffrey, Anthony Salazar; Los Alamos National Laboratories

XAPP216 (v1.0) June 1, 2000

Summary

This application note describes the use of partial reconfiguration in Virtex™ series FPGAs for the purpose of correcting Single Event Upsets to the configuration memory array induced by cosmic rays. It is essential for the reader to have a basic understanding of the Virtex SelectMAP™ interface as well as configuration and readback operations. An in-depth review of Xilinx Application Note XAPP138 is highly recommended.

Overview

- SEUs are unavoidable and must be corrected
- Using Partial Configuration for SEU Correction
- SEU Correction Methods
- SEU Detection
- SEU Scrubbing
- Design Examples
- Application of Static and Dynamic Cross-sections
- Reference Tables

Introduction

On-orbit, space based, and extra-terrestrial applications must consider the effects high energy charged particles (radiation) may have on electronic components. In Particular, Single Event Upsets (SEU) may alter the logic state of any static memory element (latch, flip-flop, or RAM cell). Since the user-programmed functionality of an FPGA depends on the data stored in millions of configuration latches within the device, an SEU in the configuration memory array may have adverse effects on the expected functionality.

A static upset in the configuration memory is not synonymous with a functional error. Upsets may have no effect on functionality. Design mitigation techniques, such as triple redundancy, can harden functionality against single events upsets. However, the upsets must be corrected so that errors do not accumulate.

The Virtex Series FPGA SelectMAP interface provides post-configuration read/write access to the configuration memory array. "Readback" is a post-configuration read operation of the configuration memory, and "Partial Reconfiguration" is a post-configuration write operation to the configuration memory. Readback and Partial Reconfiguration allow a system to detect and repair SEUs in the configuration memory without disrupting its operations or completely reconfiguring the FPGA.

Before continuing with this application note it is essential for the reader to have a full understanding of the basic configuration and readback operations, as well as the bit-stream format and command structure, of the Virtex Series configuration logic and SelectMAP interface. A careful review of Xilinx Application Note XAPP138 "Virtex FPGA Series Configuration and Readback" will provide this information. For further reading on the Virtex Series FPGAs' configuration architecture, see Xilinx Application Note XAPP151 "Virtex Configuration Architecture Advanced Users' Guide."

Partial Reconfiguration

Configuration Memory Architecture

The configuration memory array is divided into three separate segments: The "CLB Frames", "BRAM0 Frames" and "BRAM1 Frames." See Figure 1. The two BRAM segments contain only the RAM content cells for the Block RAM elements. The BRAM segments are addressed separately from the CLB Array. Therefore, accessing the Block RAM content data requires a separate read or write operation. Read/Write operations to the BRAM segments should be avoided during post-configuration operations, as this may disrupt user operation.

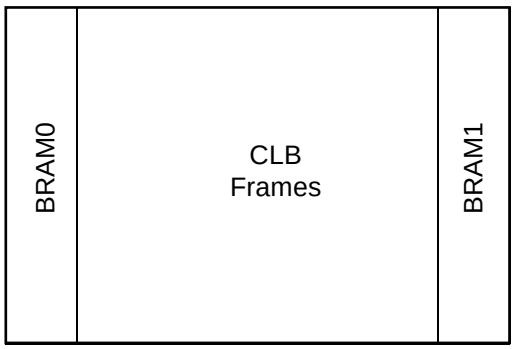

Figure 1: **Virtex Frame Segments**

The CLB Frames contain all configuration data for all programmable elements within the FPGA. This includes all Lookup Table (LUT) values, CLB, IOB, and BRAM control elements, and all interconnect control. Therefore, every programmable element within the FPGA can be addressed with a single read or write operation. All of these configuration latches can be accessed without any disruption to the functioning user design, as long as LUTs are not used as distributed RAM components.

While CLB flip-flops do have programmable features that are selected by configuration latches, the flip-flop registers themselves are separate from configuration latches and cannot be accessed through configuration. Therefore, readback and partial configuration will not effect the data stored in these registers.

However, when a LUT is used as either a distributed RAM element, or as a shift register function, the 16 configuration latches that normally only contain the static LUT values are now dynamic design elements in the user design. Therefore, the use of partial reconfiguration on a design that contains either LUT-RAM (i.e., RAM16X1S) or LUT-Shift-register (SRL16) components may have a disruptive effect on the user operation. For this reason the use of these components can not be supported for this type of operation.

However, Block RAMs (RAMB) may be used in such an application. Since all of the programmable control elements for the Block RAM are contained within the CLB Frames and the Block_RAM content is in separate frame segments, partial reconfiguration may be used without disrupting user operation of the Block RAM as design elements.

Data Frames

The configuration memory segments are further divided into columns of data frames. A data frame is the smallest portion of configuration data which may be read from, or written to, the configuration memory. The CLB array contains four categories of frame columns: one center column (eight frames), CLB columns (48 frames/column), two BRAM-Interconnect columns (27 frames/column), and two IOB columns (54 frames/column). The number of CLB columns and the size of the frames vary per device. However, the frame sizes are constant for a particular

device regardless of the column type in which it resides. The entire array may be addressed as one block, or alternatively any individual frame may be accessed as a unique block of data.

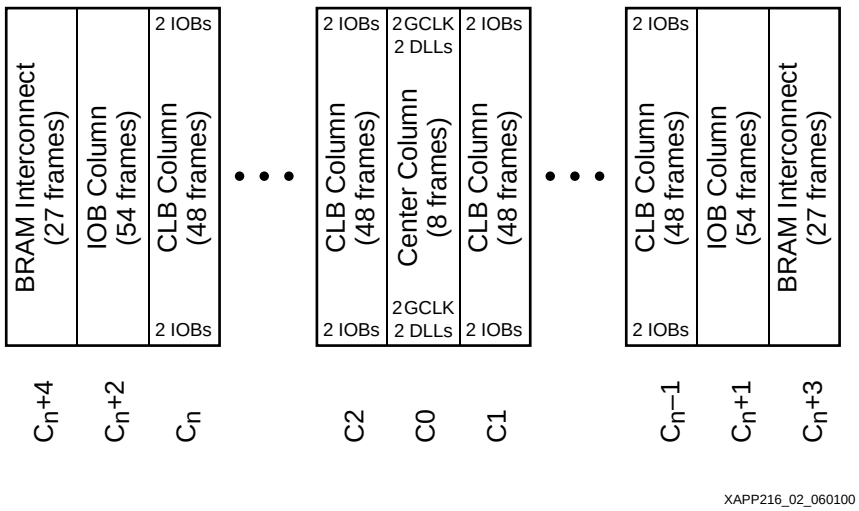

Figure 2: **CLB Frame Columns**

As shown in Figure 2, the frame columns are numbered in a "ping-pong" order which places all the even numbered columns to the left of the center column and all the odd numbered frames to the right. The frames within a column are numbered sequentially within that column away from the center. If all the frames were simply numbered sequentially in accordance with the

order of their appearance when performing a full readback of the CLB Frames, their order would be as shown in Figure 3.

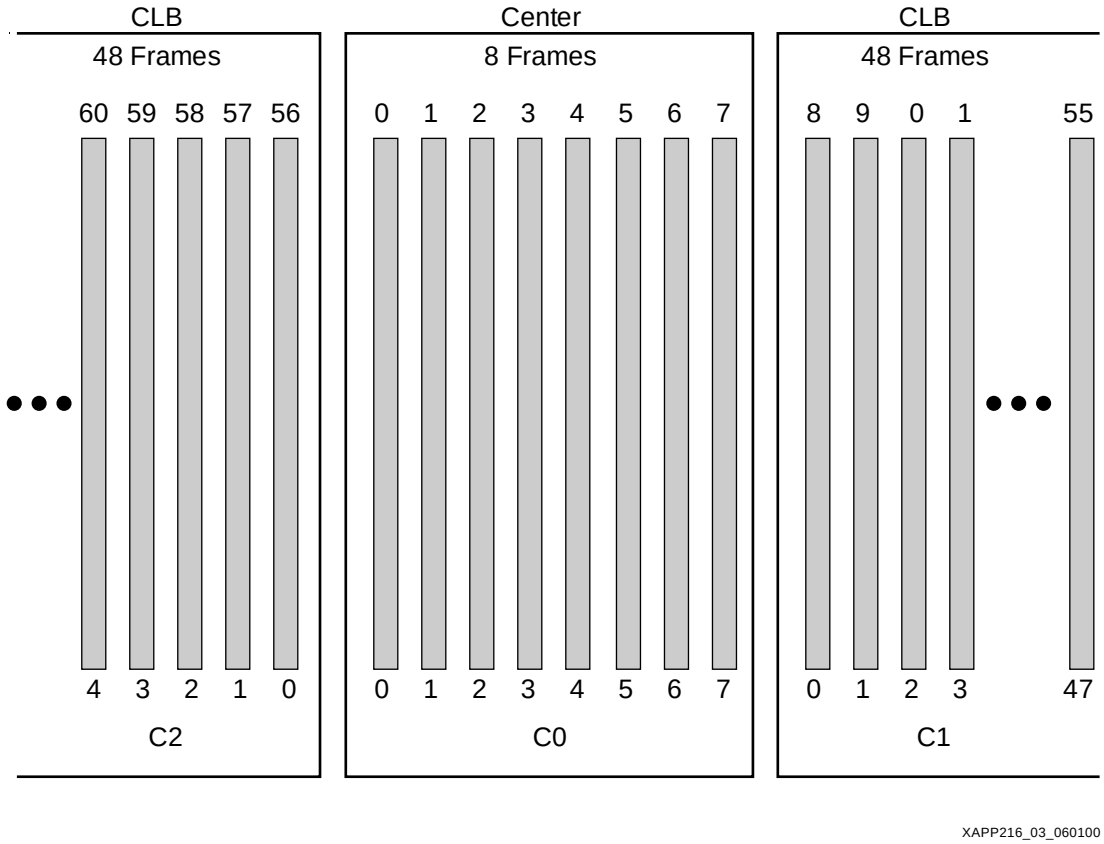

Figure 3: **Data Frame Numbering**

Partial Read/Write Operations

To write a series of data frames, the Frame Address Register (FAR) must first be set to the address of the first frame in the series. Then specify the number of data words to be written to the FDRI register followed by the data. A data-word is a 32-bit word. Therefore, the number of words to be written is the number of frames to be written times the number of words per frame (see Table 3 on page 12) plus one dummy word (typically all zeros) to follow each frame and plus one more frame of dummy words which also must be followed by a dummy word. If writing multiple frames, the first frame will be written to the address specified in the FAR and will automatically increment the address by one frame for each frame of data thereafter.

For each write operation the number of frame data words must also include a dummy word in order to complete the write operation. Data written to the FDRI register is assembled into 32-bit words and then loaded into a Frame register equal in size to one data frame. When the frame register is full the entire frame is loaded in parallel into the configuration memory latches. The last 32-bit word written is always stuck in the FDRI register. Therefore, a dummy word is needed to push the last word of the last frame of real data into the frame register in order for the entire last frame to be loaded into configuration memory.

A frame address is expressed as a major address and a minor address. The major address is the column number and the minor address is the frame number within that column. The value written to the FAR register contains a Block Type field, the major address, and the minor address. The Block Type should always be "00" to indicate the CLB Frames Segment. The Major Address is positioned in bits 17 through 24. The Minor Address is positioned in bits 9 through 16. All other bits should be "0". Therefore, to read or write the first frame of the first column, the value written to the FAR would be all zeros (00000000h).

SEU Correction Methods

SEU Detection and Single Frame Correction

One method of SEU correction is to use Readback to detect when an upset to the configuration memory has occurred. When an upset is detected only the data frame that contains the effected bit need be corrected. Using this method of writing only a single data frame, and only after an upset has occurred means that the configuration logic will be in "write mode" for the shortest amount of time. Most of the time the configuration logic will be in "read mode". This decreases the probability of an upset to the configuration logic itself from having any adverse effects to the configuration memory array. However, this method also requires some system overhead and support for the readback and detection of SEUs in the configuration memory.

Using readback for SEU detection requires a hardware implementation of algorithms for reading and evaluating each data frame. Additionally, memory space is needed to store constants and variables.

SEU Scrubbing

A simpler method to SEU correction is to omit readback and detection of SEUs and simply reload the entire CLB Frame segment at a chosen interval. This is called "scrubbing." Scrubbing requires substantially less overhead in the system, but does mean that the configuration logic is likely to be in "write mode" for a greater percentage of time. However, the cycle time for a complete scrub can be made relatively short as the SelectMAP interface is capable of operating at a throughput of 400 Mbits/s. Additionally, the chosen interval for scrub cycles should be based on the expected static upset rate for a given application or mission, and may be fairly infrequent. A longer cycle interval (time between scrubs) and shorter cycle time (scrub time) decreases the total percentage of time that the configuration logic is in "write mode."

SEU Detection

Readback and Comparison

The more traditional method of verification of the data stored in configuration memory is to readback the data and perform a bit for bit comparison. This requires the use of a mask file (.msk) and readback file (.rbb) each of which are equal in size to the original bit-stream used to configure the FPGA. This method is explained in detail in Application Note XAPP138.

This method would effectively triple the amount of system memory required for configuration and readback operations. Therefore, this method is not generally considered to be desirable for space applications.

CRC Frame Checks

Another method for readback verification and SEU detection was developed by the Los Alamos National Laboratories Space Data Systems Group. This method records a 16-bit CRC value for each data frame. During readback a new CRC value is generated for each data-frame that is read back and compared to the expected CRC result. Since a data-frame is the smallest amount of configuration memory which may be read from, or written to, the device, it is not important to know which data bit is upset but merely which data frame the upset exists in. Then only the data frame effected need be rewritten to the FPGA to correct the SEU.

This method greatly reduces the amount of system memory required to perform SEU Detection. The algorithm for calculating a CRC sum is presented in Application Note XAPP138.

SEU Correction

System Architecture

There are two different methods for implementing the CRC frame constants. For an application that will never require any update or changes to the FPGAs' design after deployment, the CRC constants for a specific FPGA design can be pregenerated in software and stored in system ROM. For applications that can accept updates for the FPGAs' bit-stream, the CRC constants

should be generated by the host system and stored in RAM. If the FPGAs' bitstream is ever updated then the CRC values can be refreshed.

Figure 4 shows a basic overview of one possible implementation of this system. The basic sub-blocks represent either logic or algorithms to interface with the Virtex SelectMAP Port, interface with the memory components, calculate and compare CRC values, and some sort of finite state machine to control the operations. The design details are left for the user to implement; however, an example design will be published by Los Alamos National Labs and posted as an addendum to this application note.

The mapping of the memory components should be done uniquely for each system. One possible method would be to store the CRC values in addresses such that the address number itself corresponds to the Frame number that the CRC value represents. This could reduce the number of processing steps, or decode logic, to access a specific CRC frame constant.

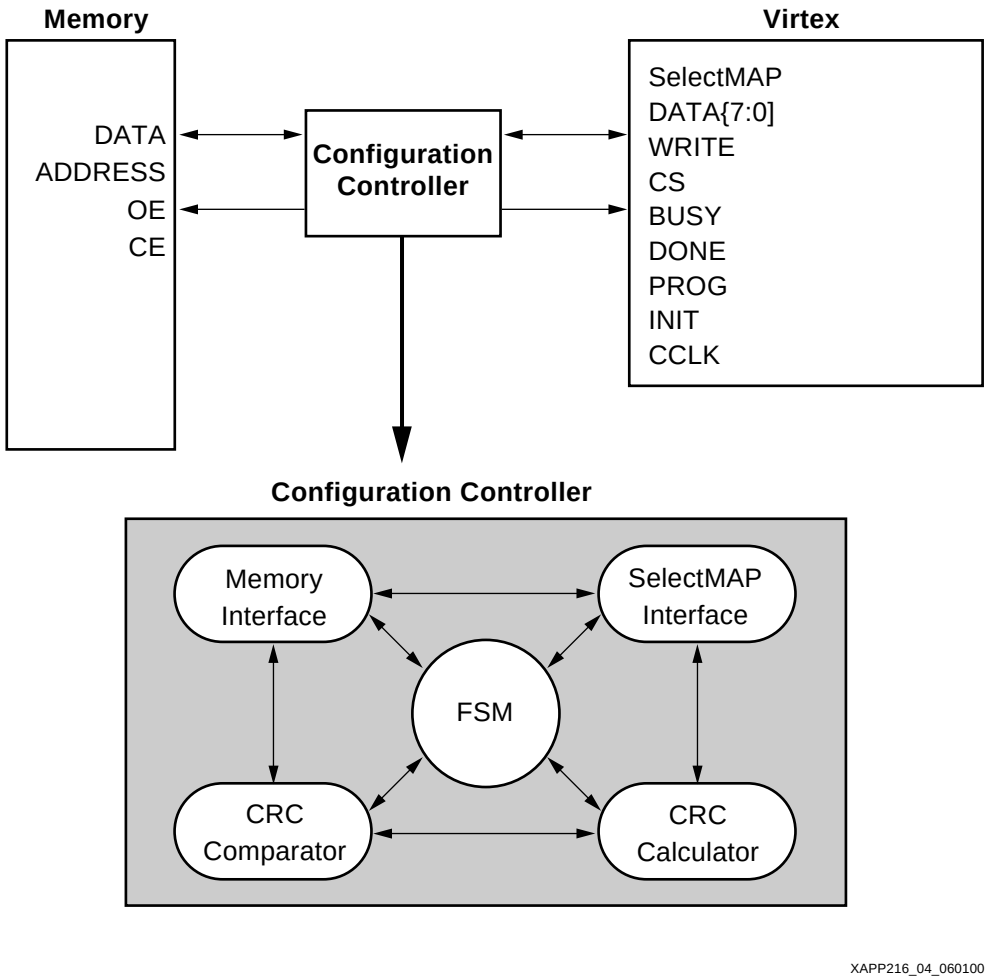

Figure 4: **Simple Configuration and SEU Correction Design**

Single Frame Correction

The process for configuration, readback and CRC calculation are omitted because these are explained in XAPP138. Whenever a data frame produces a CRC value that differs from its corresponding CRC frame constant stored in memory, the frame number should be stored for use after the readback cycle is complete. Although it is very unlikely to have more than one frame containing an SEU within one readback cycle, the CRC mismatch could potentially be produced by an SEU elsewhere in the system and not actually in the readback data. Therefore, the system should be designed to record multiple frame numbers for the correction cycle.

If the readback cycle did produce some CRC mismatches then the data for the stored frame numbers must be accessed from memory and reloaded into the FPGA. The procedure for a single frame write cycle follows:

1. **Abort**

   An Abort command is issued by holding the CS Low and the WR High for at least three clock cycles. This will reset the SelectMAP and configuration logic so that the interface may be re-synchronized. This alleviates tracking the number of clock cycles between readback and write cycles and clears any errors caused by an SEU in the configuration logic itself.

2. **Synchronize**

   Before a new process can commence the SelectMAP interface must be resynchronized by reloading the Synchronization Word.

3. **Issue WCFG Command to CMD Register**

   Enable write access to the configuration memory array by loading the WCFG command into the CMD register.

4. **Load FAR**

   Specify the frame address in the FAR with a major and minor address location. See "Frame Address Register" on page 8.

5. **Access FDRI Register**

   Use a Type 1 packet header to issue a write command to the FDRI register specifying the frame data length in 32-bit words plus one 32-bit dummy word.

6. **Load Frame Data**

   Load the data frame into the FPGA followed by one dummy frame. Each frame must be followed by a dummy word; However, the bitstream includes these dummy words at the end of each data frame.

7. **Reset CRC**

   Issue a RCRC command to the CMD register to clear the CRC register.

8. **Abort**

   Although a second Abort command may be superfluous, a resetting of the SelectMAP interface and subsequent resynchronization for any new process increases the likelihood that the process will be successful.

The data fields for the previous commands, except for the frame data, is shown in Table 1. The Abort command does not have any associated data.

Table 1: **Instruction Set for Single Frame Write Operation**

Command		Data (32 Bits)
Synchronize		AA 99 55 66
Write to CMD		30 00 80 01
WCFG		00 00 00 01
Write FAR		30 00 20 01
Frame Address		0? ?? ?? 00
Write FDRI	XQVR300	30 00 40 2A
	XQVR600	30 00 40 3C
	XQVR1000	30 00 40 4E
Frame Data		
Write CMD		30 00 80 01
RCRC		00 00 00 07

Frame Address Register

The simplest method for determining the frame address for the frame which needs to be reloaded is to count the frames during readback, starting with zero but not counting the dummy frame (see XAPP138), and then calculate the address based on that frame number.

The algorithm for calculating the frame address from the frame number needs to be conditional on which column type the frame comes from because different column types have a different number of frames and because of their organization (refer back to "Data Frames" on page 2).

Following is a description of the variables used in the subsequent equations and conditions.

N = Frame Number

Cols = The number of CLB columns in the device

Maj = Major Address portion of the FAR

Min = Minor Address portion of the FAR

DIV = Integer Division operation

MOD = Modulus remainder operation

The following algorithm is used to determine the Major and MInor Frame Address from a specific frame number and will be followed by an example exercise. It should be obvious that these conditions and equations would be greatly simplified if they were rewritten for a single device size, removing the number of columns as a variable. All variables are represented as decimal values and subsequently will need to be converted to hexidecimal before obtaining the actual FAR code.

```
Begin
   IF (0 ≤ N ≤ 7) Then {Frame is in Center Column}
      Maj = 0;
      Min = N;
   ElsIF (8 ≤ N ≤ [Cols x 48 + 7]) Then {Frame is in CLB Columns}
      Maj = (N-8)DIV(48) + 1;
      Min = (N-8)MOD(48);
   ElsIF ([Cols x 48 + 8] ≤ N ≤ [Cols x 48 + 115]) Then {Frame is in
   IOB Columns}
      Maj = (N-Colsx48-8)DIV(54) + Cols + 1;
      Min = (N-Colsx48-8)MOD(54);
   ElsIF ([Colsx48+116] ≤ N ≤ [Colsx48+169]) Then {Frame is in BRAM
   Interconnect}
      Maj = (N-Colsx48-116)DIV(27) + Cols + 3;
      Min = (N-Colsx48-116)MOD(27);
   End IF;
End;
```

Example

In this example the target device is an XQVR300. Therefore, the Cols=48. If the frame that needs to be corrected is the 2373rd valid data frame that was read back (not counting the dummy frame), then counting from zero, the frame number is N=2372.

```
Colsx48 = 48x48 = 2304 and N-2304 = 2372 - 2304 = 68;
N satisfies the third condition: 2312 ≤ N ≤ 2419; Therefore,
Maj = (N-Colsx48-8)DIV(54) + Cols + 1 = (60)DIV(54) + 49 = 50;
```

And

Min = (N-Colsx48-8)MOD(54) = (60)MOD(54) = 6;

Converting these to 8-bit Binay values gives the following major and minor addresses:

Major: 00110010; Minor: 00000110;

Inserting the Major Address into bits 17 through 24, the Minor Address into bits 9 through 16, and placing zeros in all other positions gives an FAR value of:

FAR(31:0) =0000 0000 0110 0100 0000 1100 0000 0000b = 00 64 0C 00h;

SEU Scrubbing

Scrub Data and Flow

Scrubbing is a much simpler approach to SEU correction because it does not require any readback or data verification operations, nor does it require any data generation when reloading the data frames.

In short, the process is to reload the bit-stream starting at the beginning, but stopping at the end of the first write to the FDRI register. In a standard bit-stream the first write to the FDRI register includes all the configuration data for the CLB Frames segment of the memory map. The rest of the bit-stream contains the BRAM segments, a CRC check, and the start-up sequence, all of which are not applicable to partial reconfiguration. No adjustments to the data or headers are needed.

A scrub cycle should be preceded and followed by an Abort operation. However, the Abort operation preceding the scrub cycles may be omitted if one Abort cycle is inserted after the completion of the initial configuration of the FPGA. The bitstream already contains the synchronization word at the beginning. The only support circuitry necessary is a counter to generate memory addresses (if necessary), and decode logic to toggle the control signals of the memory and SelectMAP interface at specific count values. This is the mechanism that controls how much of the bitstream is loaded. An example of this is shown in Figure 5. This example does not account for the initial configuration of the FPGA. However, the necessary additions to perform an initial configuration before the first scrub is fairly straight forward and left to the reader. For additional reading, XAPP137 provides a design example of using an interface logic device to configure a Virtex FPGA from a parallel memory source. If the FPGA is to be configured from a Serial PROM, then adding a serial-parallel converter to the interface would allow the serial prom to act as the data source for both configuration and scrubbing.

Since Scrubbing reloads the majority of the bitstream from the beginning, randomly accessible memory is not required.

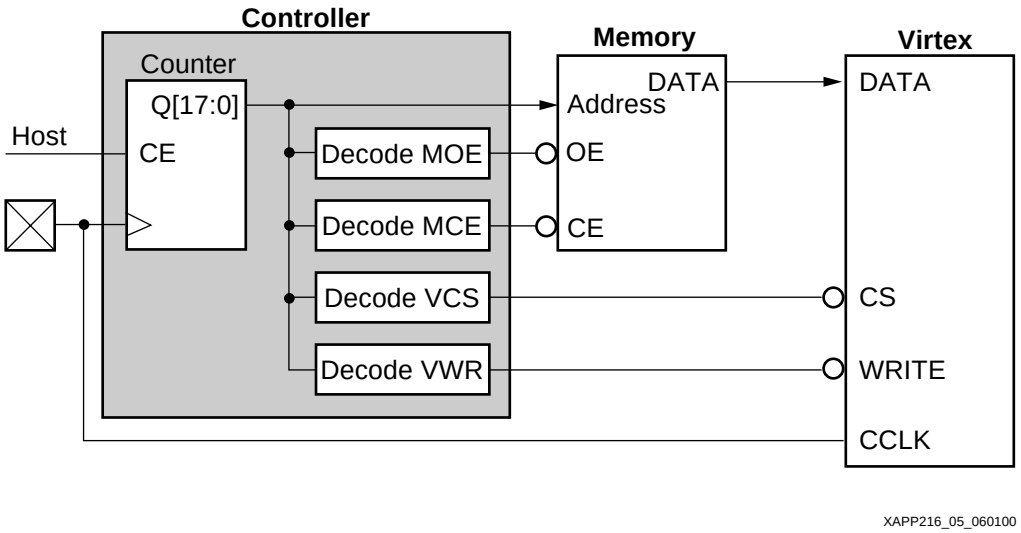

Figure 5: **Scrubbing Control Circuit**

The example shown in Figure 5 demonstrates the use of a parallel (8-bit wide) memory device. This allows the data signals to be connected directly from the memory to the Virtex SelectMAP data pins. If the memory's data ports are of any other configuration then the data should be reorganized into 8-bit words within the control chip.

For this example a simple counter is a sufficient state machine to control the scrubbing operations. The LSB outputs of the counter (number depends on the size of the memory) may be used as the address for the memory module. The example uses an 18-bit counter because this is the minimum value for a V300 bit-stream. A V600 or V1000 would require a larger counter. Additionally, the system clock may be too fast for the configuration interface (50 MHz max). In which case the address lines could be shifted to higher order bits of the count value leaving the lower order bits to serve as a clock divider.

There are four signals that need to be decoded from the Counter: MOE (Memory Output Enable), MCE (Memory Chip Enable), VCS (Virtex Chip Select), and VWR (Virtex Write). The complexity of these decoders and their associated values depends on how many Memory chips and FPGAs are being designed into the system. Since this is an entirely application specific variable we will simplify this example further by assuming a single memory chip and a single FPGA.

If the system had several memory chips, each memory would require its own MCE decoder. However, for one memory the MCE may be eliminated altogether and tied to the MOE decoder. The MOE must disable the memory's output during an Abort sequence. However, the VCS and VWR may not be combined, even for a single FPGA implementation, because the Abort sequence requires separate control of these signals.

Table 2 shows the state transitions for a complete scrubbing operation, including a trailing Abort sequence, and the associated clock cycles for each state. One clock cycle represents

one byte of data transferred. If the Counter is to be used as a Configuration Clock (CCLK) divider as well, then the number of clock transitions would need to be multiplied by the Divisor.

Table 2: **Scrubbing State Transitions**

States					Clock Cycles		
Type	**MOE**	**MCE**	**VCS**	**VWR**	**XQVR300**	**XQVR600**	**XQVR1000**
Load	L	L	L	L	207,972	435,312	745,596
Abort	H	H	L	H	4		
Disable	H	H	H	H	1		

Note: The clock cycles specified for the load operation are based on the bitstream format generated by the bitgen utility version 2.1i. If using any other version then these numbers should be manually verified in the bitstream.

The system also needs some sort of mechanism to control how often a scrub cycle takes place. In Figure 5 this is shown simply as a connection from the Host System to the CE input of the counter. Consideration is also needed for a reset control to the counter. If the desired time between scrub cycles is constant, then this could be automated by using another counter to control the CE of the scrub counter and another decoder to control a synchronous reset of the counters. Choosing how long to wait between scrub cycles (Scrub Rate) should be determined primarily from the expected upset rate for the specific application, orbit or mission.

Scrub Rates

A Scrub Rate describes how often a scrub cycle should occur. It may be denoted by either a unit of time between scrubs, or a percentage (scrub cycle time divided by the time between scrubs). The scrub rate should be determined by the expected upset rate of the device for the given application.

Upset rates are calculated from the Static Bit Cross Section (see Data Sheet) of the device and the charged particle flux the application or mission is expected to endure. For other technologies, the upset rate is an indication of how often the system will have to tolerate a functional bit error. But this is not precisely the case for an FPGA.

The static cross-section for a given device is derived by determining the cross-section per bit (obtained through experimentation and measurement) multiplied by the number of bits in the device. The static cross-section for a Virtex Series FPGA may be orders of magnitude higher than what the experienced space applications designer might be used to. This is because of the high density of configuration latches. But this upset rate does not carry the same meaning as it does for other technologies.

For example, lets compare a 6,000 flip-flop ASIC to a 6,000 flip-flop Virtex Series FPGA. If the ASIC and the FPGA have similar process geometries, then the static cross-section per bit will be similar for both devices. However, the device cross-section is the bit cross-section multiplied by the number of bits in the device. For a 6000 flip-flop ASIC the number of bits is 6000, but the a Virtex FPGA this number is 6000 plus 1.7 Million (approximately).

However, for an ASIC, a bit upset is considered to be a definite functional bit error. This would be an incorrect assumption for an FPGA. An upset in the configuration memory may or may not have any effect on the functional integrity of the user's design in the FPGA.

Design techniques may be applied to strengthen the functional integrity of the user design and protect it from the effect of any Single Event Upset. This process is called "SEU Mitigation." These design techniques are described in Xilinx Application Note XAPP186: "Space Application Design Techniques for the Virtex QPRO™ Radiation Hardened Series FPGA."

Where systems that include ASIC technology use a static upset rate to determine how often a functional bit failure may be expected, systems that use Virtex Series FPGAs should define a "Dynamic Upset Rate" for this purpose. The application of a dynamic upset rate is discussed in

the previously mentioned application note and is not covered in this paper. However, the necessary assumption is that the scrub rate should be set such that any SEU on the configuration memory will be fixed before the next will occur. Additionally, the life span of an SEU, time between the occurrence of the upset and it's subsequent correction, should be minimized. It is entirely up to the designer to choose the scrub rate. However, a good "rule of thumb" is to place the scrub rate at one order of magnitude from the upset rate. In other words, the system should scrub, on average, ten times between upsets.

For example, if we were to assume a bit upset rate of once per hour and a configuration clock frequency of 10 MHz, then the scrub rate should be once every six minutes. Thus, the scrub time, for a V1000 is 80 ms. Therefore, the scrub rate as a percentage would be 0.2%. Meanwhile, the FPGA will be capable of carrying out it's operations and functioning normally. It's ability to do so is a function of the design methodologies and mitigation strategies employed in the system.

Reference Tables

Table 3: **Device Statistics and Static Elements**

Devices	XQVR300	XQVR600	XQVR1000
CLB Array Size (RowxCol)	32 x 48	48 x 72	64 x 96
CLB Flip Flops	6,144	13,824	24,576
Select Block RAM (bits)	65,536	98,304	131,072
Frames	2474	3626	4778
Words (32-bit) per Frame (Including one dummy word)	21	30	39
Configuration Latches	1,583,360	3,364,928	5,810,048

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
06/01/00	1.0	Initial Xilinx release.