GRSCRUB – External FPGA Configuration Supervisor

Application Note

Doc. No GRLIB-AN-0012

Issue 1.0

2020-06-12

Contract N/A

Delivery N/A

CHANGE RECORD

Issue	Date	Section / Page	Description
1.0	2020-06-12		First issue of this document.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Scope of the Document

This application note presents the GRSCRUB IP functionalities and describes the system integration targeting a Xilinx Kintex UltraScale FPGA embedded in an ADA-SDEV-KIT2 development board.

The work has been performed by Cobham Gaisler AB, Göteborg, Sweden.

1.2 Reference Documents

The following documents are referred as they contain relevant information:

[RD1]	J. Heiner *et al.,* "Fault Tolerant ICAP Controller for High-Reliable Internal Scrubbing," *2008 IEEE Aerospace Conference*, Big Sky, MT, 2008, pp. 1-10.
[RD2]	F. Brosser *et al.,* "Assessing scrubbing techniques for Xilinx SRAM-based FPGAs in space applications," 2014 International Conference on Field-Programmable Technology (FPT), Shanghai, 2014, pp. 296-299.
[RD3]	A. Stoddard *et al.,* "A Hybrid Approach to FPGA Configuration Scrubbing," in IEEE TNS, vol 64, no 1, pp 497-503, Jan 2017.
[RD4]	Xilinx, "Soft Error Mitigation Controller," v4.1 LogiCORE IP Product Guide, Vivado Design Suite, PG036, Apr. 2018.
[RD5]	C. Gaisler, GRLIB IP Core User's Manual, Version 2020.1, Mar. 2020.
[RD6]	D. S. Lee *et al.,* "An Analysis of High-Current Events Observed on Xilinx 7-Series and Ultrascale Field-Programmable Gate Arrays," IEEE Rad. Effects Data Workshop (REDW), Portland, OR, USA, 2016, pp. 1-5.
[RD7]	M. Berg *et al.,* "Effectiveness of Internal Versus External SEU Scrubbing Mitigation Strategies in a Xilinx FPGA: Design, Test, and Analysis," in IEEE TNS, vol. 55, no. 4, pp. 2259-2266, Aug. 2008.
[RD8]	Xilinx, "KCU105 Board User Guide," UG917 (v1.10), Feb 2019.
[RD9]	C. Gaisler, "GRMON3 User's Manual," GRMON3-UM (Version 3.2.2), Mar. 2020.
[RD10]	AlphaData, "ADM-SDEV-BASE/XCKU060 User Manual," V1.4, 2020.
[RD11]	Xilinx, "UltraScale Architecture Configuration," UG570 (v1.12), March 31, 2020.
[RD12]	Imec, "Mixed-Signal ASICs for Harsh Environments," 2019 [Online]. Available: http://dare.imec-int.com/.

2 ABBREVIATIONS

FPGA	Field Programmable Gate Arrays
SEE	Single Event Effects
CLB	Configurable Logic Blocks
I/O	Input and Output
SEU	Single Event Upsets
SET	Single Events Transients
SDC	Silent Data Corruptions
SEFI	Single Event Functional Interrupt
SEM-IP	Soft Error Mitigation Intellectual Property
IP	Intellectual Property
BRAM	Block RAMs
FF	Flip-Flops
ECC	Error Correction Code
EDAC	Error Detection And Correction
CRC	Cyclic Redundancy Check
FFC	Full Frame Check
SMAP	SelectMap
GRLIB	Cobham Gaisler's IP library
FAR	Frame Address Register
DUT	Device Under Test
RF	Register File
IU	Integer Unit
AMBA	Advanced Microcontroller Bus Architecture
AHB	AMBA High-performance Bus
APB	Advanced Peripheral Bus
AXI	Advanced eXtensible Interface
CMOS	Complementary Metal Oxide Semiconductor

3 SOFT ERROR MITIGATION IN SRAM-BASED FPGAS

Radiation-induced soft errors are errors provoked by ionized particles that affect the system without damaging the device permanently. Even the space-grade Field Programmable Gate Arrays (FPGA) are susceptible to Single Event Effects (SEE) that may affect not only the user data but also the configuration memory of the device. SRAM-based FPGAs are particularly susceptible to soft errors due to the memory elements used to configure the design logic and architecture.

The FPGA configuration memory defines the Configurable Logic Blocks (CLB), Input and Output (I/O) interconnections, and clock lines, for instance. Single Event Upsets (SEU) affecting such elements may lead to persistent errors in the system, changing the architectural implementation of the design. The Single Events Transients (SET) are transient pulses that propagate through the combinational logic and may be captured by a memory cell, changing the storage data. Soft errors can also directly affect the memory data, registers, and flip-flops, and cause Silent Data Corruptions (SDC), which are incorrect results outputs. The Single Event Functional Interrupt (SEFI) occurs when a soft error affects the control logic or a state register and leads to hangs or crashes in the design.

The configuration memory of the Xilinx SRAM-based FPGAs is organized in frames, and each frame contains data divided into 32-bit words. Xilinx defines the configuration memory bits as non-essential, essential, and critical, and such characterization is dependent on the implemented design. Non-essential bits are related to the unused area of the FPGA configuration memory. The essential bits are the configuration bits that define the design, and soft errors in such bits modify the circuitry, which might or might not affect the design functionality. When the bit upset affects the function of the design, such a bit is defined as critical. Therefore, soft errors in critical bits are the most damage to the system since the design is directly corrupted. Another significant cause of design failure is the accumulation of upsets in the essential bits. The higher the number of bit-flips in the configuration memory, the higher the probability of the circuitry changes affect the design.

Scrubbing is a well-known technique responsible for coping with errors in the configuration memory and avoiding their build-up. Scrubbing can be defined as internal when the scrubber engine is embedded in the target FPGA being monitored, and external when the scrubber controller is located externally to the target FPGA in a different component. The literature presents several scrubbing implementations that mainly differ in the error detection, power consumption, resource usage, and correction speed [RD1, RD2, RD3]. A well-known internal scrubbing core is the Xilinx Soft Error Mitigation Intellectual Property (SEM-IP) [RD4] that is compatible with most of Xilinx FPGAs.

The memory elements that store dynamic data, such as Block RAMs (BRAM), distributed memory, and Flip-Flops (FF), are not protected by the scrubbing technique. Soft errors affecting the dynamic elements can be mitigated by applying fault tolerance techniques such as redundancy or Error Correction Code (ECC). Triplicating logic is an efficient method to cope with the effects of single faults in the design. Additional user level techniques can also be applied to deal with SDCs. Moreover, a periodic reset may be required to reestablish the system state and restore the initial state of flip-flops. Since SEFIs may also affect internal control elements of the FPGA or the configuration interface, a complete power cycle might be required to restore the system.

4 GRSCRUB IP - FPGA CONFIGURATION SUPERVISOR

The Cobham Gaisler's GRSCRUB IP core is an external FPGA configuration supervisor that features programming and scrubbing capabilities, which prevents the accumulation of errors in the configuration memory of SRAM-based FPGAs. The GRSCRUB IP targets soft errors affecting the FPGA configuration memory, and it is able to detect and correct single and multiple errors. However, one must notice that the GRSCRUB IP does not avoid bit-flips from happening or its effects on the design, as well as other scrubbers. Therefore, additional mitigation techniques at design level are recommended to decrease the number of single points of failure in the system and increase the fault masking, such as the ones described in Section 3.

The GRSCRUB IP is currently compatible with the Kintex UltraScale and Virtex-5 Xilinx FPGA families. It accesses the FPGA configuration memory externally through the SelectMap (SMAP) interface, which provides better performance in comparison with JTAG, due to the parallel data access. The GRSCRUB is part of the Cobham Gaisler's IP library (GRLIB) [RD5]. Moreover, the GRSCRUB IP will be integrated into the next version of the GR716 Microcontroller (GR716B), which is a mixed-signal fault-tolerant microcontroller based on the LEON3FT SPARC V8 processor. After the initial configuration, the GRSCRUB IP is self-standing, which releases the processor core or the primary system to perform other tasks.

4.1 GRSCRUB IP operation modes

The GRSCRUB IP implements five operation modes:

1) **Idle mode:** the IP is in idle waiting for an operation command.
2) **Programming mode:** the IP programs the configuration bitstream into the target FPGA.
3) **Scrubbing mode:** the IP executes a scrubbing operation. As described further, two scrubbing methods are supported: blind and readback scrubbing. The IP can be configured to scrub the entire FPGA configuration memory or just selected frames.
4) **Mapping mode:** the IP identifies and maps the frame addressing of the target FPGA. The frame addressing defines the frames positioning in the target FPGA, required for any scrubbing operation. Only frames that refer to configuration blocks are mapped, i.e., the memory block frames are not considered. The frame addresses are saved in the Golden memory and are accessed by the IP in scrubbing mode during reading and writing operations.
5) **Golden Cyclic Redundancy Check (CRC) mode:** the IP computes the golden CRC codes for the current frame data of the target FPGA configuration memory. The CRC code can be selected as a data check in the readback scrubbing mode. A CRC code is computed to each frame of the configuration memory, and it is verified against the golden CRC copy.

The GRSCRUB IP scrubbing operation mode supports both blind and readback scrubbing methods. In the blind scrubbing mode, the GRSCRUB IP rewrites the configuration frames without any data verification. The blind scrubbing can be performed periodically, continually refreshing the configuration data. In the readback scrubbing mode, the GRSCRUB IP verifies the integrity of each frame of the FPGA configuration memory, and then, in the event of errors, rewrites the frame with the correct data read from the Golden memory. Differently from the blind scrubbing, the readback mode allows detecting errors and correcting the frame only if necessary. The readback can also be executed periodically.

The error detection can be performed through CRC verification or by comparing a frame bit-by-bit against its golden version stored in the Golden memory. The latter option is defined as Full Frame Check (FFC). The CRC is an error detection code that applies redundancy to check inconsistencies. A standard 32-bit CRC (CRC32C) algorithm is computed for each FPGA frame and compared to the golden code saved in the Golden memory. Note that the CRC code used by the GRSCRUB IP is not related to the FRAME_ECC primitive from Xilinx FPGAs [RD4]. The CRC and FFC data verifications do not check the masked bits. Each data verification method can be configured to be enabled or not.

4.2 GRSCRUB IP additional features

The configuration interface of the target FPGA can also be affected by soft errors, which may lead to catastrophic results during the scrubbing operation. For instance, in case the FPGA Frame Address Register (FAR) is affected by an SEU during a blind scrubbing execution and its value changes to another valid address, all the subsequent frames would be overwritten wrongly, compromising the entire design. In [RD6], the authors observed high-current events in Xilinx FPGAs due to SEEs affecting the configuration interface, which led the blind scrubbing to write multiples frames in incorrect addresses.

The GRSCRUB IP was designed to decrease the probability of failures during the scrubbing operation due to a faulty interface. The GRSCRUB IP verifies the integrity of the configuration interface of the target FPGA before each new scrubbing execution. The verification is performed by reading a specific frame and checking its address. If the returned address matches the expected one, the interface is considered stable and, therefore, the scrubbing cycle starts. Otherwise, an error is reported. In addition, setting up the configuration interface for each scrubbed frame could be a safer approach instead of configuring all frames at once. For instance, writing one frame at a time during blind scrubbing avoids overwritten the entire memory in case of errors in the FAR register. Both blind and readback scrubbing can be configured to enable or disable such features.

4.3 GRSCRUB IP system setup

Fig. 1 shows the block diagram of a GRSCRUB-based system, which can be the GR716B Microcontroller or a design implemented in a flash-based FPGA, integrated with the target FPGA. The configuration memory of the target FPGA is accessed externally through the slave SelectMap configuration

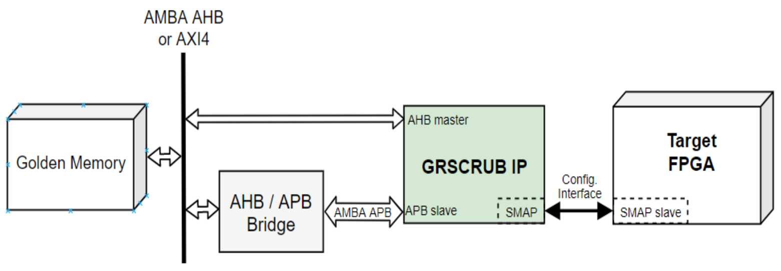


Figure 1 GRSCRUB IP system block diagram.

interface. The GRSCRUB IP can access SelectMap through all the supported bus widths (i.e., 8-, 16-, or 32-bit). The slave SelectMap clock is provided externally by the system in which the GRSCRUB IP is embedded. The GRSCRUB IP is a multiple clock domain design, which includes the internal system clock, and the SelectMap clock used for synchronization.

The GRSCRUB IP accesses through an AMBA AHB or AXI4 bus, a memory that stores the golden configuration bitstream and the mask data of the design implemented in the target FPGA (Golden memory). The golden bitstream is used both to configure the FPGA at start-up and to repair the configuration memory in the event of soft errors. The mask data information is provided by the synthesis tool and contains a description of all dynamic bits in the design. During data verification in the scrubbing operation, the GRSCRUB IP does not verify the dynamic bits in the frames, and the mask data is used to mask only these specific bits.

4.4 GRSCRUB IP and SEM-IP comparison

The Xilinx Soft Error Mitigation Intellectual Property (SEM-IP) [RD4] is an internal scrubbing core compatible with most of Xilinx FPGAs. The SEM-IP main advantage is the high-speed for single error detection and correction. As demonstrated in [RD7], internal scrubbers are susceptible to get locked and have the correction capability compromised due to faults in the scrubber interface or multiple errors in the configuration memory. In this context, external scrubbers may provide higher robustness and the ability to deal with multiple errors. Table 1 presents a comparison between the GRSCRUB IP and the SEM-IP.

Table 1 GRSCRUB and SEM-IP comparison

	GRSCRUB	SEM-IP [RD4]
Type of scrubbing	External	Internal
FPGA programming	Yes	No
Single error detection	Yes	Yes
Single error correction	Yes	Yes
Multiple errors detection	Yes	Yes
Multiple errors correction	Yes	No

Advantages	Programming;
	Frame mapping;
	MBU correction;
	Robustness
		High correction speed

5 DESIGNING WITH THE GRSCRUB IP

In the context of applications implemented in SRAM-based FPGAs demanding a high level of reliability, such as space applications, the GRSCRUB IP can be integrated into the system to maintain the FPGA configuration memory consistent by repairing the logic and correcting bit-flips. Fig. 2 presents two user-case examples for implementing the GRSCRUB IP system setup. The GRSCRUB IP can be implemented in a non-volatile-based FPGA as an IP core and connected to the target SRAM-based FPGA through SelectMap. The external memory is used as the Golden memory of the GRSCRUB IP.

Another approach is using the GR716B Microcontroller connected to the target FPGA. The GR716B is based on a fault-tolerant SPARC V8 32-bit LEON3FT processor, and it will be implemented using Imec’s DARE180 [RD12] radiation-hardened cell library in a 180nm Complementary Metal Oxide Semiconductor (CMOS) technology platform from UMC (Taiwan). In addition to the GRSCRUB IP core, the design integrates several on-chip data bus standards (SpaceWire, MIL-STD-1553, CAN-FD, I2C, SPI, UART, etc.), and other digital and analog cores, such as fault-tolerant memory controllers and digital-to-analog and analog-to-digital converters, respectively. Since the GRSCRUB IP core is self-standing, the target FPGA can be supervised without interrupting the microcontroller software execution. The GR716B Microcontroller is expected to be available during 2021.

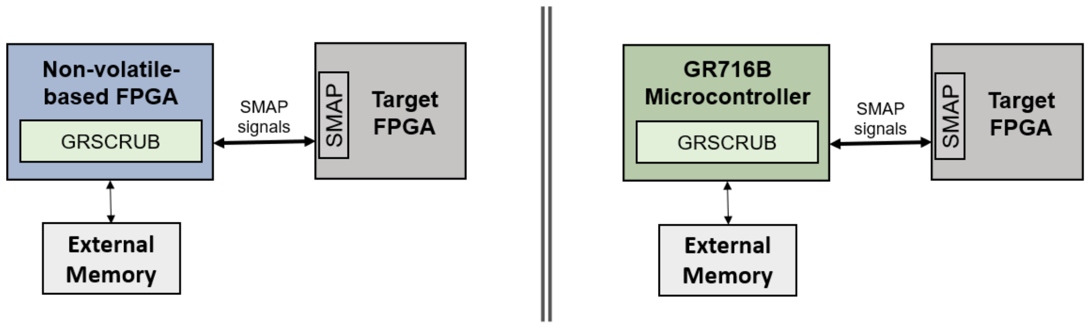


Figure 2 *User-case examples for implementing the GRSCRUB system: GRSCRUB as an IP core implemented in a non-volatile-based FPGA; and GRSCRUB embedded in the GR716B Microcontroller.*

5.1 GRSCRUB synchronization

As described in [RD5], the SelectMap clock (CCLK signal) of the target FPGA is provided by the system which embeds the GRSCRUB IP, and a dedicated register buffer must be integrated into the system design to allow controlling (enabling/disabling) the CCLK signal. This control is performed by the GRSCRUB IP to synchronize the reading and writing operations in the SelectMap interface.

The clock synchronization is critical. The data and control signals must be correctly synchronous with the CCLK signal. The synchronization can be affected by the system setup and connections to access the target FPGA. Delay due to long paths is one cause of signal desynchronization.

For example, for the experimental setup presented in the next section, it was required to invert the CCLK signal to ensure the stability of data and control signals in the rising edge of the clock. Because of the long cabling between both boards, the clock signal was delayed, which affected the synchronization. A system with short signals paths and designed with synchronization constraints should not require inverting the SelectMap clock signal.

5.2 GRSCRUB operation control

The GRSCRUB IP operation is configured through the IP's registers accessed via AHB or AXI. After the operation being configured, the GRSCRUB IP is self-standing to execute the operation mode.

The GRSCRUB's registers can also be controlled through the Cobham Gaisler's GRMON3 debug monitor [RD9]. An example of a Tcl script to configure the GRSCRUB IP using GRMON3 is presented and described in the APPENDIX A. See the GRSCRUB specification [RD5] for more information on how to configure the IP.

5.2.1 Identifying the addresses of the GRSCRUB's registers

The base address of the GRSCRUB's registers is defined in the bus specification of component instantiation. For instance, in the example of APPENDIX A, the APB base address of the GRSCRUB's registers is `0x80000D00`. See the GRSCRUB IP specification [RD5] to more details about the APB address offset of the GRSCRUB's registers.

Example of registers configuration (from APPENDIX A):

```
    [..]

    # Initialize variables
    variable REG

    # GRSCRUB registers start address
    set grscrub_regaddr 0x80000D00

    # Initialize register offsets for GRSCRUB IP
    array set REG {
        GRSCRUB.STAT        0x80000D00
        GRSCRUB.CONFIG      0x80000D04
        GRSCRUB.IDCODE      0x80000D08
        GRSCRUB.DELAY       0x80000D0C
        GRSCRUB.FCR         0x80000D10
        GRSCRUB.LFAR        0x80000D14
        GRSCRUB.LGBAR       0x80000D18
        GRSCRUB.HGBAR       0x80000D1C
        GRSCRUB.LGSFAR      0x80000D20
        GRSCRUB.LMASKAR     0x80000D24
        GRSCRUB.LFMAPR      0x80000D28
        GRSCRUB.LGCRCAR     0x80000D2C
        GRSCRUB.LGRBKAR     0x80000D30
        GRSCRUB.ECNT        0x80000D34
        GRSCRUB.SETUP       0x80000D38
        GRSCRUB.CAP         0x80000D3C
        GRSCRUB.FRAMEID     0x80000D40
        GRSCRUB.ERRFRAMEID  0x80000D44
    }

    [...]
```

5.2.2 Initial Golden memory configuration

Before starting using the GRSCRUB IP, the configuration bitstream and mask data must be loaded to the Golden memory of the system, and the corrected memory addresses should be set in the GRSCRUB registers. The Golden memory can be RAM or ROM. This application note focuses only on the RAM memory example. If the Golden memory is ROM, all data should be previously stored on the memory. See the GRSCRUB IP specification [RD5] to more details about the Golden memory storage.

The example of the Tcl script presented in the APPENDIX A access a RAM memory, in which the GRSCRUB IP can perform read and write operations. The `mem_load32` procedure loads to the Golden memory the configuration bitstream and the mask data of the target FPGA. One should add the correct path of the files and set the correct memory addresses in the `bitfolder`, `MEM_BASE`, and `BITPARAMS` variables.

The `MEM_BASE` and `BITPARAMS` variables are the addresses in the Golden memory defined by the user. In the example below, the base address of the memory component in the design (Golden memory) is `0x40000000`. Based on this address, the following load addresses are set:

- **Configuration bitstream:** `BITPARAMS(LOADAD.BIT)` is the lower address to store the configuration bitstream of the target FPGA design in the Golden memory.
- **Mask data:** `BITPARAMS(LOADAD.MSK)` is the lower address to store the mask data of the target FPGA design in the Golden memory.

In addition to the configuration bitstream and mask data addresses, the following load addresses need to be defined:

- **Mapping data:** `BITPARAMS(LOADAD.MAP)` is the lower address in the Golden memory to the GRSCRUB read the mapping information of the FPGA frames. The GRSCRUB also writes the frames mapping data at this address during Mapping operation mode (only if the Golden memory is RAM).
- **CRC codes:** `BITPARAMS(LOADAD.CRC)` is the lower address in the Golden memory to the GRSCRUB read the CRC code information of the FPGA frames. The GRSCRUB also writes the CRC code data at this address during Golden CRC operation mode (only if the Golden memory is RAM).

Example of `MEM_BASE` and `BITPARAMS` configuration (from APPENDIX A):

```
    [...]

    ### Golden memory address definition ###

    # Golden memory base address
    set MEM_BASE 0x40000000

    # Set memory addresses #

    # load configuration bitstream (generated by the synthesis tool)
    set BITPARAMS(LOADAD.BIT)       [expr $MEM_BASE + 0x00000006]
    # load mask data (generated by the synthesis tool)
    set BITPARAMS(LOADAD.MSK)       [expr $MEM_BASE + 0x01800006]
    # Address for GRSCRUB to read the frame mapped addresses.
    # The GRSCRUB might also store the frame mapped addresses. (optional)
    set BITPARAMS(LOADAD.MAP)       [expr $MEM_BASE + 0x03000000]
    # Address for GRSCRUB to read the golden CRC data.
    # The GRSCRUB might also store the golden CRC data. (optional)
    set BITPARAMS(LOADAD.CRC)       [expr $MEM_BASE + 0x04000000]

    [...]
```

The load addresses are used in the `mem_load32` procedure to store the required data information. Besides the load addresses, one should set the addresses of the relevant information of the configuration bitstream and the mask data that will be accessed by the GRSCRUB IP. Note that in the example the `LOADAD.MAP` and `LOADAD.CRC` also correspond to the addresses of the relevant information.

The following additional addresses need to be defined:

- **The starting address of the configuration bitstream:** `BITPARAMS(START.BIT)` is the address of the first *dummy* word in the configuration bitstream (`0xFFFFFFFF`), after the initial header. All configuration bitstreams have an initial header with ASCII characters that provides some file information, which is not required to program the FPGA. The synchronization phase starts at the first *dummy* word (`0xFFFFFFFF`). See the Xilinx Configuration Guide [RD11] to more details about the bitstream composition. The GRSCRUB's LGBAR register should be set with this address. See the GRSCRUB IP specification [RD5] for more details.
- **Address of the first valid data word in the configuration bitstream:** `BITPARAMS(START.GOLD)` is the address of the first golden word referent to the FPGA configuration frames to be scrubbed by the GRSCRUB IP. If all frames of the FPGA configuration memory should be scrubbed, this is the address of the first valid data word of the configuration bitstream. At the beginning of the configuration bitstream are the synchronization words to set up the FPGA configuration interface (i.e., SelectMap). The valid data words are located just after the synchronization words. See the Xilinx Configuration Guide [RD11] to more details about the bitstream composition and how to identify the *first bitstream configuration data word*. If a partial scrubbing is defined, the `BITPARAMS(START.GOLD)` is the address in the Golden memory of the first word of the first frame to be scrubbed. The GRSCRUB's LGSFAR register should be set with this address. See the GRSCRUB IP specification [RD5] for more details.
- **Address of the last valid data word in the configuration bitstream:** `BITPARAMS(END.BIT)` is the address of the last golden data word referent to the FPGA configuration frames to be scrubbed by the GRSCRUB IP. If all frames of the FPGA configuration memory should be scrubbed, this is the address of the last valid data word of the configuration bitstream. After all configuration data, the configuration bitstream contains synchronization words to set up the FPGA configuration interface (i.e., SelectMap) and finishing the programming phase correctly. See the Xilinx Configuration Guide [RD11] to more details about the bitstream composition and how to identify the *last bitstream configuration data word*. If a partial scrubbing is defined, the `BITPARAMS(END.GOLD)` is the address in the Golden memory of the last word of the last frame to be scrubbed. The

GRSCRUB's HGBAR register should be set with this address. See the GRSCRUB IP specification [RD5] for more details.

- **Address of the first valid word in the mask data:** `BITPARAMS(START.MSK)` is the address related to the first mask word related to the first golden word of the FPGA configuration frames to be scrubbed by the GRSCRUB IP. It is the same logic applied to define the `BITPARAMS(START.GOLD)` address. See the Xilinx Configuration Guide [RD11] to more details about the mask data composition. The GRSCRUB's LMASKAR register should be set with this address. See the GRSCRUB IP specification [RD5] for more details.

Example of `BITPARAMS` configuration (from APPENDIX A):

```
    [...]

    # Define the start addresses in the Golden memory #

    # Start address of the configuration bitstream in the Golden memory.
    set BITPARAMS(START.BIT)         [expr $MEM_BASE + 0x0000008C]

    # Address of the first configuration bitstream frame in the Golden memory
    set BITPARAMS(START.GOLD)        [expr $MEM_BASE + 0x000001a4]

    # Set the highest configuration bitstream address in the Golden memory
    set BITPARAMS(END.BIT)           [expr $MEM_BASE + 0x01701e74]

    # Address of the first mask data related with the first configuration bit-
stream frame in the Golden memory.
    set BITPARAMS(START.MSK)         [expr $MEM_BASE + 0x018001a4]

    [...]
```

5.2.3 Programming the target FPGA

Before enabling the programming operation mode, the GRSCRUB's registers must be configured. See the GRSCRUB IP specification [RD5] to more details about how to configure GRSCRUB's registers.

The `grscrub_init_progmode` procedure in the APPENDIX A shows a configuration example of the CONFIG, LGBAR, HGBAR, and IDCODE registers.

The `grscrub_progfpga` procedure shows the steps required for enabling the programming operation mode. The steps are the following:

1) ensure the GRSCRUB is disabled;
2) clear the done (OPDONE and SCRUND) and error (SCRERR) bitfields in the STATUS register;
3) configure the required registers (`grscrub_init_progmode` procedure); and
4) enabling the GRSCRUB IP to execute the operation.

The programming is finished when the OPDONE bitfield of the STATUS register goes high. If an error occurs during the execution, the SCRERR bitfield of the STATUS register goes high, and the

ERRID indicates the id of the error.

If the DONE signal of the target FPGA is mapped to a LED on the board, one can check if the LED is ON when the target FPGA is programmed successfully.

5.2.4 Mapping the target FPGA

Before enabling the mapping operation mode, the GRSCRUB's registers must be configured. See the GRSCRUB IP specification [RD5] to more details about how to configure GRSCRUB's registers.

The `grscrub_init_fpgamappingmode` procedure in the APPENDIX A shows a configuration example of the CONFIG, LGBAR, HGBAR, LFAR, FCR, LMASKAR, LGSFAR, LFMAPR, and IDCODE registers.

If all frames of the configuration memory should be mapped, the LFAR is set to the address of the first frame of the configuration bitstream (i.e., `0x00000000`), and the FCR is set with the total number of FPGA frames (e.g., `49030` for the KU060 FPGA). Note that although the FCR register is configured with the total number of frames of the FPGA, only the configuration frames are mapped (e.g., `37498` frames for the KU060 FPGA). One can also set the FCR register directly with the number of configuration frames. If only a partial number of frames should be mapped, the LFAR is set to the address of the first frame to be mapped (e.g., `0x00020000`), and the FCR is set with the number of FPGA frames to be mapped (e.g., `7500`). In both cases, the FCR is also set with the frame length of the target FPGA (e.g., `123` words for the KU060 FPGA).

The `grscrub_fpgamapping` procedure shows the steps require for enabling the mapping operation mode. The steps are the following:

1) ensure the GRSCRUB is disabled;
2) clear the done (OPDONE and SCRUND) and error (SCRERR) bitfields in the STATUS register;
3) configure the required registers (`grscrub_init_fpgamappingmode` procedure); and
4) enabling the GRSCRUB IP to execute the operation.

The mapping phase is finished when the OPDONE bitfield of the STATUS register goes high. If an error occurs during the execution, the SCRERR bitfield of the STATUS register goes high, and the ERRID indicates the id of the error.

To verify if the target FPGA frames were mapped correctly, one can check the address `BITPARAMS(LOADAD.MAP)` in the Golden memory and verify if the addresses of the frames are correctly stored.

5.2.5 Storing the golden CRC codes

The golden CRC codes must be stored in the Golden memory before enabling the GRSCRUB read-back with CRC detection. One can store the golden CRC codes previously or execute the GRSCRUB operation mode. For the latter, the Golden memory must be writable.

Before enabling the golden CRC operation mode, the GRSCRUB's registers must be configured. See the GRSCRUB IP specification [RD5] to more details about how to configure GRSCRUB's registers.

The `grscrub_init_readbackmode` procedure in the APPENDIX A shows a generic example of how to configure the GRSCRUB for readback scrubbing, and the same configuration is used for golden CRC operation mode. The `grscrub_init_readbackmode` procedure is detailed in the next section.

Note that the golden CRC codes refer to the target FPGA frames that will be scrubbed. Therefore, if all frames will be scrubbed, the golden CRC codes should be generated for all frames. On the other hand, if only a partial number of frames will be scrubbed, the golden CRC codes should be generated only for the partial number of frames.

The `grscrub_init_goldencrc` procedure shows the steps required for enabling the golden CRC operation mode. The steps are the following:

1) ensure the GRSCRUB is disabled;
2) clear the done (OPDONE and SCRUND) and error (SCRERR) bitfields in the STATUS register;
3) configure the required registers (`grscrub_init_readbackmode` procedure); and
4) enabling the GRSCRUB IP to execute the operation.

The execution is finished when the OPDONE bitfield of the STATUS register goes high. If an error occurs during the execution, the SCRERR bitfield of the STATUS register goes high, and the ERRID indicates the id of the error.

To verify if the golden CRC codes were generated correctly, one can check the address `BITPARAMS(LOADAD.CRC)` in the Golden memory and verify if the golden CRC codes are correctly stored.

5.2.6 Scrubbing the target FPGA

5.2.6.1 Blind Scrubbing

Before enabling the blind scrubbing operation mode, the GRSCRUB's registers must be configured. See the GRSCRUB IP specification [RD5] to more details about how to configure GRSCRUB's registers.

The `grscrub_init_blindscrubmode` procedure in the APPENDIX A shows a configuration example of the CONFIG, DELAY, LGBAR, HGBAR, LFAR, FCR, LGSFAR, LFMAPR, and IDCODE registers.

If all frames of the configuration memory should be scrubbed, the LFAR is set to the address of the first frame of the configuration bitstream (i.e., `0x00000000`), and the FCR is set with the number of configuration frames of the FPGA (e.g., `37498` for the KU060 FPGA). If only a partial number of frames should be scrubbed, the LFAR is set to the address of the first frame to be scrubbed (e.g., `0x00020000`), and the FCR is set with the number of FPGA frames to be scrubbed (e.g., `7500`). In both cases, the FCR is also set with the frame length of the target FPGA (e.g., `123` words for the KU060 FPGA).

The blind scrubbing can be configured to execute only once or periodically. The SCRUN bitfield of the CONFIG register must be 1 for a periodic run. A delay can be defined between periodic scrubbing runs. The delay period can be set in the DELAY register.

The `grscrub_blindscrubbingfpga` procedure shows the steps required for enabling the blind scrubbing operation mode. The steps are the following:

1) ensure the GRSCRUB is disabled;
2) clear the done (OPDONE and SCRUND) and error (SCRERR) bitfields in the STATUS register;
3) configure the required registers (`grscrub_init_blindscrubmode` procedure); and
4) enabling the GRSCRUB IP to execute the operation.

The SCRUND bitfield of the STATUS register goes high after each scrubbing execution in a periodic run. The OPDONE bitfield goes high only in one time execution. If an error occurs during the scrubbing, the execution is stopped, the SCRERR bitfield of the STATUS register goes high, and the ERRID indicates the id of the error.

During periodic scrubbing, one can check the HOLD bitfield of the STATUS register to identify the GRSCRUB execution. The HOLD bitfield is 0 when the GRSCRUB IP is performing the scrubbing operation on the target FPGA. The HOLD bitfield is 1 when the GRSCRUB IP is in hold waiting during the delay period.

One can also check the FRAMEID register that represents the id of the current frame of the target FPGA scrubbed by the GRSCRUB IP.

5.2.6.2 Readback scrubbing

Before enabling the readback scrubbing operation mode, the GRSCRUB's registers must be configured. See the GRSCRUB IP specification [RD5] to more details about how to configure GRSCRUB's registers.

The `grscrub_init_readbackmode` procedure in the APPENDIX A shows a configuration example of the DELAY, LGBAR, HGBAR, LFAR, FCR, LGSFAR, LMASKAR, LFMAPR, LGCRCAR, and IDCODE registers. At the initialization, one can also clean the ECNT, ERRFRAMEID, and FRAMEID registers to reset the number of detected errors and frame id of previous runs. In this example, the CONFIG register is set in the specific readback procedure, as further described.

The configuration of the number of frames to be scrubbed (FCR register) and initial frame address (LFAR) for the entire configuration memory or just partial number of frames is the same presented in the blind scrubbing section.

The readback scrubbing can also be configured to execute only once or periodically. The SCRUN bitfield of the CONFIG register must be 1 for a periodic run. A delay can be defined between periodic scrubbing runs. The delay period can be set in the DELAY register.

The readback scrubbing can be configured to detect only or to detect and correct errors. The former is configured in the `grscrub_readbackfpga_onlydetection` procedure, and the latter is configured in the `grscrub_readbackfpga_correction` procedure. The CORM bitfield of the CONFIG register defines readback mode. In both cases, the error detection can be through FFC, CRC, or both (i.e., FFC + CRC). The FFCEN and CRCEN bitfields of the CONFIG register configure the detection options.

The steps required for enabling the readback scrubbing operation mode are the following:

1) ensure the GRSCRUB is disabled;
2) clear the done (OPDONE and SCRUND) and error (SCRERR) bitfields in the STATUS register;
3) configure the required registers (`grscrub_init_readbackmode` procedure);
4) configure the CONFIG register; and
5) enabling the GRSCRUB IP to execute the operation.

The SCRUND bitfield of the STATUS register goes high after each scrubbing execution in a periodic run. The OPDONE bitfield goes high only in one time execution. If an error occurs during the scrubbing, the execution is stopped, the SCRERR bitfield of the STATUS register goes high, and the ERRID indicates the id of the error.

During periodic scrubbing, one can check the HOLD bitfield of the STATUS register to identify the GRSCRUB execution. The HOLD bitfield is 0 when the GRSCRUB IP is performing the scrubbing operation on the target FPGA. The HOLD bitfield is 1 when the GRSCRUB IP is in hold waiting during the delay period.

One can also check the FRAMEID register that represents the id of the current frame of the target FPGA scrubbed by the GRSCRUB IP.

The ECNT register presents the number of errors detected during the readback scrubbing. If the error correction is enabled, the ECNT register shows the number of correctable and uncorrectable errors. The error counters accumulate over scrubbing runs. One should clear the register to initiate a new count.

6 CONNECTING WITH THE TARGET FPGA

The GRSCRUB IP must be connected to the slave SelectMap interface of the target FPGA. Fig. 3 shows an example of the GRSCRUB and the SelectMap connection in a Xilinx UltraScale FPGA. In the example, the GRSCRUB port signals are directly attached to the SelectMap pins. The function of the SelectMap pins is described in the GRSCRUB IP specification [RD5]. One should always check the documentation of the Xilinx FPGA family for detailed information [RD11]. Also, refer to the FPGA Data Sheet to define the proper voltage connection.

The interface mode pins M[2:0] of the target FPGA must be configured to slave SelectMap. Thus, these pins must be connected to the GRSCRUB IP or directly tied to high/low levels externally, as in the example.

The top system that embeds the GRSCRUB IP provides the clock to the SelectMap interface (CCLK signal) and the GRSCRUB (SMAPCLKI signal). As previously described in section 5.1, a clock buffer should be used to allow the GRSCRUB IP to control the CCLK signal. The GRSCRUB IP enables or disables the CCLK through the CLK_EN signal. This control is required for the synchronization of operations.

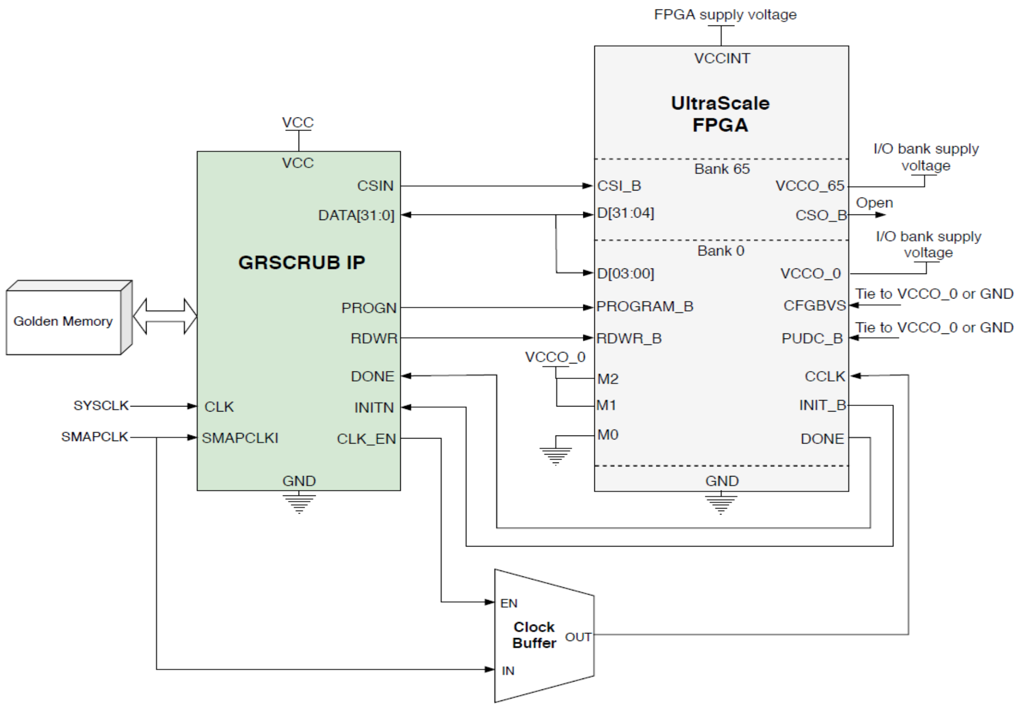


Figure 3 Example of connection of the GRSCRUP IP and slave SelectMap interface for Xilinx UltraScale FPGA.

To allow the GRSCRUB IP to access and control the slave SelectMap interface, the generated configuration bitstream of the target FPGA must be constrained by following the requirements below:

1) set the slave SelectMap interface;
2) set the SelectMap pins to persistent: the persistent property keeps the slave SelectMap enabled after configuration;
3) do not compress the configuration bitstream;
4) do not use encryption in the configuration bitstream; and
5) do not prohibit readback in the security settings of the configuration bitstream.

Example of a constraint file used in the Vivado Design Suite tool:

```
# Select Slave SelectMAP interface
set_property CONFIG_MODE {S_SELECTMAP} [current_design]

# Configuration interface pins are persistent
set_property BITSTREAM.CONFIG.PERSIST {YES} [current_design]

# Do not compress the bitstream
set_property BITSTREAM.GENERAL.COMPRESS {FALSE} [current_design]

# Do not encrypt the bitstream
set_property BITSTREAM.ENCRYPTION.ENCRYPT {NO} [current_design]

# Do not apply security
set_property BITSTREAM.READBACK.SECURITY {NONE} [current_design]
```

7 EXPERIMENTAL EVALUATION SETUP

The GRSCRUB IP evaluation setup consists of a host FPGA embedding the GRSCRUB IP in a system similar to the one presented in Fig. 1, and the Device Under Test (DUT), which is the target FPGA under evaluation, a Xilinx Kintex UltraScale FPGA. Fig. 4 presents the block diagram and the view of the experimental setup. The test controller and the target FPGA block elements are detailed in the following sections.

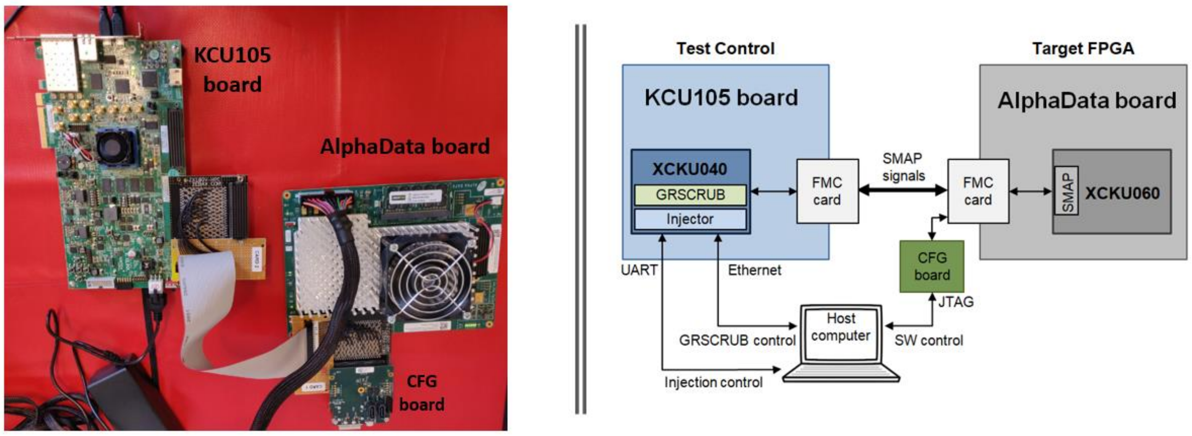


Figure 4 GRSCRUB IP evaluation test setup.

7.1.1 Test controller

A Xilinx KCU105 evaluation board [RD8] is used as the test controller. The board features a Xilinx Kintex UltraScale XCKU040 FPGA, in which the GRSCRUB IP and a fault injection engine are implemented. Table 2 presents the resource usage of the GRSCRUB IP embedded in the XCKU040 FPGA.

Table 2 Resource usage of GRSCRUB IP implemented in the XCKU040 FPGA

LUT	FF	Carry	DSP	BRAM
4,550	2,678	117	5	1

The fault injection engine is controlled via UART and is responsible for emulating upsets in the configuration memory of the target FPGA. It also uses the SelectMap interface to access the configuration frames and flip bits, one at time. The target frame and target bit inside the frame are selected randomly. The injection engine reads the selected frame, flips the target bit, and then rewrites the frame to the FPGA. Since both GRSCRUB IP and injector uses the SelectMap interface to access the FPGA configuration memory, only one can be enabled at a time.

Besides the GRSCRUB IP and the fault injection system, the test controller design also contains other IP cores from the GRLIB IP library [RD5], such as AHB bus, DDR3 memory controller, Debug Support Unit (DSU), Ethernet, and UART. In this setup, the GRSCRUB IP is controlled through

Ethernet using the Cobham Gaisler's GRMON3 debug monitor [RD9] that configures the IP to execute the operation modes presented in Section 4.2.

Two FPGA Mezzanine Card (FMC) breakout boards are used to allow the communication between the test controller and the DUT board. The SelectMap signals from the target FPGA are accessed and controlled via the FMC cards. The 8-bit bus width of the SelectMap interface is used for reading and writing operations.

The test controller frequency is *100 MHz*, and the provided SelectMap clock is *10 MHz*. The maximum SelectMap clock frequency depends on the system setup. Due to the cabling to connect both boards and long signal paths, the SelectMap frequency is restricted in the experimental setup. Higher speeds can be achieved in a system integrating the target FPGA and GRSCRUB IP on the same board.

The fault injection campaigns aim first to evaluate the GRSCRUB IP and test the scrubbing functionality, and second to ensure that the IP operates transparently in dynamic designs. In all test campaigns, the GRSCRUB IP programs the target FPGA, and then the test controller starts the execution. For each injection run, one or more random faults are injected in the configuration memory of the target FPGA. In sequence, the GRSCRUB IP is released to scrub the faulty bits. At the end of the scrubbing execution, the configuration memory is verified to check if all bits were corrected. After that, a new injection run starts, and the loop is repeated.

7.1.2 Target FPGA – Xilinx Kintex UltraScale XCKU060

An AlphaData ADM-SDEV-BASE development kit [RD10] embedding a Xilinx Kintex UltraScale FPGA (XCKU060-1-FFVA1517I industrial part, equivalent to the XQRKU060-CNA1509 space-grade part) is the adopted target FPGA. An FMC card is also attached to the board, providing access to the JTAG interface. The JTAG connection is used to control the software execution when required.

Two test designs were implemented for the evaluation experiments, as described below:

- *Static design*: the design does not implement any dynamic function, and therefore most of the configuration bitstream is empty. The functionality of the design is not evaluated since the goal is only to validate the GRSCRUB IP features. The fault injection targets all FPGA configuration frames, and the IP also monitors the entire configuration memory.
- *LEON3FT-based design*: the design implements a LEON3FT processor core. In addition to the LEON3FT processor, the design also contains other IP cores from GRLIB [RD5], such as DSU, fault-tolerant SRAM module, AHB bus, JTAG, and UART. The 16 KB Instruction and Data L1 caches and the processor Register File (RF) are implemented in BRAMs and are protected by Error Detection And Correction (EDAC). The LEON3FT runs a test software that monitors and tests the Integer Unit (IU) of the processor. The software is controlled using the GRMON3 via JTAG. The floorplanning of the design is constrained to a specific area, and both fault injection and GRSCRUB IP only target this area.

The resource usage of Static and LEON3FT designs implemented in the target FPGA are presented in Table 3.

Table 3 Resource usage of Static and LEON3FT designs implemented in the XCKU060 FPGA

Design	LUT	FF	Carry	DSP	BRAM
Static	23	521	3	0	0
LEON3FT	8,852	6,016	43	4	53

8 EVALUATION RESULTS

Table 4 presents the fault injection results for the Static design implemented in the target FPGA. The blind, readback FFC, and readback CRC scrubbing modes of the GRSCRUB IP were evaluated. Single or multiple random faults were injected per run, and then the GRSCRUB IP scrubbing mode was enabled to correct the faults. In all tests, the GRSCRUB IP was able to detect and correct all injected faults.

Table 4 Fault injection results for Static design and GRSCRUB IP in different scrubbing modes

GRSCRUB IP Scrubbing	# Inj. faults per run	# Total runs	# Total faults corrected
Blind	1	2,000	2,000
Blind	10	15,735	157,350
Readback FFC	10	12,086	120,860
Readback CRC	10	7,220	72,200

The tests with the LEON3FT-based design implemented in the target FPGA demonstrated that 99.6% of the software runs were successful. The software executed continuously while single random faults were injected in the target FPGA. After each injection, the GRSCRUB IP readback FFC scrubbing was enabled to clear the bit-flip. A total of 11,399 faults were injected, and the GRSCRUB IP was able to correct all injected faults.

The large amount of injected faults not leading to errors in the target design confirms that the GRSCRUB IP scrubbing operation allows uninterrupted software execution in the presence of correctable faults in the FPGA configuration memory by preventing the error build-up. The software errors presented refer to critical points of failure related to non-protected modules in the target FPGA design (the literature usually refers to such bits as "critical bits") that lead to errors before the GRSCRUB IP be able to correct the fault. One must notice that such software errors are application-dependent, i.e., different software benchmarks may lead to different results.

In this context, the GRSCRUB IP minimizes the latency of single points of failure in the system, but it does not avoid errors happening and neither their effects on the design. Additional mitigation techniques at the design level are recommended to decrease the number of single points of failure and increase the fault masking.

8.1 Performance analysis

Table 5 presents the approximated performance, in seconds, of the GRSCRUB operations targeting the XCK060 in the experimental setup using 8-bit data in the SelectMap interface and targeting all configuration memory frames.

The performance of the readback scrubbing operation is related to faulty-free configuration memory. As detailed in [RD5], the scrubbing period depends on several factors, such as the number of scrubbed frames, the data bus width, the GRSCRUB and SelectMAP frequencies operation, and the required time to access the Golden memory. In addition, the required time for reading and writing operations in the SelectMap interface should be considered. The performance of the readback operation is also directly affected by the number of faults in the FPGA configuration memory.

Table 5 Performance of GRSCRUB operation targeting the XCK060 FPGA in the experimental setup using 8-bit SelectMap bus width

GRSCRUB IP Operation	Performance (aprox.)	Number of frames	GRSCRUB freq.	SMAP freq.	SMAP bus width
Programming	13.5 s	49,030	100 MHz	10 MHz	8-bit data
Mapping	7.1 s	37,498			
Blind scrubbing	9.8 s				
Readback (FFC / CRC) scrubbing	3.8 s				

9 CONCLUSION

The Cobham Gaisler's GRSCRUB IP is an FPGA configuration supervisor that features programming and scrubbing capabilities. The GRSCRUB IP will be included in the new version of the Cobham Gaisler's GR716B Microcontroller, and it is also available as an IP core in the GRLIB. Fault injection tests targeting a Xilinx Kintex UltraScale FPGA demonstrated the GRSCRUB IP capability to correct all injected faults in the FPGA configuration memory. Tests in a LEON3FT design confirms that the GRSCRUB IP scrubbing operation allows uninterrupted software execution in the presence of correctable errors in the FPGA configuration memory by preventing the error build-up. The GRSCRUB IP reduces the persistent effects of errors in critical points of failure. However, the impact on the design is not mitigated. Therefore, additional mitigation techniques at the design level are recommended for that and to increase the fault masking.

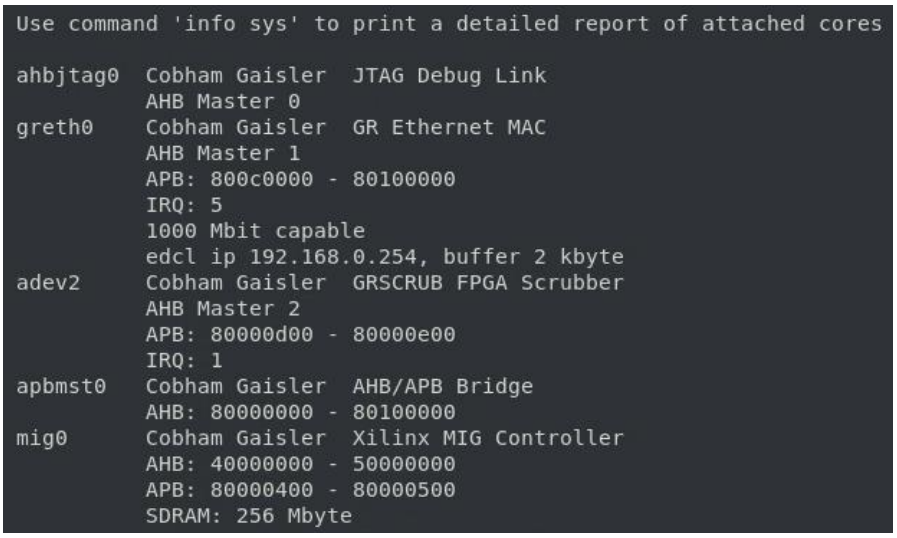
APPENDIX A EXAMPLE OF TCL SCRIPT TO CONFIGURE THE GRSCRUB IP USING GRMON3

The Tcl source code presented below is an example of how to configure the GRSCRUB IP to execute the operational modes. See the GRSCRUB specification [RD5] for more information on how to configure the IP.

The Golden memory addresses used in the example depend on the memory space defined on the memory controller component in the design.

The addresses of GRSCRUB registers depends on the address space defined on the AHB bus.

Fig. 5 presents the *info sys* command in GRMON3 that shows the information of the components of the design.



```
Use command 'info sys' to print a detailed report of attached cores

ahbjtag0  Cobham Gaisler  JTAG Debug Link
          AHB Master 0
greth0    Cobham Gaisler  GR Ethernet MAC
          AHB Master 1
          APB: 800c0000 - 80100000
          IRQ: 5
          1000 Mbit capable
          edcl ip 192.168.0.254, buffer 2 kbyte
adev2     Cobham Gaisler  GRSCRUB FPGA Scrubber
          AHB Master 2
          APB: 80000d00 - 80000e00
          IRQ: 1
apbmst0   Cobham Gaisler  AHB/APB Bridge
          AHB: 80000000 - 80100000
mig0      Cobham Gaisler  Xilinx MIG Controller
          AHB: 40000000 - 50000000
          APB: 80000400 - 80000500
          SDRAM: 256 Mbyte
```

Figure 5 Report in the GRMON3 of the core components in the example design.

- **User adaptions:**

In order to use the example source code, one should adapt the following parameters:

- Golden memory addresses;
- GRSCRUB registers addresses;
- Include the correct path of the configuration bitstream and mask files in the mem_load32 procedure.

- **GRSCRUB_config.tcl source code:**

```
#--GAISLER_LICENSE
#-------------------------------------------------------------------------------------
#-- File:        GRSCRUB_config.tcl
#-- Author:      Adria Barros de Oliveira - Cobham Gaisler AB
#-- Description: Tcl functions to configure the GRSCRUB IP using GRMON3
#-------------------------------------------------------------------------------------


#####################################################################################
### GRSCRUB CONFIGURATION
#####################################################################################

namespace eval grscrub {

    # Clear old variables
    catch {unset REG}
    catch {unset bitfolder}

    # Initialize variables
    variable REG

    # GRSCRUB registers start address
    set grscrub_regaddr 0x80000D00

    # Initialize register offsets for GRSCRUB IP
    array set REG {
        GRSCRUB.STAT        0x80000D00
        GRSCRUB.CONFIG      0x80000D04
        GRSCRUB.IDCODE      0x80000D08
        GRSCRUB.DELAY       0x80000D0C
        GRSCRUB.FCR         0x80000D10
        GRSCRUB.LFAR        0x80000D14
        GRSCRUB.LGBAR       0x80000D18
        GRSCRUB.HGBAR       0x80000D1C
        GRSCRUB.LGSFAR      0x80000D20
        GRSCRUB.LMASKAR     0x80000D24
        GRSCRUB.LFMAPR      0x80000D28
        GRSCRUB.LGCRCAR     0x80000D2C
        GRSCRUB.LGRBKAR     0x80000D30
        GRSCRUB.ECNT        0x80000D34
        GRSCRUB.SETUP       0x80000D38
        GRSCRUB.CAP         0x80000D3C
        GRSCRUB.FRAMEID     0x80000D40
        GRSCRUB.ERRFRAMEID  0x80000D44
    }


    # Choose correct patch to golden bitfiles.
    variable bitfolder ./DUT


    ### Golden memory address definition ###

    # Golden memory base address
    set MEM_BASE 0x40000000

    # Set memory addresses #

    # load configuration bitstream (generated by the synthesis tool)
    set BITPARAMS(LOADAD.BIT)       [expr $MEM_BASE + 0x00000006]
    # load mask data (generated by the synthesis tool)
    set BITPARAMS(LOADAD.MSK)       [expr $MEM_BASE + 0x01800006]
    # Address for GRSCRUB to read the frame mapped addresses.
    # The GRSCRUB might also store the frame mapped addresses. (optional)
    set BITPARAMS(LOADAD.MAP)       [expr $MEM_BASE + 0x03000000]
    # Address for GRSCRUB to read the golden CRC data.
    # The GRSCRUB might also store the golden CRC data. (optional)
    set BITPARAMS(LOADAD.CRC)       [expr $MEM_BASE + 0x04000000]


    # Define the start addresses in the Golden memory #

    # Start address of the configuration bitstream in the Golden memory.
    # This must be the address of the first dummy word in the configuration bitstream
    # (0xFFFFFFFF), after the initial header.
    # All configuration bitstreams have an initial header with ASCII characters that
    # provides some file information, which is not required to program the FPGA.
    # The synchronization phase starts at the first dummy word (0xFFFFFFFF).
    # The LGBAR register is set with this address.
    set BITPARAMS(START.BIT)        [expr $MEM_BASE + 0x0000008C]

    # Address of the first configuration bitstream frame in the Golden memory
    # The LGSFAR register is set with this address.
    set BITPARAMS(START.GOLD)       [expr $MEM_BASE + 0x000001a4]

    # Set the highest configuration bitstream address in the Golden memory
    # The HGBAR register is set with this address.
    set BITPARAMS(END.BIT)          [expr $MEM_BASE + 0x01701e74]

    # Address of the first mask data related with the first configuration bitstream
    # frame in the Golden memory.
    # The LMASKAR register is set with this address.
    set BITPARAMS(START.MSK)        [expr $MEM_BASE + 0x018001a4]


        ### Other configurations ###

  # Total number of configuration frames of KU060
  set fcnt 49030

        # Frame length of KU060
        set flen 123

        # Number of mapped frames
        # (Only mapped frames can be scrubbed)
        set numbermappedframes 37498

        # Define periodic scrubbing runs
        # periodic = 1
        # one time = 0
        set scrun 0

        # Enable partial scrubbing
        # patial = 1
        # full   = 0
        set partial_en 0

        # Opdone bitfield position on Status register
        set done 0x10

        # KU060 FPGA IDCODE
        set FPGA_IDCODE 0x03919093


        #####################################################################################
        ### Initialization procedures
        #####################################################################################

        # Initial configuration
        proc init_config {{design "static"}} \
        {
          # Load bitstream and mask data
          mem_load32 $design

          # Program the target FPGA
          grscrub_progfpga

          # Map the frame addresses
          grscrub_fpgamapping
        }

        # Load the configuration bitstream and mask data in the Golden memory
        proc mem_load32 {{design "static"}} \
        {
          variable BITPARAMS
          variable BITPARAMSTMR
          variable bitfolder
          variable MEM_BASE
          variable partial_en

          if {$design == "static"} {
            load ${bitfolder}/static.bit              $BITPARAMS(LOADAD.BIT)
            load ${bitfolder}/static.msk              $BITPARAMS(LOADAD.MSK)

            verify -max 2 ${bitfolder}/static.bit     $BITPARAMS(LOADAD.BIT)
            verify -max 2 ${bitfolder}/static.msk     $BITPARAMS(LOADAD.MSK)

            puts "BIT-Files Loaded in RAM"

          } elseif {$design == "leon3mp"} {
            load ${bitfolder}/leon3mp.bit             $BITPARAMS(LOADAD.BIT)
            load ${bitfolder}/leon3mp.msk             $BITPARAMS(LOADAD.MSK)

            verify -max 2 ${bitfolder}/leon3mp.bit    $BITPARAMS(LOADAD.BIT)
            verify -max 2 ${bitfolder}/leon3mp.msk    $BITPARAMS(LOADAD.MSK)

            # Adjust addresses
            # Set partial scrubbing (Only the design frames are scrubbed)
            set BITPARAMS(START.GOLD)                 [expr $MEM_BASE + 0x00384FB8]
            set BITPARAMS(START.MSK)                  [expr $MEM_BASE + 0x01B84FB8]
            set partial_en 1

            puts "BIT-Files Loaded in RAM"

          } else {
            puts "This design option is not supported."
          }
        }


        ##################################################################################
        ### Read and Write registers procedures
        ##################################################################################

        # Write a register. Takes a Register name from REG array, and 32-bit value
        proc reg_write {reg val} \
        {
          silent wmem $reg $val

          return 0
        }

        # Read a register. Takes a Register name from REG array
        proc reg_read {reg} \
        {
          set val [silent mem $reg 4]

          return $val
        }


        ##################################################################################
        ### General procedures
        ##################################################################################

        # GRSCRUB IP enable
        proc grscrub_enable {} \
        {
          variable REG

          set config_reg [expr ([reg_read $REG(GRSCRUB.CONFIG)])]

```

```
      #configuration reg -> en bitfild = 1
      reg_write $REG(GRSCRUB.CONFIG) [expr $config_reg | 0x1]

      puts "GRSCRUB ip enabled"
    }

    # GRSCRUB IP disable
    proc grscrub_disable {} \
    {
      variable REG

      set config_reg [expr ([reg_read $REG(GRSCRUB.CONFIG)])]

      #configuration reg -> en bitfild = 0
      reg_write $REG(GRSCRUB.CONFIG) [expr $config_reg & 0xFFFFFFFE]

      puts "GRSCRUB ip disabled"
    }

    # Clean OPDONE and SCRUND bitfields of Status register
    proc grscrub_doneclear {} \
    {
      variable REG
      variable done

      set status_reg [expr ([reg_read $REG(GRSCRUB.STAT)])]

      #clear both dones
      reg_write $REG(GRSCRUB.STAT) [expr $status_reg | 0x1010]

      grscrub_checkdoneclear

      puts "GRSCRUB done clean"
    }

    # Verify if done bitfiled is clean
    proc grscrub_checkdoneclear {} \
    {
      variable REG
      variable done

      set status_reg [expr ([reg_read $REG(GRSCRUB.STAT)])]
      set donecheck [expr $status_reg & $done]

      if {$donecheck == 0x0} {
        puts "GRSCRUB done is clean!"
      } else {
        puts "GRSCRUB done is NOT clean!"
      }
    }

    # Clean the SCRERR bitfield of Status register
    proc grscrub_errorclear {} \
    {
      variable REG

      set status_reg [expr ([reg_read $REG(GRSCRUB.STAT)])]

      reg_write $REG(GRSCRUB.STAT) [expr $status_reg | 0x8]

      puts "GRSCRUB error clean"
    }

    # Show GRSCRUB registers
    proc grscrub_showregs {} \
    {
      variable grscrub_regaddr

      puts "\nGRSCRUB registers:"

      mem $grscrub_regaddr 80
    }
```

```

        ##################################################################################
        ### Programming the target FPGA
        ##################################################################################

        # Configure the GRSCRUB for programming operation mode
        proc grscrub_init_progmode {} \
        {
          variable REG
          variable BITPARAMS
          variable FPGA_IDCODE

          puts "GRSCRUB init program mode"

          #configuration reg -> opmode = 0001
          reg_write $REG(GRSCRUB.CONFIG) 0x00000010

          #golden bitstream addresses
          reg_write $REG(GRSCRUB.LGBAR) $BITPARAMS(START.BIT)
          reg_write $REG(GRSCRUB.HGBAR) $BITPARAMS(END.BIT)

          reg_write $REG(GRSCRUB.IDCODE) $FPGA_IDCODE
        }

        # Configure GRSCRUB to program the target FPGA
        proc grscrub_progfpga {} \
        {
          variable REG
          variable done

          puts "\nStarting FPGA Programming"

          grscrub_disable
          grscrub_doneclear
          grscrub_errorclear
          grscrub_init_progmode

          # wait
          after 100

          grscrub_enable

          puts "FPGA Programming..."

          # wait OPDONE or SCRERR bitfield of Status register
          while {([expr { $done & [expr [reg_read $REG(GRSCRUB.STAT)]]}] != $done) &&
                 ([expr [reg_read $REG(GRSCRUB.STAT)]] != 0x14) &&
                 ([expr [reg_read $REG(GRSCRUB.STAT)]] != 0x00000060) &&
                 ([expr [reg_read $REG(GRSCRUB.STAT)]] != 0x80000060) &&
                 ($grmon::interrupt != 1)} {
            # wait if not done
            after 100
          }

          # check if programmed successfully
          if {([expr { 0x00000060 & [expr [reg_read $REG(GRSCRUB.STAT)]]}] != 0x00000060)} {
            puts "GRSCRUB FPGA programmed successfully!"
          } else {
            puts "ERROR to program FPGA!!!"
          }

          grscrub_disable
        }


        ##################################################################################
        ### Mapping the target FPGA
        ##################################################################################

        # Configure the GRSCRUB for mapping operation mode
        proc grscrub_init_fpgamappingmode {} \
        {
```

```
    variable REG
    variable BITPARAMS
    variable fcnt
    variable flen
    variable FPGA_IDCODE
    variable partial_en

    puts "GRSCRUB FPGA mapping init"

    #configuration reg -> opmode = 0011
    reg_write $REG(GRSCRUB.CONFIG) 0x00000030

    #golden bitstream address
    reg_write $REG(GRSCRUB.LGBAR) $BITPARAMS(START.BIT)
    reg_write $REG(GRSCRUB.HGBAR) $BITPARAMS(END.BIT)

    if {$partial_en==1} {
      # Set partial scrubbing: Example of partial scrubbing in frames of row1 only (KU060)

      #set specific frame address
      reg_write $REG(GRSCRUB.LFAR) 0x00020000

      #set specific number of frames
      reg_write $REG(GRSCRUB.FCR) [expr [expr 7500 << 9] | [expr $flen << 2]]

    } else {
      # Full scrubbing of the configuration memory

      #set frame address 0x0
      reg_write $REG(GRSCRUB.LFAR) 0x00000000

      # Set the total number of FPGA configuration frames
      # However, only the frames defined in the configuration block are mapped,
      # that means that all block memories are excluded from the mapping phase.
      reg_write $REG(GRSCRUB.FCR) [expr [expr $fcnt << 9] | [expr $flen << 2]]

    }

    #mask addr
    reg_write $REG(GRSCRUB.LMASKAR) $BITPARAMS(START.MSK)

    #start frame addr
    reg_write $REG(GRSCRUB.LGSFAR) $BITPARAMS(START.GOLD)

    #map addr in the golden memory
    reg_write $REG(GRSCRUB.LFMAPR) $BITPARAMS(LOADAD.MAP)

    reg_write $REG(GRSCRUB.IDCODE) $FPGA_IDCODE
  }

  # Configure GRSCRUB to map frame addresses of the target FPGA
  proc grscrub_fpgamapping {} \
  {
    variable REG
    variable done

    puts "\nGRSCRUB starting FPGA address mapping"

    grscrub_disable
    grscrub_doneclear
    grscrub_errorclear

    grscrub_init_fpgamappingmode

    #wait
    after 100

    grscrub_enable

    puts "Mapping FPGA configuration memory..."

    # wait OPDONE or SCRERR bitfield of Status register
    while {([expr { $done & [expr [reg_read $REG(GRSCRUB.STAT)]]}] != $done) &&
           ([expr { 0x00000008 & [expr [reg_read $REG(GRSCRUB.STAT)]]}] != 0x8) &&
           ([expr { 0x00000020 & [expr [reg_read $REG(GRSCRUB.STAT)]]}] != 0x20) &&
           ($grmon::interrupt != 1)} {
      # wait if not done
      after 100
    }

    grscrub_disable

    # check if mapped successfully
    if {([expr [reg_read $REG(GRSCRUB.STAT)]] == $done)} {
      puts "GRSCRUB FPGA mapping successfully"
    } else {
      puts "ERROR to map FPGA!!!"
    }
  }


  #####################################################################################
  ### Golden CRC codes
  #####################################################################################

  # Configure the GRSCRUB for golden CRC operation mode
  proc grscrub_init_goldencrc {} \
  {
      variable REG
      variable shift_en
      variable done

      puts "\nStarting golden CRC - to memory"

      grscrub_disable
      grscrub_doneclear
      grscrub_errorclear

      grscrub_init_readbackmode

      #configuration reg
      #    -> opmode = 100 golden crc
      reg_write $REG(GRSCRUB.CONFIG) 0x00000040

      grscrub_enable

      puts "Readding FPGA configuration memory..."

      # wait OPDONE bitfield of Status register
      while {([expr { $done & [expr [reg_read $REG(GRSCRUB.STAT)]]}] != $done) &&
             ($grmon::interrupt != 1)} {
        # wait if not done
        after 100
      }

      # Check error
      if {([expr [reg_read $REG(GRSCRUB.STAT)]] == $done)} {
        puts "GRSCRUB FPGA golden CRC successfully"
      } else {
        puts "ERROR to compute golden CRC!!!"
      }

      grscrub_disable
  }

  #####################################################################################
  ### Readback Scrubbing
  #####################################################################################

  # Configure the GRSCRUB for readback scrubbing operation mode
  proc grscrub_init_readbackmode {} \
  {
    variable REG
    variable BITPARAMS
```

```
        variable fcnt
        variable flen
        variable FPGA_IDCODE
        variable numbermappedframes
        variable partial_en

        puts "GRSCRUB init readback mode"

        # clear error counter and frame id registers
        reg_write $REG(GRSCRUB.ECNT)        0x00000000
        reg_write $REG(GRSCRUB.ERRFRAMEID)  0x00000000
        reg_write $REG(GRSCRUB.FRAMEID)     0x00000000

        #delay (optional, only used in periodic scrubbing)
        reg_write $REG(GRSCRUB.DELAY) 0x10000000

        #golden bitstream addresses
        reg_write $REG(GRSCRUB.LGBAR) $BITPARAMS(START.BIT)
        reg_write $REG(GRSCRUB.HGBAR) $BITPARAMS(END.BIT)


        if {$partial_en==1} {
          # Set partial scrubbing: Example of partial scrubbing in frames of row1 only (KU060)

          #set specific frame address
          reg_write $REG(GRSCRUB.LFAR) 0x00020000

          #set specific number of frames
          #note: only mapped frames
          reg_write $REG(GRSCRUB.FCR) [expr [expr 7500 << 9] | [expr $flen << 2]]

        } else {
          # Full scrubbing of the configuration memory

          #set frame address 0x0
          reg_write $REG(GRSCRUB.LFAR) 0x00000000

          # only mapped frames
          reg_write $REG(GRSCRUB.FCR) [expr [expr $numbermappedframes<<9] | [expr $flen<<2]]

        }

        #mask addr
        reg_write $REG(GRSCRUB.LMASKAR) $BITPARAMS(START.MSK)

        #start frame addr
        reg_write $REG(GRSCRUB.LGSFAR) $BITPARAMS(START.GOLD)

        #map addr in the golden memory
        reg_write $REG(GRSCRUB.LFMAPR) $BITPARAMS(LOADAD.MAP)

        reg_write $REG(GRSCRUB.IDCODE) $FPGA_IDCODE

        #required if crc on
        reg_write $REG(GRSCRUB.LGCRCAR) $BITPARAMS(LOADAD.CRC)

      }

      # Configure GRSCRUB to readback only detection
      proc grscrub_readbackfpga_onlydetection {{datacheck "ffc"}} \
      {
        variable REG
        variable shift_en
        variable done
        variable scrun

        puts "\nStarting readback GRSCRUB IP - only detection"

        grscrub_disable
        grscrub_doneclear
        grscrub_errorclear

```

```
    grscrub_init_readbackmode

    #data verification
    #bit 12  -> FFC
    #bit 11  -> CRC
    if {$datacheck == "ffc"} {
      reg_write $REG(GRSCRUB.CONFIG) 0x0000102C
      puts "FFC selected"
    } elseif  {$datacheck == "crc"} {
      reg_write $REG(GRSCRUB.CONFIG) 0x0000082C
      puts "CRC selected"
    } else {
      #all
      reg_write $REG(GRSCRUB.CONFIG) 0x0000182C
      puts "All methods selected: FFC + CRC"
    }

    # if periodic scrubbing
    if {$scrun == 1} {
      set config_reg [expr ([reg_read $REG(GRSCRUB.CONFIG)])]
      #configuration reg -> scrun = 1
      reg_write $REG(GRSCRUB.CONFIG) [expr $config_reg | 0x2]
      puts "Periodic scrubbing enabled"
      puts "CTRL+C to exit, and grscrub_disable to disable the IP."
    }

    grscrub_enable

    puts "Readding FPGA configuration memory..."

    # wait OPDONE or SCRERR bitfield of Status register
    while {([expr { $done & [expr [reg_read $REG(GRSCRUB.STAT)]]}] != $done) &&
           ($grmon::interrupt != 1) &&
           ([expr { 0x00000020 & [expr [reg_read $REG(GRSCRUB.STAT)]]}] != 0x00000020)} {
      # wait if not done
      after 100
    }

    # check if readback successfully
    if {([expr [reg_read $REG(GRSCRUB.STAT)]] == $done) ||
        ([expr [reg_read $REG(GRSCRUB.STAT)]] == 0x00001010)} {
        set run_error [expr [reg_read $REG(GRSCRUB.ECNT)] & 0x0000FFFF]
        puts "GRSCRUB FPGA readback successfully"
        puts "GRSCRUB Last readback mismatches: $run_error"
    } else {
        puts "ERROR to readback FPGA!!!"
    }

    grscrub_disable
  }

  # Configure GRSCRUB to readback detection and correction
  proc grscrub_readbackfpga_correction {{datacheck "ffc"}} \
  {
    variable REG
    variable shift_en
    variable done
    variable scrun

    puts "\nStarting GRSCRUB readback - correction"

    grscrub_disable
    grscrub_doneclear
    grscrub_errorclear

    grscrub_init_readbackmode

    #data verification
    #bit 12  -> FFC
    #bit 11  -> CRC
    if {$datacheck == "ffc"} {
      reg_write $REG(GRSCRUB.CONFIG) 0x00001024
```

```
          puts "FFC selected"
        } elseif  {$datacheck == "crc"} {
          reg_write $REG(GRSCRUB.CONFIG) 0x00000824
          puts "CRC selected"
        } else {
          #all
          reg_write $REG(GRSCRUB.CONFIG) 0x00001824
          puts "All methods selected: FFC + CRC"
        }

        # if periodic scrubbing
        if {$scrun == 1} {
          set config_reg [expr ([reg_read $REG(GRSCRUB.CONFIG)])]
          #configuration reg -> scrun = 1
          reg_write $REG(GRSCRUB.CONFIG) [expr $config_reg | 0x2]
          puts "Periodic scrubbing enabled"
          puts "CTRL+C to exit, and grscrub_disable to disable the IP."
        }

        grscrub_enable

        puts "Readding FPGA configuration memory..."

        # wait OPDONE or SCRERR bitfield of Status register
        while {([expr { 0x00000010 & [expr [reg_read $REG(GRSCRUB.STAT)]]}] != 0x00000010) &&
               ($grmon::interrupt != 1) &&
               (([expr { 0x000001E0 & [expr [reg_read $REG(GRSCRUB.STAT)]]}] == 0x00000000) ||
               ([expr { 0x000001E0 & [expr [reg_read $REG(GRSCRUB.STAT)]]}] == 0x000000A0))} {
          # wait if not done
          after 100
        }

        # check error
        if {(([expr { 0x00000020 & [expr [reg_read $REG(GRSCRUB.STAT)]]}] == 0x00000020) &&
             ([expr { 0x000001E0 & [expr [reg_read $REG(GRSCRUB.STAT)]]}] != 0x000000A0))} {
          puts "ERROR to readback FPGA!!!"
        } else {
          set run_error [expr [reg_read $REG(GRSCRUB.ECNT)] & 0x0000FFFF]
          set uncor_error [expr [expr [reg_read $REG(GRSCRUB.ECNT)] & 0xFFFF0000] >> 16]
          set corect_errors [expr $run_error-$uncor_error]
          puts "GRSCRUB FPGA readback successfully"
          puts "GRSCRUB Last readback mismatches: $run_error"
          puts "GRSCRUB Correctable errors: $corect_errors"
          puts "GRSCRUB Uncorrectable errors: $uncor_error"
        }

        grscrub_disable
      }

      ##################################################################################
      ### Blind Scrubbing
      ##################################################################################

      ## Configure the GRSCRUB for blind scrubbing operation mode
      proc grscrub_init_blindscrubmode {} \
      {
        variable REG
        variable BITPARAMS
        variable fcnt
        variable flen
        variable scrun
        variable FPGA_IDCODE
        variable numbermappedframes
        variable partial_en

        puts "GRSCRUB init blind scrub mode"

        # if periodic scrubbing
        if {$scrun == 1} {
          reg_write $REG(GRSCRUB.CONFIG) 0x00000022
        } else {
          reg_write $REG(GRSCRUB.CONFIG) 0x00000020
        }

        #delay (optional, only used in periodic scrubbing)
        reg_write $REG(GRSCRUB.DELAY) 0x10000000

        #golden bitstream addresses
        reg_write $REG(GRSCRUB.LGBAR) $BITPARAMS(START.BIT)
        reg_write $REG(GRSCRUB.HGBAR) $BITPARAMS(END.BIT)


        if {$partial_en==1} {
          # Set partial scrubbing: Example of partial scrubbing in frames of row1 only (KU060)

          #set specific frame address
          reg_write $REG(GRSCRUB.LFAR) 0x00020000

          #set specific number of frames
          #note: only mapped frames
          reg_write $REG(GRSCRUB.FCR) [expr [expr 7500 << 9] | [expr $flen << 2]]

        } else {
          # Full scrubbing of the configuration memory

          #set frame address 0x0
          reg_write $REG(GRSCRUB.LFAR) 0x00000000

          # only mapped frames
          reg_write $REG(GRSCRUB.FCR) [expr [expr $numbermappedframes<<9] | [expr $flen<<2]]

        }

        #start frame addr
        reg_write $REG(GRSCRUB.LGSFAR) $BITPARAMS(START.GOLD)

        #map addr in the golden memory
        reg_write $REG(GRSCRUB.LFMAPR) $BITPARAMS(LOADAD.MAP)

        reg_write $REG(GRSCRUB.IDCODE) $FPGA_IDCODE
      }

      # Configure GRSCRUB to blind scrubbing
      proc grscrub_blindscrubbingfpga {} \
      {
        variable REG
        variable done

        puts "\nStarting GRSCRUB blind scrubbing"

        grscrub_disable
        grscrub_doneclear
        grscrub_errorclear

        grscrub_init_blindscrubmode

        grscrub_enable

        puts "Bling Scrubbing FPGA..."

        # wait OPDONE or SCRERR bitfield of Status register
        while {([expr { $done & [expr [reg_read $REG(GRSCRUB.STAT)]]}] != $done) &&
               ($grmon::interrupt != 1) &&
               ([expr { 0x00000020 & [expr [reg_read $REG(GRSCRUB.STAT)]]}] != 0x00000020)} {
          #wait if not done
          after 10
        }

        # check error
        if {([expr [reg_read $REG(GRSCRUB.STAT)]] == $done) ||
            ([expr [reg_read $REG(GRSCRUB.STAT)]] == 0x00001010)} {
          puts "GRSCRUB FPGA Bling Scrubbing successfully"
        } else {
          puts "ERROR to Bling Scrubbing FPGA!!!"
          }

          grscrub_disable
        }

        #####################################################################################
        ### Bit-flip simulation
        #####################################################################################

        # Simulate a bit-flip to quick test the GRSCRUB scrubbing functionality
        # The simplest way to simulate a bit-flip is changing the original golden configuration
        # bitstream in the Golden memory. Thus, the GRSCRUB will check, detect a mismatch, and
        # overwrite the FPGA frame "correcting" the bit-flip.
        # word_addr: address in the golden memory (word must be in the scrubbed frames)
        # bit_pos: bit position between 0 and 31
        proc bitflip_sim {word_addr bit_pos} \
        {

          puts "\nBit-flip simulation"

          # read 32-bit word from golden memory
          set golden_word [silent mem $word_addr 4]
          # puts "Original word: $golden_word"
          puts [format "Original word:0x%08x" $golden_word]

          set faulty_word [expr $golden_word ^ [expr 1 << $bit_pos]]
          # puts "Faulty word: $faulty_word"
          puts [format "Faulty word: 0x%08x" $faulty_word]

          puts "\nWrite faulty word in the Golden memory. A bit-flip should be detected."
          silent wmem $word_addr $faulty_word

          # Select the scrubbing method to test
          # Note: to test the CRC detection, the golden CRC codes must be regenerated with the
faulty word

          # The GRSCRUB should detect and "correct" the word in the target FPGA
          grscrub_readbackfpga_correction "ffc"

          puts "\nWrite original word in the Golden memory. Another bit-flip should be de-
tected."
          silent wmem $word_addr $golden_word

          # The GRSCRUB should detect and "correct" the word in the target FPGA
          grscrub_readbackfpga_onlydetection "ffc"
          #before blind scrubbing, one bit-flip should be detected
          grscrub_blindscrubbingfpga
          #after blind scrubbing, the bit-flip should be corrected
          puts "The bit-flip should be corrected after blind scrubbing."
          grscrub_readbackfpga_onlydetection "ffc"
        }
      }


      #####################################################################################
      ### MAIN
      #####################################################################################

      # Clear old variables
      catch {unset setdesign}
      catch {unset affected_word_addr}
      catch {unset affected_bit}

      # Init variables
      # Select static or leon3mp design
      set setdesign "static"

      # optional bit-flip simulation
      set affected_word_addr 0x400001a4
      set affected_bit 0

      # Select the data check for readback
```

```
    # ffc, crc, or all (ffc + crc)
    #set datacheck "ffc"

    ## Alias to sub namespace procedures
    interp alias {} init_config       {} grscrub::init_config
    interp alias {} grscrub_enable    {} grscrub::grscrub_enable
    interp alias {} grscrub_disable   {} grscrub::grscrub_disable
    interp alias {} grscrub_showregs  {} grscrub::grscrub_showregs
    interp alias {} grscrub_init_goldencrc              {} grscrub::grscrub_init_goldencrc
    interp alias {} grscrub_readbackfpga_onlydetection  {} grscrub::grscrub_readbackfpga_onlydetection
    interp alias {} grscrub_readbackfpga_correction     {} grscrub::grscrub_readbackfpga_correction
    interp alias {} grscrub_blindscrubbingfpga          {} grscrub::grscrub_blindscrubbingfpga
    interp alias {} bitflip_sim                         {} grscrub::bitflip_sim


    ## Execute main procedures ##

    # See system components
    info sys

    # Initial configuration: configure golden memory, programming, and mapping target FPGA
    init_config $setdesign

    # Show GRSCRUB registers
    grscrub_showregs

    #Set golden CRC codes
    grscrub_init_goldencrc

    # Example of how to configure readback scrubbing only detection
    # Select the data check for readback
    # ffc, crc, or all (ffc + crc)
    grscrub_readbackfpga_onlydetection "ffc"
    grscrub_readbackfpga_onlydetection "crc"
    grscrub_readbackfpga_onlydetection "all"

    # Example of how to configure readback scrubbing detection + correction
    # Select the data check for readback
    # ffc, crc, or all (ffc + crc)
    grscrub_readbackfpga_correction "ffc"
    grscrub_readbackfpga_correction "crc"
    grscrub_readbackfpga_correction "all"

    # Example of how to configure blind scrubbing
    grscrub_blindscrubbingfpga

    # Bit-flip simulation to test the scrubbing correction (optional)
    bitflip_sim $affected_word_addr $affected_bit

    ### End of example of GRSCRUB IP configuration ###
```