GR716B FPGA Scrubber Controller Application Note

Application Note
Doc. No GRHA-AN-0001
Issue 1.0

2024-11-25

Contract 4000130767/20/NL/MM/gm
Deliverable TN-7

	Function	Name	Signature and date
Prepared	Radiation Effects	Ádria B. de Oliveira	2024-11-25
Approved	Radiation Effects	Lucas A. Tambara	2024-11-26
Checked	Hardware	Anandhavel Sakthivel	2024-11-26

CHANGE RECORD

Issue	Date	Section / Page	Description
1.0	2024-11-25		First issue of this document.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 Purpose and Scope of the Document

This document presents the application note for the Field Programmable Gate Array (FPGA) scrubber controller featured in the Frontgrade Gaisler's GR716B radiation-hardened microcontroller [AD1]. The scope of this document is limited to demonstrating the programming and scrubbing capabilities of GR716B targeting an AMD/Xilinx Kintex UltraScale KU060 FPGA.

This document is part of the deliverables within the activity "GR716B Rad-Hard Microcontroller for Space Applications" initiated by the European Space Agency (ESA) under Advanced Research in Telecommunications Systems (ARTES) Competitiveness & Growth, contract 4000130767/20/NL/MM/gm.

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

1.2 Applicable Documents

The following documents, listed in order of precedence, contain requirements applicable to the contents of the document:

[AD1] Frontgrade Gaisler, "GR716B Advanced Data Sheet and User's Manual", version 0.8, 2024.

[AD2] Frontgrade Gaisler, "GRLIB VHDL IP Core Library", version 2024.2, July 2024.

[AD3] Frontgrade Gaisler, "GRMON4 User's Manual", version 4.0.1, 2024.

[AD4] Frontgrade Gaisler, "GR716B Preliminary User's Manual", GR716B-BOARD-UM, version 0.1, July 2024.

[AD5] Frontgrade Gaisler, "GR-CPCIS-XCKU Data Sheet and User's Manual", GR-CPCIS-XCKU-DSUM, version 1.5, Nov. 2023.

[AD6] Frontgrade Gaisler, "GRSCRUB FPGA Error Injection Framework User Manual", GRSCRUB-SPEC-0001, issue 1.0, Dec. 2022.

1.3 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 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] AMD/Xilinx, "UltraScale Architecture Soft Error Mitigation Controller LogiCORE IP Product Guide," PG187, June 2024.

[RD5] Á. Oliveira *et al.*, "NOEL-V FT and GRSCRUB IP: Fault Tolerance Characterization of a Complex System-on-Chip on Xilinx Kintex UltraScale FPGA," 2022 RADECS, Venice, Italy, 2022, pp. 1-5, doi: 10.1109/RADECS55911.2022.10412477.

[RD6] AMD/Xilinx, "UltraScale Architecture Configuration User Guide," UG570 (v1.19), June 2024.

2 ABBREVIATIONS

ARTES	Advanced Research in Telecommunications Systems
CMOS	Complementary Metal Oxide Semiconductor
CRAM	Configuration memory RAM
CRC	Cyclic Redundancy Check
DSU	Debug Support Unit
ECC	Error Correction Code
EDAC	Error Detection and Correction
EIFW	Error Injection Framework
ESA	European Space Agency
ESTEC	European Space Research and Technology Center
FF	Flip-Flop
FFC	Full Frame Check
FPGA	Field Programmable Gate Array
GPIO	General-Purpose Input/Output
I/O	Input and Output
IP	Intellectual Property
LUT	Lookup Table
NDSEE	Non-Destructive Single Event Effects
PLL	Phase-Locked Loop
PSU	Power Supply Unit
SDC	Silent Data Corruption
SEE	Single Event Effects
SEFI	Single Event Functional Interrupt
SEM-IP	Soft Error Mitigation Intellectual Property
SET	Single Events Transient
SEU	Single Event Upset
SMAP	SelectMap
SoC	System-on-Chip
SPI	Serial Peripheral Interface
SPIMCTRL	SPI Memory Controller
TMR	Triple Modular Redundancy
XO	Crystal Oscillator

3 BACKGROUND

This section provides a background of Non-Destructive Single Event Effects (NDSEE) in SRAM-based FPGAs and possible mitigation methods.

3.1 Soft Error Mitigation in SRAM-based FPGAs

The FPGA configuration memory defines the architecture of the design implemented in lookup tables (LUT), flip-flops (FF), input and output (I/O) interconnections, routing tables, and clock lines, for instance. To map such elements, the configuration memory is usually divided into frames that contains 32-bit word data. The number of words in a frame and the number of frames in the configuration memory varies depending on the FPGA family.

NDSEEs are radiation-induced soft errors provoked by ionized particles that affect the system without damaging the device permanently. FPGAs 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 on their configuration memory RAM (CRAM) due to the memory elements used to configure the design logic.

Single Event Upsets (SEU) affecting the CRAM may lead to persistent errors in the system, changing the architectural implementation of the design. 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, latches, and flip-flops, and cause Silent Data Corruptions (SDC), which are incorrect application results. The Single Event Functional Interrupt (SEFI) occurs when a soft error affects the control logic or a state register and leads to a hang or a crash in the design.

CRAM scrubbing is a well-known fault tolerance technique responsible for coping with errors in the configuration memory and avoiding their accumulation. It restores the CRAM frames by rewriting the correct configuration frame-by-frame. The scrubbing operation does not interfere in the design execution since it targets the static layer of the FPGA configuration memory. There are two scrubbing approaches. The blind scrubbing rewrites the CRAM frames without a prior error check, which means that all frames are refreshed whether they present an error or not. On the other hand, the readback scrubbing first reads a frame, check for errors (error detection), and the frame is refreshed only in case of bit-flip detection (error correction). The primary difference between both approaches is that readback provides the error rate per scrubbing cycle.

A scrubbing is defined as internal when the scrubber engine is embedded inside the FPGA being monitored. An external scrubbing is performed when the scrubber engine is located externally to the target FPGA. The literature presents several scrubbing implementations that mainly differ in error detection, power consumption, resource usage, and correction speed [RD1, RD2, RD3]. The AMD/Xilinx Soft Error Mitigation Intellectual Property (SEM-IP) core [RD4] is an example of internal scrubbing included in most AMD/Xilinx FPGAs.

One must notice that the scrubbing technique does not avoid bit-flips from happening or its effects on the design. Additionally, memory elements that store dynamic data, such as Block RAMs (BRAM), distributed memory, and flip-flops, are not protected by the CRAM 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 the logic is also 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 reprogramming or power cycle might be required to restore the system.

In this context, the GR716B microcontroller features an FPGA scrubber controller with programming and scrubbing capabilities that aims at monitoring the target FPGA configuration memory, correcting errors, and avoiding the accumulation of upsets. The next section details the features of the GR716B FPGA scrubber controller.

4 GR716B MICROCONTROLLER

The Frontgrade Gaisler's GR716B is a fault-tolerant mixed-signal microcontroller implemented using Imec's DARE180 radiation-hardened cell library in a 180nm Complementary Metal Oxide Semiconductor (CMOS) technology. Figure 1 shows the GR716B block diagram. The GR716B microcontroller is based on the fault-tolerant LEON3FT SPARC V8 processor featuring a 128KiB Error Detection and Correction (EDAC) protected tightly coupled memory, and a double precision IEEE-754 floating point unit (FPU). The GR716B also features memory protection units, non-intrusive advanced on-chip debug support unit (DSU), real-time accelerators, 2-port SpaceWire router, MIL-STD-1553B interface, CAN FD interface, 10/100 Ethernet, FPGA scrubber controller, programmable PWM interface, DACs and ADCs, and fast analog comparators. More information is presented in the GR716B data sheet and user's manual [AD1].

This document focuses on the FPGA supervisor and scrubber controller feature of the GR716B microcontroller. The FPGA supervisor is highlighted in red in Figure 1 and is further described in the following sections.

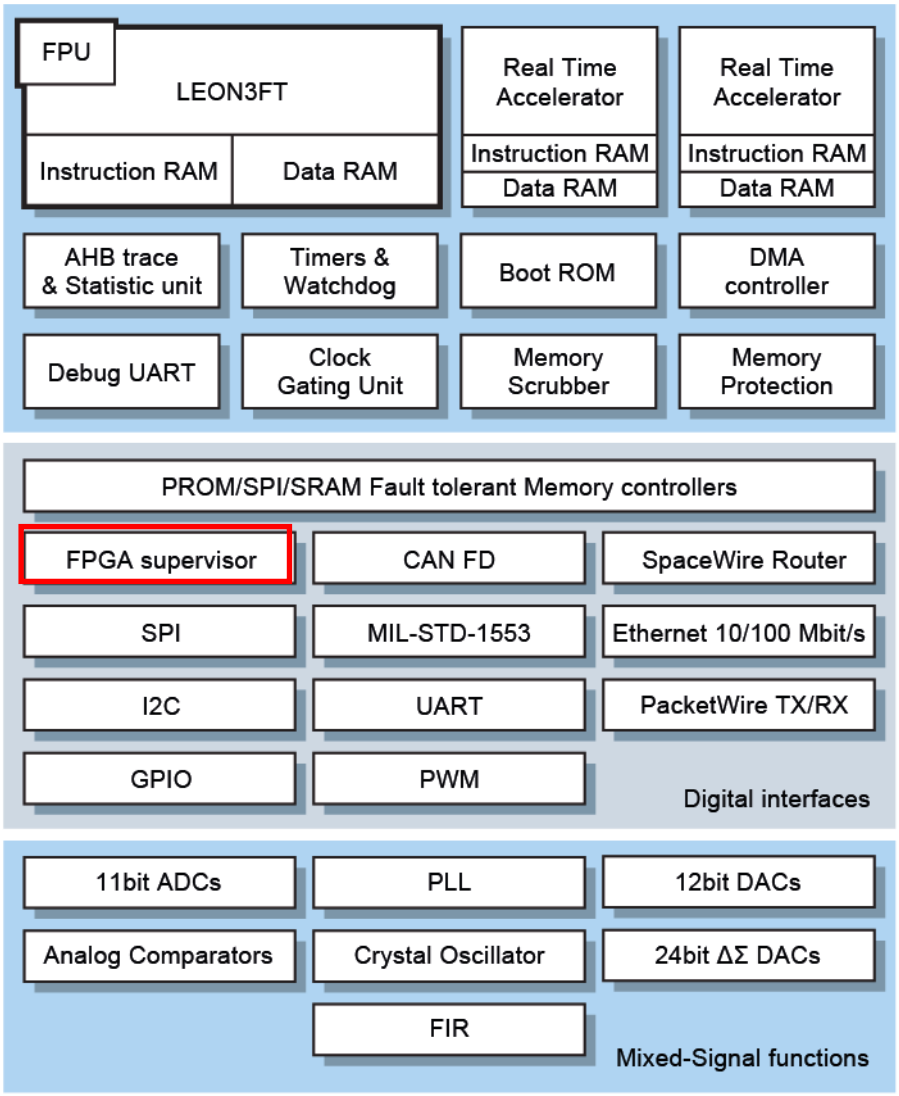

Figure 1. GR716B block diagram.

4.1 FPGA Scrubber Controller

The GR716B FPGA scrubber controller is based on the GRSCRUB IP core from the GRLIB [AD2]. For simplicity, the GR716B FPGA scrubber controller is hereafter referred to as GRSCRUB. One should refer to the GR716B data sheet [AD1] for further information about the GRSCRUB usage in the GR716B microcontroller.

The GRSCRUB is an external FPGA configuration monitor that features programming and scrubbing capabilities. After the initial configuration, the GRSCRUB is self-standing, which releases the processor core to perform other tasks. The GRSCRUB is compatible with the AMD/Xilinx Kintex UltraScale and Virtex-5 FPGA families. It accesses the target FPGA configuration memory externally through the SelectMap (SMAP) interface.

As previously introduced, CRAM scrubbing prevents the accumulation of radiation-induced soft errors in the configuration memory of SRAM-based FPGAs. The GRSCRUB can detect and correct single and multiple errors affecting the FPGA configuration memory. However, the scrubbing technique does not prevent upsets from happening or its effects on the design. Therefore, additional mitigation techniques at the design level are recommended to decrease the number of single points of failure in the system and increase the fault masking, as described in Section 3.

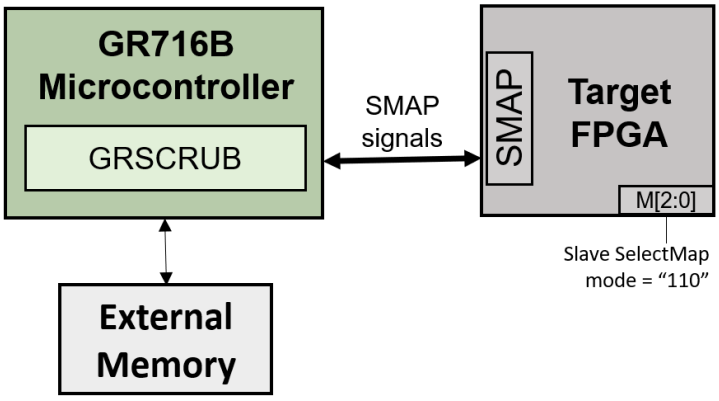

Figure 2. Simplified example of user-case setup.

Figure 2 shows a simplified example of a setup using the GRSCRUB feature of the GR716B microcontroller for supervising a target FPGA. Note that the FPGA M[2:0] pins must be pre-configured to Slave SelectMap mode (i.e., M[2:0]=”110”). The GR716B is connected to the SelectMap interface of the FPGA via General-Purpose Input/Output (GPIO) pins (from GPIO[25] to GPIO[38]). The GR716B allows using 8-bit data and all control pins required to program and scrub the target FPGA. Table 1 describes the functionality of the GR716B GPIOs used to interface the SelectMap. The proper configuration of the GR716B GPIO pins are presented later in this document.

An external Serial Peripheral Interface (SPI) Flash memory stores the bitstream used to program and scrub the target FPGA. The external memory is referred to as golden memory since it stores all the golden data required for the FPGA supervising functions. Besides the bitstream, the external memory should store the FPGA mask data, the FPGA frame addresses (mapping information), and the Cyclic Redundancy Check (CRC) codes (used for error detection during scrubbing). The external memory must have enough storage space and be loaded prior to the usage of GRSCRUB. More information about the required data to be stored in the external memory is available later in this document. One should refer to the GR716B data sheet [AD1] for further information about the usage of the SPI memory controller and memory connections.

Table 1. GR716B GPIO pins description.

GR716B GPIO	Pin direction	Interface name	Function
GPIO[25]	Input	INITN	FPGA initialization
GPIO[26]	Input	DONE	FPGA programming done
GPIO[34:27]	Input/Output	DATA[7:0]	In/out 8-bit data
GPIO[35]	Output	PROGN	FPGA configuration clear
GPIO[36]	Output	RDWR	SelectMap read/write
GPIO[37]	Output	CSIN	SelectMap chip select
GPIO[38]	Output	SCLK	SelectMap clock

4.1.1 Operation Modes

The main operation modes of GRSCRUB in the GR716B microcontroller are:

- **Programming mode:** GRSCRUB programs the configuration bitstream into the target FPGA.
- **Scrubbing mode:** GRSCRUB executes a scrubbing operation. Two scrubbing methods are supported: blind and readback scrubbing. In both cases, the scrubbing can be performed targeting the entire FPGA configuration memory or just selected frames, and the execution can be one-time or periodic.
  - *Blind scrubbing:* GRSCRUB rewrites each configured frame without prior verification. The frames are rewritten with the golden data stored in the golden memory.
  - *Readback scrubbing:* GRSCRUB detects and corrects errors in the configured frames. The error detection is performed by reading a frame and checking for inconsistencies by comparing the read data with the golden data. The error detection can be performed through Full Frame Check (FFC) or CRC verification. In the FFC mode, the error detection occurs word-by-word since the GRSCRUB compares each read 32-bit frame word with the corresponding 32-bit golden word stored in the golden memory. At the first error detected, the entire frame is corrected. In the CRC mode, the GRSCRUB reads the frame from the FPGA, computes a 32-bit CRC code based on all frame words, and compares it with the corresponding golden CRC. In case of CRC mismatches, the frame is corrected. All error detection and correction counters are saved in the GRSCRUB registers.

4.1.2 External Memory Configuration

All the required data must be stored in the external memory so that GRSCRUB can properly execute its operations. The required data depends on the applicable GRSCRUB operations planned for a specific system. Table 2 details the required golden memory data per GRSCRUB operation. Section 5.2 describes an example of how to load the external memory data using the GR716B microcontroller.

Table 2. Required memory data per GRSCRUB operation.

Memory data	Prog.	Blind scrub.	Readback scrub.		Description
			FFC	CRC	
Bitstream	X	X	X	X	FPGA configuration bitstream (*.bit* file from Vivado tool) is mandatory for all operations since it is necessary for programming and scrubbing the FPGA. The entire bitstream must be loaded to the golden memory.
Mask			X	X	FPGA configuration mask (*.msk* file from Vivado tool) is required for readback scrubbing. GRSCRUB uses the mask data to identify the dynamic bits in the FPGA CRAM frames. Only the data related to the scrubbed frames are needed to be stored. For simplicity, the entire mask file can be loaded to the golden memory. The mask file has the same format as the bitstream file. If the user decides to store only the mask data related to the scrubbed frames, the correlation between mask and bitstream must be matched. Misalignment will cause erroneous verification of the bits.
Frame mapping		X*	X	X	The frame mapping stores the addresses of all FPGA configuration frames in the CRAM. The frame addresses are required in case of error correction of a specific frame. Therefore, frame mapping is mandatory for the readback scrubbing. *The blind scrubbing only uses the frame mapping if a limited number of frames is configured to be scrubbed at once (BLKFRAME>0) in the SETUP register. If no maximum limit is established (BLKFRAME=0), the frame mapping is not required for blind scrubbing.
CRC codes				X	The golden CRC codes are only required to be stored in the golden memory for the readback scrubbing with CRC error detection.

5 SYSTEM CONFIGURATION

This section describes the required steps for system configuration in order to use the GRSCRUB in GR716B microcontroller. The how to steps focuses on the golden data generation, the storage of the data in the external memory, and the configuration of the GR716B microcontroller and GRSCRUB registers. Section 55 of the GR716B data sheet provides further details of the GRSCRUB configuration [AD1].

Annex A provides demonstration scripts to configure and control the GR716B and GRSCRUB. The scripts are written in TCL and are intended to be used on the Frontgrade Gaisler's GRMON debug tool [AD3]. The scripts can be used as a baseline for software development.

5.1 Data Generation

5.1.1 FPGA Design Implementation

For the validation presented in this document, a KU060 FPGA design was implemented based on the fault-tolerant NOEL-V processor and GRLIB IPs. The design has been triplicated using distributed Triple Modular Redundancy (TMR) synthesis strategy. Demonstrating the design implementation is outside the scope of this document. For further information, refer to [RD5].

Table 3 presents the KU060 FPGA resource usage of the triplicated NOEL-V System-on-Chip (SoC).

Table 3. Resource usage of example design implemented in the KU060 FPGA.

LUT	FF	Carry	BRAM	DSP
211,643	74,842	2,744	126	39

5.1.2 Bitstream and Mask Files

The FPGA bitstream and mask files can be generated using the AMD/Xilinx Vivado tool. It is not in the scope of this document to demonstrate this step.

One should notice that to allow the GRSCRUB to access and control the slave SelectMap interface, the generated FPGA bitstream must be configured following the requirements below:

- Enable the mask file generation in the tool.
- Do not prohibit readback in the configuration bitstream security settings.
- Do not use compression or encryption in the configuration bitstream.
- Set the SelectMap pins to persistent in the configuration bitstream generator.
- Configure the slave SelectMap interface.

Below is an example of design constraints that can be used:

```
# Do not compress the bitstream
set_property BITSTREAM.GENERAL.COMPRESS FALSE [current_design]
# Configuration interface pins are persistent
set_property BITSTREAM.CONFIG.PERSIST YES [current_design]
# Select Slave SelectMap interface
set_property CONFIG_MODE S_SELECTMAP [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]
```

5.1.3 Frame Mapping

The mapping file with the frame addresses of the KU060 FPGA was previously generated using the GRSCRUB IP. The GRSCRUB mapped and saved the frame addresses from the KU060 into a DDR memory. The DDR memory content was dumped to a *.srec* file via GRMON.

The *GNU objcopy* utility was used to update the memory addresses in the *.srec* file to the external memory addresses used in this document (i.e., start address=0x13D0000 – more information is provided in Section 5.2). The *.srec* file is stored in the package provided in Annex A.

5.1.4 Golden CRC Codes

The golden CRC codes for the FPGA configuration frames must be recomputed for each specific bitstream and mask files since the encoding varies with the design data. A Python script was developed to simplify the CRC code generation (see Annex A). The script computes the CRC32 code as per GRSCRUB and each CRC32 code is related to the respective CRAM frame. The user should input the generated bitstream and mask files (*.bit* and *.msk* files from the Vivado tool) to the script. The output is a binary *.bin* file with the golden CRC codes. This file can be directly loaded to the golden memory.

5.2 Loading External Memory Data

Before starting using the GRSCRUB in the GR716B microcontroller the user must ensure that the external SPI Flash memory is properly loaded. As previously described in Table 2, the external memory should store the FPGA bitstream and mask files, the frame mapping, and the golden CRC codes, depending on the required operations executed by GRSCRUB. This section shows an example of how to load all required data, considering an AMD/Xilinx Kintex UltraScale KU060 FPGA, to an external SPI Flash memory connected to the GR716B microcontroller.

This example uses a TCL script to be executed on GRMON. In this setup, the target SPI Flash memory has a density of 512 Mb (64 MB) and is connected to the SPI Memory Controller 0 (SPIMCTRL0) of the GR716B microcontroller (refer to the GR716B data sheet for further information [AD1]). Therefore, the SPIMCTRL0 must be configured to use 4-byte addresses for the memory space.

Table 4 shows the size and the start address used to load the required data into the external SPI Flash memory. The selected base address is 0x10D00000. For simplicity, the start addresses are selected with enough room between the data. Additionally, the entire mask file is loaded to avoid misalignment issues.

Table 4. Size and start address of data stored in the external SPI Flash memory.

Memory data	Size (KU060)	Start memory address
Bitstream	23 MB	0x10D00006
Mask	23 MB	0x12500006
Frame mapping	146 kB	0x13D00000
CRC codes	146 kB	0x13F00000

One should note that the FPGA configuration bitstream and mask files include an initial header with ASCII characters that provides some file information. The header information may vary in size and content, which can misalign the binary data into the memory words. In order to load the binary data to the memory with the correct alignment, one can consider shifting the start load address (e.g., the start address shift of 0x6 for the bitstream and mask data in Table 4). To define the correct shift, the first binary *dummy* word (0xFFFFFFFF) can be used. See the AMD/Xilinx configuration user guide [RD6] for more details about the bitstream composition.

The TCL script below shows an example of how to load the data to the SPI Flash memory. One can source the script to GRMON using: *grmon> source <script name>.tcl*.

```
############################################################
## Script to load required data to the SPI Flash memory
## One must update the required file names where defined (< >)
############################################################

# Set SPI Flash memory addresses (SPIMCTRL0)
set LOADAD_BIT      0x10D00006
set LOADAD_MSK      0x12500006
set LOADAD_MAP      0x13D00000
set LOADAD_CRC      0x13F00000

# SPIMCTRL0 clock gate enable
grcg enable 5 grcg0
# Set SPIMCTRL0 4-byte address and alternate scaler
wmem 0xfff00100 0x131003
wmem 0xfff00104 0x0000000c

# Detect memory
spim flash detect

# Load bitstream data
spim flash load -erase <KU060_bitstream>.bit $LOADAD_BIT
verify -max 10 <KU060_bitstream>.bit $LOADAD_BIT
# Load mask data
spim flash load -erase <KU060_mask>.msk $LOADAD_MSK
verify -max 10 <KU060_mask>.msk $LOADAD_MSK
# Load frame mapping data
spim flash load -erase <KU060_map>.srec $LOADAD_MAP
verify -max 10 <KU060_map>.srec $LOADAD_MAP
# Load CRC data
spim flash load -erase <CRC_codes>.bin $LOADAD_CRC
verify -max 10 <CRC_codes>.bin $LOADAD_CRC
```

5.3 GR716B Microcontroller Configuration

The GR716B microcontroller requires a few configurations after reset:

- Enable clock for the SPIMCTRL0 core.
- Configure SPIMCTRL0 for 4-byte address.
- Enable the SPIMCTRL0 alternate scaler for the dividing the SPI clock (optional). If the alternate scaler is enabled, the system clock frequency is divided by 2. The default scaler is system clock divided by 8. Note that the alternate scalar should only be enabled with internal system clock equal or below 50 MHz (i.e., the maximum SPIMCTRL0 clock is 25 MHz).
- Enable clock for the GRSCRUB core.
- Configure the external SelectMap clock (SCRUBBER_CLK). In this example, the reference clock divisor is set to 4.
- Configure the GRSCRUB GPIOs.
- Configure the GRSCRUB registers. This step is described in Section 5.4.

The example below shows a TCL script that can be used to configure the clock and registers of the SPIMCTRL0 and GRSCRUB cores. Note that the SCRUBBER_CLK is dependent on the GR716B input system clock, and the SPIMCTRL0 alternate scaler depends on the internal system clock and Phase-Locked Loop (PLL) settings. See the GR716B data sheet for more information [AD1].

```
############################################################
## GR716B microcontroller configuration
############################################################

## SPIMCTRL0 configuration
# enable SPIMCTRL0 clock
grcg enable 5 grcg0
# force 4 byte address mode (F4B)
set spim_reg [silent mem 0xFFF00100 4]
silent wmem 0xFFF00100 [expr $spim_reg | [expr 1 << 12]]
# use 4 address bytes (ADDRBYTES)
set spim_reg [silent mem 0xFFF00100 4]
silent wmem 0xFFF00100 [expr $spim_reg & ~[expr 3 << 8]]
# enable alternate scaler (EAS)
set spim_reg [silent mem 0xFFF00104 4]
silent wmem 0xFFF00104 [expr $spim_reg | [expr 1 << 2]]

## GRSCRUB configuration
# enable GRSCRUB clock
grcg enable 22 grcg1
# configure SCRUBBER_CLK config (external SYS_CLK/4)
silent wmem 0x8010D010 0x04
```

```
############################################################
## GR716B microcontroller configuration (cont.)
############################################################

## Configure the IO matrix
# set GRSCRUB GPIOs in SYS.CFG.GP3
set cfg_gp_reg [silent mem 0x8000D00C 4]
set cfg_gp_reg [expr $cfg_gp_reg & ~0xFFFFFFF0]
set cfg_gp_reg [expr $cfg_gp_reg | 0xEEEEEEE0]
silent wmem 0x8000D00C $cfg_gp_reg
# set GRSCRUB GPIOs in SYS.CFG.GP4
set cfg_gp_reg [silent mem 0x8000D010 4]
set cfg_gp_reg [expr $cfg_gp_reg & ~0x0FFFFFFF]
set cfg_gp_reg [expr $cfg_gp_reg | 0x0EEEEEEE]
silent wmem 0x8000D010 $cfg_gp_reg
# no pull resistors
silent wmem 0x8000D028 0x0
silent wmem 0x8000D02C 0x0
# disable schmitt trigger
silent wmem 0x8000D038 0x0
silent wmem 0x8000D03C 0x0
```

5.4 GRSCRUB Configuration

Before starting using the GRSCRUB, the correct configuration should be set in its registers. After its initial configuration, the GRSCRUB can run autonomously from the microcontroller core. This section shows some examples of how to configure the GRSCRUB registers based on the data loaded in the golden memory and the selected operation mode. In the following example, a TCL script is used in GRMON tool for the GRSCRUB configuration. This section refers to portions of the TCL script, and Annex A points to its complete version.

5.4.1 Registers Settings

The GRSCRUB base address in GR716B is 0x80404000. For simplicity, shift addresses can be set in a TCL script for the register's addresses configuration, as *REG* in the example below.

The GRSCRUB registers must be configured based on the addresses defined in the golden memory (e.g., the external SPI Flash memory addresses described in Section 5.2). In our example, the base data address in the SPI Flash memory is 0x10D00000, and the start addresses are defined in Table 4. Pre-defined variables can be stablished in TCL to be reused during the registers configuration.

```
############################################################
## GRSCRUB register configuration
############################################################

# GRSCRUB registers base address and offsets
set GRSCRUB_REGADDR 0x80404000
array set REG {
      GRSCRUB.STAT          0x00000000
      GRSCRUB.CONFIG        0x00000004
      GRSCRUB.IDCODE        0x00000008
      GRSCRUB.DELAY         0x0000000C
      GRSCRUB.FCR           0x00000010
      GRSCRUB.LFAR          0x00000014
      GRSCRUB.LGBAR         0x00000018
      GRSCRUB.HGBAR         0x0000001C
      GRSCRUB.LGSFAR        0x00000020
      GRSCRUB.LMASKAR 0x00000024
      GRSCRUB.LFMAPR        0x00000028
      GRSCRUB.LGCRCAR  0x0000002C
      GRSCRUB.LGRBKAR  0x00000030
      GRSCRUB.ECNT          0x00000034
      GRSCRUB.SETUP         0x00000038
      GRSCRUB.CAP           0x0000003C
      GRSCRUB.FRAMEID   0x00000040
      GRSCRUB.ERRFRAMEID  0x00000044
   }

# Golden memory base address
set MEM_BASE 0x10D00000
# configuration bitstream
set BITPARAMS(LOADAD.BIT)       [expr $MEM_BASE + 0x00000006]
# mask data
set BITPARAMS(LOADAD.MSK)       [expr $MEM_BASE + 0x01800006]
# frame map addresses
set BITPARAMS(LOADAD.MAP)       [expr $MEM_BASE + 0x03000000]
# golden CRC data (required for readback mode with CRC check)
set BITPARAMS(LOADAD.CRC)       [expr $MEM_BASE + 0x03200000]
```

The addresses for the bitstream, mask, frame mapping, and golden CRC codes are a reference to the golden data. Additionally, the following information must also be configured to GRSCRUB registers (all addresses refer to the golden data in the external memory):

- **The bitstream starting address (after header):** as previously informed, the configuration bitstream has an initial ASCII header that has readable information about the bitstream but is not used to configure the FPGA. Therefore, the address of the first binary word (*dummy* word 0xFFFFFFFF) in the bitstream should be configured to the GRSCRUB LGBAR register.
- **The bitstream last address:** the GRSCRUB HGBAR register should be set to the address of the last word in the bitstream.
- **Address of the first word of the first scrubbed frame in the bitstream:** the GRSCRUB LGSFAR register should be set to the address of the first word of the first frame to be scrubbed in the bitstream. For example, if the first frame to be scrubbed is frame id 0 (first bitstream frame), the LGSFAR should point to the address of the first word in the 1st frame. This case is simpler since the first frame starts just after the initial synchronization words in the bitstream. In case the first frame to be scrubbed is frame id 9, for instance, then the LGSFAR

should point to the address of the first word in the 10th frame. One should make sure that the correct alignment is set.

- **Address of the first mask word of the first scrubbed frame in the mask data:** since the scrubbed frames should have their corresponding mask data, the GRSCRUB LMASKAR register should be set to the address of the first mask word of the first frame to be scrubbed. The logic is similar to the LGSFAR register. The LMASKAR and LGSFAR must be aligned in order to point to the same frame id, i.e., addresses in the external memory that correspond to the same frame id in the bitstream and mask.

Since defining such addresses can be a complex task, a *config_gold_addr* procedure in a TCL script (presented in Annex A) has been created to automatically compute them based on the base addresses of the primary data. Below is an example of how to manually set these addresses.

```
############################################################
## GRSCRUB register configuration (cont.)
############################################################

## Define the start addresses in the golden memory
# Start address of the configuration bitstream in the golden memory.
# The LGBAR register is set with this address.
set BITPARAMS(START.BIT)       [expr $MEM_BASE + 0x00000090]
# 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 + 0x01701e78]
# Address of the first scrubbed frame in the golden memory (here is set to frame id 0)
# The LGSFAR register is set with this address.
set BITPARAMS(START.GOLD)    [expr $MEM_BASE + 0x000001a8]
# Address of the first mask data related with the first scrubbed frame in the golden memory.
# The LMASKAR register is set with this address.
set BITPARAMS(START.MSK)     [expr $MEM_BASE + 0x018001a8]
```

Other configurations are also required, such as the total number of frames presented in the FPGA (*FCNT*), the number of words per FPGA frame (*FLEN*), the number of frames to be scrubbed by GRSCRUB (*NUM_SCRUB_FRAMES*), and the address of the first FPGA frame to be scrubbed. The FPGA id code (*FPGA_IDCODE*) must also be configured to the GRSCRUB. The example below shows the correct configuration for KU060 FPGA. One should refer to the specific FPGA data sheet to find its correct configuration if a different FPGA is used.

If all frames of the FPGA configuration memory should be scrubbed by GRSCRUB, then *FSTART_ADDR* is set to the address of the first frame of the configuration bitstream (i.e., `0x00000000`), and the *NUM_SCRUB_FRAMES* 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, then *FSTART_ADDR* and *NUM_SCRUB_FRAMES* should be set accordingly.

The GRSCRUB SETUP register should be configured to reflect the FPGA correct settings. For instance, the number of row boundaries in the FPGA frame addressing, and whether bit swapping the data is enabled or not. The KU060 FPGA has the SelectMap data bit-swapped, and there are two rows between frame regions on the FPGA mapping architecture. One can also select the GRSCRUB waiting time for the FPGA response after starting the programming phase. A minimum of one microsecond is recommended.

The maximum number of frames to be scrubbed at once (BLKFRAME) can also be configured in the SETUP register. This value establishes the number of frames that are read sequentially during readback or written sequentially during blind scrubbing. If BLKFRAME > 0, the frames are scrubbed by blocks, and a maximum of BLKFRAME frames is scrubbed each time. Therefore, the number of frames defined in the FCNT bitfield of the Frame Configuration Register (FCR) is split into the BLKFRAME size. This feature can be used to have more control of the FPGA configuration interface and to reduce the impact of soft errors affecting the interface. If BLKFRAME = 0, all frames set in the FCR are scrubbed sequentially at once.

```
############################################################
## GRSCRUB register configuration (cont.)
############################################################

# Total number of configuration frames of KU060
set FCNT 49030
# Frame length of KU060
set FLEN 123
# FPGA id code of KU060
set FPGA_IDCODE 0x03919093
# Number of frames to be scrubbed (only mapped frames can be scrubbed)
set NUM_SCRUB_FRAMES 37498
# address of the first FPGA frame to be scrubbed
set FSTART_ADDR 0x0

# set block frame constant
set SET_BLKFRAME 0 ; # (0) all frames at once; (>0) limited frames

proc set_default_setup { } {
    variable REG
    variable SET_BLKFRAME

    # clear setup register
    reg_write $REG(GRSCRUB.SETUP) 0

    # ROWBND = 2
    set setup [reg_read $REG(GRSCRUB.SETUP)]
    reg_write $REG(GRSCRUB.SETUP) [expr $setup | [expr 2 << 4]]
    # TPROG = 150
    set setup [reg_read $REG(GRSCRUB.SETUP)]
    reg_write $REG(GRSCRUB.SETUP) [expr $setup | [expr 150 <<12]]
    # BITSWPEN = 1
    set setup [reg_read $REG(GRSCRUB.SETUP)]
    reg_write $REG(GRSCRUB.SETUP) [expr $setup | [expr 1 <<21]]
    # block frame config
    if {$SET_BLKFRAME} {
        set setup [reg_read $REG(GRSCRUB.SETUP)]
        reg_write $REG(GRSCRUB.SETUP) [expr $setup | [expr $SET_BLKFRAME <<22]
    }
}
```

5.4.2 Operation Control

5.4.2.1 FPGA Programming

Before enabling the programming operation mode, the GRSCRUB registers must be configured. The *grscrub_init_progmode* procedure configures the GRSCRUB CONFIG, LGBAR, HGBAR, and IDCODE registers based on the example provided in this document. The registers are configured with values defined in Section 5.4.1.

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 GRSCRUB STATUS register;
3) configure the required registers (*grscrub_init_progmode* procedure); and
4) enable the GRSCRUB 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.

See an example of script procedures below.

```
############################################################
## Configuration for FPGA programming mode
############################################################

proc grscrub_init_progmode {} {
    variable REG
    variable BITPARAMS
    variable FPGA_IDCODE

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

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

    reg_write $REG(GRSCRUB.IDCODE) $FPGA_IDCODE
}
```

```
#############################################################
## Configuration for FPGA programming mode (cont.)
#############################################################

proc grscrub_progfpga {} {
    variable REG
    variable done

    grscrub_disable
    grscrub_doneclear
    grscrub_errorclear
    grscrub_init_progmode

    # GRSCRUB operation enable
    grscrub_enable

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

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

    grscrub_disable
}
```

5.4.2.2 FPGA Scrubbing

5.4.2.2.1 Blind Scrubbing

Before enabling the blind scrubbing operation mode, the GRSCRUB registers must be configured. The *grscrub_init_blindscrubmode* procedure configures the GRSCRUB CONFIG, DELAY, LGBAR, HGBAR, LFAR, FCR, LGSFAR, LFMAPR, and IDCODE registers based on the example provided in this document. The registers are configured with values defined in Section 5.4.1.

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) enable the GRSCRUB 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 internal 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 is performing the scrubbing operation on the target FPGA. The HOLD bitfield is 1 when the GRSCRUB is on 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.

See an example of script procedures below.

```
############################################################
## Configuration for GRSCRUB blind scrubbing operation mode
############################################################

proc grscrub_init_blindscrubmode {}  {
    ## variable definition here (removed due to limited space – see Annex A for complete example) ##

    # set one time scrubbing and no delay
    reg_write $REG(GRSCRUB.CONFIG) 0x00000020
    reg_write $REG(GRSCRUB.DELAY) 0x0

    #golden memory addresses
    reg_write $REG(GRSCRUB.LGBAR) $BITPARAMS(START.BIT)
    reg_write $REG(GRSCRUB.HGBAR) $BITPARAMS(END.BIT)
    reg_write $REG(GRSCRUB.LMASKAR) $BITPARAMS(START.MSK)
    reg_write $REG(GRSCRUB.LGSFAR) $BITPARAMS(START.GOLD)
    reg_write $REG(GRSCRUB.LFMAPR) $BITPARAMS(LOADAD.MAP)

    #set frame address and configuration
    reg_write $REG(GRSCRUB.LFAR) $FSTART_ADDR
    reg_write $REG(GRSCRUB.FCR) [expr [expr $NUM_SCRUB_FRAMES << 9] | [expr $FLEN << 2]]

    reg_write $REG(GRSCRUB.IDCODE) $FPGA_IDCODE
}
```

```
#############################################################
## Configuration for GRSCRUB blind scrubbing operation mode (cont.)
#############################################################

# example with continually monitoring
proc grscrub_blindscrubbingfpga {} {
    variable REG
    variable done

    grscrub_disable
    grscrub_doneclear
    grscrub_errorclear
    grscrub_init_blindscrubmode

   # GRSCRUB operation enable
    grscrub_enable

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

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

    grscrub_disable
}
```

5.4.2.2.2 Readback Scrubbing

Before enabling the readback scrubbing operation mode, the GRSCRUB's registers must be configured. The *grscrub_init_readbackmode* procedure shows a configuration example of the DELAY, LGBAR, HGBAR, LFAR, FCR, LGSFAR, LMASKAR, LFMAPR, LGCRCAR, and IDCODE registers based on the example provided in this document. The registers are configured with values defined in Section 5.4.1. At the initialization procedure, one can also clean the ECNT, ERRFRAMEID, and FRAMEID registers to reset the number of detected errors and frame id of previous runs.

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 for two modes: (1) only detect errors, or (2) detect and correct errors. The former is configured in the *grscrub_readbackfpga_onlydetection* procedure (shown in Annex A), and the latter is configured in the *grscrub_readbackfpga_correction* procedure (shown below and in Annex A). The CORM bitfield of the CONFIG register defines the readback mode. In both cases, the error detection can be set through FFC, CRC, or FFC and CRC checks in

parallel. The FFCEN and CRCEN bitfields of the CONFIG register set 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) enable the GRSCRUB 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 internal 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 is performing the scrubbing operation on the target FPGA. The HOLD bitfield is 1 when the GRSCRUB 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. 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 counting.

```
############################################################
## Configuration for GRSCRUB readback scrubbing operation mode
############################################################

proc grscrub_init_readbackmode {} {
    ## variable definition here (removed due to limited space – see Annex A for complete example) ##

    # 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

    # no delay
    reg_write $REG(GRSCRUB.DELAY) 0x0

    #golden memory addresses
    reg_write $REG(GRSCRUB.LGBAR) $BITPARAMS(START.BIT)
    reg_write $REG(GRSCRUB.HGBAR) $BITPARAMS(END.BIT)
    reg_write $REG(GRSCRUB.LMASKAR) $BITPARAMS(START.MSK)
    reg_write $REG(GRSCRUB.LGSFAR) $BITPARAMS(START.GOLD)
    reg_write $REG(GRSCRUB.LFMAPR) $BITPARAMS(LOADAD.MAP)
    reg_write $REG(GRSCRUB.LGCRCAR) $BITPARAMS(LOADAD.CRC)

    #set frame address and configuration
    reg_write $REG(GRSCRUB.LFAR) $FSTART_ADDR
    reg_write $REG(GRSCRUB.FCR) [expr [expr $NUM_SCRUB_FRAMES << 9] | [expr $FLEN << 2]]

    reg_write $REG(GRSCRUB.IDCODE) $FPGA_IDCODE
}
```

```
################################################################
## Configuration for GRSCRUB readback scrubbing operation mode (cont.)
################################################################

proc grscrub_readbackfpga_correction {{datacheck "ffc"}} {
    variable REG
    variable done
    variable scrun

    grscrub_disable
    grscrub_doneclear
    grscrub_errorclear
    grscrub_init_readbackmode

    #data verification: FFC (bit 12), CRC (bit 11)
    if {$datacheck == "ffc"} {
        reg_write $REG(GRSCRUB.CONFIG) 0x00001024
    } elseif  {$datacheck == "crc"} {
        reg_write $REG(GRSCRUB.CONFIG) 0x00000824
    } else {
        reg_write $REG(GRSCRUB.CONFIG) 0x00001824
    }

    # GRSCRUB operation enable
    grscrub_enable

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

    # check errors
    if {(([expr { 0x00000020 & [expr [reg_read $REG(GRSCRUB.STAT)]]}] == 0x00000020) &&
        ([expr { 0x000001E0 & [expr [reg_read $REG(GRSCRUB.STAT)]]}] != 0x000000A0))} {
           log_puts [format "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 correct_errors [expr $run_error-$uncor_error]
           log_puts [format "GRSCRUB FPGA readback successfully"]
           log_puts [format "GRSCRUB Last readback mismatches: $run_error"]
           log_puts [format "GRSCRUB Correctable errors: $correct_errors"]
           log_puts [format "GRSCRUB Uncorrectable errors: $uncor_error"]
           log_puts [format "FPGA readback time: %s ms" [expr $end_time-$start_time]]
    }

    grscrub_disable
}
```

6 EXPERIMENTAL SETUP

This section provides an overview of the test setups and conditions used during the validation of the GRSCRUB feature in the GR716B microcontroller.

6.1 Validation Test Setup

Two validation test setups have been used to verify the GRSCRUB functionalities in the GR716B microcontroller.

6.1.1 Setup I: Preliminary Test Setup

The preliminary test setup, hereafter called Setup I, is composed of a GR716B board featuring the GR716B microcontroller connected to a GR-CPCIS-XCKU board featuring a KU060 FPGA. Figure 3 shows a block diagram of Setup I.

The GR716B board [AD4] is an updated version (rev. 1.5) of the GR716-BOARD development board. Besides the GR716B microcontroller, the GR716B board features a 512 Mbit SPI Flash memory, power supply configuration and monitoring, communication interfaces, and two 2x32 pin stackable 0.1” headers allowing access to all GR716B I/O pins. The on-board SPI Flash memory is used as the GRSCRUB golden memory, and it is configured as per Section 5.2. A GR716-DSU-USB adapter is connected to the GR716B board to interface the DSU and GRMON software via UART. The TCL scripts described in the previous sections run over the GRMON in the test computer and control the GR716B and GRSCRUB.

The GR-CPCIS-XCKU board [AD5] features an AMD/Xilinx KU060 Kintex UltraScale FPGA (FCBGA package), external access to the FPGA SMAP interface (I/O header), FPGA interface to DDR3 SDRAM via two SODIMM connectors, SPI and Parallel Flash memory, and embedded power control and monitoring circuitry. This board also provides the possibility to fit a GR716 microcontroller, being compatible with both GR716A and GR716B versions. The board version (rev. 1.0) used in Setup I has a GR716A mounted.

A custom flat cable is used to connect the SelectMap signals between the I/O headers of the GR716B board and the GR-CPCIS-XCKU board. Both boards are simultaneously powered up by an external Power Supply Unit (PSU) to avoid power connection issues.

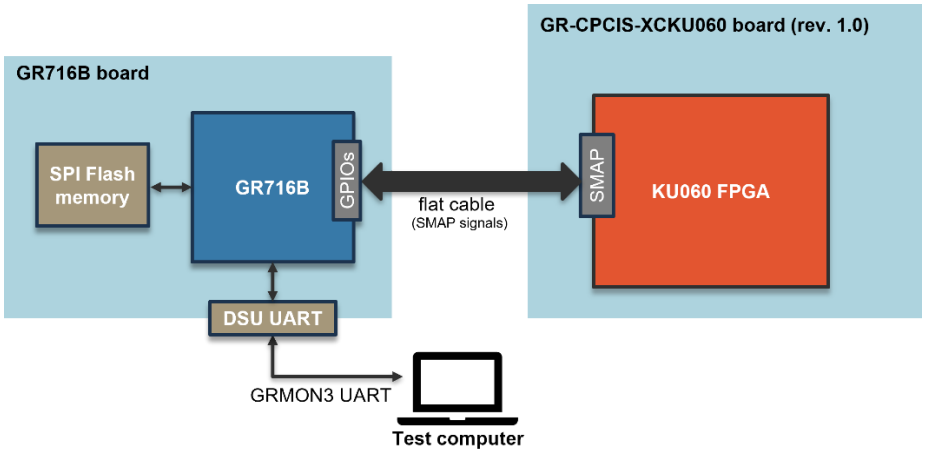

Figure 3. Block diagram of Setup I: GR716B board (rev. 1.5) connected to the GR-CPCIS-XCKU board (rev. 1.0).

On the GR716B board, the GR716B input system clock (SYS_CLK) and SpaceWire clock (SPW_CLK) are defined by the crystal oscillator (XO) output, which is 20 MHz. The PLL is configured to provide a 50 MHz internal system clock, and the SPI memory controller is set to use the alternate scaler, which leads to a 25 MHz clock (maximum value). The SelectMap clock is generated from the SYS_CLK and is set to 10 MHz (maximum value based on the SYS_CLK input). One should refer to the GR716B data sheet for further information on how to configure the clocks [AD1]. Table 5 gives an overview of the clock configuration used in Setup I.

Table 5. GR716B clock configuration in Setup I.

SYS_CLK	SPW_CLK	PLL internal system clock	SPI memory clock	SMAP clock
20 MHz	20 MHz	50 MHz	25 MHz	10 MHz

6.1.2 Setup II: Final Test Setup

The final test setup, hereafter called Setup II, is composed of a GR-CPCIS-XCKU board featuring a KU060 FPGA and a GR716B microcontroller. The GR716B microcontroller has been assembled on the GR-CPCIS-XCKU board (rev. 2.1). On this board version, the GR716B has direct on-board connection to the FPGA SelectMap signals. Therefore, the SelectMap I/O header is no longer needed but it is still useful for signalling monitoring and debugging. Figure 4 shows a block diagram of Setup II, and Figure 5 depicts a view of the GR-CPCIS-XCKU board (rev. 2.1).

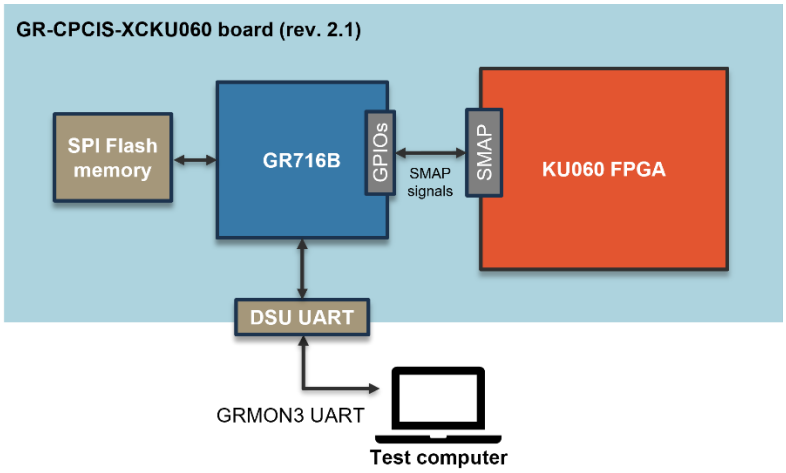

Figure 4. Block diagram of Setup II: GR-CPCIS-XCKU board (rev. 2.1) featuring a KU060 FPGA and a GR716B microcontroller.

Figure 5. View of the GR-CPCIS-XCKU board (rev. 2.1).

The GR716B is also connected to a SPI Flash memory that is used as the GRSCRUB golden memory, and it is configured as per Section 5.2. The GR-CPCIS-XCKU board has an on-board FTDI chip that allows UART connection to the GR716B DSU. Like the preliminary setup, the TCL scripts described in the previous sections run over the GRMON in the test computer and control the GR716B and GRSCRUB.

On the GR-CPCIS-XCKU board, the GR716B input SYS_CLK and SPW_CLK are defined by the on-board oscillators, which are 100 and 200 MHz, respectively. The PLL is configured to provide a 50 MHz internal system clock, and the SPI memory controller is set to use the alternate scaler, which leads to a 25 MHz clock (maximum value). The SelectMap clock is generated from the SYS_CLK and is set to 25 MHz (maximum recommended value in GR716B). One should refer to the GR716B data sheet for further information on how to configure the clocks [AD1]. Table 6 gives an overview of the clock configuration used in Setup II.

Table 6. GR716B clock configuration in Setup II.

SYS_CLK	SPW_CLK	PLL internal system clock	SPI memory clock	SMAP clock
100 MHz	200 MHz	50 MHz	25 MHz	25 MHz

6.2 Error Injection Setup

An error injection system has been prepared to validate the scrubbing capabilities of GRSCRUB in the GR716B microcontroller. The error injection targets to flip bits in the FPGA design in order to simulate SEUs in the CRAM. Two distinct setups have been configured depending on the Setup I and B characteristics.

6.2.1 Setup I - Error Injection

An error injection framework (EIFW) [AD6] has been developed by Frontgrade Gaisler that allows emulating errors on the CRAM via the SelectMap interface. The EIFW is implemented in VHDL and can run on an external FPGA for validation purposes. The injection engine is controlled via Python scripts running on the test computer and communicating over UART. More information about the EIFW is provided in its user manual [AD6].

In order to use the EIFW in Setup I, an external FPGA has been added to the system. A GR-XC6S board, featuring an AMD/Xilinx Spartan-6 FPGA, is used as a test controller (TC) to implement the EIFW engine and manage a multiplexing of the SelectMap signals. Since both GRSCRUB and the EIFW use the SelectMap to access the FPGA configuration memory, they cannot be used in parallel. Therefore, the GR716B GPIOs have been connected to the GR-XC6S board, and the signals are routed through the TC FPGA. An I/O header from the TC FPGA is connected to the SelectMap interface of the KU060 on the GR-CPCIS-XCKU board. The I/O signals are multiplexed between the GR716B and the EIFW. Figure 6 shows the block diagram of the test setup.

Additional TCL scripts have been integrated to the primary scripts presented in the previous sections to coordinate the error injection and the GRSCRUB scrubbing cycles. The error injection control scripts are presented in Annex A. The error injection can be performed deterministically or randomly, and errors are injected only between the scrubbing cycles. Since both GRSCRUB and EIFW use the SelectMap interface to access the FPGA configuration memory, only one can be enabled at a time.

For the example presented in this document, only random bit errors are injected since they better reproduce real scenarios. For each injection run, one or more random errors can be injected at once. In sequence, the GRSCRUB is released to scrub the entire FPGA CRAM. At the end of the scrubbing execution, the GRSCRUB counters are checked and logged (in case of readback scrubbing). In sequence, a new injection run starts, and the loop is repeated.

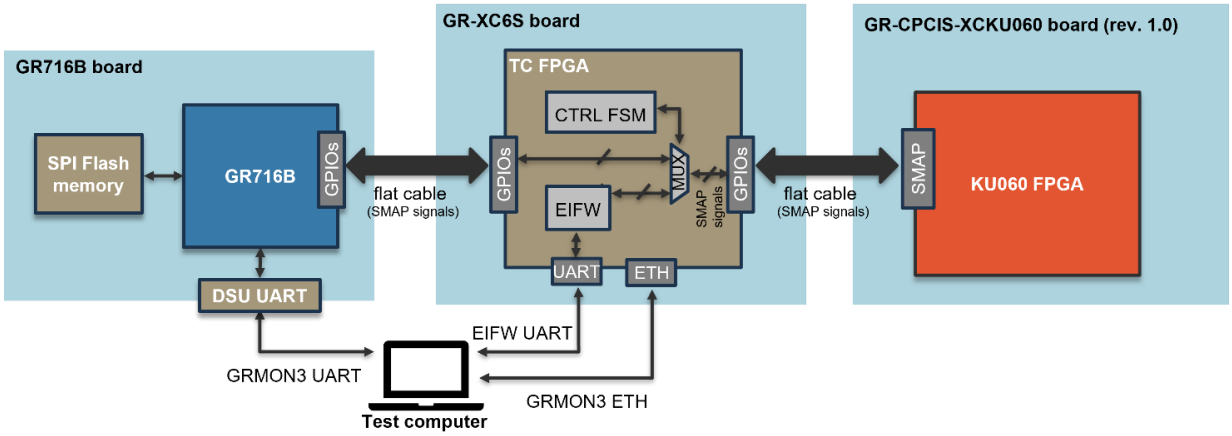

Figure 6. Block diagram of error injection in Setup I: GR-XC6S board connected between the GR716B board and the GR-CPCIS-XCKU board.

6.2.2 Setup II - Error Injection

The GR716B on the GR-CPCIS-XCKU board (rev. 2.1) is tied to the FPGA SelectMap pins, which makes it unfeasible to use the SelectMap I/O header for signal connection. In this scenario, the EIFW presented in the previous section cannot be used.

For validation purposes, a workaround has been implemented to inject errors into the FPGA CRAM using the GRSCRUB itself. Additional TCL scripts have been developed to control the GRSCRUB in the GR716B to emulate errors in selected CRAM frames. The error injection control scripts are presented in Annex A.

Similar to the EIFW, errors can be injected deterministically or randomly, and errors are injected only between the scrubbing cycles. For the example presented in this document, only random bit errors are injected since they better reproduce real scenarios. One random error is injected per injection run. In sequence, the GRSCRUB is released to scrub the entire FPGA CRAM. At the end of the scrubbing execution, the GRSCRUB counters are checked and logged (in case of readback scrubbing). In sequence, a new injection run starts, and the loop is repeated.

The random error injection using the GRSCRUB follows the approach below:

1) A random target frame is selected considering all the CRAM frames (i.e., a random id from the 37498 frames – total KU060 CRAM frames).
2) A random target bit is selected in the frame (i.e., a random bit from the 3936 bits in a frame – each KU060 frame has 123 32-bit words).
3) The target frame is read back from the FPGA CRAM using GRSCRUB and saved in the GR716B LEON3 on-chip RAM (start address 0x30000000).
4) The target bit is flipped on the on-chip RAM.
5) The GRSCRUB is configured to write back the target frame from the on-chip RAM into the FPGA CRAM.

7 EXPERIMENTAL RESULTS

7.1 Validation Results

The GRSCRUB features have been successfully validated on the GR716B microcontroller. All modes and configurations presented in this document are functional on Setup I and Setup II.

The KU060 FPGA can be programmed as expected. After programming the FPGA using GRSCRUB, the design implemented in the KU060 has been dynamically validated by accessing the DSU of the NOEL-V SoC using GRMON and running test software.

All the scrubbing modes are functional. The blind and readback modes have been primarily validated by targeting the entire CRAM. Tests have also been performed targeting partial scrubbing. Different tests have been made with BLKFRAME equal to 0 (all frames scrubbed at once) and to 1 (one frame scrubbed at a time).

Table 7 presents the approximated performance, in milliseconds, of the GRSCRUB operations targeting the KU060 FPGA in the final test setup (Setup II – GR-CPCIS-XCKU board rev. 2.1). This performance data is gathered using the GRMON monitoring the GR716B. All configuration memory frames are scrubbed at once (BLKFRAME=0) during blind and readback modes. The performance of the readback scrubbing operation is related to error-free configuration memory. Longer periods may be observed depending on the number of errors being corrected at a specific scrubbing cycle.

The performance of the GRSCRUB operations may be improved if the external SPI Flash memory has a higher clock frequency.

Table 7. Performance of GRSCRUB operations targeting the KU060 FPGA on the GR-CPCIS-XCKU board (rev. 2.1) – final test setup (Setup II).

Operation		Performance
FPGA programming		25609 ms
Blind scrubbing		19408 ms
Readback scrubbing	FFC	31651 ms
	CRC	15928 ms
	FFC and CRC	31856 ms

7.2 Error Injection Results

During the error injection tests on both setups, the FPGA design was kept in static mode, i.e., the NOEL-V SoC was kept in reset. However, no difference in the outcome results is expected when the FPGA design is in a dynamic state.

Table 8 presents the error injection results using the EIFW to inject bitflips in the KU060 CRAM in Setup I. The blind and readback scrubbing modes of the GRSCRUB were evaluated. Both modes were configured for scrubbing all CRAM frames at once (BLKFRAME=0). The readback scrubbing was configured for FFC and CRC parallel checks. Campaigns of 10 and 100 random errors per run were executed. In all tests, the GRSCRUB was able to detect and correct all injected errors.

Table 8. Error injection results using the EIFW in Setup I with different GRSCRUB scrubbing modes. All injected faults were successfully corrected.

BLKFRAME	Scrubbing mode		# errors injected per run	# total errors	All faults corrected
0 (all frames scrubbed at once)	Blind scrubbing		10; 100	10400	PASS
	Readback scrubbing	FFC and CRC	10; 100	38100	PASS

Table 9 shows the error injection results using the GRSCRUB to flip bits in the KU060 CRAM in Setup II. The blind and readback scrubbing modes of the GRSCRUB were evaluated. Both modes were tested with different block frame configurations: scrubbing all frames at once (BLKFRAME=0), scrubbing one frame at a time (BLKFRAME=1), and scrubbing a hundred frames at a time (BLKFRAME=100). All CRAM frames were scrubbed in all modes. The readback scrubbing was configured for FFC check, CRC check, and FFC and CRC parallel checks. One error was injected per run in all tests. In all tests, the GRSCRUB was able to detect and correct all injected errors.

Table 9. Error injection results using the GRSCRUB error injection in Setup II with different GRSCRUB scrubbing modes. All injected faults were successfully corrected.

BLKFRAME	Scrubbing mode		# errors injected per run	# total errors	All faults corrected
0 (all frames scrubbed at once)	Blind scrubbing		1	10000	PASS
	Readback scrubbing	FFC	1	1000	PASS
		CRC	1	1000	PASS
		FFC and CRC	1	10000	PASS
1 (one frame scrubbed at a time)	Blind scrubbing		1	1000	PASS
	Readback scrubbing	FFC and CRC	1	1000	PASS
100 (hundred frames scrubbed at a time)	Blind scrubbing		1	1000	PASS
	Readback scrubbing	FFC and CRC	1	1000	PASS

8 CONCLUSION

The Frontgrade Gaisler's GR716B rad-hard microcontroller, featuring the FPGA scrubber controller – GRSCRUB, has been functionally validated targeting an AMD/Xilinx KU060 Kintex UltraScale FPGA. This document provides examples of how to configure the GR716B and GRSCRUB and demonstrates the configuration steps for each operational mode.

The GRSCRUB programming and scrubbing capabilities have been exercised under two different setups. The first preliminary setup uses a GR716B board connected via custom cabling to a GR-CPCIS-XCKU board (rev 1.0). The second final setup is a reworked GR-CPCIS-XCKU board (rev 2.1) featuring a GR716B directly connected to the SelectMap signals of the KU060 FPGA. The GRSCRUB functionalities have been successfully validated on both setups. Error injection has also been performed to ensure the GRSCRUB error correction capability during scrubbing. In all tests, the GRSCRUB successfully detected and corrected all injected errors, demonstrating the effective implementation of the solution.

9 ANNEX A – SCRIPTS PACKAGE

This annex points to example scripts used to demonstrate the GRSCRUB capabilities on the GR716B rad-hard microcontroller. The scripts provide the possibility to reproduce the user-case scenario described in this application note. It includes scripts to program and scrub the target FPGA using the GR716B microcontroller. Scripts to perform error injection in the FPGA configuration memory (CRAM), simulating radiation-induced upsets, are also included.

The demonstration scripts to configure and control the GR716B and GRSCRUB are written in TCL and are intended to be used on GRMON. The scripts can be used as a baseline for software development.

The scripts package is available for download from the GR716B website (www.gaisler.com/gr716b).

Figure 7 shows a view of the package contents and Table 10 describes the files.

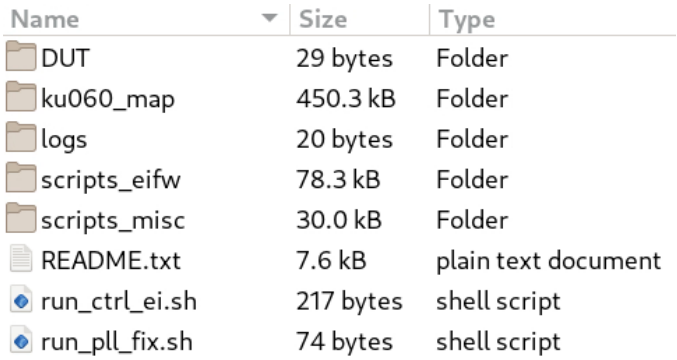

Figure 7. View of the GR716B_GRSCRUB_AppNote_EXTERNAL package.

Table 10. Content description of the scripts package.

Folder/File	Description	Reference section
DUT	This folder is for the user to optionally store the FPGA design files, such as bitstream, mask, and CRC data that should be loaded to the external SPI Flash memory.	0, 5.1.4, 5.2
ku060_map	The addresses of the AMD/Xilinx KU060 FPGA configuration memory are saved in a *.srec* file, which should be loaded to the external SPI Flash memory. Note that the *.srec* file is configured to be loaded at address 0x13D00000. One can use the *GNU objcopy* utility to update the memory address if needed.	5.1.3, 5.2
logs	This folder stores the log files for each subrun. The README file describes the content of each log file.	
scripts_eifw	Scripts used for GRSCRUB validation with error injection.	
scripts_eifw/grscrub-ctrl.tcl	Script that configures the GR716B and GRSCRUB	5.3, 5.4
scripts_eifw/	Scripts used in the Setup I only. The TCL script	6.1.1, 6.2.1

fpgaei_ctrl.tcl and fpgaei_campaign.py	configures the error injection campaign, and the Python script controls the error injection framework (EIFW).	
scripts_eifw/grscrub-ei.tcl	Script used in the Setup II only. The TCL script controls the error injection using GRSCRUB.	6.1.2, 6.2.2
scripts_eifw/gr-log.tcl	Script to save logs.	
scripts_eifw/gr-start.tcl	Top TCL script. It sources the other scripts from the scripts_eifw folder. One can update this script to either source fpgaei_ctrl.tcl (for Setup I) or grscrub-ei.tcl (for Setup II).	
scripts_misc	Miscellaneous scripts.	
scripts_misc/ gr716b_pll_fix.tcl	Script used to configure the GR716B PLL.	6.1.1, 6.1.2
scripts_misc/ load_spi_flash.tcl	Script used to load the data to the external SPI Flash memory.	5.2
scripts_misc/crc_gen	Scripts used to generate the CRC codes of the user-defined FPGA bitstream.	5.1.4
REAME.txt	README file for reference.	
run_ctrl_ei.sh	Bash script to automatically launch GRMON and run the TCL scripts.	
run_pll_fix.sh	Bash script to automatically launch GRMON and run the PLL configuration script.	