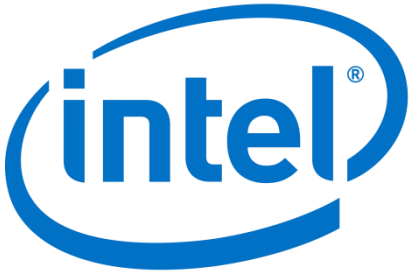

Generic Serial Flash Interface (GSFI) and Nios II Booting Quick Start Guide

Author: Tzy Way Ooi, Chun Hsien Ng

Date: Nov 2019

Revision: 1.0

Contents

1 Introduction 3
2 Prerequisite 3
3 Hardware and Software Requirements 3
3.1 Hardware Requirements 3
3.2 Software Requirements 3
4 Design Architecture 4
5 Running the design 4
5.1 Using the hardware image .sof and .elf files 4
5.2 Running Nios II booting using master image .jic 6
6 Update design with other flash device for Generic Serial Flash Interface IP 6
6.1 Hardware design update requirement 6
6.2 Software design update requirement 7
7 Update design with different boot address for Nios II Booting 9
7.1 Hardware Design update requirement 9
7.2 Software Design update requirement 11
7.3 HEX File Generation 12
7.4 JIC file generation with Generic Flash Programmer 13
8 Revision History 18

1 Introduction

This design example demonstrates basic operations that can be performed by the Generic Serial Flash Interface (GSFI) IP to various serial flash devices such as Macronix, Cypress and Micron. A simple software example is created to perform the following flash operation.

1. Read ID
2. Read register
3. Write register
4. Read memory
5. Program
6. Erase
7. Sector protect
8. Sector un-protect

The above flash operations are selected with the purpose to help designers in implementing their own designs to access the flash devices.

This design example shows how to perform a Nios II boot with GSFI controller. This design example will cover the settings required in the controller and the BSP editor setting in Nios II Software build tools.

The application data must be programmed into flash devices before it can be used by the design. There are several ways to program data to flash devices and in this design example, the designer will use the Generic Flash Programmer available in Intel Quartus Prime software. The Generic Flash Programmer supports targeting any flash using a JTAG indirection configuration (JIC) file enabling programming of devices that are not in the default list.

2 Prerequisite

Designers should have knowledge in instantiating and developing a system with Platform Designer and familiar with the IPs used in this design example.

Related information:

- Generic Serial Flash Interface Intel FPGA IP Core User Guide
- Generic Flash Programmer User Guide (Quartus Prime Standard Edition)
- Generic Flash Programmer User Guide (Quartus Prime Pro Edition)
- Embedded Design Handbook

3 Hardware and Software Requirements

3.1 Hardware Requirements

This design requires the following hardware:

- Cyclone V E Development Kit
- USB type B cable for connecting from the host PC to the development kit

3.2 Software Requirements

This design requires the following software:

- Quartus Prime Standard 18.1.1
- Nios II Embedded Design Suite (EDS)

4 Design Architecture

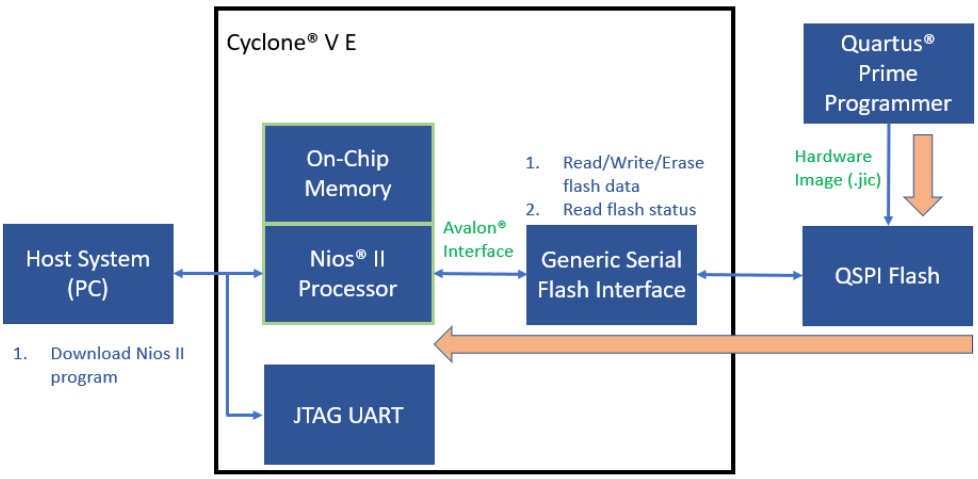

Figure 1: Design example block diagram

Figure 1 shows the major IP blocks used in this design example. You may see the detailed connection between each IP blocks in the Platform Designer project included in this design example.

This design example contains a Nios II processor, on-chip memory, JTAG UART and Generic Serial Flash Interface IP core connected by using Platform designer tool. The GSFI IP core is used to access the QSPI flash devices on the Cyclone V E development kit using the Nios II processor as an Avalon master to control the GSFI IP core. The GSFI IP core supports all flash devices and hence this example shows the controller access to three flash from different vendor. The hardware image .sof file and software image .elf file is program and downloaded from the host system (PC) to FPGA through JTAG interface.

The orange color arrow shows the Nios II boot flow. There are two types of supported boot processes Execute-in-Place (XIP) and boot copier. In this example, the boot copier method is used. The application data is generated as a .hex file and converted to the .jic format and then programmed to flash using the Quartus programmer. Upon devices power up, the Nios II processor starts executing the boot copier software after a system reset. The software copies the application from the boot flash to on-chip memory. Once this is complete, the Nios II processor transfers the program control over to the application.

5 Running the design

The following section describes how to run the design with hardware image .sof, .elf and .jic files

5.1 Using the hardware image .sof and .elf files

The hardware image (.sof) and software (.elf) files are provided in the master_image folder

1. Download the design from design store https://fpgacloud.intel.com/devstore/platform/18.1std.1/Standard/generic-serial-flash-interface-and-nios-ii-booting/

2. Unqar the design and the hardware image .sof and .elf is located in the master_image folder

3. Launch Quartus Programmer and program the generated .sof file to the FPGA device. Master_image folder provided in the project directory contain the .sof and .elf which are ready to be downloaded to the board.

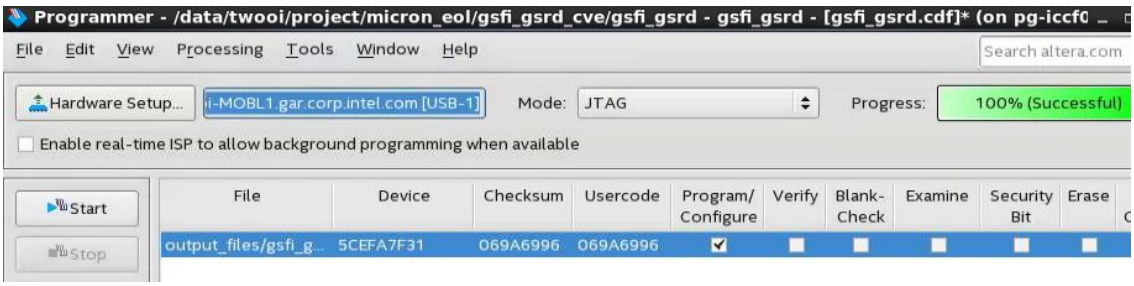

Figure 2: Quartus Programmer

4. Launch Nios II Command Shell by going to Window Start menu and looking for Quartus installation folder. Download the .elf file and invoke the terminal by typing ***nios2-download -g -r flash_test.elf && nios2-terminal***

5. You will be able to see the JTAG UART generate a running status for the flash operations. Please take note that the .elf file provided in master_image folder is an .elf file for performing flash operation x1 with Micron flash.

```
GSFI Controller Test

Device ID is expected: 1019ba20

Status register: 0x0
Flag status register: 0x81
NVCR register: 0xafee
Flash address bytes is 4 byte address

GSFI register initialization...
GSFI addressing mode is 4-byte
GSFI operating protocol 1-1-1

data read in address 0 is: 0xabcd1234
erasing sector due to current memory address is not empty...
erased sector data: 0xffffffff
erase verification SUCCESSFUL

program operation with data abcd1234
read data = 0xabcd1234
write verification SUCCESSFUL

Perform sector protect
Erase sector...
Erase error : a3
erase_operation failed
data is retained = 0xabcd1234
sector protect verification SUCCESSFUL

Perform sector unprotect
Erase sector...
data is erased = 0xffffffff
sector unprotect verification SUCCESSFUL

program operation with data abcd1234
```

Figure 3: JTAG UART print out log for executing the .elf file

The software program will first read the flash ID and check whether it matches with the flash ID stated in the software program. Once the flash ID is matched, the software will proceed to read

status register, flag status register (for Micron devices only), NVCR register(for Micron devices only), and check the addressing mode that the flash device is operating in. Next, the software program will setup the GSFI IP with the addressing mode and operating protocol. Next, the software will perform the erase, write, read, sector protect and sector unprotect flash operation. A message with “SUCCESSFUL” will be printed out if the operation is executed without any issue. If an error occurs, then a message with “FAILED” will be printed out.

From the above snapshot, the addressing mode and operating protocol is 4 byte addressing mode and 1-1-1 (1 command line, 1 address line and 1 data line) operating protocol. All operations were executed successfully and the message “SUCCESSFUL” was printed out.

5.2 Running Nios II booting using master image .jic

You may run the Nios II booting with the .jic file provided in the master_image folder

1. Launch Quartus Programmer and program the generated .jic file. Master_image folder provided in the project directory contains the .jic file which is ready to be program into the flash devices

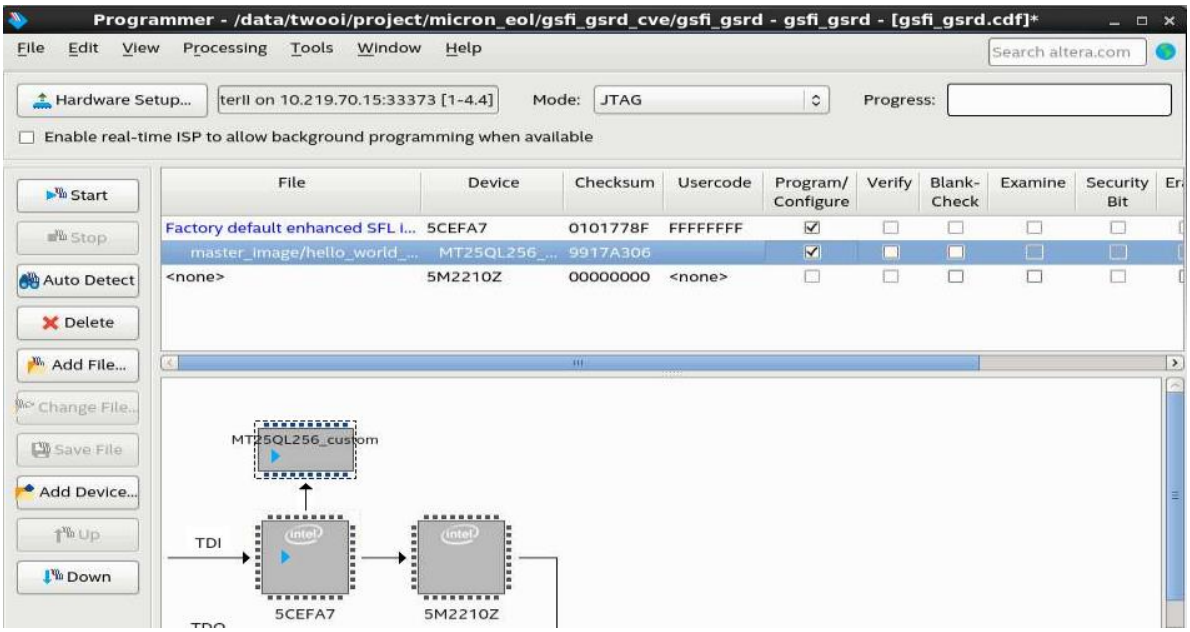

Figure 4: Program the .JIC file

2. Power cycle the board after Quartus programmer has programmed the flash devices
3. Launch Nios II terminal from Nios II command shell and you may observe “Hello World from GSFI” is printed on the terminal

6 Update design with other flash device for Generic Serial Flash Interface IP

6.1 Hardware design update requirement

The design provided has selected the GSFI IP with device density of 256Mb. Hence, you may need to regenerate the hardware design if you would like to use the GSFI IP with flash devices of other density. To regenerate the hardware design to access different flash device

1. Once you downloaded the design, you may unqar the design using Quartus by opening the .par file that was downloaded
    a. File → Open project
    b. Select the .par file downloaded from design store
2. Open the .qsys design file in Platform Designer and modify the device density parameter in the GSFI IP

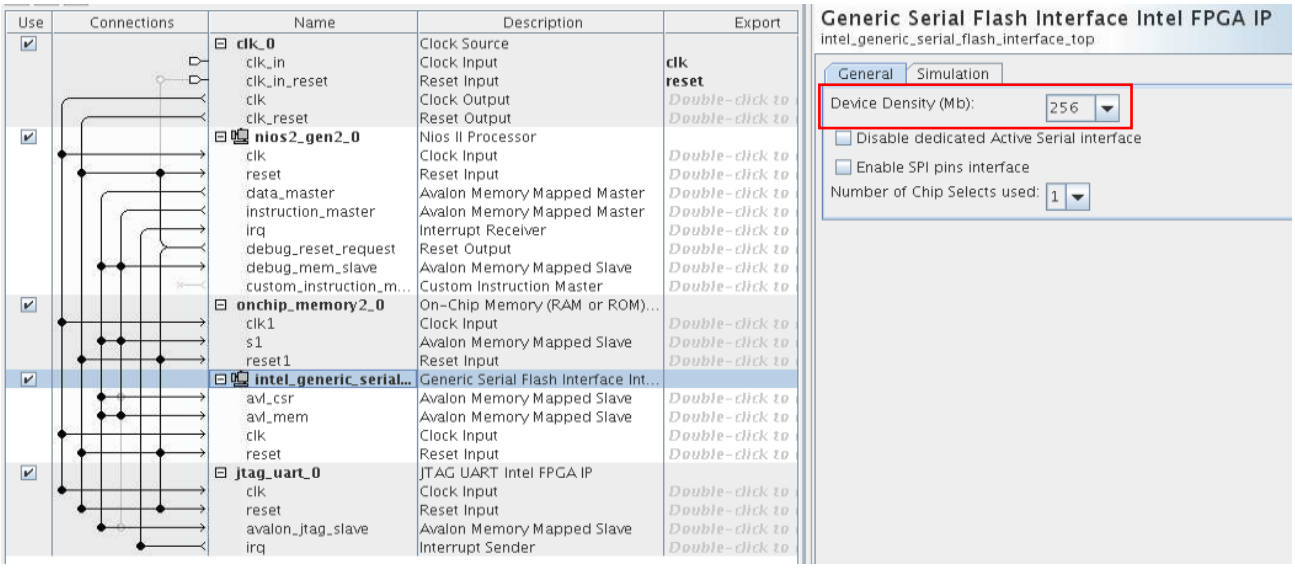

Figure 5: Platform designer design file

3. Regenerate the design by clicking on the “Generate HDL…”.
4. Once the design is regenerated in Platform Designer, you may proceed to compile the design in Quartus Prime software

6.2 Software design update requirement

The .elf file provided in the master image folder performs all flash operations using Standard mode for Micron flash devices. You need to modify the C source code so that the .elf file generated can be used with vendor flash devices as well as different I/O modes (Standard, Dual and Quad). The .c files provided in this design example are shown as below:

Table 1: Software C programming file used in design example

File	Description
software_example/main.c	Main C source code for Nios II to perform operations with Micron, Macronix and Cypress
software_example/operation_common.c	C source code for common flash operations for Micron, Macronix and Cypress
software_example/operation_micron_specific.c	C source code for flash operation that are specific for Micron flash devices

software_example/operation_macronix_specific.c	C source code for flash operation that are specific for Macronix flash devices
software_example/operation_cypress_specific.c	C source code for flash operation that are specific for Cypress flash devices

To modify the source for a different operating protocol or vendor, import the software code to the Nios II Eclipse. After that, open the main.c file and modify the following parameters:

Table 2: Definition of each define

C code	Description
#define expected_device_id	Set the flash device ID. For example, Micron 256Mb devices is 0x1019BA20.
#define addressing	Set the addressing mode. 0 for 3-byte addressing while 1 for 4-byte addressing.
#define operating	Set the flash operating protocol. 0 for 1-1-1, 1 for 1-2-2 and 2 for 1-4-4
#define chip_select	Set the flash chip select if more than one flash device is in the design.
#define data	Set the test data to be written to the flash device.
#define Micron	Enable the vendor specific C source code to be compiled. For example, defining Micron will enable the Micron code, defining Cypress will enable the Cypress code and defining Macronix will enable the Macronix code.

```
// Define Expected device id -- micron MT25QL256: 0x1019ba20, cypress 256: 0x4d190201, macronix 256: 0xc21920c2
// Define Addressing mode -> 0x0: 3 byte addressing, 0x1: 4 byte addressing
// Define Operating mode -> support (command, address, data) 0: 1-1-1, 1: 1-2-2, 2: 1-4-4
// Define chip select -> 0x0: first flash, 0x1: second flash, 0x2: third flash
#define expected_device_id  0x1019ba20
#define addressing          0x1
#define operating           0x0
#define chip_select         0x0

// for testing purpose data test
#define data 0xabcd1234

// Define vendor to compile the correct sample program (Micron, Cypress, Macronix)
#define Micron
```

Figure 5: Example code for define

The device ID is based on the flash device datasheet. Addressing mode and operating protocol is based on your preference. As described in table 2, you may select 3 byte addressing mode or 4 byte addressing mode and you may select the operating protocol to be 1-1-1, 1-2-2 or 1-4-4 (command lines-address lines-data lines). In this design example, the C source code only supports these operating protocols. Please take note that you need to specify the flash vendor correctly so that the correct portion of the C source code will be compiled.

In Figure 5, the expected device ID is set to 0x1019ba20 which is Micron MT25Q devices. The addressing is set to 0x1 which mean that 4 bytes addressing mode is used. In the operating protocol, the value is set to 0x0 which represents standard mode (1-1-1). The chip select is set to 0 as only one flash on the board. Data can be set to any value with the size of 4 bytes and this is the data to be written to flash devices. Lastly, define the correct vendor. Micron is defined in the example above as the expected device ID is refer to Micron flash devices.

The example C source code provided is tested with the all the operating protocol and flash devices described in the table 3. Cypress and Macronix devices do not support the 1-2-2 write operation protocol and this option should not be used with those devices.

Table 3: Flash devices and operating protocol used in design

	Micron	Cypress	Macronix
Device ID of 256Mb flash	0x1019ba20	0x4d190201	0xc21920c2
Operating protocol (command lines-address lines-data lines)	Read operation: 1-1-1, 1-2-2, 1-4-4 Write operation: 1-1-1, 1-2-2, 1-4-4	Read operation: 1-1-1, 1-2-2, 1-4-4 Write operation: 1-1-1, 1-4-4	Read operation: 1-1-1, 1-2-2, 1-4-4 Write operation: 1-1-1, 1-4-4

Once you have finished modifying the #define to your desired operation, you may regenerate the software executable file (.elf) by building the project in the Nios II Eclipse software.

7 Update design with different boot address for Nios II Booting

The notes captured in this section assumed that you are familiar with the Nios II boot flow as documented in section **5.2.4 Nios II Processor Booting from QSPI Flash** in Embedded Design Handbook. The following guidelines will be focusing on modification required to boot Nios II from QSPI flash with GSFI IP, in comparison with Generic Quad SPI Controller.

7.1 Hardware Design update requirement

1. In the Nios II Processor parameter editor, set the **Reset vector memory** and **Exception vector memory** according to your boot option. Set the **Reset vector offset** according to your desired location in the QSPI flash.

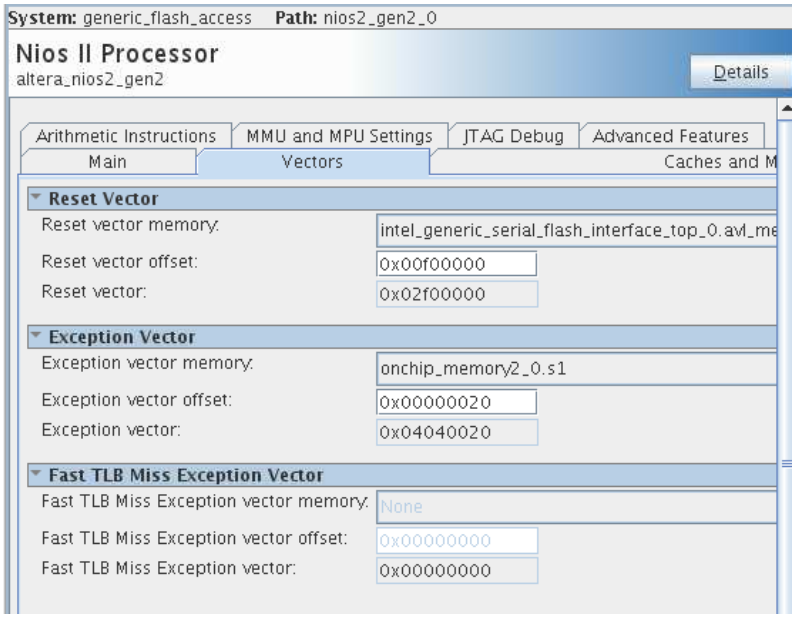

Figure 6: Nios II processor parameter editor

Table 4: Reset vector and exception vector configuration

Boot Option	Reset Vector Configuration	Exception Vector Configuration
Nios II processor application execute-in-place from QSPI flash	QSPI Flash (GSFI IP)	Choose between: ▪ On-Chip RAM (OCRAM)/ External RAM ▪ QSPI Flash (GSFI IP)
Nios II processor application copied from QSPI flash to RAM using boot copier	QSPI Flash (GSFI IP)	OCRAM/ External RAM

2. Right-click on the GSFI IP parameters editor, select **Show Hidden Parameters**.

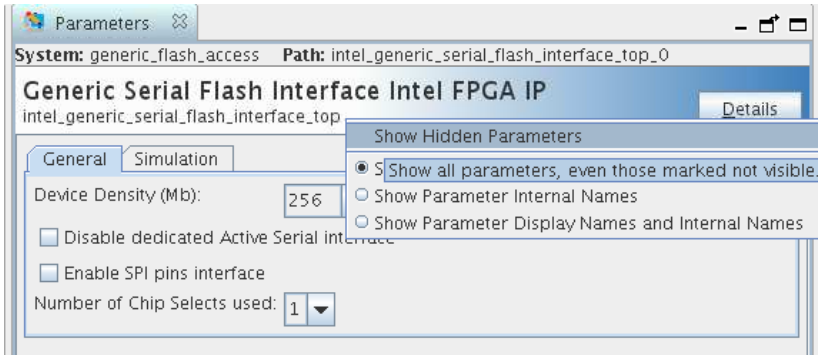

Figure 7: Hidden parameter

3. Refer to **Intel Supported Configuration Devices** tab → **Intel Supported Third Party Configuration Devices** tab in Device Configuration Support Center to check the byte addressing mode supported for each flash device in each FPGA device. For example, Cyclone V E FPGAs when used with Micron MT25QL256 devices support the 4-byte addressing mode. The Cyclone V E FPGA dev kit used in this design has been modified to include a flash socket and Micron MT25QL256 flash devices is used.

.

The addressing mode is set by modifying bit 8 of the Control Register value in the Default Setting parameter section. Changing bit 8 to 0x0 enables 3-byte addressing and 0x1 enables 4-byte addressing.

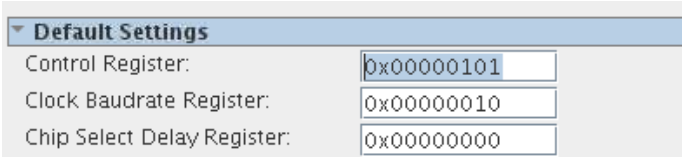

Figure 8: Control Register in Default Settings

Offset (Hex)	Register Name	R/W	Field Name	Bit	Default Value (Hex)	Description
0	Control Register	Reserved		31:8	Reserved	
		R/W	Addressing mode	8	0x0	Addressing mode for read and write operation: • 0x0: 3-bytes addressing • 0x1: 4-bytes addressing For 4-byte addressing mode, you must enable 4-byte address by sending command to the flash. This bit affects direct access to memory via the Avalon-MM interface for both write and read operation.
		R/W	Chip select	7:4	0x0	Selects the flash device • 0x0: To select first device • 0x1: To select second device • 0x2: To select third device
		Reserved		3:1	Reserved	
		R/W	Enable	0	0x1	Set this bit to 0 to disable the output of the IP and put all output signal to high impedance state. This can be used to share bus with other devices.

Figure 9: Addressing mode in Control Register

4. Click **Generate HDL**.

7.2 Software Design update requirement

1. Once you have created your application, you must edit the BSP editor settings according to the selected Nios II processor boot option for booting Nios II successfully

Table 5: BSP editor settings

Boot Option	BSP Editor Setting: Settings.Advanced.hal.linker	BSP Editor Setting: Linker Script
Nios II processor application execute-in-place from QSPI flash	If the exception vector memory is set to OCRAM/ External RAM, enable the following settings: ▪ *allow_code_at_reset* ▪ *enable_alt_load* ▪ *enable_alt_load_copy_rodata* ▪ *enable_alt_load_copy_rwdata* ▪ *enable_alt_load_copy_exceptions* If the exception vector memory is set to QSPI Flash (GSFI IP), enable the following settings: ▪ *allow_code_at_reset* ▪ *enable_alt_load* ▪ *enable_alt_load_copy_rodata* ▪ *enable_alt_load_copy_rwdata*	▪ Set .text Linker Section to QSPI flash (GSFI IP) ▪ Set other Linker Sections (.heap, .rwdata, .rodata, .bss, .stack) to OCRAM /External RAM
Nios II processor application copied from QSPI flash to RAM using boot copier	Make sure all settings are left unchecked	Make sure all Linker Sections are set to OCRAM /External RAM

2. Build the project to generate the ELF file.

7.3 HEX File Generation

A HEX file must be generated from the ELF file so that the HEX file can be used in generating the .JIC file and program to the flash devices

1. Launch Nios II Command Shell.
2. For Nios II processor application execute-in-place (XIP) from QSPI flash, use the following command line to convert the ELF to HEX for your application.

elf2hex <input ELF filename> <base address of GSFI AVL MEM> <end address of GSFI AVL MEM> --width=8 --little-endian-mem --create-lanes=0 <output HEX filename>

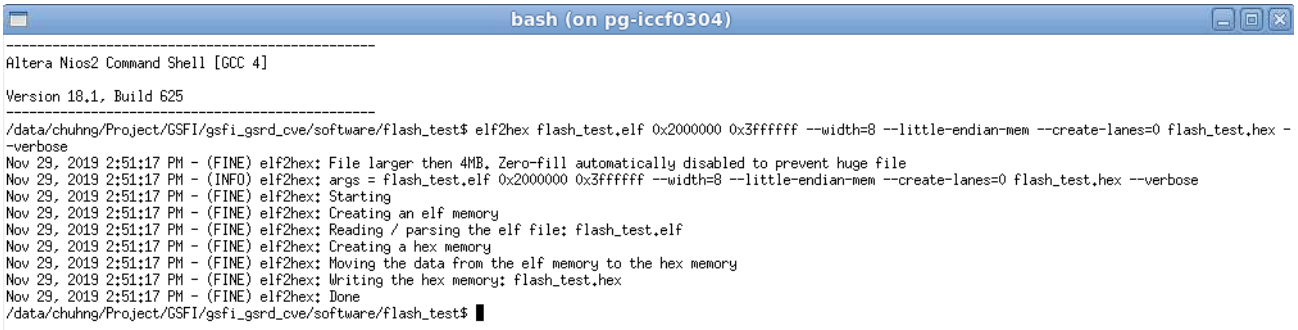

Figure 10: elf2hex example

3. For Nios II processor application copied from QSPI flash to RAM using boot copier, use the following command to convert the ELF to HEX for your application.

alt-file-convert -I elf32-littlenios2 -O hex --input=<ELF filename> --output=<HEX filename> --base=<address of GSFI AVL MEM> --end=<address of GSFI AVL MEM> --reset=<reset vector offset> --out-data-width=8 --boot=<../boot_loader_cfi.srec>

```
/data/chuhng/Project/GSFI/gsfi_gsrd_cve/software/flash_test$ alt-file-convert -I elf32-littlenios2 -O hex --input=flash_test.elf --output=flash_test.hex --base=0x20000
00 --end=0x3fffffff --reset=0x2f00000 --out-data-width=8 --boot=/acds_releases/acds/18.1std/current.linux/linux64/ip/altera/nios2_ip/altera_nios2/boot_loader_cfi.srec
Converting Nios II ELF file to HEX file. Appending boot file.
/data/chuhng/Project/GSFI/gsfi_gsrd_cve/software/flash_test$
```

Figure 11: alt-file-convert example

7.4 JIC file generation with Generic Flash Programmer

Starting from Quartus 18.1.1 Standard and 19.1 Pro software, the Generic Flash Programmer tool can be used to generate the appropriate secondary programming file like .jic file and program the .jic file into the flash device. The setting and controls for the Generic Flash Programmer may be accessed through the Programmer → Convert Programming Files menu. The Convert Programming File may be used to generate the required programming file and implement a custom flash programming flow.

For Nios II booting, a .jic file with the .sof file and .hex file must be generated. The .jic file will be programmed into the flash devices with Quartus Programmer. In this design example, the Micron MT25QL256 device with a default flow is supported in Convert Programing Files tool. However, in order to show an example of using the custom flow in the Generic Flash Programmer, the next steps will ignore that flash device is supported

The Generic Flash Programmer custom flow will create an XML file with the custom programming details. It is critical that the XML and the JIC are in the same directory for correct operation. The Generic Flash Programmer User Guide has additional details on the custom flow that are not covered here.

To generate the .jic file:

1. Go to File → Convert Programming Files
2. Select the Programming file type to be JTAG Indirect Configuration File (.jic)

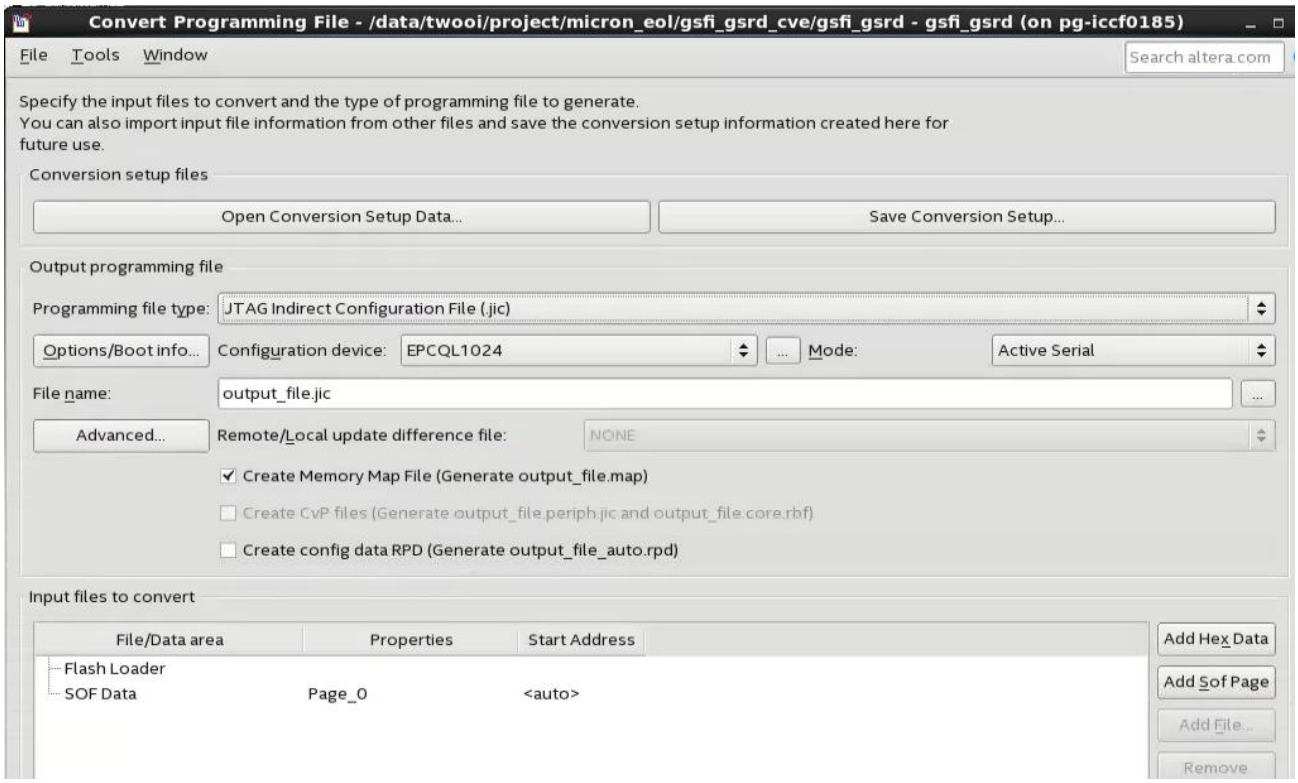

Figure 12: View of convert programming file tool

3. The Configuration device field allows for choosing a specific supported device or alternatively “<<new device>>” for an unsupported device. The flexibility allows users to create a custom flow for the programmer to work with unsupported flash devices. To create customize flow:
    a. Select <<new device>>
    b. Enter the information about Device name, Device ID, Device I/O voltage, Device density, Total device die and dummy clock for single IO or quad IO

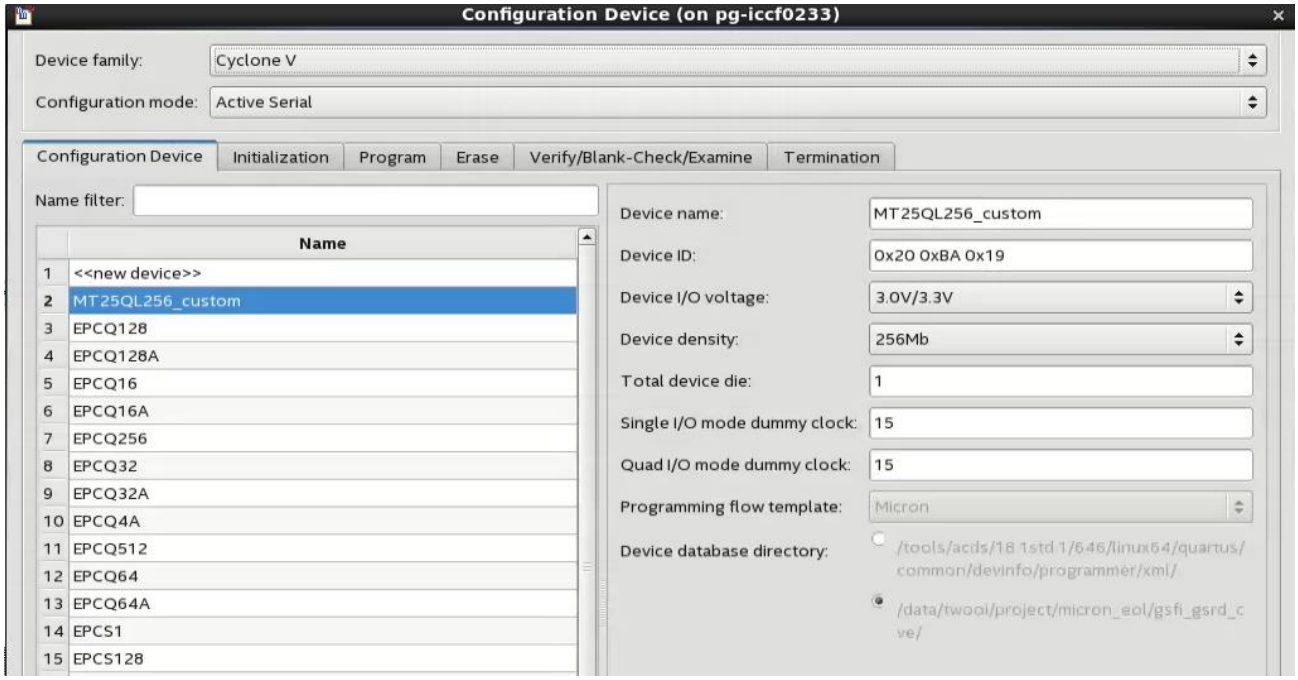

Figure 13: View of creating a customize programming flow

c. The Programming flow template helps users to define a template for flash operation in Initialization, Program, Erase, Verify/Blank-Check/Examine and Termination. Four templates are provided which are Micron, Macronix, Cypress and Winbond. In this example, we choose Micron

d. The Initialization tab contains all the flash operation defined in Micron template. Users can add operation by dragging the boxes which represent the read register operation (RR) or write register operation (WR) to the flow

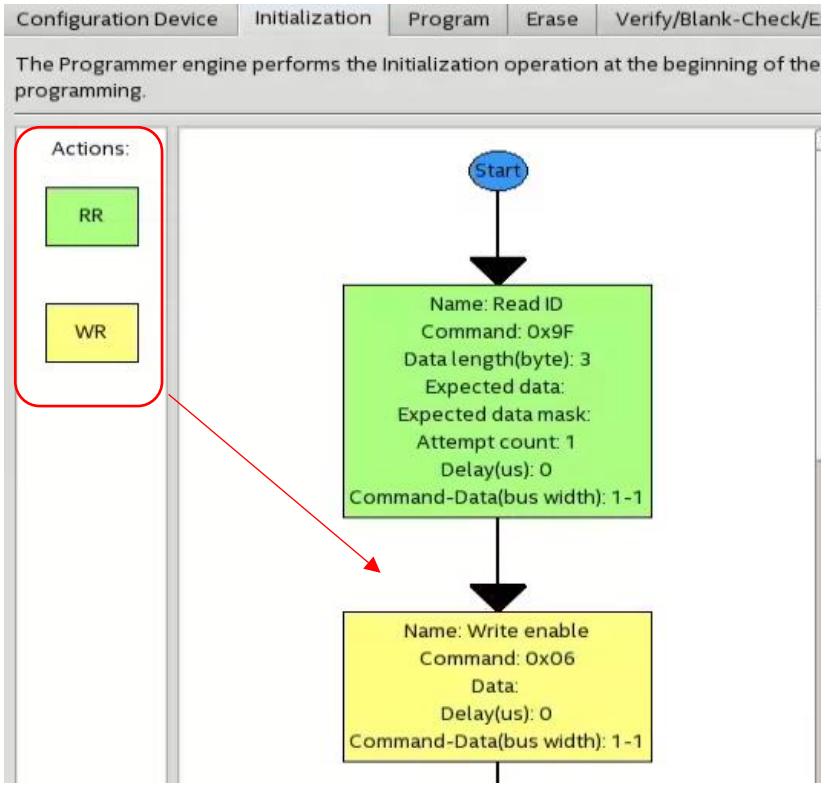

Figure 14: Initialization tab

e. In this example, a read NVCR operation is added by dragging the read register box (RR) into the flow where you want programmer to perform the operation. The Name is changed to “Read NVCR”, the Command is set to 0xB5, the Data length is set to “2”, the Attempt count is set to “1”, the Delay is set to “0” and the Command-Data bus width is set to “1-1”.

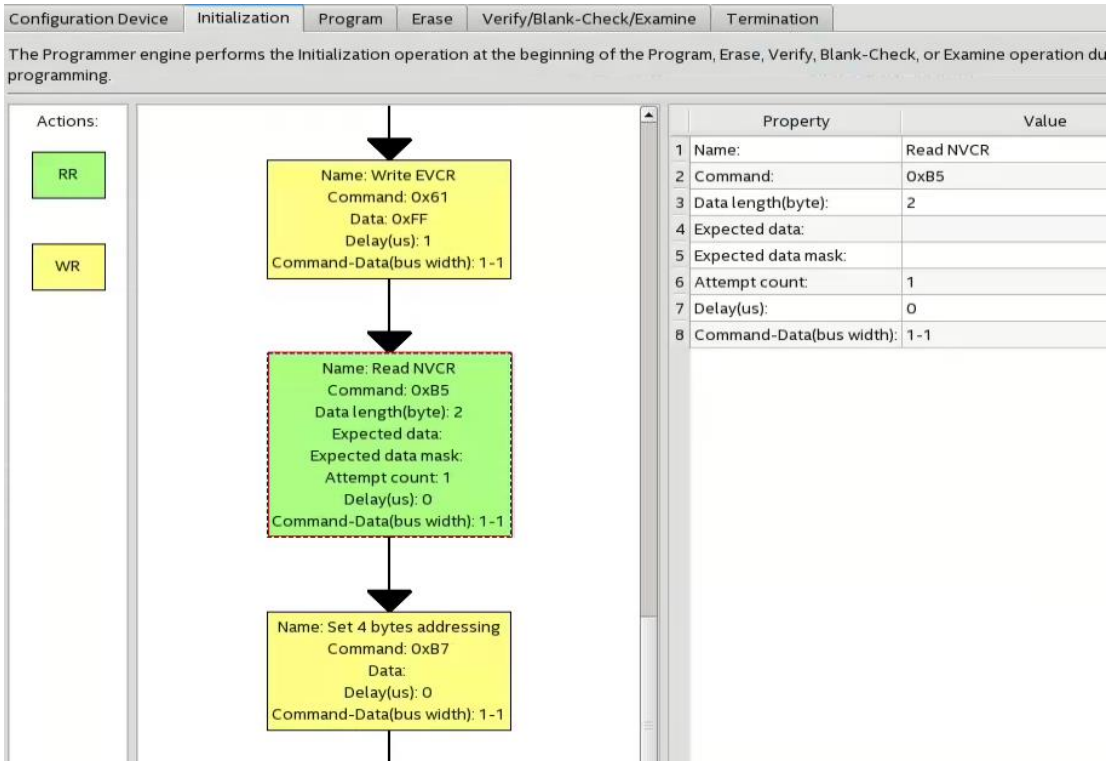

Figure 15: Addition on read register operation

4. Additional operations may be added as needed for the programmer to execute

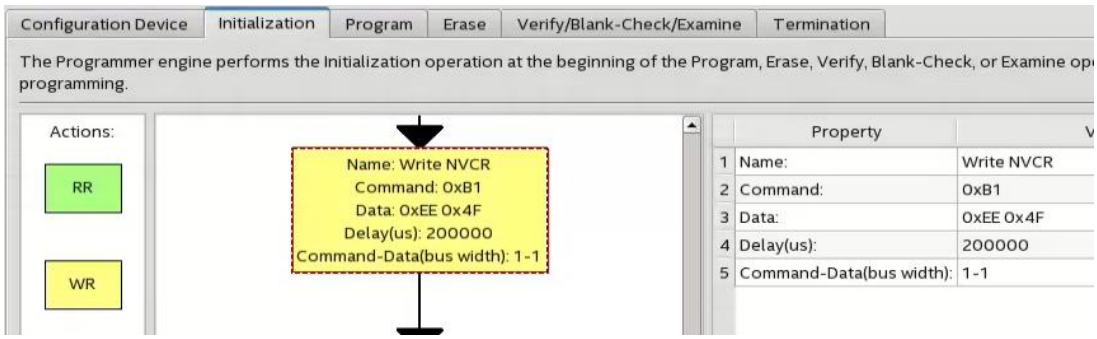

Figure 16: Modification on the write operation

5. Click on Apply after all the necessary operations has been added.
6. Choose the Flash Loader for the FPGA used and add the .sof file to the SOF Data.
7. Click Add Hex Data to add .hex file.
8. Click on Generate to generate the .jic file.

Once the .jic file is generated, the flash devices can programmed using the Quartus Programmer.

To program the .jic file

1. Open Quartus Programmer and click on “Auto-Detect” so that the FPGA is detected
2. Choose the .jic file to be programmed into the flash part
3. Click Start to start the program operation
4. Please take note that you will be able to see the NVCR register value with the read register for NVCR operation is added earlier

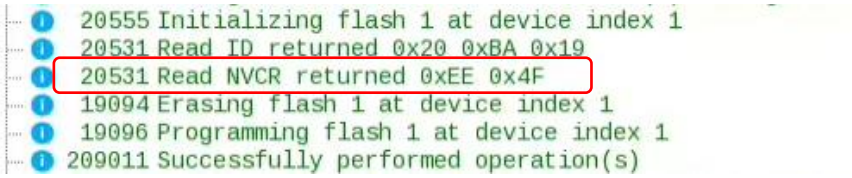

Figure 17: Print out messages from Quartus system messages

You will see a green bar with the word Finished when programmer successfully programmed the flash device

8 Revision History

Revision	Description
1.0	First version