Firmware Upgrade User Guide

Introduction

The *NI In-Field Firmware Upgrade* procedure allows the end-user to update the FPGA bitstream in the field, and without the need to access vendor-specific tools (i.e., Vivado from Xilinx). An ad-hoc multi-stage procedure is designed, with the objective to keep user intervention at a bare minimum.

The delivered archive contains an example design that can be used as a starting point for any PCIe design that has similar requirements:

1. PCIe interface available at power-on;
2. Ability to replace the bitstream in the Flash after deployment, without the use of third-party tools.

Please note that the example design targets Ultrascale architecture, and it is unknown whether the same applies to other families as well.

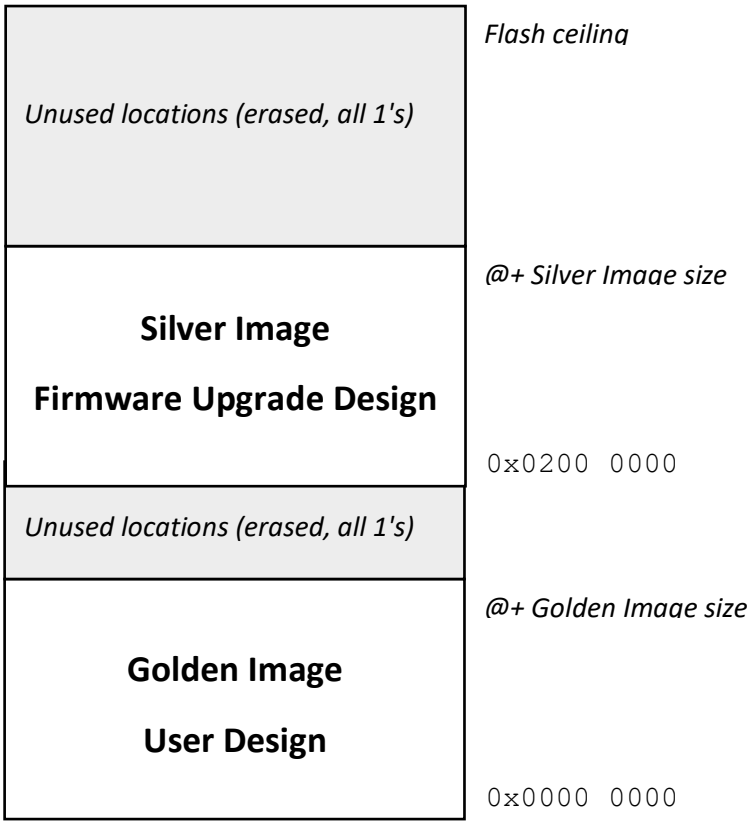
The example design has been fully validated on part number **xcku035-ffva1156-2-e** with 512-MiB **N25Q** family Micron QSPI NOR Flashes.

Flash layout

The Flash memory range is organized to store two images: the *Golden Image* and the *Silver Image*. The *Golden Image* configures the FPGA to run the user-specific application, with logic to support the first stages of the upgrade process. The *Silver Image*, on the other hand, contains the logic related to the latest stages of the upgrade process.

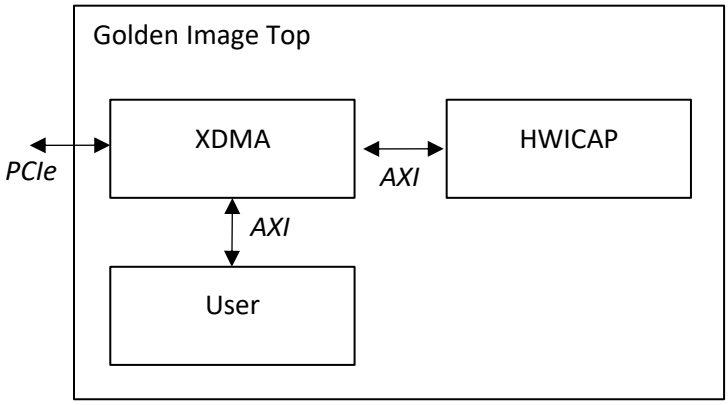
The Golden Image is likely to be (and will be) modified as new features are added or bugs are fixed. A replacement of the Silver Image is an exceptional event, instead.

By design convention, the Golden and Silver images are stored starting at address 0x0000_0000 respectively 0x0200_0000 (i.e., at an offset +32MiB).


Unused locations (erased, all 1's)
Flash ceiling
Silver Image
Firmware Upgrade Design
@+ Silver Image size
0x0200 0000
Unused locations (erased, all 1's)
Golden Image
User Design
@+ Golden Image size
0x0000 0000

Golden Image Design Notes

The simplified architecture is depicted next:



The XDMA block implements the PCIe end-point and exchanges PCIe packets with the host. The XDMA acts as an AXI bridge to the User logic **and** the HWICAP. The HWICAP instance gives user access to the ICAP port, through which dynamic reconfiguration can be triggered.

Constraints

1. The PCIe DMA/Bridge XDMA subsystem must be configured as Tandem PROM, so that the 100ms enumeration window from PCIe specs is satisfied;
2. An instance of the ICAP (AXI HWICAP IP) must be present and memory mapped over XDMA BARs at address 0xc001_0000; the HWICAP must be configured in AXI Lite mode;
3. It is not possible to use STARTUPE3 primitive to gain access to the Flash pins over Bank#0.

Memory map

Memory map for Golden Image:

BASE ADDRESS	CEILING (RANGE)	CONTENTS
`0xc000_0000`	`0xc000_ffff (64KiB)`	8KiB BRAM range, AXI Lite, 32-bit address
`0xc001_0000`	`0xc001_ffff (64KiB)`	GPIO for reconfiguration tests, AXI Lite, 32-bit address
`0xc002_0000`	`0xc002_ffff (64KiB)`	HWICAP, AXI Lite, 32-bit address
`0xc000_0000`	`0xc000_7fff (32KiB)`	8KiB BRAM range, AXI Full, 64'bit address

Functional test before and after upgrade

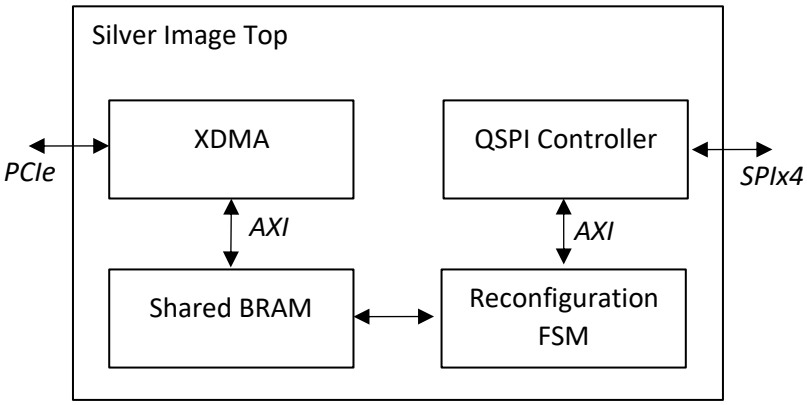
The Golden Image in the example design contains a simple bitwise operation on a 2-bit value. The Boolean operator is applied to the outputs of the memory-mapped GPIO on Channel 1. The result is stored on the input of the same memory-mapped GPIO, but on Channel 2.

There are two versions of the Golden Image: the first will perform a bit-wise AND, while the second will perform a bit-wise OR.

The `fpga_golden_image_check.sh` utility can be used to read the result and verify which image has been loaded. Input values are written at offset +0x10000, and output value is read from offset +0x10008.

Silver Image Design Notes

The simplified architecture is depicted next:



The XDMA block is now configured withouth the Tandem support, to gain access to the configuration pins (on Bank#0) after configuration. A centralized reconfiguration controller listens on commands sent by the host through the XDMA. Commands are stored in a Shared BRAM like in a message passing system. Access to the QSPI Controller is granted only to the reconfiguration controller for safety reasons.

Constraints

1. The PCIe DMA/Bridge XDMA subsystem must not contain any Tandem configuration, i.e. the Tandem configuration selection must be set to None;
2. An instance of the STARTUPE3 primitive must be present and memory mapped over the system bus at address 0xc002_0000.

Principles of operation

The Reconfiguration FSM waits for commands from the host. Commands are sent by the remote host via PCIe over the pre-defined BARs. Commands and command arguments are stored in the local BRAM range, that acts as a mailbox in a simplified message passing system.

The internal RAM also holds (a portion of) the new bitstream that will be eventually written to the QSPI Flash memory.

The RAM is partitioned into *Control* and *Data* groups. The former is mapped to the lowest 64 Bytes (rows 0 through 7), and contains status and control values for the exchange protocol; the latter (rows 8 through 8191) is mapped to the remaining range, with Bytes of the new FPGA image. The RAM contains plenty of space for data (8184 rows for a total range of nearly 32KiB), however Flash programming always happens at the page boundary, so that only a small portion of the data rows will be used. This is typically 64 rows (for 256 Byte pages).

ROW #	OFFSET	CONTENTS
8191	+0x7ffc	Data Bytes 32765 through 32762
…		
8	+0x0020	Data Bytes 3 through 0
7	+0x001c	Watermark
6	+0x0018	Reserved
5	+0x0014	Initialization status

4	+0x0010	Command result
3	+0x000c	Command status
2	+0x0008	Command argument 2/2
1	+0x0004	Command argument 1/2
0	+0x0000	Pending command

Upon successful boot, a Read to the Watermark (offset +0x001c) returns the special code `0x0D15EA5E` meaning that the Silver Image is up and running and the reconfiguration FSM is ready to accept commands.

Requests are sent through the command mailbox at +0x0000. The control Software reacts when a command different than `FWU_NOP` is found at that offset. Commands are accompanied with none, one or two arguments. Arguments are stored in the RAM and they must be programmed before programming the command opcode at offset +0x0000. Available commands are reported in the following table.

NUMERIC	OPCODE	BRIEF
0x00000000	`FWU_NOP`	Idle, no pending command. Command arguments are not valid
0xffff1111	`FWU_ERASE_SECTOR`	Erase an entire sector of the Flash. The sector identifier is derived from the address at offset +0x0004; any valid address within the target sector will trigger the sector erase operation. the number of sectors to erase is read from offset +0x0008. The highest address that has been erased will be stored at +0x0010
0x22222222	`FWU_FLASH_TEST`	Run a simple Erase/Write/Read cycle on a single page of the SPI Flash, to test connectivity and functionality mandatory for the Firmware upgrade process. The starting address is read from offset +0x0004
0xffff2222	`FWU_PROGRAM_PAGE`	Program a number of Flash pages (i.e., a multiple of 256 Bytes). The starting address is read from +0x0004, and the number of pages is read from +0x0008. Pages will be programmed sequentially. Data is taken from the local BRAM starting at +0x0020
0xffff3333	`FWU_READ_PAGE`	Read data from a number of Flash pages (i.e., a multiple of 256 Bytes). The base address is read from +0x0004, and the number of pages is read from +0x0008. Data will be stored in the local BRAM starting at +0x0020

Completion of a command is stored at offset +0x000c. Upon successful completion, every command generates a `0x00000001` value at this offset; otherwise `0xE330E330` code is used to identify a generic error. The value at offset +0x000c is valid if and only if the command has generated a `0x00000001` value as final status. The host shall then keep polling this register after a new command is issued.

The host is responsible of clearing the value at +0x000c **before** issuing a command to avoid race conditions. Failure to meet this requirement leads to unexpected behavior.

The FSM clears the value at +0x0000 every time the in-flight command is ended, to avoid unwanted re-execution.

Microblaze memory map

The memory map that the Microblaze sees is defined next:

BASE ADDRESS	CEILING (RANGE)	CONTENTS
0xc000_0000	0xc000_1fff (8KiB)	8KiB BRAM range, AXI Lite, 32-bit address
0x44a0_0000	0x44a0_ffff (64KiB)	QSPI controller, AXI Lite, 7-bit address

XDMA memory map

The memory map that the XDMA sees is defined next:

BASE ADDRESS	CEILING (RANGE)	CONTENTS
0x0000_0000_c000_0000	0x0000_0000_c000_1fff (8KiB)	8KiB BRAM range, AXI Full, 64-bit address
0xc000_0000	0xc000_1fff (8KiB)	8KiB BRAM range, AXI Lite, 32-bit address
0xc001_0000	0xc001_1fff (8KiB)	Dual-channel GPIO, AXI Lite, 32-bit address

For safety reasons the direct path from the XDMA to the QSPI controller are excluded, and an AXI bus exception will be generated causing the system to hang. For the same reason the direct path from the AXI Full port of the XDMA to the GPIO has been excluded, since the GPIO accepts only AXI Lite transaction.

How-to

FPGA operation in Normal mode

In Normal mode, the FPGA works following user specification, and access to its internals happen as expected over the PCIe link, either through the AXI4 Full interface or the AXI4 Lite interface. The XDMA driver is the only Software to be trusted while accessing the FPGA over PCIe. In Normal mode no user intervention on the Flash is allowed (nor expected).

FPGA operation in FWU mode

When a new bitstream is available from the designers, the FPGA must be operated in Firmware Upgrade mode (FWU). While in this mode, limited access to FPGA resources is available, and any data stored in the FPGA configuration is lost.

The Firmware Update process is an interactive one. The following lines will guide the user across all steps, assuming he/she is running Linux on a host machine and the XDMA driver has been previously loaded without errors.

1. Receive the `golden_image_new.bit` file from the design supplier; copy the image into a known folder, e.g. `$WORK/golden_image_new.bit`;
2. Open a Linux shell, and launch the following commands to reboot the FPGA in Firmware Upgrade mode; the Silver Image will be loaded:

```
; Reboot FPGA in Firmware Upgrade mode
$> ./fpga_reboot_silver.sh
```

3. Upon completion, a message invites the user **not** to power-off the machine, but to reboot it. This step is mandatory since the FPGA re-programming caused loss of the PCIe link, and the PC needs to re-enumerate the nodes;
4. Once reboot is complete, use the following Linux commands to double check PCIe node enumeration completed:

```
; List attached PCI devices
$> lspci
```

Expected output shall list Xilinx devices, something similar to:

```
01:00.0 Serial controller: Xilinx Corporation Device 8038 (rev
ff)
```

5. Run the following, to double check the Silver Image has been loaded as expected and PCIe link is healthy:

```
; Check Silver Image
$> ./fpga_silver_image_check.sh
```

6. If successful, the following text will be eventually printed:

```
          info: Silver Image connection test: PASS
```

7. In case of errors do **not** continue and ask support from supplier immediately. Otherwise, reconfigure the FPGA with the up-to-date bitstream using the provided utility:

```
; Program and reboot
$> ./fpga_upgrade_golden $WORK/golden_image_new.bit 0
```

8. If successful, the following text will be eventually printed:

```
      info: Golden Image has been successfully updated
```

9. Perform a full power-cycle on the host and enjoying your brand-new FPGA image!

Appendix

Archive contents

The following table reports the contents of the delivered archive. Contents have been generated using Vivado 2021.2.

TYPE	NAME	CONTENTS
Folder	golden_image/	Golden Image v1 design files (AND version)
Folder	golden_image_new/	Golden Image v2 design files (OR version)
Folder	fwu_image/	Firmware Upgrade design files
Tcl script	gen_memcfg.tcl	Vivado TCL script ro generate the Flash configuration file using two bitstreams (please refer to the Flash layout section): one from golden_image and one from fwu_image
Bash script	gen_memcfg.sourceme	Utility script to launch memory configuration file creation
Folder	scripts	Contains scripts to run the Firmware Upgrade process
Folder	xdma_driver	XDMA driver sources
File	FirmwareUpgrade.docx	This file
File	FirmwareUpgrade.pdf	PDF version of this manual
Folder	ni_fwu	Contains C source files for the Firmware Upgrade FSM

Running the example design

Once the archive has been extracted, the following steps will give all files required to be able to play with the Firmware Upgrade utility.

1. Launch Vivado and generate bitstream for the following projects:
    a. golden_image/project_1.xpr
    b. golden_image_new/project_1.xpr
2. Launch Vivado opening the fwu_image/project_1.xpr and run implementation;
3. With fwu_image/project_1.xpr still open, generate the XSA description file (File -> Export -> Export Hardware);
4. Launch the Vitis IDE and create a new workspace;
5. Create a new application project named ni_fwu using the XSA generated so far and the Hello World template;
6. Import the following sources from the ni_fwu folder:
    a. ni_fwu.c
    b. qspi_flash.h and qspi_flash.c
7. Remove the default helloworld.c file and compile the ni_fwu project;
8. From Vivado, associate the Microblaze in the block design with the ni_fwu/Debug/ni_fwu.elf binary and run bitstream generation;
9. Open a shell and point to the folder containing the gen_memcfg.tcl file, and run the following command:

```
$> source gen_memcfg.sourceme
```

10. No errors nor critical warnings shall arise. Two files will be generated: `multi_boot_image.mcs` and `multi_boot_image.prm`. Use them to program the Flash of the target board using Vivado; be sure to select "entire configuration memory" when asked;
11. Run a full power-cycle;
12. Compile the XDMA driver and the utilities

```
$> cd xdma_driver/dma_ip_drivers/XDMA/linux-kernel/xdma
$> make
$> cd ../tools
$> make
```

13. Insert the XDMA driver module

```
$> cd xdma_driver/dma_ip_drivers/XDMA/linux-kernel/xdma
$> sudo insmod ./xdma.ko
$> sudo dmesg
```

14. The output of `dmesg` shall reveal the XDMA has been successfully loaded, BARs have been detected and configuration BAR has been found;
15. Now everything is ready to follow the HowTo from previous sections.

Limitations and points of improvement

- Ultrascale+ architecture is **not** supported. The HWICAP does not boot properly in the Golden Image; it is therefore not possible to swap the FPGA mode from Normal to FWU;
- An instance of the HWICAP in the Silver Image is **not** supported;
- The HWICAP in the Golden Image must be configured in AXI Lite mode;

- The Silver Image offset (0x0200_0000, i.e. @+32MiB) is way over provisioned. 32MiB is plenty of room for a bitstream. Although this gives practically no problems with any updated image in the future, it slows down the upgrade process since the algorithm erases the entire Golden Image range;
- The provided Software for the Golden Image upgrade process contains AXI Lite only accesses over the XDMA BARs. This reduced the mean throughput. Some portions of the algorithm can be sped up by using DMA access, namely the new bitstream's Byte transfer in the remote BRAM;