Flashy
User Guide

Utility to programme the flash memory on the DTH

Revision: 17 (8 Jun 2020)

Authors: Christoph Schwick, Dominique Gigi, Jeroen Hegeman

Table of Contents

Document Revisions

Revision	Date	Comment
9	2020-01-14	First version of the document
14	2020-04-17	Added support for the TCDS fpga. New batch mode.
16	2020-06-08	Version for the first release of the DTH Kit

1 Introduction

The DTH contains two main FPGAs. The first is mainly performing tasks for the DAQ system whereas the other one performs the tasks for the TCDS system. Both FPGAs load their firmware at power up from a dedicated flash memory chip. The capacity of these memory chips is large enough to hold multiple firmware images. Since the images are compressed their size varies and therefore it is not possible to exactly say how many images can be loaded into the the flash chips.

This User Guide describes the software utilities used to write images into the flash chip, to erase regions of the flash chip or to protect regions of the flash chip from accidental overwriting. Further the utilities contains a series of functions for debugging or advanced usage.

Flashy is a python command line utility which uses the package “consolemenu” to display ASCII menus. In order to accelerate the process of programming an image file into the flash chip Flashy is calling a programme written in C++ with the name “FlashProgrammer”.

The packages contains an additional C++ programme called “flashConverter”. This programme can be used to convert an image file for the FPGA firmware in INTEL format (.mcs) to a binary file which can be written to the flash memory by the Flashy programme.

1.1 The flash chip of the DTH

The flash memory chips deplyed on the DTH are of the type SL29GL01GS (Cypress). This chip features 1Gbit of flash memory organised in 1024 sectors of 128kBytes each. Sectors can be individually erased or written or protected in various ways against accidental programming or erasing. The Flashy utility supports one mode of protection where a non volatile bit per sector is used to inhibit programming or erasing the sector by setting the bit to 1. This protection can be removed by resetting the respective protection bit to ‘0’.

As usual with these type of flash chips, erasing a sector sets all bits in the sector to ‘1’ and programming means setting selected bits to ‘0’. Programming or erasing the chip takes substantial time (order of minutes), so please be patient during these operations.

Data is organised in words of 16bit (i.e. a 16 bit data bus is available at the output of the chip). This has consequences on how to specify addresses for the flash chip in the software. Addresses are in terms of 16 bit words (i.e. one address addresses 2 bytes. Addresses are incremented by one to go to the next 16 bit word.)

2 Installation

The software is distributed as part of the DTH-Kit software via an RPM. The yum repository for this software is housed at gitlab at CERN:

https://gitlab.cern.ch/dth_p1-v2/cmd_dth.yum

The DTH-Kit software is pre-installed on the DTH-Kits delivered to sub-detectors. Therefore these installation instructions are only given for reference and can be skipped by users of the DTH-Kit.

To install the DTH-Kit software you can add the yum-repository to your system and then install the RPM with the following commands executed as super user:

```
curl https://gitlab.cern.ch/dth_p1-v2/cmd_dth.yum/raw/master/DTH-Kit_p1_v2.repo
-o /etc/yum.repos.d/DTH-Kit_p1_v2.repo

yum -y install dth_software_kit
```

Flashy depends on the XDAQ HAL library and the xpci kernel driver to access the flash chip of the DAQ FPGA. Sources and binaries of the xpci driver are available in the yum repository. HAL needs to be installed via the standard xdaq installation procedure.

The configuration file of the xdaq repository is listed in the Appendix. It should be installed into *etc*/ yum.repos.d/. To install the HAL library the core and the worksuite groups should be installed with the following commands (as root):

```
yum -y group install --skip-broken cmsos_core cmsos_worksuite
```

(the –skip-broken flag avoids that the installation fails due to some missing dependencies of the AMC13 software which is not needed)

For the TCDS FPGA uHAL and the XILINX xdma driver need to be installed. uHAL can be installed via a dedicated yum repository. The repository and the software can be installed with the following commands (executed as root):

```
curl http://ipbus.web.cern.ch/ipbus/doc/user/html/_downloads/ipbus-
sw.centos7.x86_64.repo -o /etc/yum.repos.d/ipbus-sw.repo

yum -y groupinstall uhal
```

The xdma driver is provided by the DAQ group in form of 2 RPMs available from the yum repository of the DTH-Kit software. They are isntalled with the commands (as root)

```
yum install kmod-xdma_tcds2 xdma_tcds2
```

Be aware that the installation of the xdma driver (kernel module) takes a very long time (many minutes) on ComExpress board. Do not interrupt this installation.

2.1 Installation from the sources:

The sources are included in the git repository for the DTH software. This can be checked out with the command

```
git clone https://gitlab.cern.ch/dth_p1-v2/dth_software.git
```

Before using the utility the C++ programmes need to be compiled. For this it is assumed that XDAQ together with the worksuite and the uHAL software are installed on the computer (see above).

To compile the C++ programms simply type do

```
cd flash_config
make
```

To use the software for debugging purposes from the directories of the git repository the target “pylocal” can be used. It sets up a directory structure with symbolic links to the software together with a file setting environment variables when sourced. To use this feature type

```
make pylocal
cd pylocal
source pythonpath.sh
```

Then the DTH_Flashy can be executed with

```
python ./DTH_Flashy.py --fpga {daq|tcds}
```

3 Usage

To use the software the LD_LIBRARY_PATH of your environment must include the path the the XDAQ libraries:

```
export LD_LIBRARY_PATH=/opt/xdaq/lib
```

The following paragraphs explain the usage of the various utilities.

3.1 FlashConverter

3.1.1 Invocation:

```
FlashConverter {filename}
```

filename is a firmware file created with the Vivado tools in INTEL mcs format (extension “.mcs”). The utility will create a new file with the same name but the extension “.bin” which will the same firmware in binary format ready to be written into the flash chips of the DTH board.

3.2 Flashy

3.2.1 Invocation

```
Flashy.py --fpga {daq|tcds} [--debug]
```

The required parameter --fpga chooses the flash chip to work with: the flash of the DAQ FPGA or the flash of the TCDS FPGA.

The DAQ and the TCDS FPGAs implement different PCIe endpoints with their respective Device Ids, therefore the parameters change according to the flash chip you want to access with Flashy.

Once started the main menu is presented to the user:

```
┌──────────────────────────────────────────────────────────────────────────────────┐
│                                                                                  │
│                                    Flashy                                        │
│                                                                                  │
│                        Tools for the flash memory on the DTH                     │
│                                                                                  │
├──────────────────────────────────────────────────────────────────────────────────┤
│                                                                                  │
│    1 - Dump Information about flash memory                                       │
│    2 - Write content of binary bitstream file to the flash                       │
│    3 - Compare file content to content in the flash                              │
│    4 - Check for blank sectors in flash chip                                     │
│    5 - Erase one or many sectors                                                 │
│    6 - Erase entire flash chip                                                   │
│    7 - Load FPGA firmware from Flash                                             │
│    8 - Advanced and Debug menu                                                   │
│    9 - Exit                                                                      │
│                                                                                  │
│                                                                                  │
└──────────────────────────────────────────────────────────────────────────────────┘
SELECT>
```

The menu items are mostly self explaining. They are selected by typing the number of the item on the keyboad. Some actions will display information on the screen. To get back to the menu the user should hit “Enter”.

The first menu point is useful to see if the general interface to the flash chip is operational.

Several menu points ask you to specify an address of the flash chip. In general two formats are accepted: If you want to specify the startaddress of a given sector you can so this by typing the letter ‘s’ followed by the sector number (starting with sector number 0, being the first sector). A decimal or hexadecimal address can also be given (hexadecimal preceeded by the prefix 0x). Note that addresses given as numbers address 16bit words: Hence to address the byte 32 and 33 in the flash you would have to specify the address 16 or 0x10.

3.2.2 Checking for empty sectors

The output of the 4th menu point is compressed in order to avoid to display 1024 data records for the 1024 sectors in the chip. Starting with sector ‘0’ the programm will go through the flash memory and check for each sector if it is erased (all bits on ‘1’) or not. It will only print out a message when the state of a sector is different from the previous sector. This will allow you to determine the empty and the programmed regions of the flash chip (with a granularity of an entire sector). An example output for a flash chip containing 5 firmware images is given below:

```
An empty sector:       92      11.5MB          adr: 00b80000
A non empty sector:  100      11.5MB          adr: 00c80000
An empty sector:     194      24.2MB          adr: 01840000
A non empty sector:  200      24.2MB          adr: 01900000
An empty sector:     294      36.8MB          adr: 024c0000
A non empty sector:  300      36.8MB          adr: 02580000
An empty sector:     392      49.0MB          adr: 03100000
A non empty sector:  400      49.0MB          adr: 03200000
An empty sector:     492      61.5MB          adr: 03d80000
Dump of Flash Timer
  1)  Chip blank check                      00h 00m  3.935s      acc: 00h 00m
3.935s
```

3.2.3 Load FPGA firmware from the Flash

In this option the software is saving the contents of the PCI configuration to internal memory and then reloading the FPGA with an image from the flash. The user has to specify the sector where the firmware image starts. After programmation the PCI configuration is written back to the PCI interface of the firmware so that the host computer does not crash when trying to access the DTH. Note that this only works if previously the DTH was running a firmware with the same PCIe configuration parameters and the DTH was successfully detected by the PC. During routine firmware upgrades this is normally the case. But exceptionally a reboot might be necessary to make the PC aware of PCI configuration changes (e.g. the requested memory space of a BAR has been increased).

3.2.4 The Advanced menu

In the advanced menu small regions of the flash memory can be accessed (either for reading or writing). In this menu you also find the possibility to check the protection bits of the sectors or to set or unset them. It is only possible to unprotect all sectors of the flash chip at onces. Hence to change the protection ALL protected regions need to be re-specified after the erasing of the protection (this is due to the fact that the protection bits themselves are flash memory bits which can only be erased as a group, like the bits in a sector).

In this menu you also find special commands to try to get the flash in a well defined state in case some previous operation failed: you can exit the ASO (Address Space Overlay) mode which is used by most of the commands of the flash chip, and you can abort an ongoing writing of a line Buffer (a mode used to efficiently write large blocks to the flash memory).

3.2.5 Further command line options

DTH_Flashy accepts some additional command line options.

--help	: prints usage information

--debug	: allows to inspect standard output and error output of the programme for debugging purposes. (During normal operation the console menus regularly clear the screen which makes reading the output impossible)
--batch	: batch mode operation (see next section)

3.2.6 Batch mode

The option –batch allows some operations to be executed in batch mode (for usage in scripts for example)

To operate in batch mode DTH_Flashy needs to be invoked in the following way:

```
DTH_Flashy.py --batch --command {command} [parameter options]
```

The following table lists the supported batch commands with their required parameter options

command	parameters	comment
`program`	`--file` `--start_adr`	Programs the flash chip with the binary content of a file containing a firmware image. `--file` indicates the path to the binary image file. `--start_adr` indicates the first sector in the flash where to start the programming. As in interactive mode the start address can be given as a sector number in which case it must be preceded by the letter ‘s’ (e.g. s100 indicates sector 100) or it can be given as an address (decimal or hexadecimal with the prefix 0x. Be aware that addresses have to indicate the first address of a sector. Addresses are given in units of bytes.)
`verify`	`--file` `--start_adr`	Compares the flash chip contents with the binary content of a file. `--file` indicates the path to the binary image file. `--start_adr` indicates the first sector in the flash where to start the programming. As in interactive mode the start address can be given as a sector number in which case it must be proceeded by the letter ‘s’ (e.g. s100 indicates sector 100) or it can be given as an address (decimal or hexadecimal with the prefix 0x. Be aware that addresses have to indicate the first address of a sector. Addresses are given in units of bytes.)
`unprotect`		Unprotects all sectors of the chip (including the sectors containing the “golden design”!) Normally this is only used in order to change the protected sectors, i.e. this command is followed by a protect command.
`protect`	`--start_sector` `--num_sector`	Protect a region of the flash chip against accidental erasing

command	parameters	comment
		or programming. start_sector is a number between 1 and 1023 and num_sector specifies the number of sectors to protect starting from start_sector.
`eraseAll`		Erases the entire chip (except for sectors which are protected)
`erase`	`--start_sector` `--num_sector`	Erases a region of the flash chip. start_sector is a number between 1 and 1023 and num_sector specifies the number of sectors to erase starting from start_sector.
`loadFPGA`	`--start_adr`	Loads the firmware which is at the specified address in the Flash memory. The format of the start_adr is the same as for the “program” and “verify” commands.

4 Preparing a firmware image for the Flashy utility

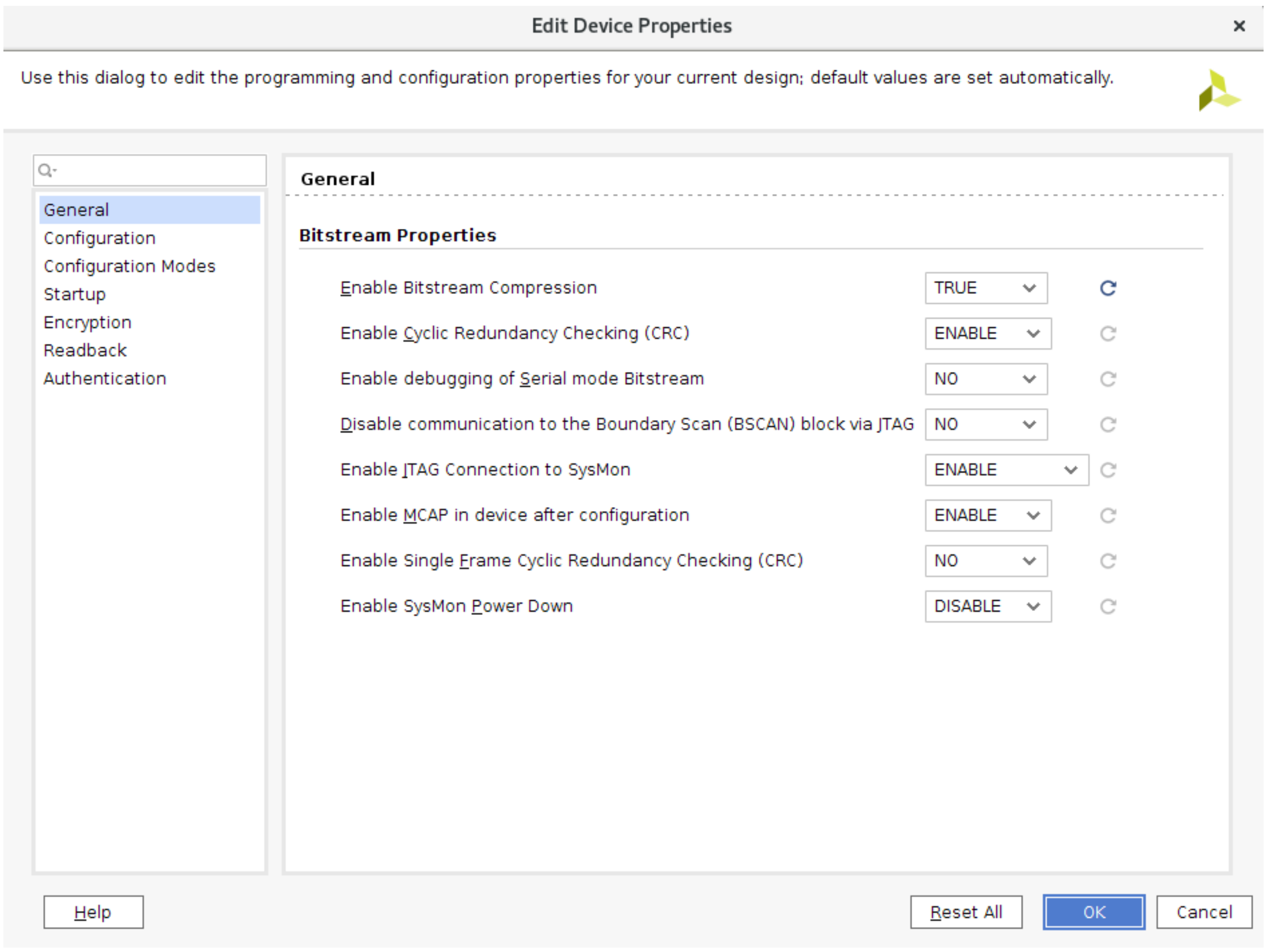
This chapter documents how to prepare a DTH firmware file ready to be written to the flash memory with Flashy. This operation is executed by the DAQ group and users of the DTH can skip this chapter.

4.1 Bitstream options to choose when generating the bitfile of the firmware

If the implemented design is opened choose the Settings of the Project Manager and subsequently the Bitstream settings. Inside go to the link “Configure additional bistream settings”)

What follows is a set of screen shots for the settings in the various categories.

4.1.1 General



Edit Device Properties

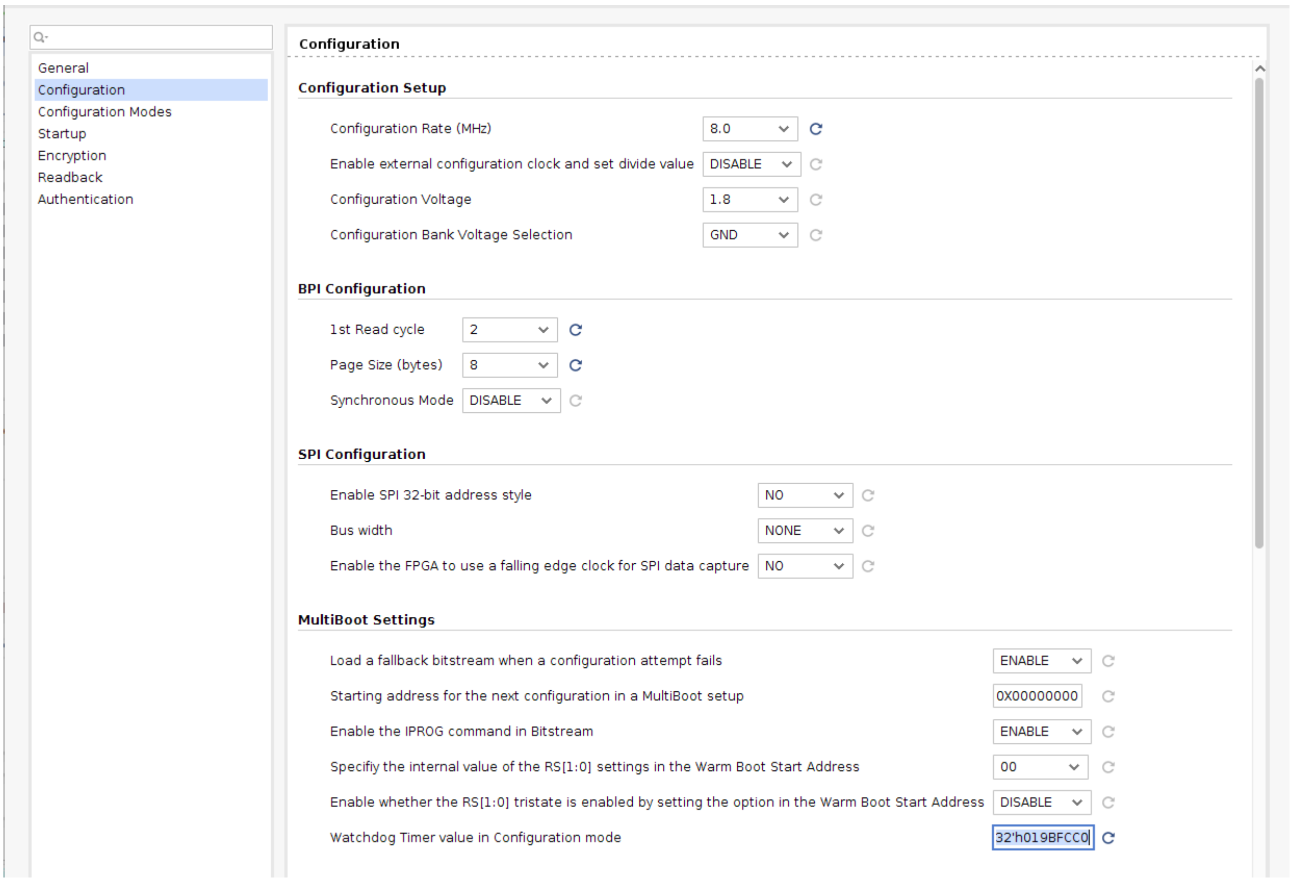
Use this dialog to edit the programming and configuration properties for your current design; default values are set automatically.

General

Bitstream Properties

Enable Bitstream Compression	TRUE
Enable Cyclic Redundancy Checking (CRC)	ENABLE
Enable debugging of Serial mode Bitstream	NO
Disable communication to the Boundary Scan (BSCAN) block via JTAG	NO
Enable JTAG Connection to SysMon	ENABLE
Enable MCAP in device after configuration	ENABLE
Enable Single Frame Cyclic Redundancy Checking (CRC)	NO
Enable SysMon Power Down	DISABLE

4.1.2



Configuration

Configuration Setup

Configuration Rate (MHz)	8.0
Enable external configuration clock and set divide value	DISABLE
Configuration Voltage	1.8
Configuration Bank Voltage Selection	GND

BPI Configuration

1st Read cycle	2
Page Size (bytes)	8
Synchronous Mode	DISABLE

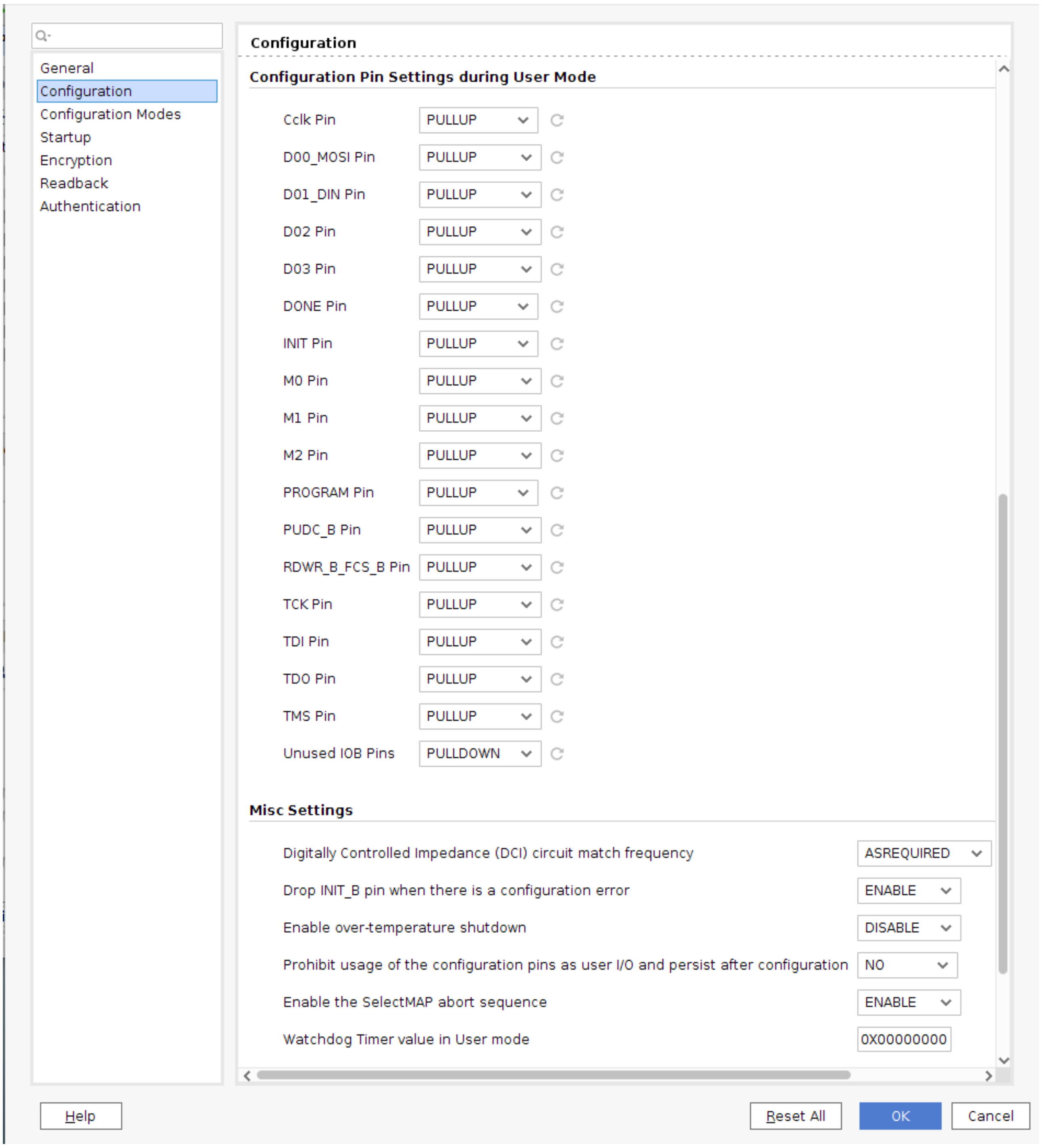
SPI Configuration

Enable SPI 32-bit address style	NO
Bus width	NONE
Enable the FPGA to use a falling edge clock for SPI data capture	NO

MultiBoot Settings

Load a fallback bitstream when a configuration attempt fails	ENABLE
Starting address for the next configuration in a MultiBoot setup	0X00000000
Enable the IPROG command in Bitstream	ENABLE
Specifiy the internal value of the RS[1:0] settings in the Warm Boot Start Address	00
Enable whether the RS[1:0] tristate is enabled by setting the option in the Warm Boot Start Address	DISABLE
Watchdog Timer value in Configuration mode	32'h019BFCC0

4.1.3 Configuration



Configuration

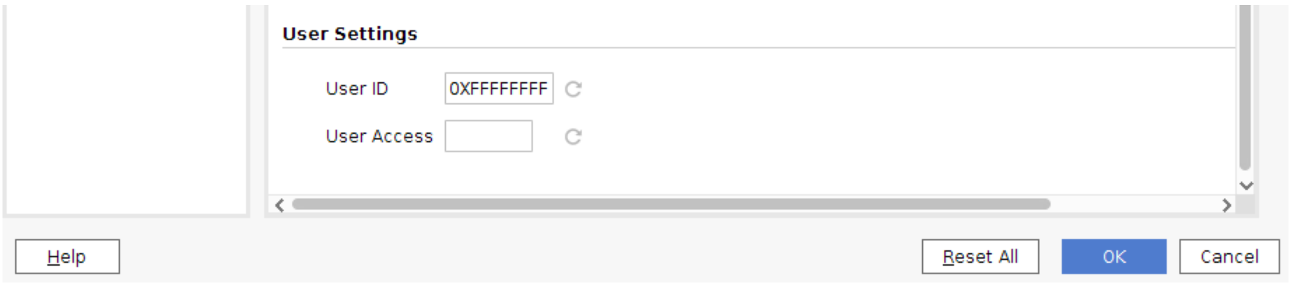
Configuration Pin Settings during User Mode

Cclk Pin	PULLUP
D00_MOSI Pin	PULLUP
D01_DIN Pin	PULLUP
D02 Pin	PULLUP
D03 Pin	PULLUP
DONE Pin	PULLUP
INIT Pin	PULLUP
M0 Pin	PULLUP
M1 Pin	PULLUP
M2 Pin	PULLUP
PROGRAM Pin	PULLUP
PUDC_B Pin	PULLUP
RDWR_B_FCS_B Pin	PULLUP
TCK Pin	PULLUP
TDI Pin	PULLUP
TDO Pin	PULLUP
TMS Pin	PULLUP
Unused IOB Pins	PULLDOWN

Misc Settings

Digitally Controlled Impedance (DCI) circuit match frequency	ASREQUIRED
Drop INIT_B pin when there is a configuration error	ENABLE
Enable over-temperature shutdown	DISABLE
Prohibit usage of the configuration pins as user I/O and persist after configuration	NO
Enable the SelectMAP abort sequence	ENABLE
Watchdog Timer value in User mode	0X00000000

Remark: The hex value under “Watchdog Timer value in Configruation mode” corresponds to 27000000 which results in 3.375s timeout with a 8MHz configuration clock. This means that if the configuration does not terminate successfully after 3.375 seconds the golden design at addresse 0 of the flash memory will be automatically loaded.



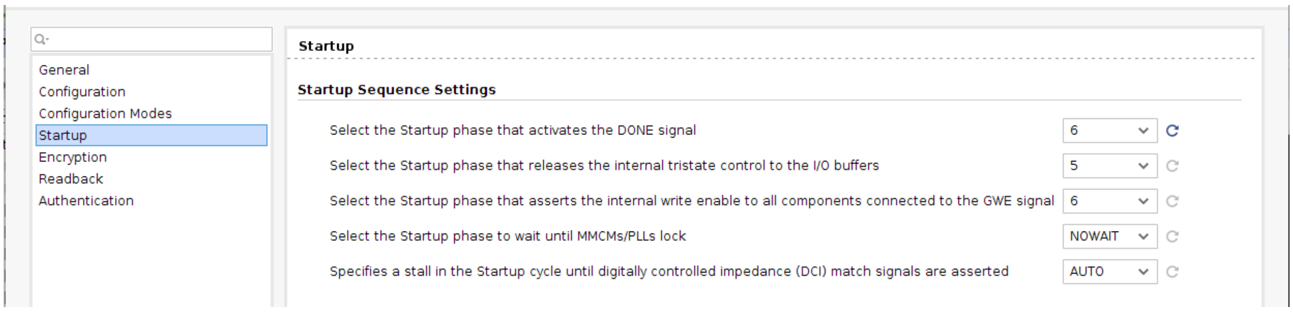
User ID 0XFFFFFFFF

User Access

4.1.4 Configuration Modes

Choose Mster BPI x16

4.1.5 Startup



Select the Startup phase that activates the DONE signal	6
Select the Startup phase that releases the internal tristate control to the I/O buffers	5
Select the Startup phase that asserts the internal write enable to all components connected to the GWE signal	6
Select the Startup phase to wait until MMCMs/PLLs lock	NOWAIT
Specifies a stall in the Startup cycle until digitally controlled impedance (DCI) match signals are asserted	AUTO

4.1.6 Encryption, Readback and Authentication

Do not change any settings from the defaults (security features selected)

4.2 Generate an mcs file with Vivado

1. In Vivado open the implemented design for which you want to create a firmware image.
2. Under Tools choose “Generate Memory Configuration file”

3. Choose MCS format (default)
4. Choose an output filename (use the file browser button on the right)
5. For Memory Part choose s29gl01gs-bpi-x16 (Spansion – 1024Mb)
6. Click on “Load bistream files” and direct the Bifile browser to your toplevel bitfile ({top_hierachy_module_name}.bit in the implementation directory)

The programme will produce an mcs file from the bitfile of your design. In the TCL console you find the command to do the same thing programmatically on the console.

This file you can use as an input to the programme “FlashConverter” which will produce a .bin file which you can use to programme the flash with the Flashy.py utility as described above.

5 Appendix

5.1 XDAQ yum repository for xdaq version 15

```
[xdaq]
name=XDAQ Software Core
baseurl=http://xdaq.web.cern.ch/xdaq/repo/core/15/cc7/x86_64/RPMS/
enabled=1
gpgcheck=0

[xdaq-worksuite]
name=XDAQ Worksuite
baseurl=http://xdaq.web.cern.ch/xdaq/repo/worksuite/15/cc7/x86_64/RPMS/
enabled=1
gpgcheck=0

[xdaq-development]
name=XDAQ development master
baseurl=http://xdaq.web.cern.ch/xdaq/repo/development/core/master/cc7/x86_64/RPMS/
enabled=1
gpgcheck=0

[worksuite-development]
name=Worksuite development master
baseurl=http://xdaq.web.cern.ch/xdaq/repo/development/worksuite/master/cc7/x86_64/RPMS/
enabled=1
gpgcheck=0
```