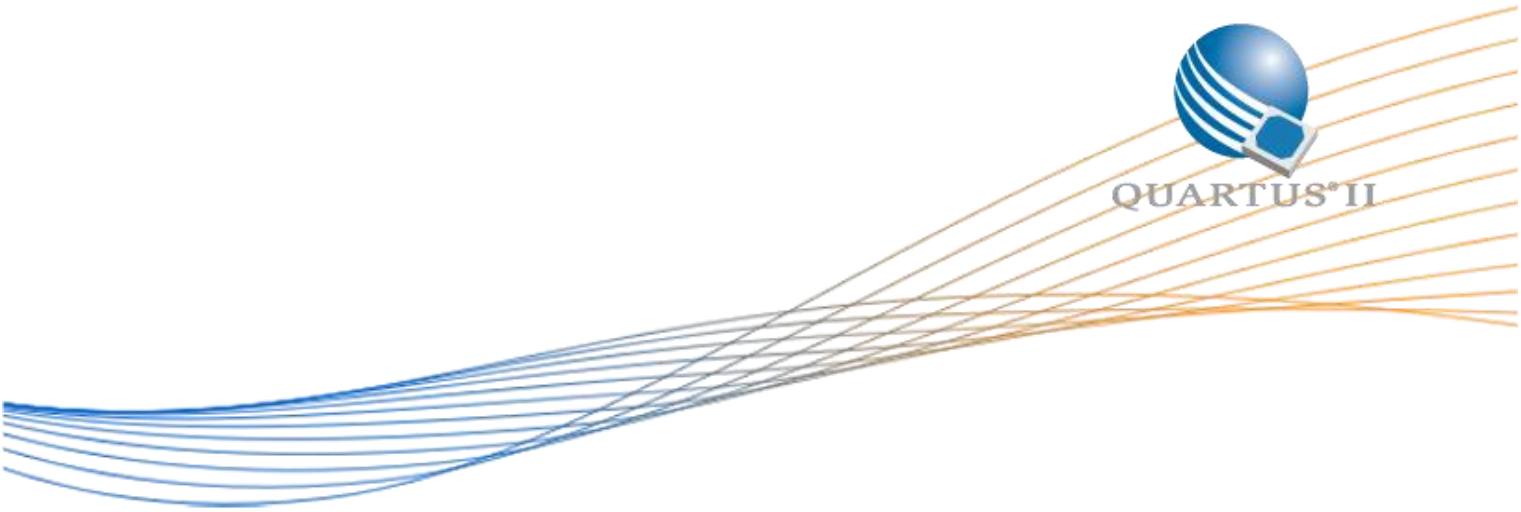


I2C Remote System Upgrade Example (Max 10)

Author: Isaac Kruger

Date: 11/11/15

Revision: 1.1

Table of Contents

Introduction

This example implements a remote system upgrade on a Max 10 device via I2C protocol. This example requires TWO Max 10 kits: a Max 10M50 Development Kit is used as the I2C master device, and a Max 10M08 Evaluation Kit is used as the I2C slave device. The master device will store an updated image for the slave device in its on-chip flash. When a push-button is pressed, the master device will send this image over I2C, and the slave device will execute a remote update.

Prerequisites

You will need the following software to implement this example:

- Quartus II v15.1 with the Nios II Software Build Tools for Eclipse installed

You will need the following hardware to implement this example:

- Max 10M50 Development Kit
- Max 10M08 Evaluation Kit
- Wires
- Two 5kohm resistors
- Some method of building a simple circuit (e.g. breadboard or soldering iron)

Theory of Operation

The on-chip flash on Max 10 devices is divided into five sectors. This is true regardless of the overall size of the on-chip flash, which varies greatly between devices (in this example, the Max 10M50 has much more on-chip flash available than the Max 10M08). Three of these five sectors are designated as configuration flash memory or CFM (CFM0, CFM1, and CFM2). By default, CFM0 contains the "factory image" for the Max 10 device- essentially the image that runs whenever the device turns on and boots up. CFM1 and CFM2 combined are exactly the same size as CFM0. The two remaining sectors are designated as user flash memory or UFM (UFM0 and UFM1). The UFM is much smaller than the CFM- approximately one tenth the overall size.

The Max 10 presents the user with many options on how to utilize the on-chip flash. For the purposes of this example, the important option is how to utilize CFM1 and CFM2. One possibility is to combine them into a single block (referred to as CFM1/2). This block is exactly the same size as CFM0, which allows for a second image to be stored. The Max 10 can then boot up from this image instead of the factory image. On the Max 10M08 Evaluation Kit switch 6 of the user switches (SW3) controls which image is used for booting ("off" = CFM0, "on" = CFM1/2). The Max 10M08 slave device has CFM1 and CFM2 set up in this way, as seen in Figure 1.

An alternative possibility is to keep CFM1 and CFM2 separate, and utilize them as "virtual user flash". In effect, this makes both CFM1 and CFM2 additional sectors of UFM. Anything can be stored in these sectors, and they cannot be used for booting. This can be useful because CFM1 and CFM2 are comparatively much larger than the default UFM. The Max 10M50 master device has CFM1 and CFM2

set up in this way, as seen in Figure 1. Refer to the Max 10 User Flash Memory User Guide for more information on the flash memory built in to the Max 10 family of chips.

Figure 1 below shows the setup of the on-chip flash on both master and slave devices at the start of the example. Note that CFM0, CFM1, and CFM2 are all roughly to scale with each other, but UFM0 and UFM1 have been enlarged to make the diagram easier to read. This figure will be modified later in this guide to reflect changes in the contents of the on-chip flash.

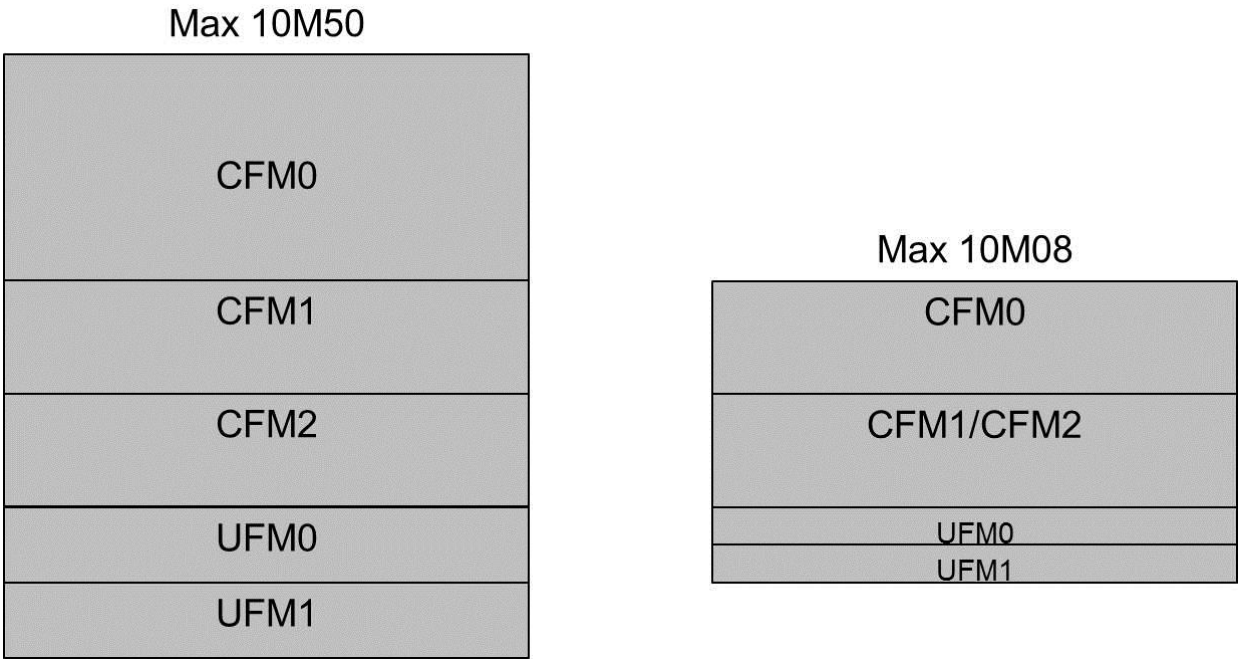


Figure 1

The flow for this example can be divided into four distinct stages. These stages will be discussed at a high level in this section, however the actual instructions will be in the "Executing Example Design" section of this guide.

Stage 1 – Program Master Image into Max 10M50

The first stage is to program the Max 10M50 master device with the I2C master image. This will utilize CFM0 for the hardware image, and UFM0/1 for the software component. Figure 2 contains an illustration of the on-chip flash contents after this step, and Figure 3 contains a block diagram of the I2C master image.

The master image contains a Nios II processor which pulls bytes from CFM1 and CFM2 (which will contain images for the 10M08 slave device as described later in this guide) and transmits them over

I2C.

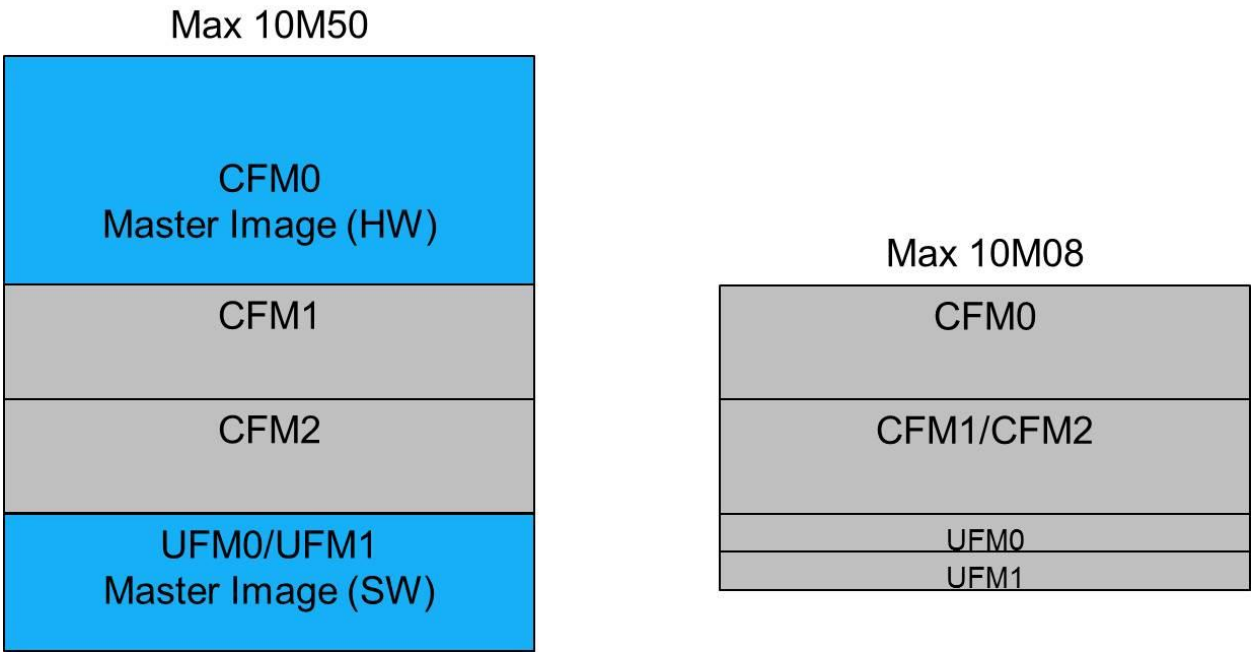


Figure 2

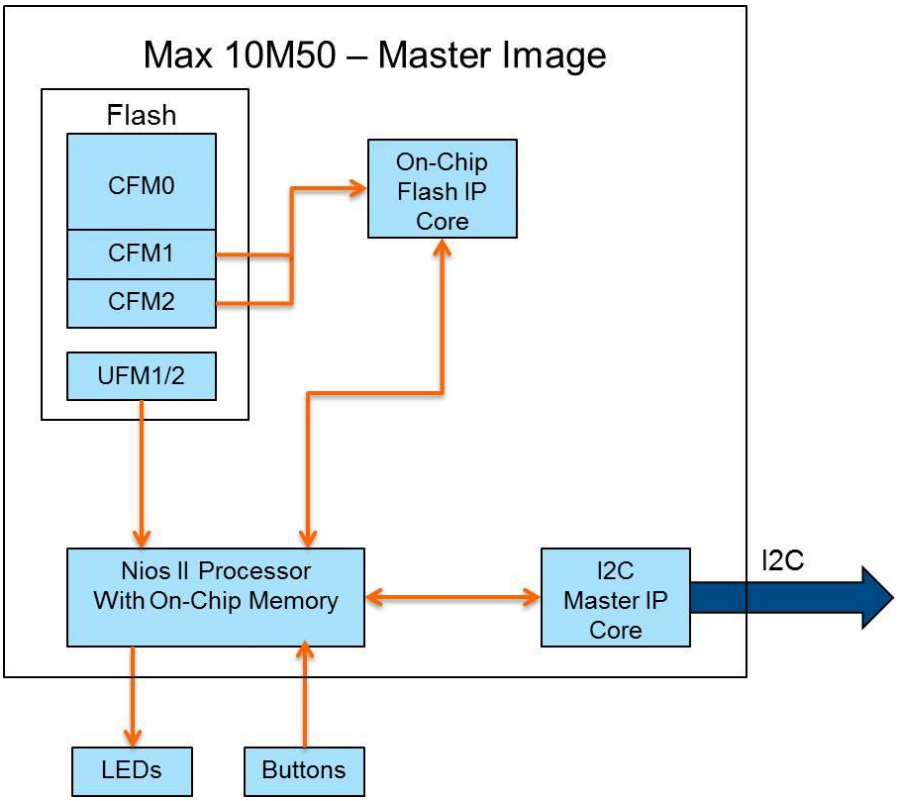


Figure 3

Stage 2 – Program Slave Image 1 into Max 10M08

The second stage is to initialize the slave device with the default image. The default image (Slave Image 1) will be programmed into both CFM0 and CFM1/2. The Max 10M08 will be set to boot from CFM1/2 and this is what will be modified in the remote system upgrade. CFM0 will be used purely as a backup in case of an incorrect I2C transmission or other corrupting error.

Figure 4 below illustrates the contents of the on-chip flash after this stage. Figure 5 contains a block diagram of both the default image (Slave Image 1) and the remote upgrade image (Slave Image 2). These images are identical except for the LED Driver- in Slave Image 1 the LEDs will blink in a binary counting pattern, and in Slave Image 2 a single lit LED will traverse back and forth across the row of five. The two images therefore have the same block diagram. In addition to the LED driver, the slave image contains a custom FLASH_LOADER IP component which handles the I2C communication, the interface with on-chip flash, and the remote update.

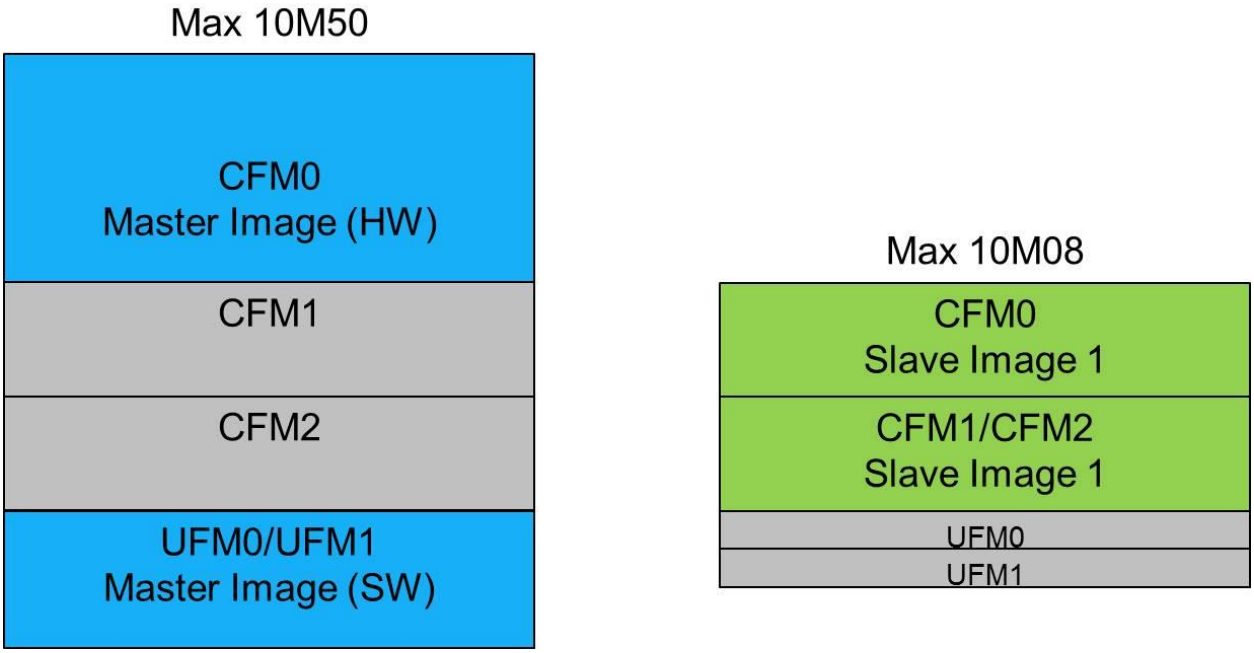


Figure 4

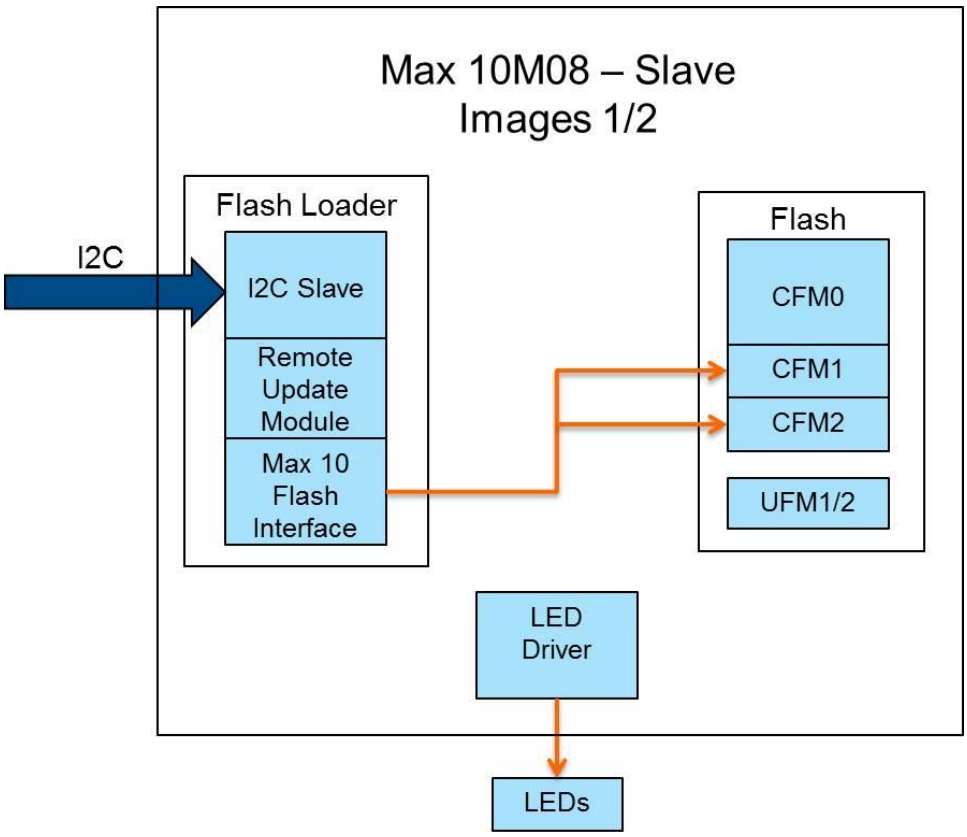


Figure 5

Stage 3 – Program Slave Images 1 and 2 into CFM 1 and 2 on Max 10M50

As seen back in Figure 2, the Nios II in the master image pulls bytes out of CFM1 and/or CFM2 to send over I2C. Slave Image 1 will be stored in CFM1 on the 10M50, and Slave Image 2 will be stored in CFM2 on the 10M50, as seen below in Figure 6.

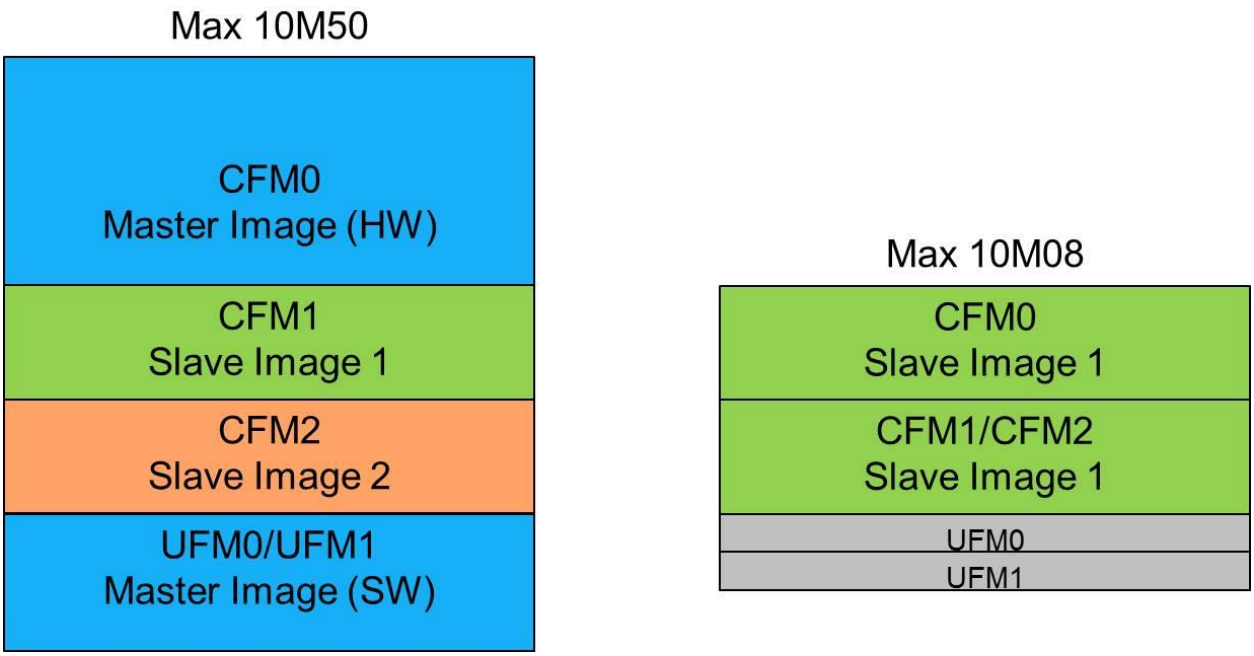


Figure 6

The process for doing this is slightly complicated, as it cannot be done with the built in Quartus tools. The Quartus programmer only allows you to program flash with images appropriate for the device- it does not allow for flash on a 10M50 to be programmed with an image for a 10M08. To get around this problem, the programming was done via UART. A modified version of the design created in *AN741: Remote System Upgrade for Max 10 FPGA Devices over UART with the Nios II Processor* was used which allows for a Max 10M08 programming to be sent over UART and stored in CFM1 or CFM2 on a Max 10M50. This design also makes use of the QSPI flash on the Max 10 Development Kit for storing Nios II firmware. A block diagram of this design is below in Figure 7.

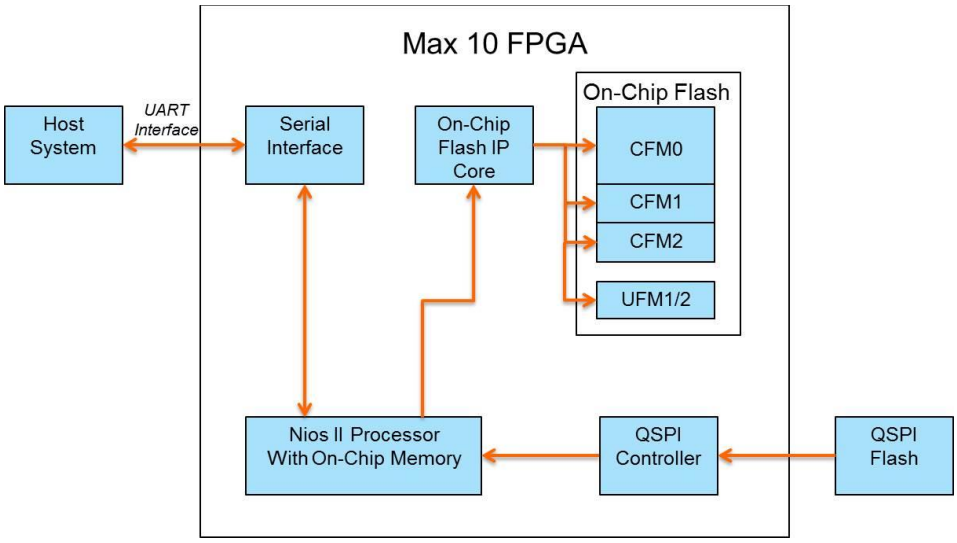


Figure 7

Stage 4 – Trigger I2C Transmission

The final stage is to trigger the I2C transmission by pressing a button on the Max 10M50. This will trigger the Nios II to begin pulling bytes from CFM2 and sending them over I2C. The Max 10M08 will receive these bytes, process them, and store them it its own CFM1/2. Once the entire image has been received, the Max 10M08 will reset and reconfigure itself from this new image. This process is illustrated below in Figure 8. Figure 9 illustrates the contents of on-chip flash after this process is complete.

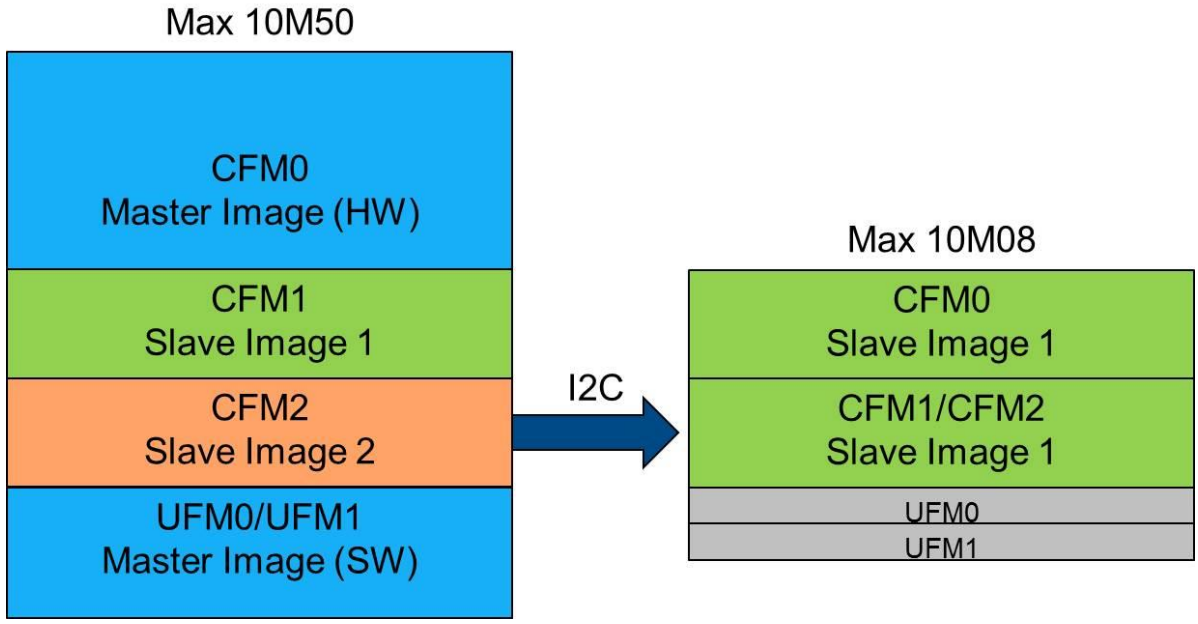


Figure 8

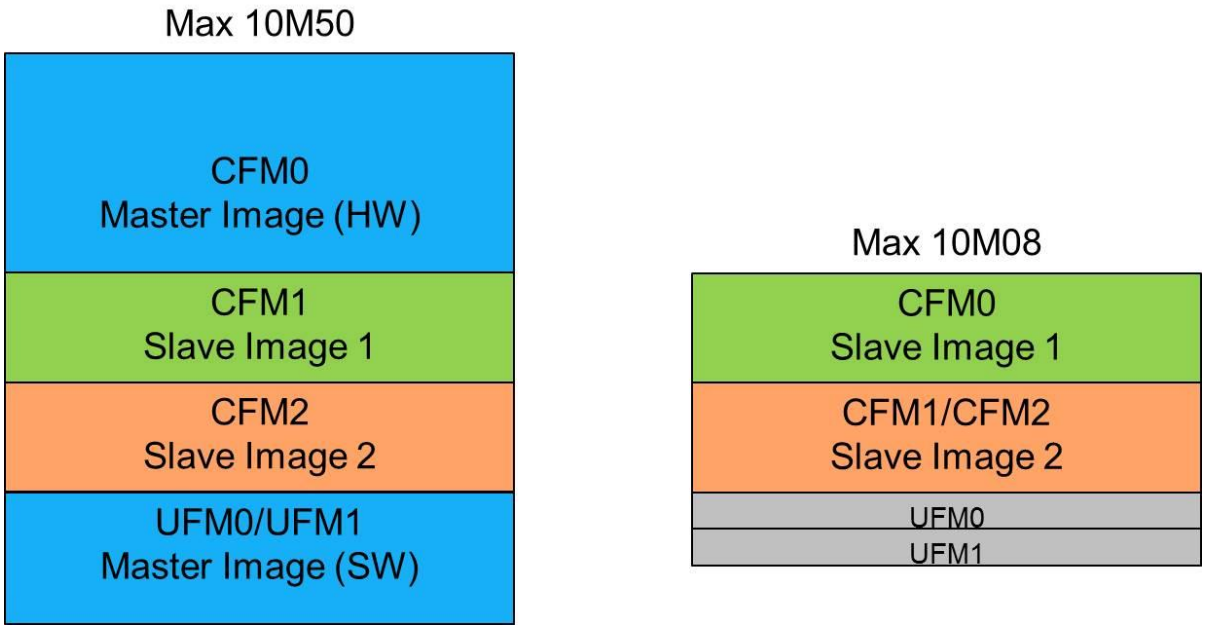


Figure 9

A different button can then be pressed to send Slave Image 1 over I2C and effectively “reset” the 10M08 back to the default image. Either image can be sent as many times as desired. All images are stored in non-volatile memory, so after reaching this stage the kits can be disconnected from the computer or power cycled and they will still function.

Executing Example Design

All programming files needed for this design have been included in the folder “final_programming_files”. This section will assume that you are using these pre-made files. The “Generating/Modifying Programming Files” section of this guide will cover how to recompile or regenerate these programming files.

Hardware Setup

Before you can execute this example design, you first have to connect the two development kits. I2C is a two wire interface, so connecting the kits requires a very simple circuit, as seen in Figure 10.

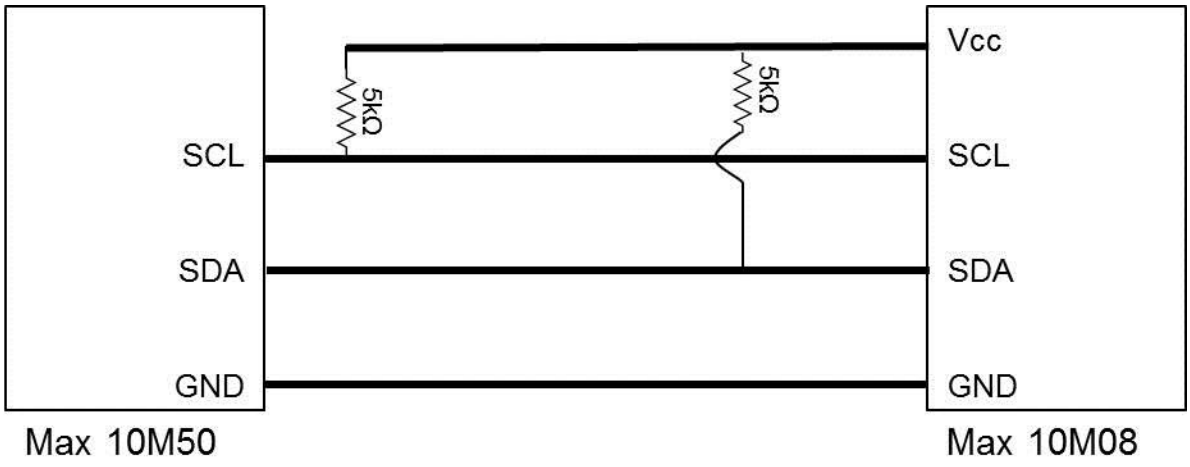


Figure 10

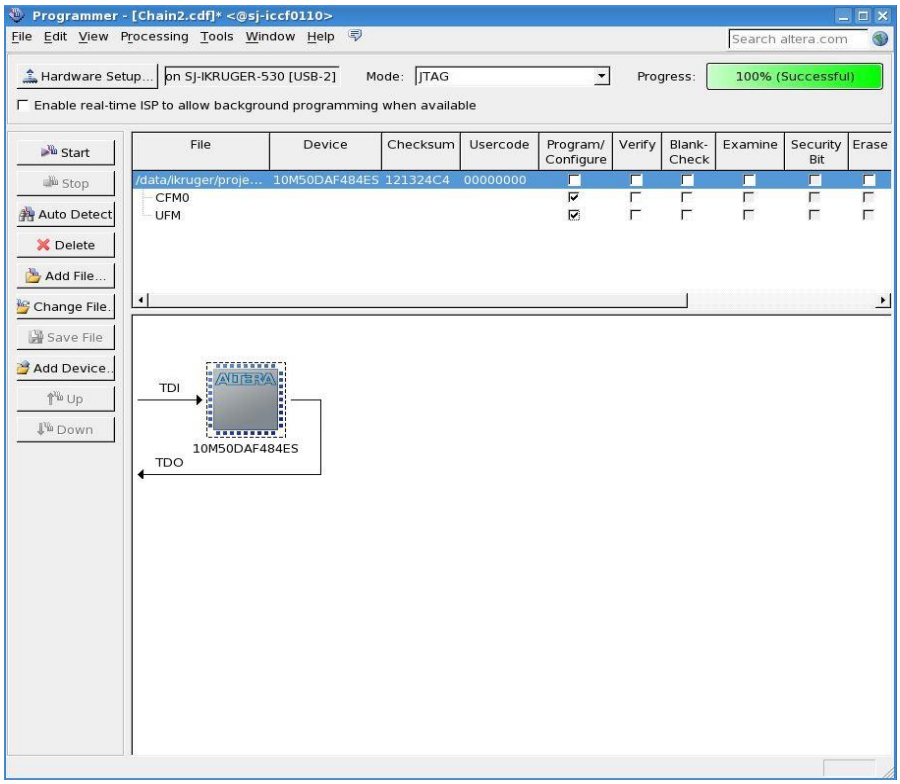
The chart below shows the pins used on the two kits for SCL, SDA, GND, and Vcc. On the Max 10M50 Development Kit, the PMOD A GPIO is used (marked as J4 on the kit). The Max 10M08 Evaluation Kit does not have PMOD GPIO, so pins from Arduino header section J3 are used instead. Note that both the PMOD and the Arduino header begin with Pin 1 (ie there is no Pin 0). Also note that due to differing voltage standards Vcc must be connected to the Max 10M08 and not the Max 10M50. Finally, note that ground on both kits must be tied together so that the two devices share a common reference voltage.

	Max 10M50 Development Kit	**Max 10M08 Evaluation Kit**
SCL	PMODA.1	J3.4
SDA	PMODA.2	J3.5
Vcc	XXX	J3.3
GND	PMODA.5	J3.2

Programming Master Image into Max 10M50

Follow the steps on the Design Store web page to extract and install the i2c_rsu platform file. For the purposes of this section, it is not necessary to open any projects or compile anything. For information on that process, consult the "Generating/Modifying Programming Files" section of this guide. The following steps describe how to program the Max 10M50 with the I2C master image.

1. Launch the Quartus II Programmer from the tools menu by clicking Tools → Programmer.
2. Plug in the Max 10M50 Development Kit and make sure the appropriate hardware (usually "USB-Blaster II") is displayed next to the "Hardware Setup…" button.
3. Once the hardware is selected, click "Auto-Detect". If a dialog box appears asking for which device is on the JTAG chain, select 10M50DAES.
4. Double click on the File list next to the device you just selected where it says "<none>". Navigate to the "final_programming_files" subfolder and select "master_image.pof".
5. Under the "Program/Configure" heading there should be three checkboxes. Check the boxes that correspond to CFM0 and UFM. CFM0 will be programmed with the hardware portion of the design, and UFM will be programmed with the software.
6. Click "Start" and wait for the device to finish programming. All LEDs should turn off.

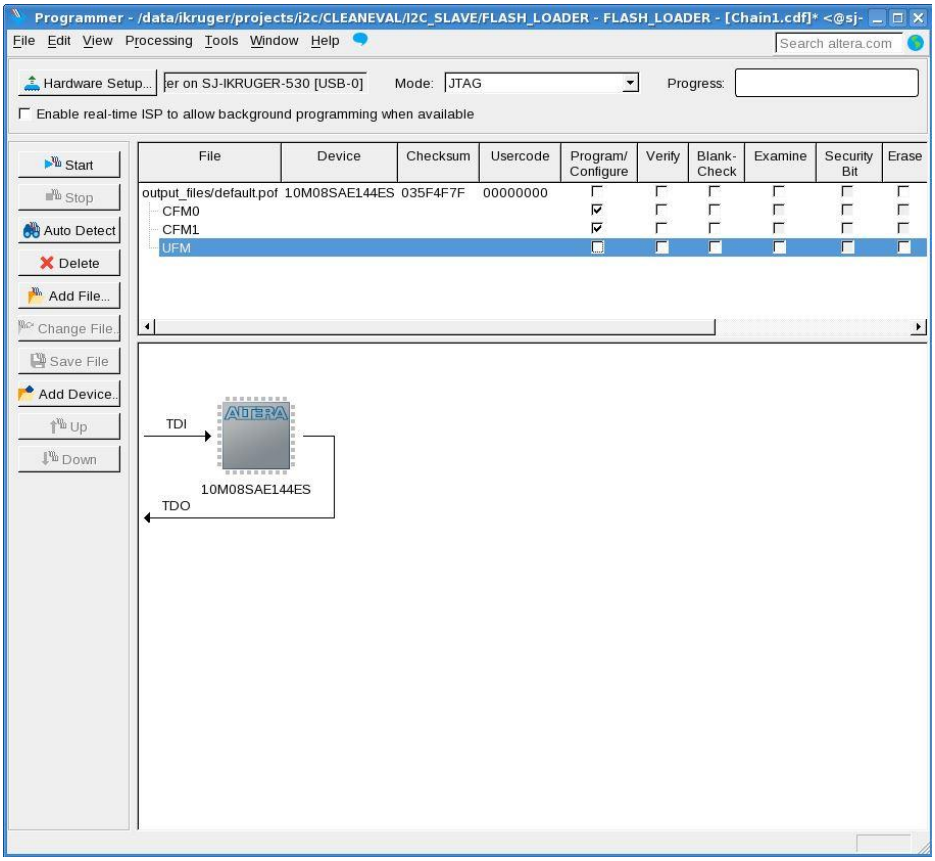


7. Keep the programmer window open.

Programming Slave Image 1 into Max 10M08

The following steps describe how to program the Max 10M08 with the I2C Slave Image 1 or "default image".

1. Plug in the 10M08 Evaluation Kit. You do not need to unplug the 10M50 Development Kit in order to do this, both kits can be plugged in at the same time if you have enough USB ports on your host system.
2. Click “Hardware Setup”. Ensure that you have the correct device selected from the dropdown menu. If you have two devices plugged in, make sure you select the 10M08. Typically, whichever device you plugged in first will be tagged with “[USB-0]” and whichever one you plugged in second will be tagged with “[USB-1]”. Choose the correct one and click “Close”.
3. Click “Auto Detect”. From the list of devices, choose “10M08SAES” (if the list of devices all begin with “10M50”, you probably selected the wrong device in step 2). If a window pops up informing you that the auto-detected device chain does not match, click “Yes”.
4. Double click on the File list next to the device you just selected where it says “<none>”. Navigate to the “final_programming_files” subfolder and select “default.pof”.
5. Under the “Programming/Configure” heading, check the boxes that correspond to CFM0 and CFM1.



6. Click on Start and wait for the device to finish programming. You should now see the LEDs blinking in a binary counter pattern.
7. Confirm that the default image is located in both configuration regions by changing switch 6 of SW3 on the kit, and then turning the board off and on. The same LED blinking pattern should be displayed. Before continuing, change the same switch to the “ON” position and restart the board to make sure the image being booted is from CFM1/2.

8. Keep the programmer window open.

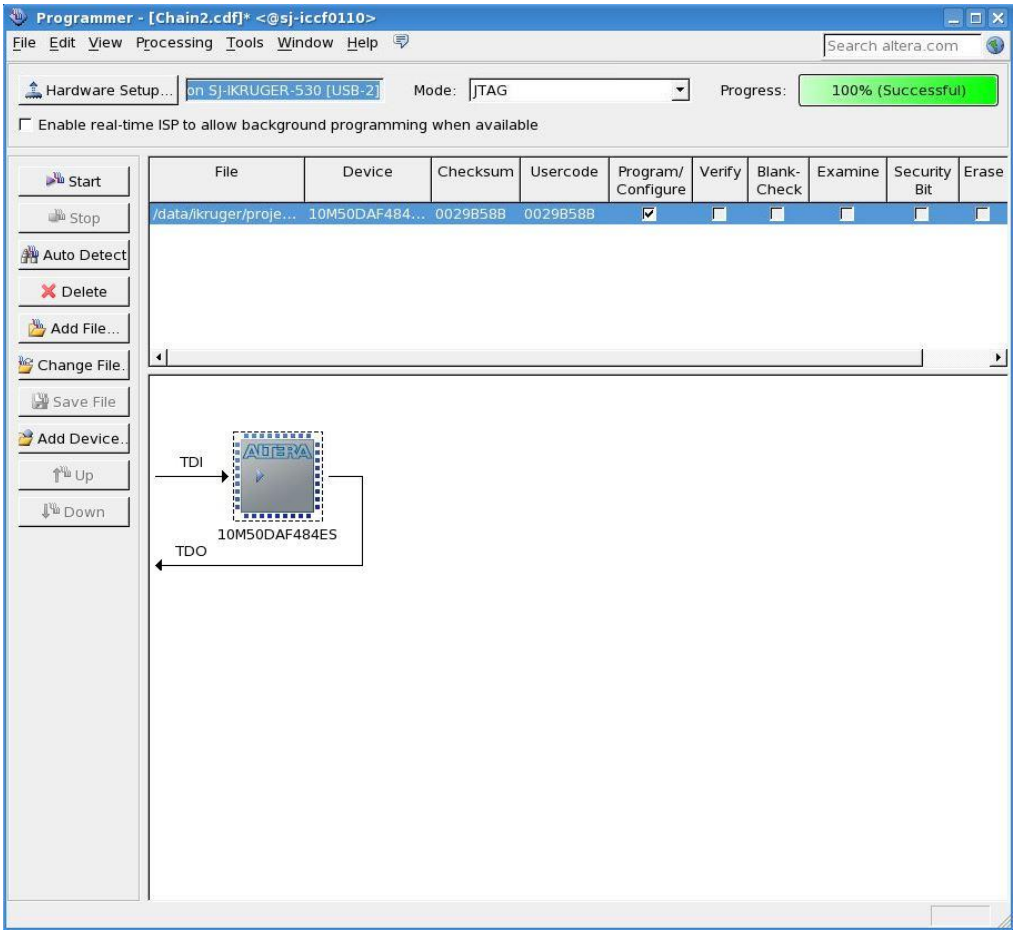
Programming Slave Images 1 and 2 into CFM 1 and 2 on Max 10M50

Programming the 10M08 images into the 10M50 on-chip flash is the most complicated portion of this example. It is divided into three sub-sections:

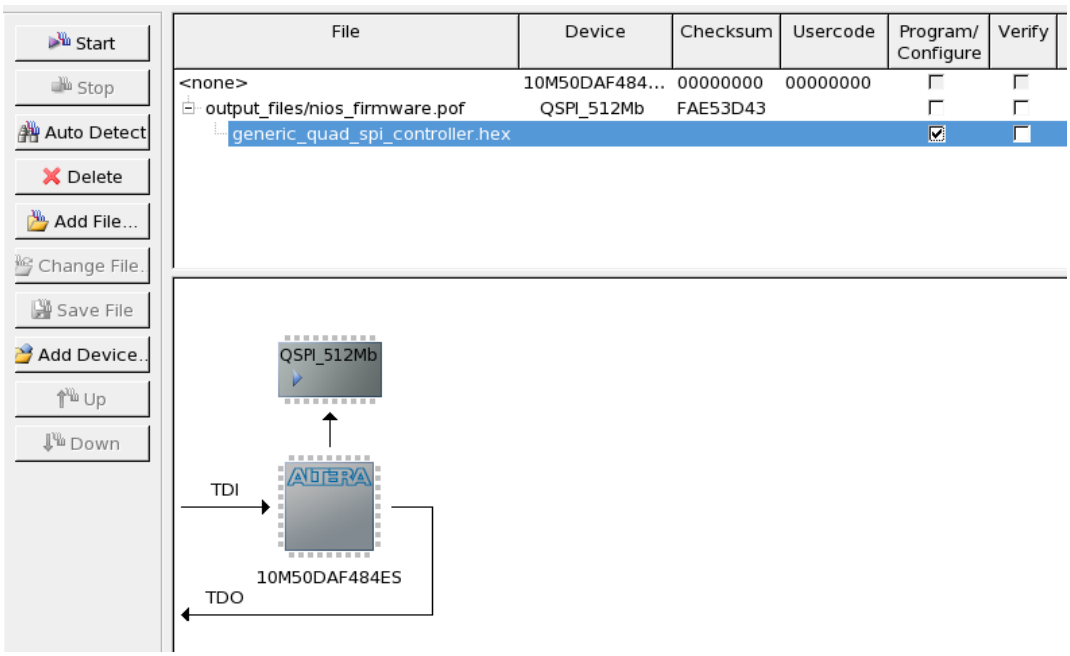
Programming Nios II Firmware into QSPI Flash

The 10M08 images will be programmed into the 10M50 on-chip flash via UART. The UART communication is controlled by a Nios II processor. In order to not interfere with the contents of on-chip flash, the firmware for the Nios II will be stored in external QSPI flash. The steps to program the QSPI flash are below.

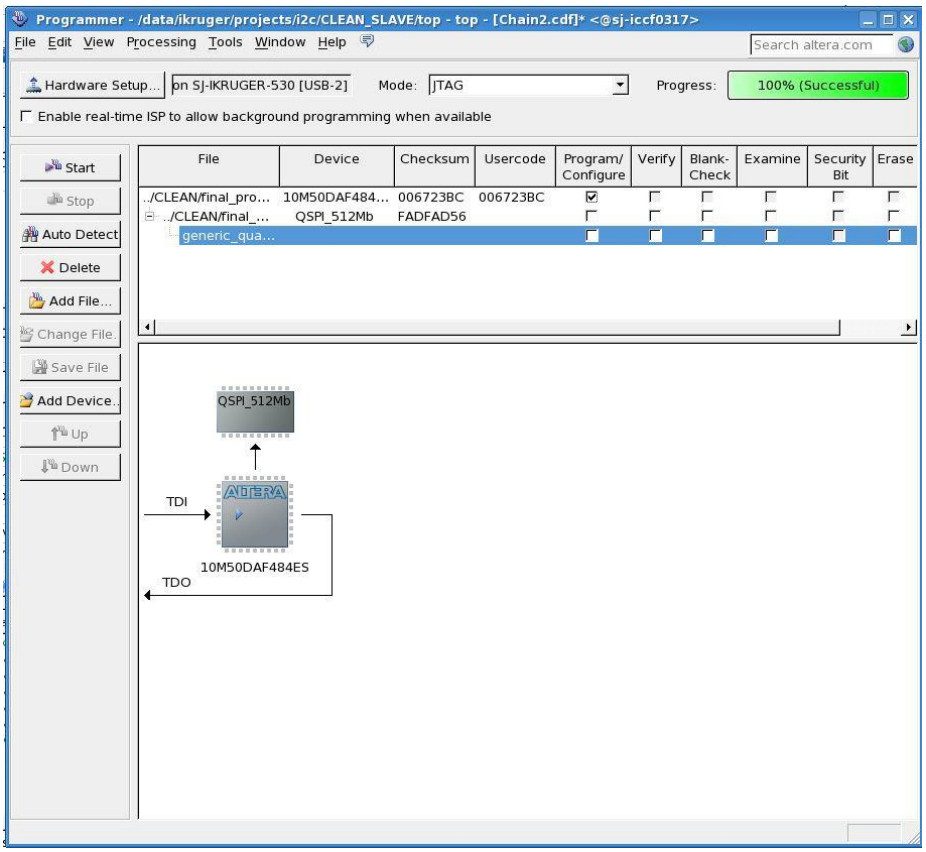
1. If the Programmer isn't already open, open it by clicking Tools->Programmer from Quartus.
2. Ensure that the 10M50 is plugged in and selected in the "Hardware Setup" window.
3. Click "Auto Detect" and select the 10M50DAES. If a window pops up informing you that the auto-detected device chain does not match, click "Yes".
4. Double click under "File" where it says "<none>". Navigate to the "final_programming_files" folder and select "pfl.sof". This programming file was created from a Qsys system with nothing but a Parallel Flash Loader IP instantiated inside of it and a Quartus project that connected the Qsys system up to the QSPI flash pins on the board.
5. Check the "Program/Configure" checkbox for the programming file, then click "Start".



6. Once the programming is done, click on the Auto Detect button. If a window pops up informing you that the auto-detected device chain does not match, click “Yes”.
7. Double click on the “<none>” in the file list next to the QSPI_512Mb device (it should be the second “<none>”) and select the “nios_firmware.pof” file from inside of the “final_programming_files” directory. Click Open.
8. The programmer analyzed the *pof* file and found the *hex* file containing the Nios II firmware. To program only the firmware and not the entirety of the flash, select the “Program/Configure” checkbox next to the “generic_quad_spi_controller.hex” file only as show below. Click “Start” and wait for the flash to program (this will take up to a few minutes).



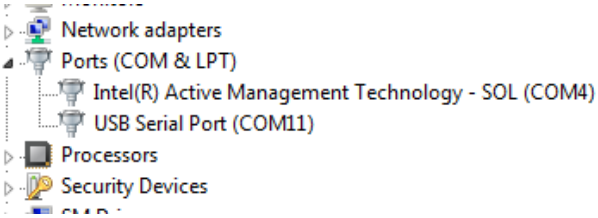
9. Now you need to program in the hardware image that will handle UART communication. Double click on the “<none>” in the file list next to the 10M50 (should be the first in the list). Select the file “RSU.sof” from inside the “final_programming_files” directory. Click open.
10. Check the “Program/Configure” checkbox next to the file you just added (in the same row as the 10M50 device). Uncheck the “Program/Configure” checkbox next to the “generic_quad_spi_controller.hex” file from step 8. Click start and wait for the programming to complete.



Setup the USB-To-UART System

In this section of the design we are sending the two slave images to the Max 10M50 over a serial UART connection. The Max 10 Development Kit has an onboard USB-To-UART converter. Follow the steps below to connect the converter to the board and to determine which COM port it represents in Windows.

1. Connect a mini-USB cable to the UART1 connector on the dev kit board and your PC.
2. Open up Device Manager by opening up the start menu, searching for “Device Manager”, and clicking on the first result.
3. Under “Ports (COM & LPT)” there should be a USB Serial Port (or similar) device with a COM number next to it. For instance, the serial port below is utilizing COM11:



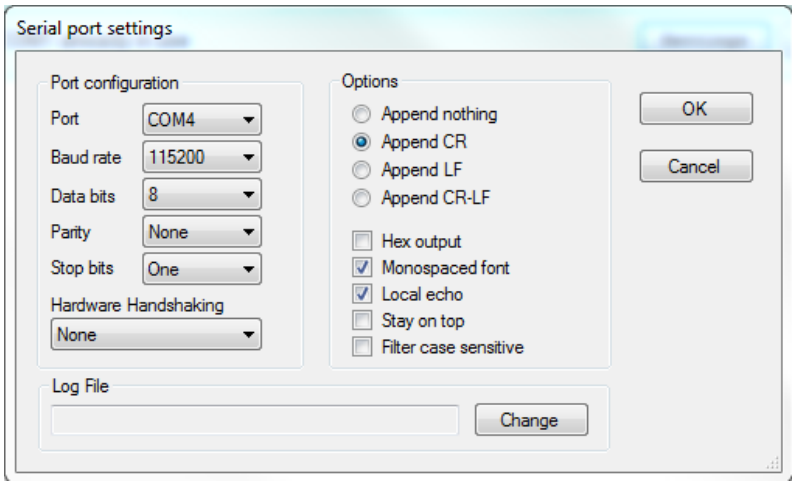
4. Remember this COM port number for later.

Send Slave Image 1 and Slave Image 2 Via UART

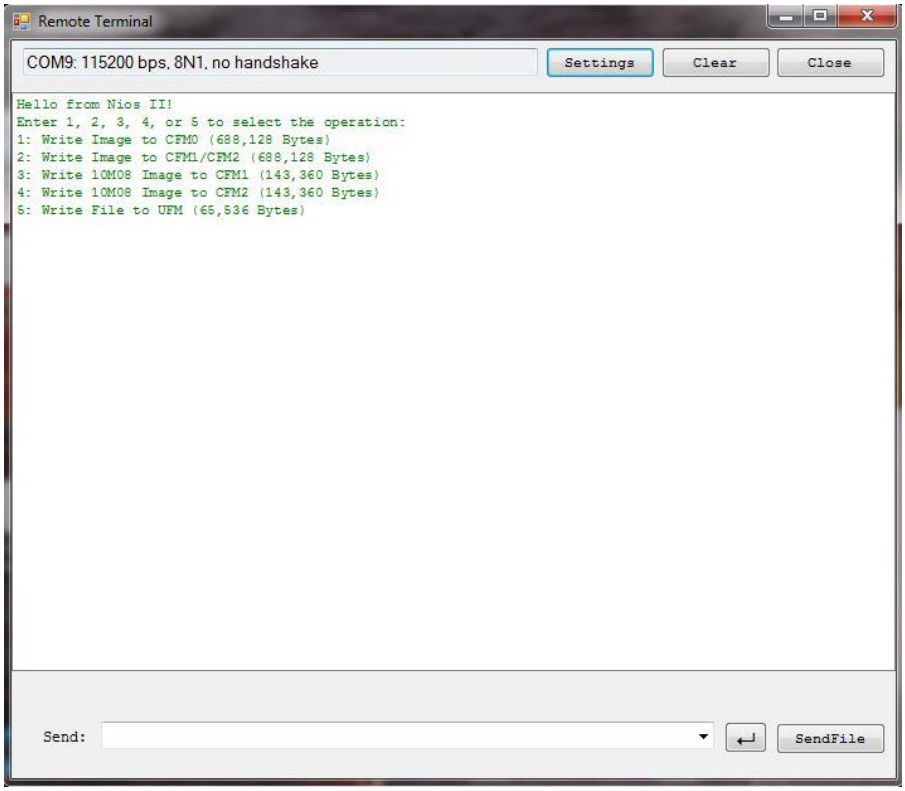
1. Open up the *RemoteTerminal.exe* software located in the “TerminalSoftware” folder. This program will be used to interact with the Nios II firmware and to send the application image over UART. In

reality, any terminal program that supports sending files should work fine. This terminal software is derived from the Termie open source project. **Note: if you are getting errors when trying to run this software from a network drive, first copy and paste the RemoteTerminal.exe file to your local drive before running.**

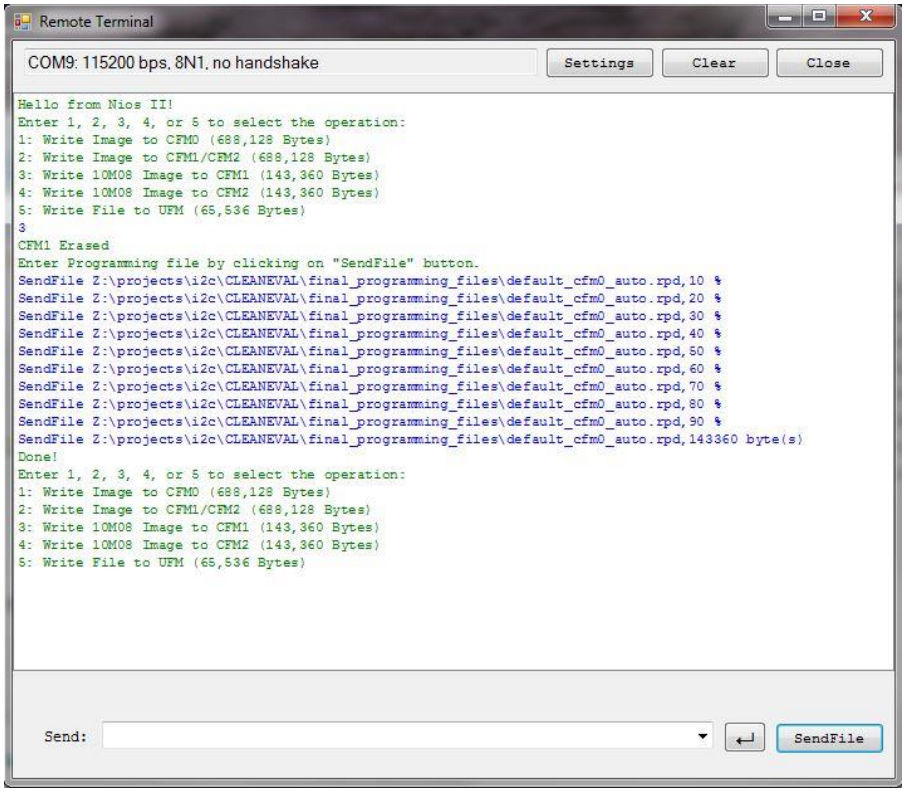
2. Click on the Settings button and change the "Port" to the COM port you noted down in the previous section. Leave the rest of the settings at their defaults. Your settings should like the following (besides the COM port):



3. Press the USER_PB3 button (second button from the top) to reset the Nios II processor. Inside the terminal window you should now be presented with a menu.



4. Enter “3” in the Send textbox and press enter. This will start the “Write File to CFM1” operation.
5. When prompted, click on the “SendFile” button and browse for “default_cfm0_auto.rpd” in the “final_programming_files” directory. This file contains the default binary-counting-LED image for the Max 10M08.
6. The transmission process will now begin. This process will take a few minutes, and in this time the terminal will become unresponsive. The terminal will update after every 10% of the file that it sends.
7. Once the terminal has completed sending the file, the menu will reappear. This time, enter “4” to start the “Write File to CFM2” operation.



8. When prompted, click on the “SendFile” button and browse for “knightrider_cfm0_auto.rpd” in the “final_programming_files” directory. This file contains the new 10M08 image (Slave Image 2), which is identical to Slave Image 1 except for the pattern of blinking LEDs.
9. After this file finishes sending, close the terminal.

Perform the Remote System Upgrade over I2C

All of the above steps need only be performed once. The remote upgrade can be performed and reset multiple times, as all images are stored in non-volatile memory.

1. If you have just finished the UART section of this example, make sure you power cycle the 10M50 development kit. This will cause the kit to reboot from CFM0, which should contain the I2C Master image.

2. Connect the 10M50 and 10M08 kits via I2C, using the circuit described earlier in this document, and ensure that both kits are turned on.
3. Press button PB2 (third from the top) on the 10M50. This will trigger the transmission of Slave Image 2 over I2C. The 10M08 will automatically receive and store the image in its own CFM1/2. Upon completion of the transmission, the 10M08 will reboot and reconfigure itself from Slave Image 2. The entire process should take about 10 seconds.
4. Upon completion, the LEDs on the 10M08 should begin flashing in a new pattern, with a single lit LED traversing back and forth across the row of 5.
5. To "reset" the 10M08, press button PB3 (second from the top) on the 10M50. This will trigger the transmission of Slave Image 1 over I2C. The process will be the same, and after about 10 seconds the 10M08 should return to the original binary blinking LED pattern.
6. Steps 3-5 can be repeated as many times as you like.

Generating/Modifying the Programming Files

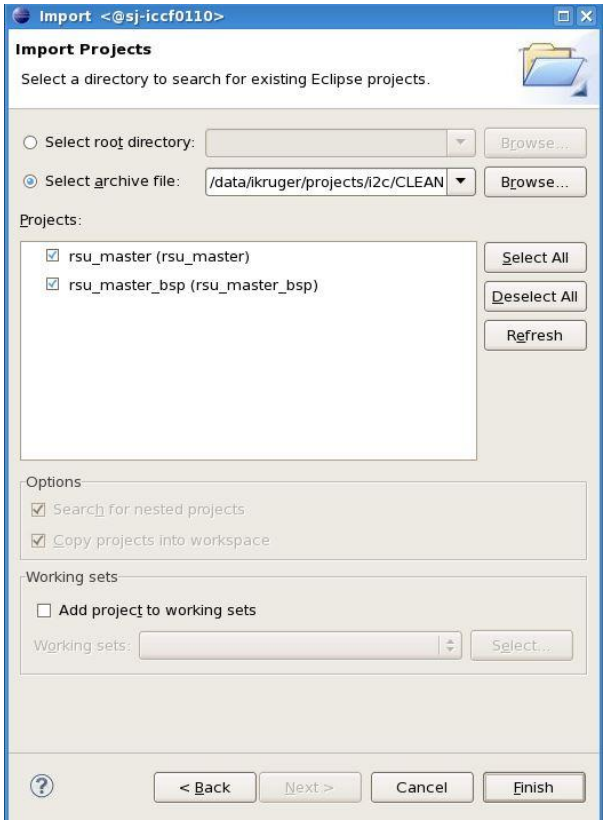
This section will guide you through the process of generating most of the necessary programming files for this design example, in case you ever want to make modifications or just better understand the flow. This section assumes that you have already followed the instructions on the Design Store website to extract and install the i2c_rsu platform.

Note that this section does not contain instructions for regenerating the programming files used for the UART section of this example ("pfl.sof", "RSU.sof", and "nios_firmware.pof"). These files are part of a separate project which can be found separately on the Altera Design Store.

Generating "master_image.pof"

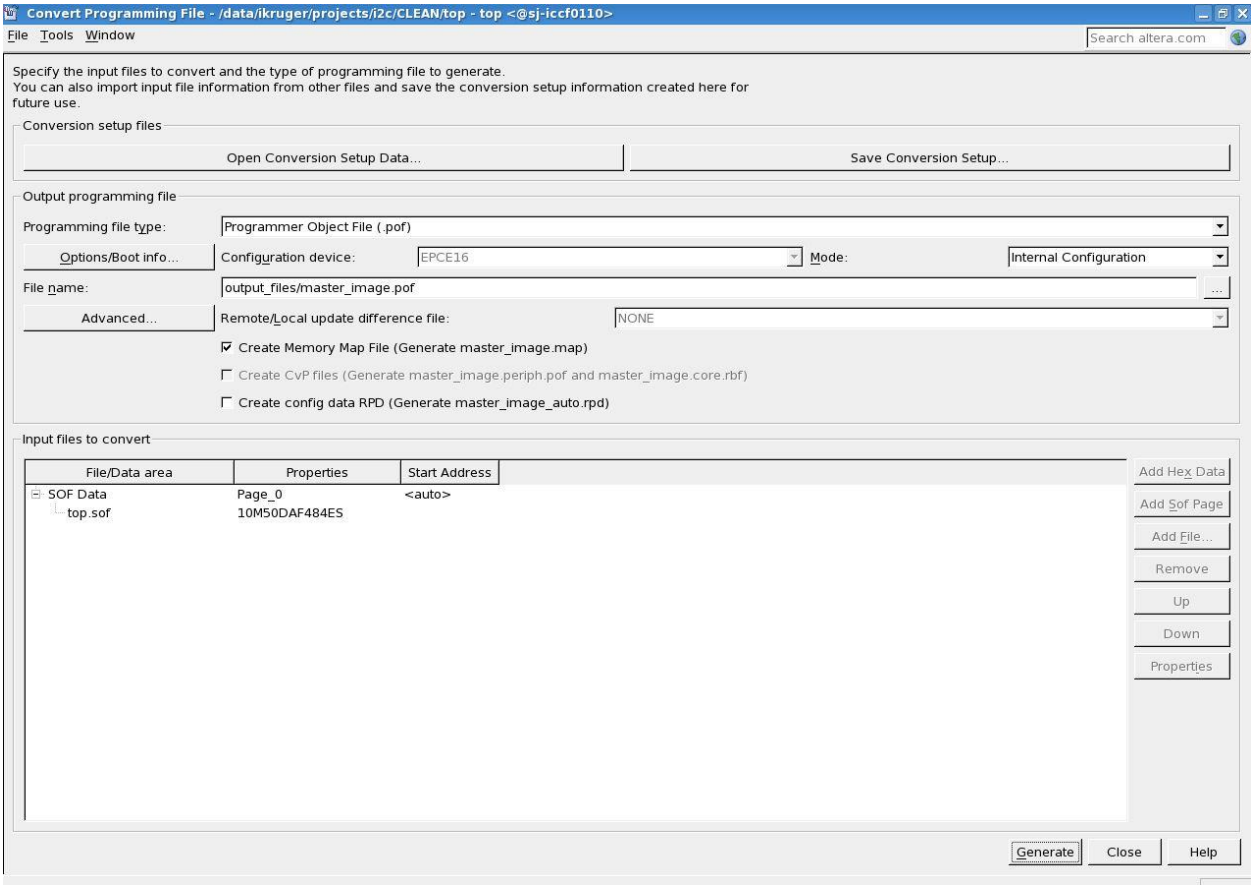
Creating "master_image.pof" requires you to compile the hardware design, generate a hex file for the Nios II software, then combine them into a single pof file that will be programmed into CFM0 and UFM. The steps are below.

1. Open the top level project top.qpf
2. Compile the design by clicking Processing → Start Compilation
3. Open the Nios II IDE by clicking Tools → Nios II Software Build Tools for Eclipse
4. When prompted for a Workspace, Browse to the subfolder "i2c_master_software" within the main project folder
5. Click File → Import. Under the "General" tab, select "Existing Projects into Workspace" and click Next. Click the "Select archive file" radio button, then Browse to the subfolder "i2c_master_software" and choose the file "rsu_master_software.zip". Make sure both "rsu_master" and "rsu_master_bsp" are checked and click Finish.



6. Click Project → Build All
7. In order to combine the software with the hardware into a single .pof file, a .hex file must be generated for the on-chip UFM. To do this, right-click on the project rsu_master in the Project Explorer at the left side of the window, then click Make Targets → Build. Select "mem_init_generate" and click Build.

8. The next step is to combine the .sof file generated during compilation in Step 2 with the .hex file generated in step 7 into a single .pof file that can be programmed into on-chip flash. Switch back to the main Quartus II window and click File → Convert Programming Files.
9. Ensure that the "Programming file type" is set to Programmer Object File (.pof) and the "Configuration device" is set to EPCE16. Set the "Mode" to Internal Configuration
10. Click Options/Boot info. Under "User Flash Memory", set "UFM source" to "Load memory file". Next to "File path" click the "…" button and from the main project folder browse to "i2c_master_software/rsu_master/mem_init/" and select the file "onchip_flash_0.hex". This is the file you created in Step 7. Click OK.
11. Set the "File name" to output_files/master_image.pof
12. In the "File/Data area" highlight the "SOF Data" and click Add File at the right. From the main project folder, browse to the "output_files" subfolder and select "top.sof". Your settings should look like this:

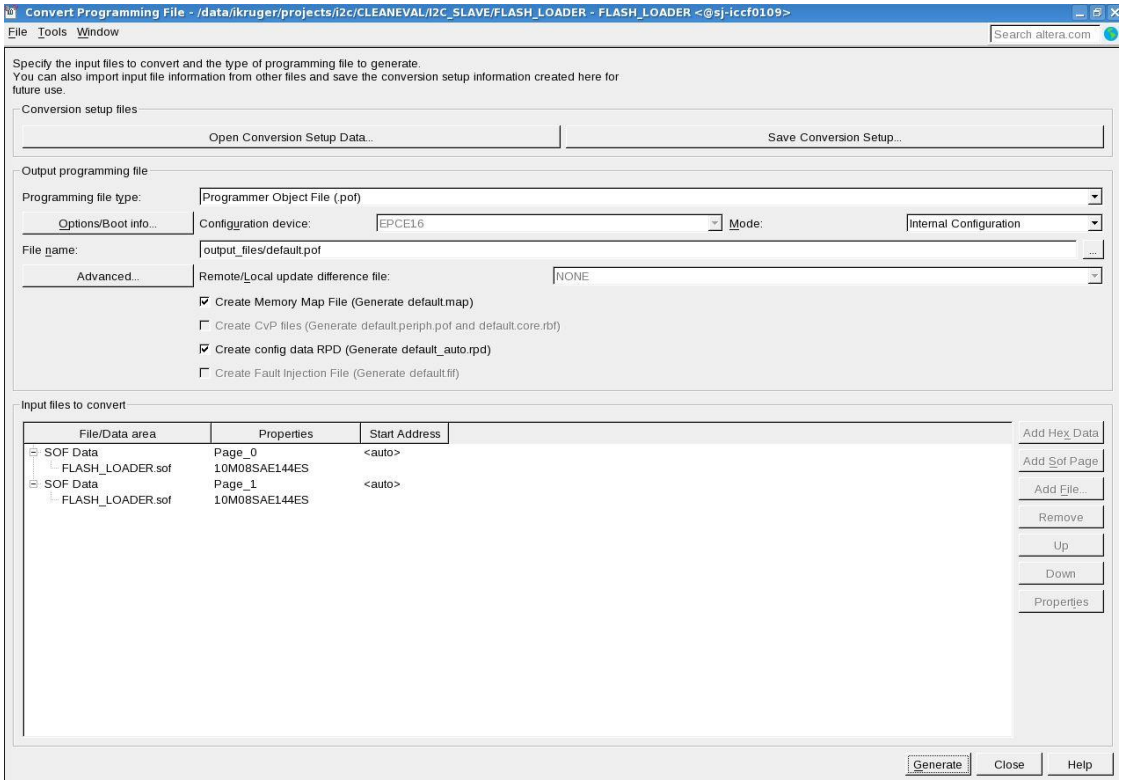


13. Click Generate. The created master image "master_image.pof" will be located in the "output_files" subfolder.

Generating Slave Images 1 and 2

This steps below will walk you through creating both images for the 10M08 slave device (“default.pof” and “knightrider.pof”) as well as the raw programming data (.rpd) files which will be used for the remote update. Firstly, to create “default.pof”:

1. From the main project folder, browse to the “I2C_SLAVE” subfolder. This folder contains the I2C_SLAVE project. Open “FLASH_LOADER.qpf”.
2. Compile the design by clicking Processing → Start Compilation. This will create a .sof file “FLASH_LOADER.sof” in the “output_files” subfolder of the I2C_SLAVE folder.
3. Click on File → Convert Programming Files
4. Ensure that the “Programming file type” is set to Programmer Object File (.pof) and the “Configuration device” is set to EPCE16. Set the “Mode” to Internal Configuration
5. Set the “File name” to output_files/default.pof
6. Check the box for “Create config data RPD”
7. In the “File/Data area” highlight the “SOF Data” and click Add File at the right. From the I2C_SLAVE folder, browse to the “output_files” subfolder and select “FLASH_LOADER.sof”.
8. Click Add Sof Page, highlight the new “SOF Data” that appears (with Page_1 in the Properties field), and click Add File at the right. As in step 8, select “FLASH_LOADER.sof” from the “output_files” subfolder. This is necessary because “default.pof” will be used to program both CFM0 and CFM1/2, so two pages need to be generated.
9. Make sure your settings look like those below, then click Generate at the bottom.



10. You have now created “default.pof”, which you can use to program CFM0 and CFM1/2 on the Max 10M08. You have also created two .rpd files: “default_cfm0_auto.rpd” and “default_cfm1_auto.rpd”. These files contain the programming data for CFM0 and CFM1/2 respectively, in a raw hex format. Since “default.pof” contains the same image in both CFM0 and CFM1/2, the two .rpd files are identical.
11. You can now use either of these .rpd files in the “Programming Slave Images 1 and 2 into CFM 1 and 2 on Max 10M50” section of this guide.

You will follow much the same process to create the programming files for Slave Image 2. The source files for Slave Image 2 are almost exactly the same as the source files for Slave Image 1, with the exception of the LED driver. Therefore there is not a new project file for Slave Image 2. The steps below will walk you through creating “knightrider.pof” and “knightrider_cfm0_auto.rpd”:

1. Return to the main Quartus II window, and open the project “FLASH_LOADER.qpf” from the I2C_SLAVE subfolder if you have not already done so.
2. Click Project → Add/Remove Files in Project. In the list of files, highlight “binary_leds.v” and click Remove at the right.
3. Next to “File name”, click the “…” button and browse to the main I2C_SLAVE folder. Select “knight_rider.v” and click Open. Click Add at the right, then click OK at the bottom. This will replace the binary-counting LED drive with the new LED driver.
4. Compile the project by clicking Processing → Start Compilation.
5. Repeat Steps 4-10 of the process for creating “default.pof” and “default_cfm0_auto.rpd”, but in step 5 set the output file name instead to “knightrider.pof”.
6. As before, two identical .rpd files will be created by this process (“knightrider_cfm0_auto.rpd” and “knightrider_cfm1_auto.rpd”). As before, these files will be identical and either can be used in the “Programming Slave Images 1 and 2 into CFM 1 and 2 on Max 10M50” section of this guide.