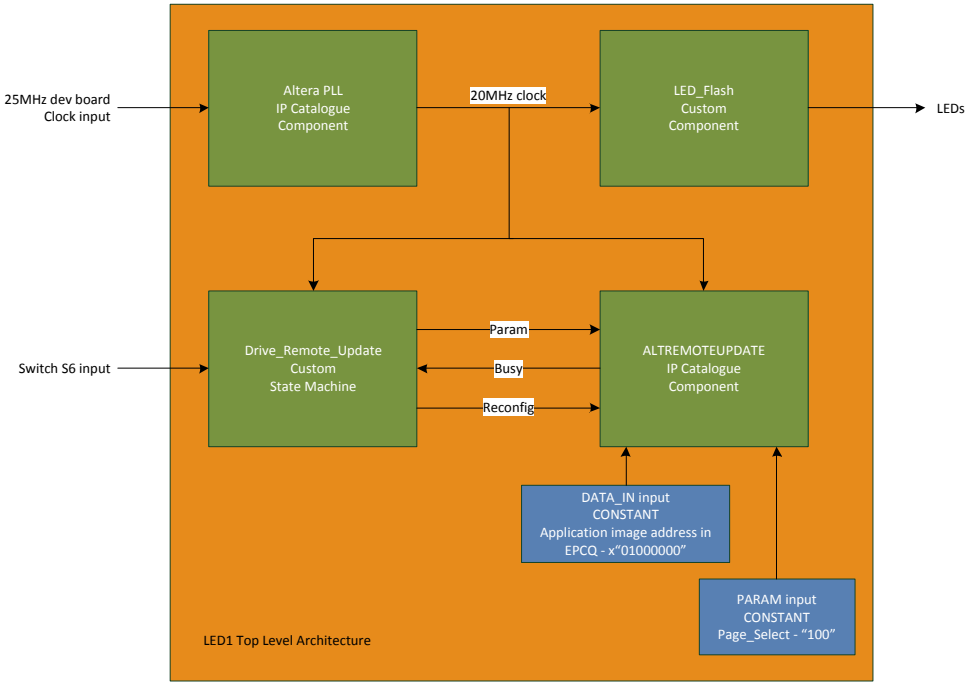
Simple Remote Update Example Design

Overview

The following design demonstrates how to use the ALTREMOTEUPDATE IP to trigger a reconfiguration of the FPGA from a pre-programmed serial flash device. It does not include any mechanism to update the flash device remotely (that is the subject of a future example, but you could use this as the foundation). Most of this is basic stuff, using the ALTREMOTEUPDATE user guide and application note AN603 as starting points.

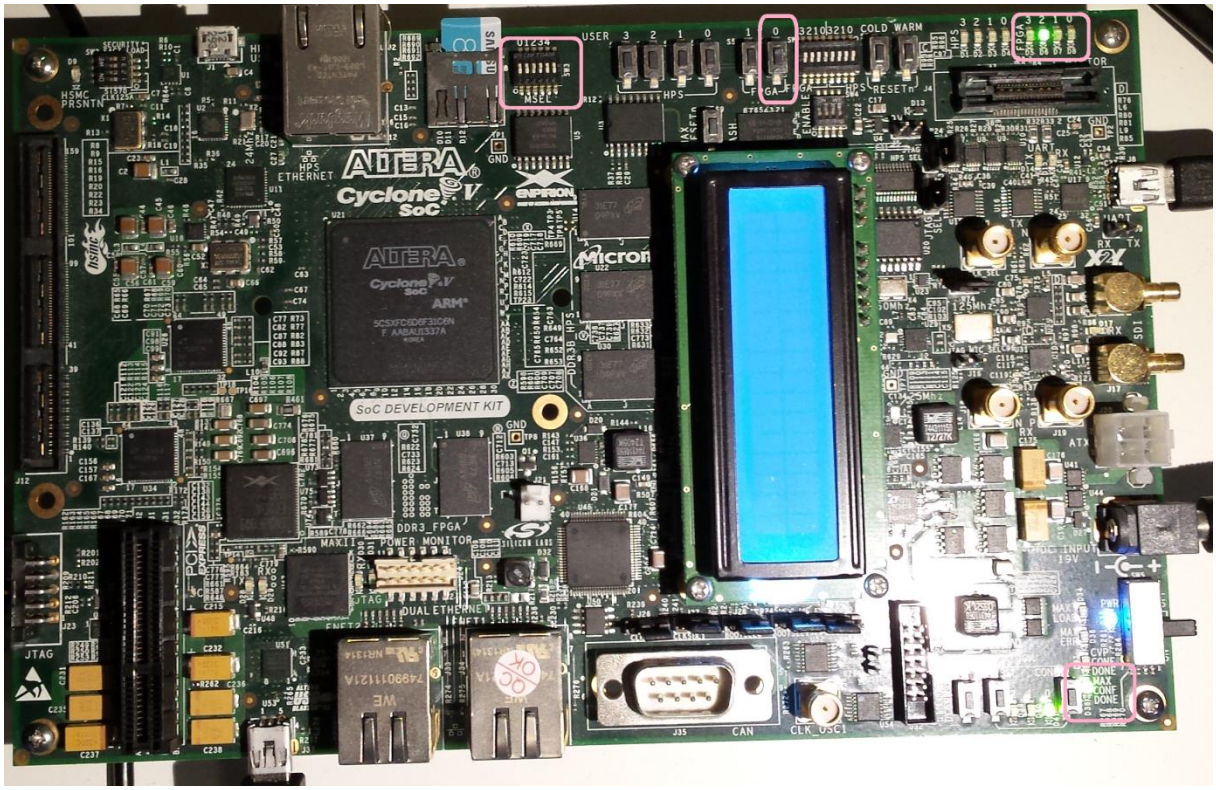
The design and description has been targeted at the Cyclone V SoC Terasic development board fitted with EPCQ256 flash, but it could be very easily retargeted to a different board. Only the basic Cyclone V features have been used and none of the SoC portion has been used.

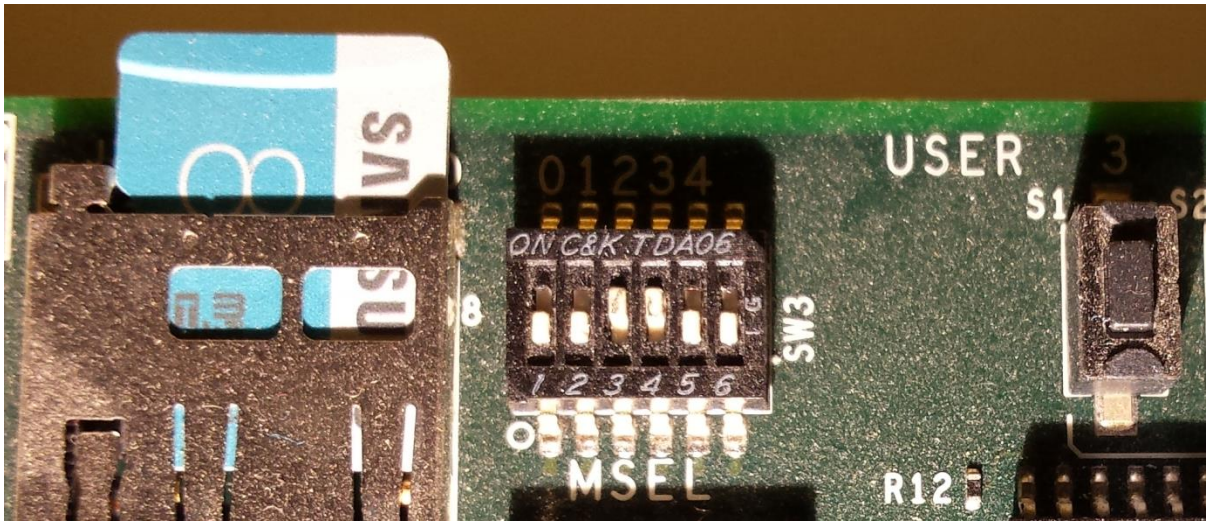
The following diagram gives a simple overview of the design.



The two photos below show the development board used with the key features highlighted for this design:

- MSEL
- Switch S6 to trigger the reconfiguration process
- FPGA LEDs [3:0]
- Configuration Done LED





Design Description

This example uses two top-level designs:

- LED1 – the main “factory default” design, as detailed in the diagram in the overview. This image has the reconfiguration IP that on a press of switch S6 reconfigures the FPGA with application LED2. This design also flashes LEDS [1:0]
- LED2 – this is the “application” design that simply flashes the other two LEDs [3:2]. This does not include any reconfiguration IP, although there is no reason why you could not add some in.

As LED2 does not really do anything other than indicate a new image has been loaded, the remainder of the description in this section is for LED1.

The main top-level for LED1 is LED1.vhd and it includes all of the component instantiations and a simple state machine (Drive_Remote_Update) that drives the ALTREMOTEUPDATE IP.

Project settings

There are no specific or special project settings, apart from the Configuration Mode Setting; Look in Assignments, Device, Device & Pin Options and then the Configuration tab – the Configuration Mode setting must be REMOTE and not LOCAL for the remote update to work. Whilst you are here, also check the correct configuration device is specified – in this case EPCQ256.

PLL

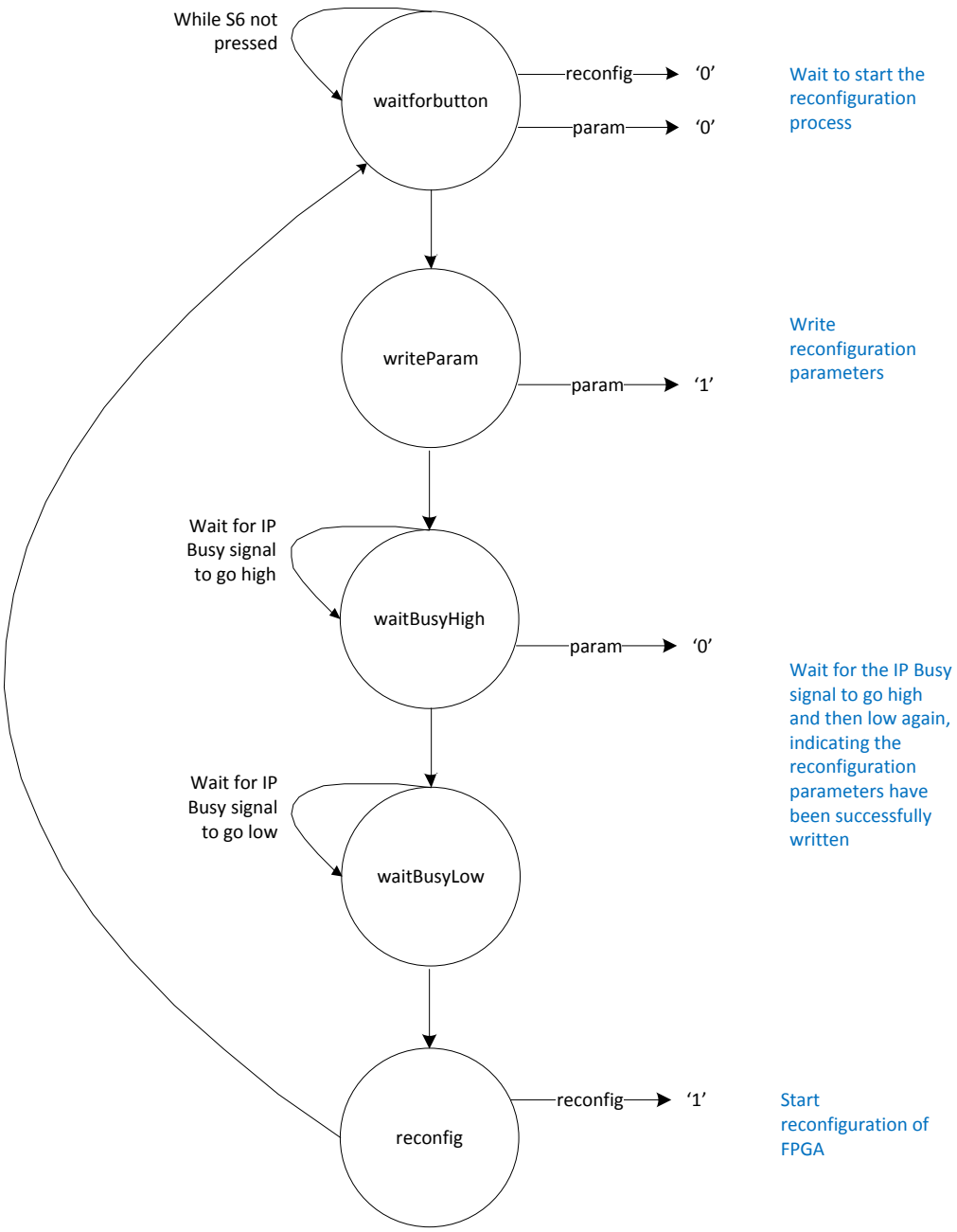
A PLL has been included because the design uses the 25MHz reference clock input provided by the development board, but the ALTREMOTEUPDATE IP has a maximum recommended clock frequency of 20MHz. The PLL takes 25MHz and generates 20MHz. *Note – take care to get your reset polarity correct for this IP component.*

Driving the ALTREMOTEUPDATE IP

The design keeps this as basic as possible:

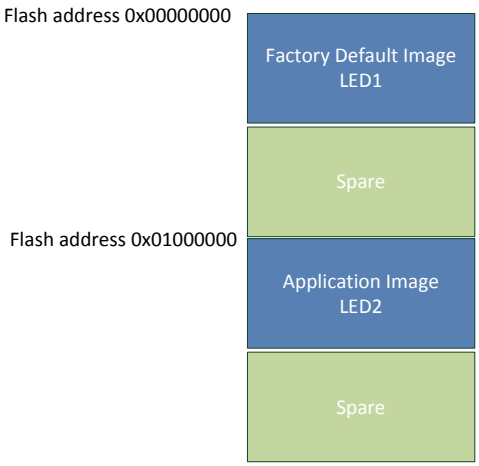
- The functionality is fixed – the IP is just set to reconfigure the FPGA from a fixed address in the flash
- None of the status or reconfiguration error functionality is used. See AN603 for more details as you can do a lot more with this IP that has been shown here.

To reconfigure the FPGA on detecting switch S6 is pressed a simple state machine was created. The timing of the signals is described in application note AN603.



Creating and programming the EPCQ image

This example assumes that the EPCQ256 fitted to the development board is already configured with two images, the factory default (LED1) and the application image (LED2). The expected location of these images in the flash device is given below:

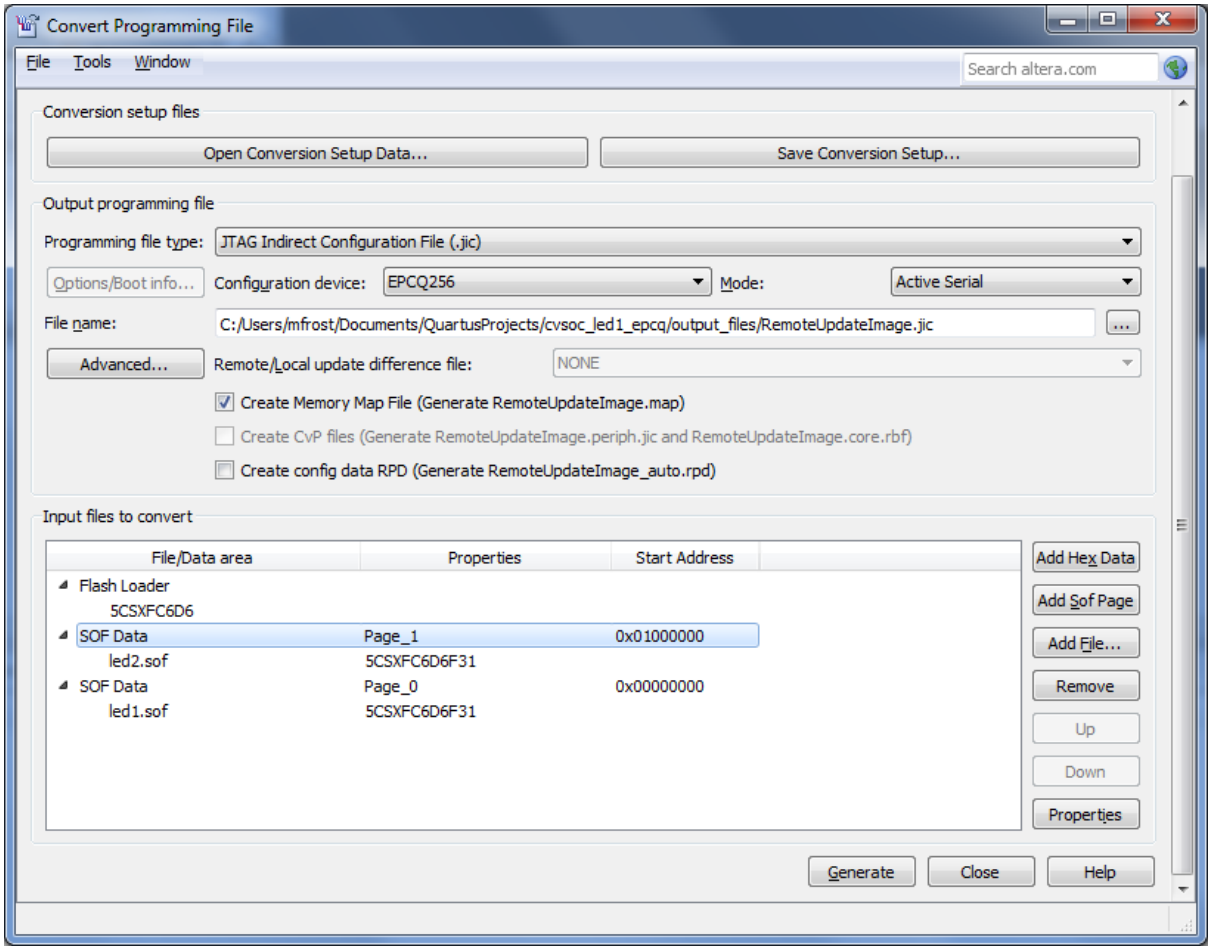


The following describes how to take the two image .SOF files (LED1.sof and LED2.sof), create the EPCQ image (.JIC) and finally how to program the .JIC into the flash.

Creating the .JIC file

Fire up mighty Quartus, and go to File, Convert Programming Files. The do the following:

- Select programming file type to be .JIC
- Select the EPCQ256 and specify an output file name
- In the “input Files to Convert’ GUI, click on the Flash Loader, and Add Device and select the correct FPGA (5CSXFC6D6). This specifies the serial flash loader which will allow programming of the flash device via the JTAG interface. This flash loader is a temporary image and not a permanent part of your design.
- Now click SOF Data and add file, and select LED1.SOF.
- Then click Properties, and set the address mode to Start and select the base address in the flash for this image to be 0x00000000. Select OK
- Finally, we need to add the second image, LED2.sof, so select Add SOF Page and repeat the above for LED2.sof, but this time select the base address for this image to be 0x01000000.
- Before you hit Generate, your screen should look like the screenshot below. If it does, then hoorah, and hit Generate to create the .JIC file.



Programming the .JIC into the EPCQ device

Depending on your revision of development board, you may have an un-cooperative MAX V image, which will prevent you from programming the EPCQ from JTAG. If this occurs reprogram the MAX V with the POF file included with this design (max_13_1_0.pof) – use the Quartus Programmer and select the MAX V and the POF and program.

The other potential issue you might encounter could be unreliable programming of the .JIC into the EPCQ. This can be caused by signal integrity issues on some earlier development boards, leading to all sorts of erratic programming behaviour. The way around this is to lower the JTAG clock speed:

- Open an NIOS II Command Shell (there will be a link under you Altera installation)
- Confirm your current JTAG clock setting with the following command:
  `jtagconfig --getparam 1 JtagClock`
- This will return a value, most probably `16M`, or 16MHz.
- Set the JTAG clock frequency to a lower value, say 6MHz using the following command and then confirm the change using the previous command.
  `jtagconfig --setparam 1 JtagClock 6M`
- Programming should now be reliable – NOTE that you will have to make this change with every power cycle of the board!

OK, so now we have everything setup OK to actually burn the .JIC into the EPCQ. Fire up the Quartus programmer and auto detect the hardware. You should see the FPGA on the scan chain (5CSXFC6D6). Right click on the FPGA, select Change File and select your .JIC file.

The programmer view will change slightly and you will now see the EPCQ256 connected to the FPGA. In the top window, select Program/Configure (both the FPGA and the EPCQ should have a tick box next to them) and hit Start. The programmer should then burn the .JIC into the EPCQ.

Testing on the dev board

If you have correctly programmed your EPCQ flash device…. On switching on the development board you should now see the CONF done LED lit (see photo #1) and LEDs 1 and 0 slowly flashing (counting up). This shows that image LED1 is running OK.

You can test the reconfiguration process by hitting user switch S6 (see photo #1) and after a short while the CONF done LED should go off and then back on, but this time LEDs 3 and 2 should be flashing (also counting but much faster); this indicates that image LED2 is running. i.e. image LED1 has reconfigured the FPGA with image LED2.

Now go and make yourself a nice cup of tea.

Files includes with the example

- LED1.QAR – the Quartus archive of the main project
- LED2.QAR – the Quartus archive of the application project (i.e. a basic LED flasher)
- MAX_13_1_0.pof – the MAX V image to use in case of JTAG programming issues with the FPGA
- RemoteUpdateImage.JIC – the example .JIC which should just work on the development board used for this example