**Setting up the AudioBoard**

**6\_20\_2016.**

This is a work in progress tutorial on taking a new audio recording collar and building the full system out of the GIT repo.

Tutorial Developed with Quartus 15.0.1 Build 150 06/03/2015 SJ Full Version.

It is recommended you proceed with 15.0.

Todo: More detail added to the 100% full system compile.

Sections.

1)Powering the Device.

2)Programming the CPLD/FLASH

a) Programming correct images to CPLD and Flash

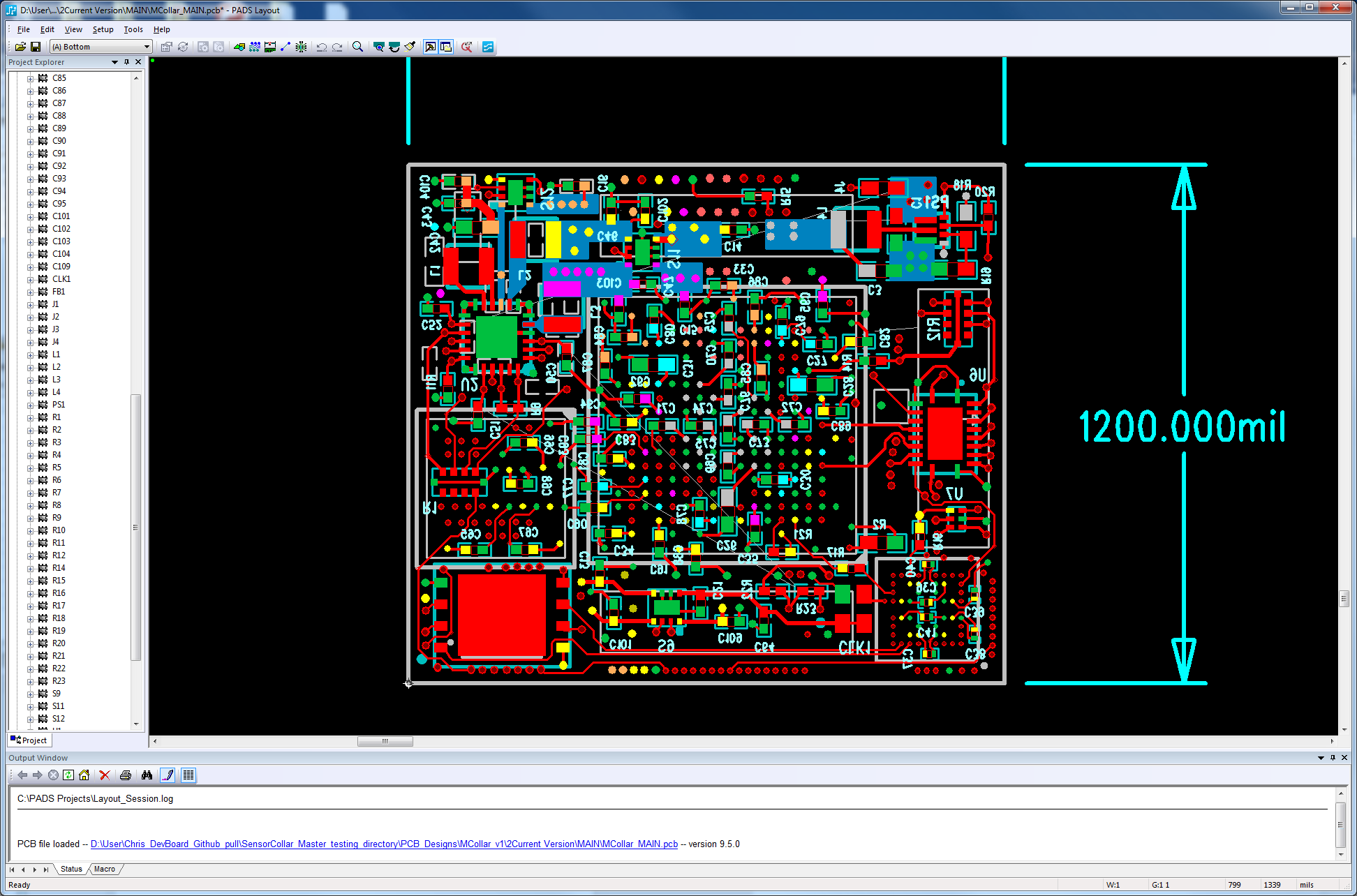
4)Booting and Programming the FPGA.

5)Interpreting the Source Code Tree and Building the Full System

**1A) Powering the Device.**

A solder bridge is used to route power from the ESH\_AUX\_VCC input pin of the 10 pin connector onto the PWR\_MAIN bus. ESH\_AUX\_VCC normally routes to the audio recording collar daughter board where it is input to the solar controller before coming back to the main board. Jumping allows us to power the main board without a sensor board attached. We can keep the jumper throughout development, even with a sensor board attached. The jumper is J4 located on the back of the board.

The force\_start switch on the interface board (S2) must be switched on to pull up JTAG. If the force\_start switch output is not on JTAG through the interface board will not work.



An interface board is used to program the audio recording collar stack.

To program, connect a USB Blaster to the interface board.

The Blaster ribbon cables sits away from the board.

The interface board setup also includes a ribbon cable going to a USB 3.0 connector which interfaces with the collar board. Pin 1 is designated on the boards throughout. The ribbon cable pin 1 stripes should be matched everywhere.

The interface board and the audio recording collar stack can be powered using 4.1V on the VCC pin of the EXT SERIAL header of the interface board. Connect ground from the supply as well to the GND pin of the EXT SERIAL header. The current limit should always be set to low (~100mA) to test for shorts in the system initially. After the shorts have been ruled out, current limiting can be set higher (~500mA) to properly power the system.

A full system with FPGA powered can consume 100mA idling. Current spikes can reach up about 200mA and greater.

No physical modifications to the board are needed to get a CPLD image booting the FPGA.

Start with the CPLD\_INIT directory of the GIT repository.

**2A) Programming the CPLD/FLASH**

Source\_Code\AudioRecordingCollar\CPLD\_Init

Used for loading flash with an FPGA image.

Before opening the Quartus project. Copy AudioRecordingCollarCPLDInit\_IO.qsf and rename to AudioRecordingCollarCPLDInit.qsf

A note on the QSF files: The git project attempts to keep the QSF which Quartus maintains apart from the QSF which defines the board specific pins/ports/names as defined by the physical board.

For this reason, one should copy the Toplevel\_name\*\_IO.qsf to Toplevel\_name.qsf when starting from git for the first time for any project.

**This rename should be done before opening the project for the first time. This is important step.**

Files needed for this stage: These files need to be added to the Quartus project.

../../MainCollar/General/Utilities\_pkg.vhd

../../MainCollar/PowerController/FlashWrite.vhd

../../MainCollar/PowerController/FlashWrite.qip

../../MainCollar/PowerController/FlashInit.vhd

AudioRecordingCollarCPLDInit\_TopLevel.vhd

Several Project wide .TCL scripts are needed. These include at\_compile\_start.tcl and set\_vhdl\_constants.tcl. These are found in the /QuartusII directory.

Of note is that the CPLD used on the audio recording collar IS different than the one used for the DevBoard Power Monitor. Device name and part number is different for this CPLD. This is handled in the QSF file.

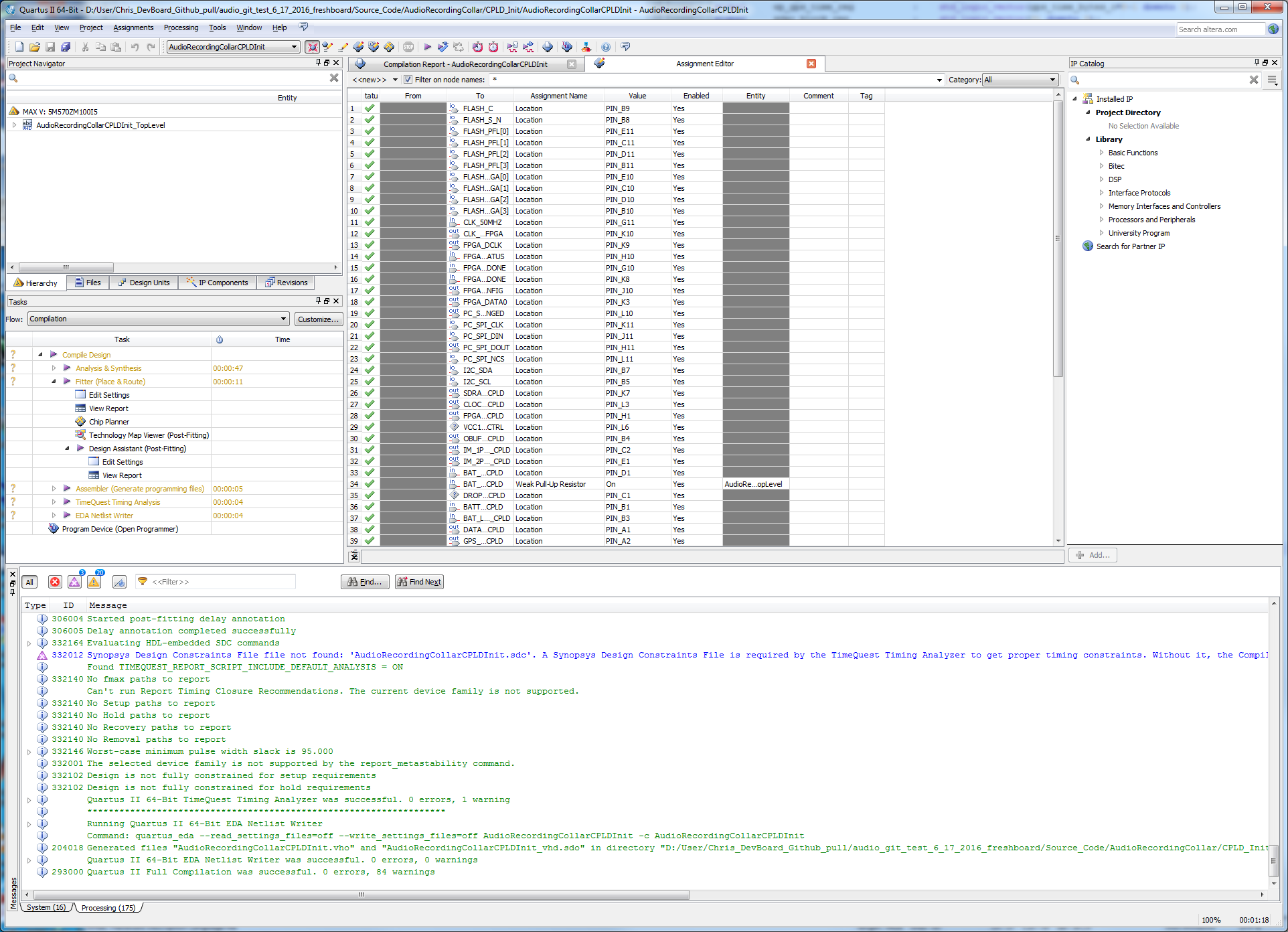
Copy the TCL scripts from the GIT QuartusII directory to your working directory. If one of these file is missing, check the DevBoard\_PowerMonitor\FPGA directory. Additionally, a log file must be created. In the working directory, create a commit\_timestamp.log file. (Alternatively, a commit\_timestamp.txt file can be used if the file name extension on line 34 of the at\_compile\_start.tcl file is changed)

You should be able to compile now. A POF should now exist in the output\_files directory.

Before programming the CPLD check that the assignments did come in from the QSF file correctly.

This can be done by opening the Assignments->Assignment Editor and checking for assignments. Verify the dialogue is not blank.

Also double check the device type at this point. This CPLD is a 5M570ZM100I5



Open the programmer and program the POF to the CPLD. The CPLD is the first device on the JTAG chain. Its device identifier is 5M570ZM100. A POF can now be selected by double clicking under the FILE section of the main center window of the corresponding JTAG device. A file browser will open.

The image below shows the FPGA in the chain too. However, it won’t be there when you first initially program a new board. Notice the check boxes used to program. Ensure real-time ISP is enabled. Program/Configure the CFM of the CPLD. We don’t need/want to program the UFM of the CPLD. Note, If an error occurs where the silicon ID does not match the chip ID and you’re using a USB Blaster II, try instead switching to a USB Blaster I.

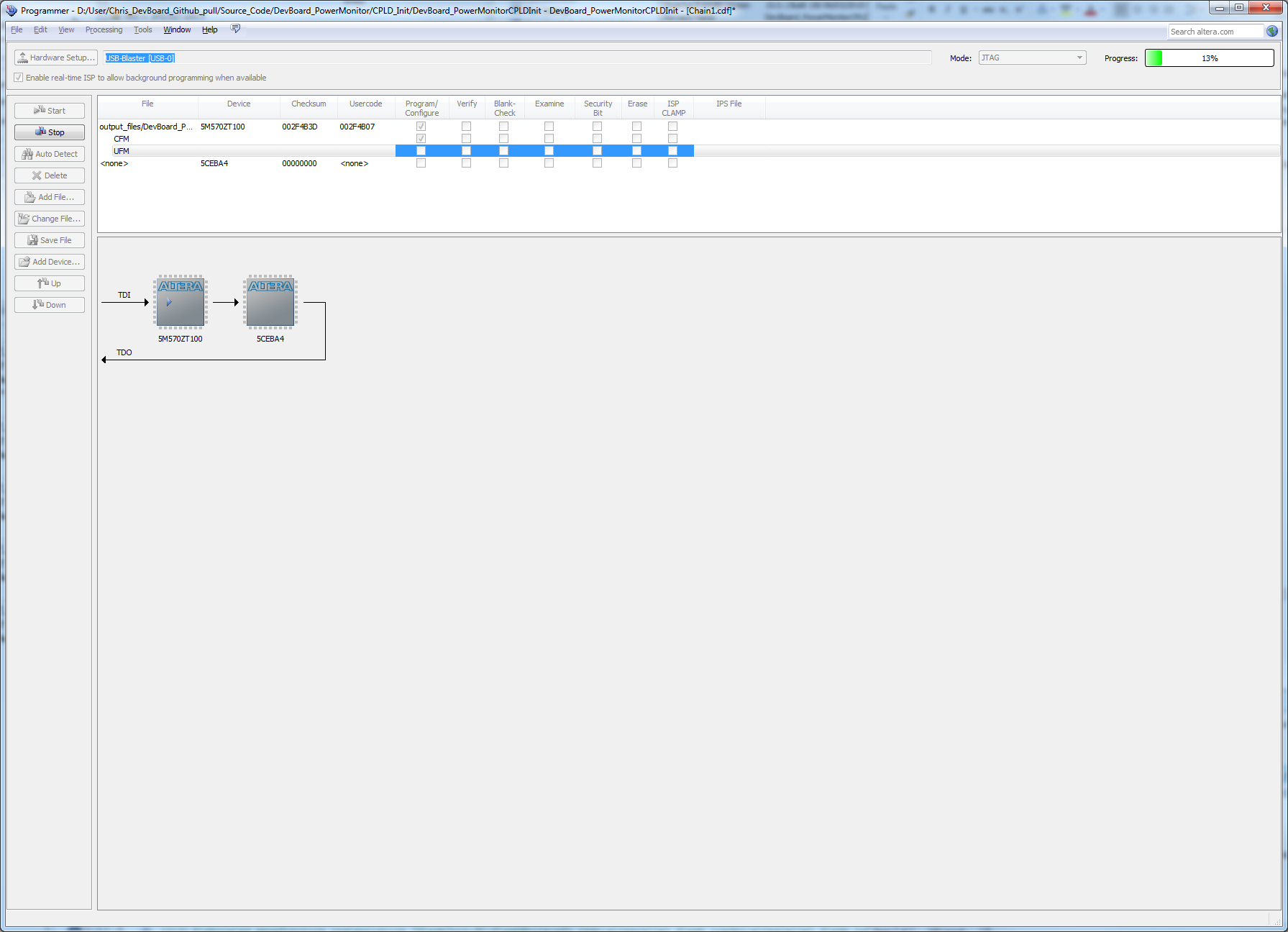


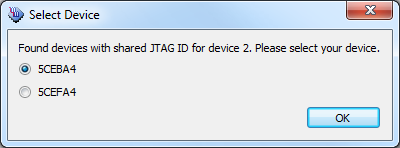
Figure 1 Programming CPLD with CPLD\_INIT

Reboot the board after flashing the CPLD. The new CPLD image is loaded after a reboot of the device. Remove and reapply power to the board.

Now we can program the flash chip with an FPGA image.

Use the auto-detect button in the programmer to rescan the JTAG chain.

The programmer needs us to select the ID for the FPGA.



If we take a look at the JTAG chain now:

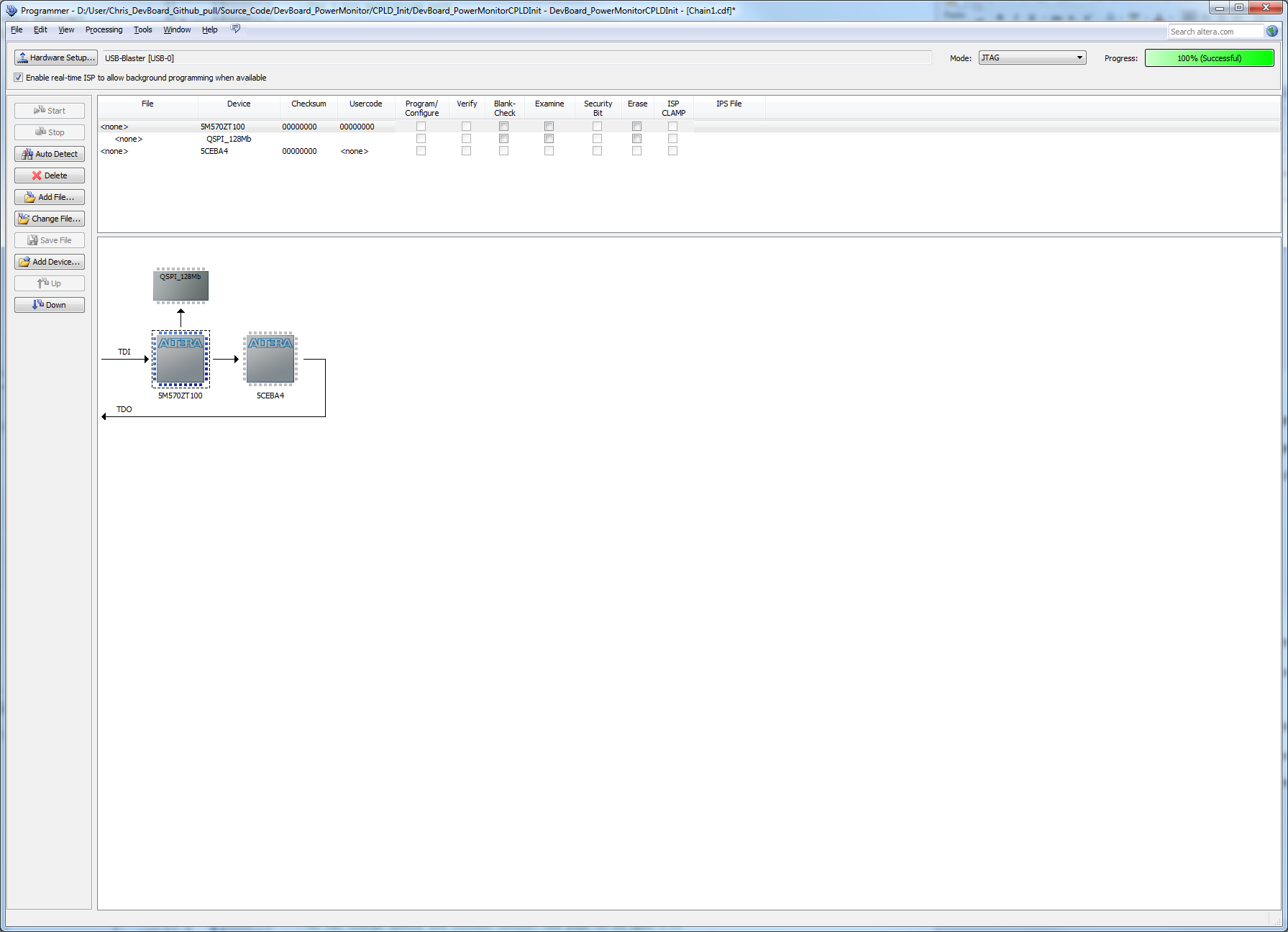


Figure 2 JTAG chain after programming CPLD\_INIT and repowering

Now the JTAG chain has the piece of flash and the FPGA on it. The flash is accessed through the parallel flash loader which is now loaded into the CPLD. The FPGA has now turned on. The Parallel Flash loader is a piece of configurable Altera IP which is loaded into the CPLD. It allows us to interact with the micron flash also connected to the CPLD.

On my setup I see about 50mA at the meter with CPLD\_INIT programmed to CPLD.

Now a converted FPGA image must be programmed to the attached flash.

This will involve:

Source\_Code\AudioRecordingCollar\FPGA\_INIT

Bare bones FPGA project for loading to flash.

Copy AudioRecordingCollarFPGA \_IO.qsf and rename AudioRecordingCollarFPGA.qsf. This is an important step.

Upon opening the project after copying and renaming the QSF, the correct device name (5CEBA4U15C7) should be in the project. If not, try again from scratch. Quartus reads the QSF on boot, populating device name, bank options, and all pin assignments. Thus it is crucial the QSF be there and named correctly so that Quartus finds it upon boot.

Include files: The VHD file should be added explicitly to the project.

AudioRecordingCollarFPGA \_TopLevel.vhd

sdc\_values.tcl

sdc\_values.tcl should already be in the FPGA\_INIT git directory.

Copy the .tcl scripts from: /QuartusII

First a bare-bones bootable image of the FPGA must be compiled. The FPGA\_INIT directory exists for this purpose. It does not include any systems or the sdc timing framework. It simply displays a counter on a GPIO pin. This is a good way to check that your FPGA has booted and it booted with the image from flash.

Attempt to compile the design. It will fail, but two package files will be generated in the main directory:

compile\_start\_time\_pkg.vhd

shared\_sdc\_values\_pkg.vhd

These are being generated by the at\_compile\_start.tcl script being linked to by the QSF. Include these package files in the quartus project and recompile.

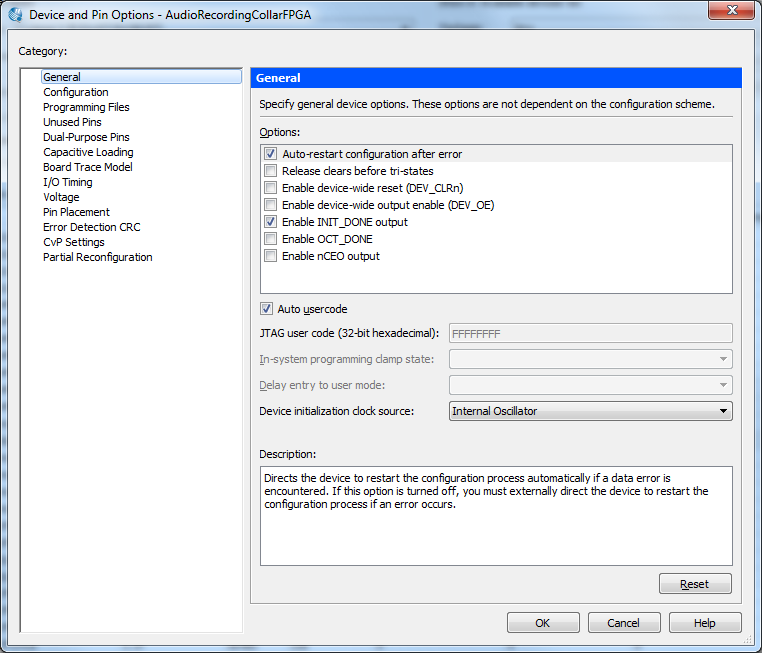
Compilation should be successful. The flow of steps as described here should be almost identical for a proper image to be generated.

An FPGA image booted from flash looks to a different source to begin clocking its design. For a FPGA SOF programmed over JTAG, the internal FPGA oscillator is used to boot the image. For an image booted from flash, the FPGA looks to the DCLK FPGA pin for boot. These differences are reflected in the QSF files. The QSF’s of FPGA\_INIT vs FPGA differ.

set\_global\_assignment -name DEVICE\_INITIALIZATION\_CLOCK INIT\_INTOSC

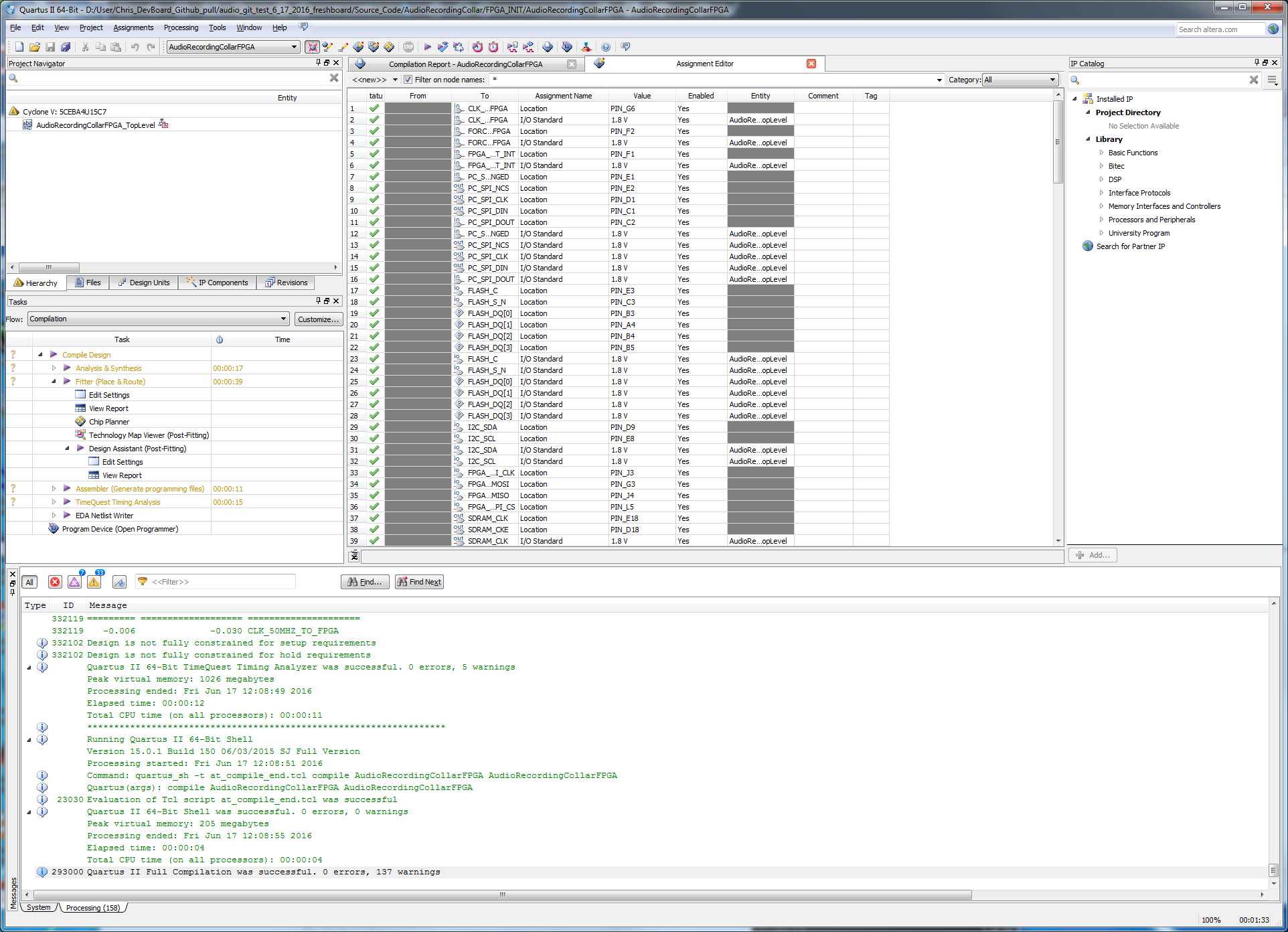
set\_global\_assignment -name DEVICE\_INITIALIZATION\_CLOCK INIT\_DCLK

The other pin that must be enabled on the FPGA for the PFL in booting from flash is the init\_done pin. These options are included in the \_IO.qsf, but the corresponding options are shown in Quartus here. **This is all handled in the qsf’s**. I am including the Quartus options here for documentation.



You should now be able to compile. This will result in an FPGA .SOF image. However the flash needs a .POF image.

The conversion of SOF to POF has been added to the script at\_compile\_end.tcl which is included in the QSF file. It generates the POF for our board automatically now. See the appendix for how to do this manually. Look for the AudioRecordingCollarCPLDInit.pof in the /output\_files directory after the design is finished compiling.



Again, check the Device Part Number and that the assignments came in. The FPGA is a 5CEBA4U15C7. Programming a bad image to flash (incorrectly driven FPGA pins) can lock you out of the JTAG chain requiring board soldering work to unlock the chain. This is because the next CPLD image will autoboot from flash to FPGA on every boot.

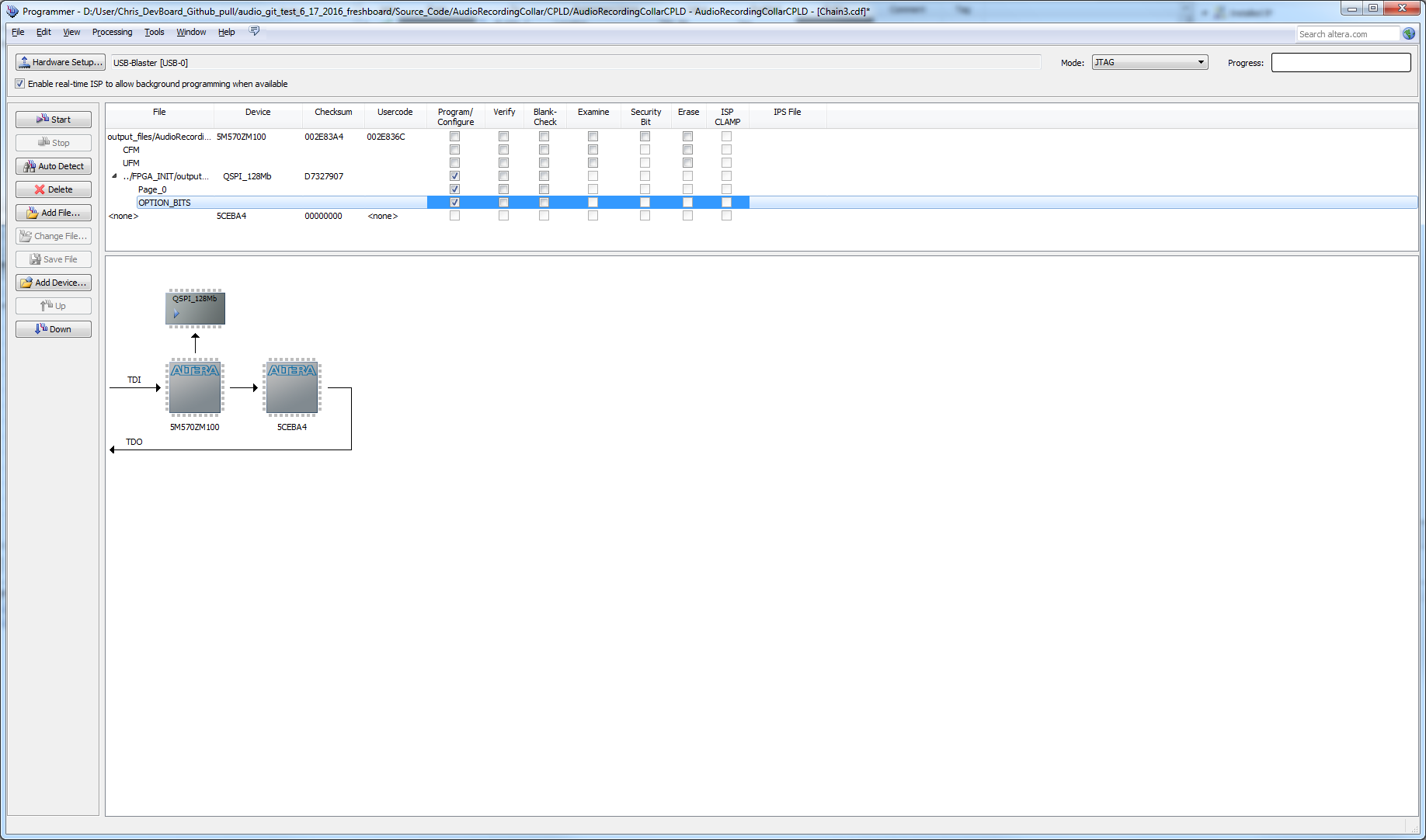


Figure 3 Programming Flash with FPGA POF

Program the QSPI\_128Mb with the AudioRecordingCollarCPLDInit.pof. Check the Page\_0 and OPTION\_BITS.

It is good practice to always File->Save the chain descriptor file before hitting the Start button.

After programming the flash, verify the image in flash. This can be done by checking the Verify boxes next to the QSPI\_128Mb. Press “start”. The PAGE and Option Bits should both verify successfully. Hard-coding the option bits and SOF page allow this to be done. The project had less success with auto generated option bit location.

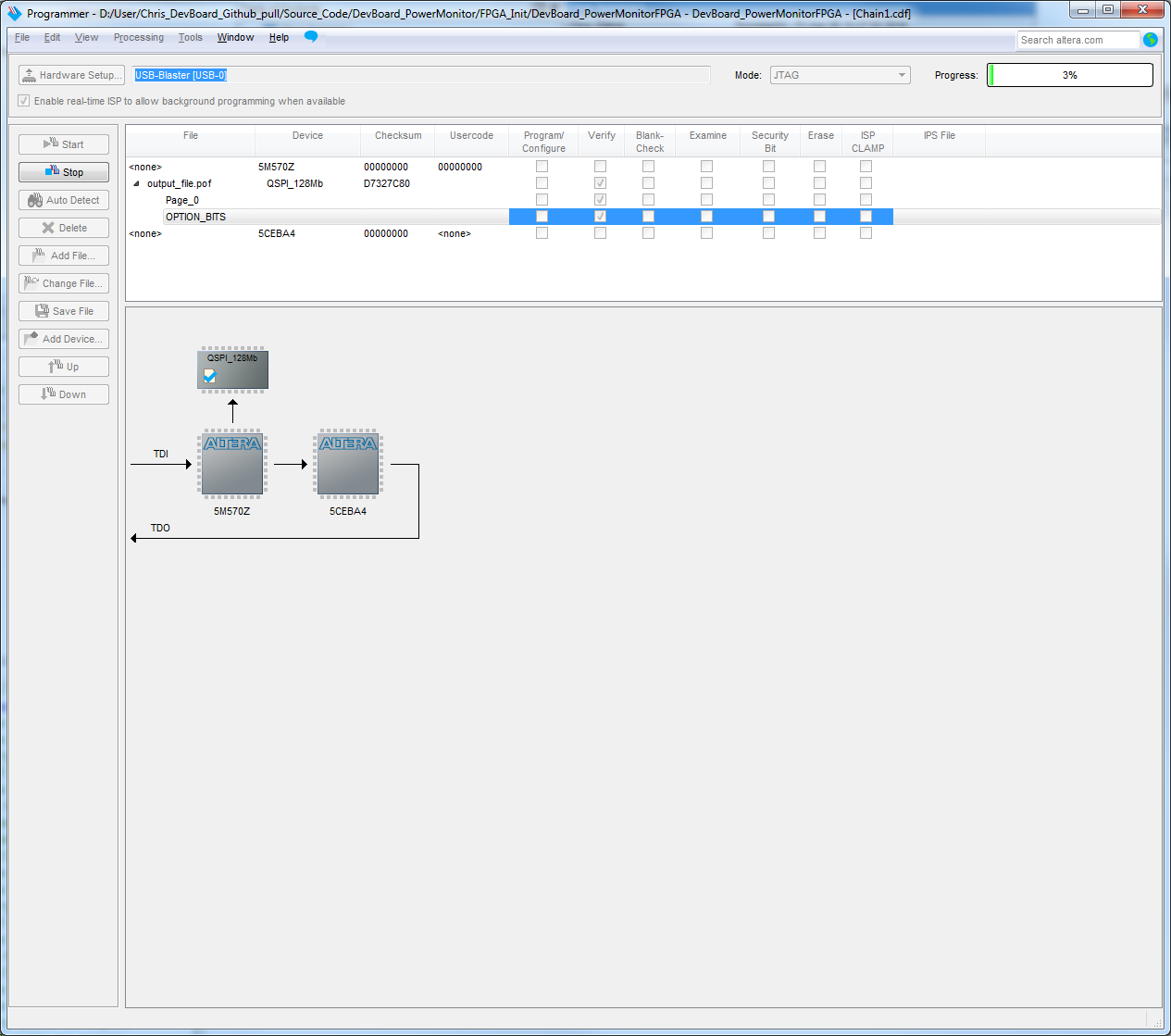


Figure 4 Verifying FPGA POF in FLASH

**4)Booting and Programming the FPGA.**

Now that the FPGA image is in flash, the image on the CPLD must be changed. The CPLD image is changed to the run-time image. This is done because the CPLD is only large enough to hold a flash program version of the PFL or a FPGA program version of the PFL, not both. This image includes capability to load FPGA from flash as well as handle all power controller operations as well as talk to the FPGA.

The git directory of interest is:

Source\_Code\ AudioRecordingCollar \CPLD

Used for loading FPGA from Flash on Boot and Normal Operation

Again rename the Toplevel\_name\*\_IO.qsf to Toplevel\_name.qsf so all the pins are defined for the CPLD. This should be done before opening the Quartus project.

Do this before opening the project for the first time. Upon opening quartus, the device name should have been read from the renamed QSF file.

For CPLD, these are the include files.

set\_global\_assignment -name VHDL\_FILE ../../MainCollar/General/Utilities\_pkg.vhd

set\_global\_assignment -name VHDL\_FILE AudioRecordingCollarCPLD\_TopLevel.vhd

set\_global\_assignment -name VHDL\_FILE ../../MainCollar/PowerController/PC\_StatusControl\_pkg.vhd

set\_global\_assignment -name VHDL\_FILE ../../MainCollar/PowerController/StatCtlSPI.vhd

set\_global\_assignment -name VHDL\_FILE ../../MainCollar/PowerController/PC\_UFM.vhd

set\_global\_assignment -name QIP\_FILE ../../MainCollar/PowerController/PFL.qip

set\_global\_assignment -name QIP\_FILE ../../MainCollar/PowerController/PFL.vhd

set\_global\_assignment -name VHDL\_FILE ../../MainCollar/PowerController/PowerController.vhd

set\_global\_assignment -name QIP\_FILE ../../MainCollar/PowerController/InternalFlash.qip

set\_global\_assignment -name QIP\_FILE ../../MainCollar/PowerController/InternalFlash.vhd

Again copy the .tcl scripts from the QuartusII directory.

This time a small amount of timing constraint has been added to the CPLD project. It is auto included in the project by being named the same as the top level. Nothing needs to be done here.

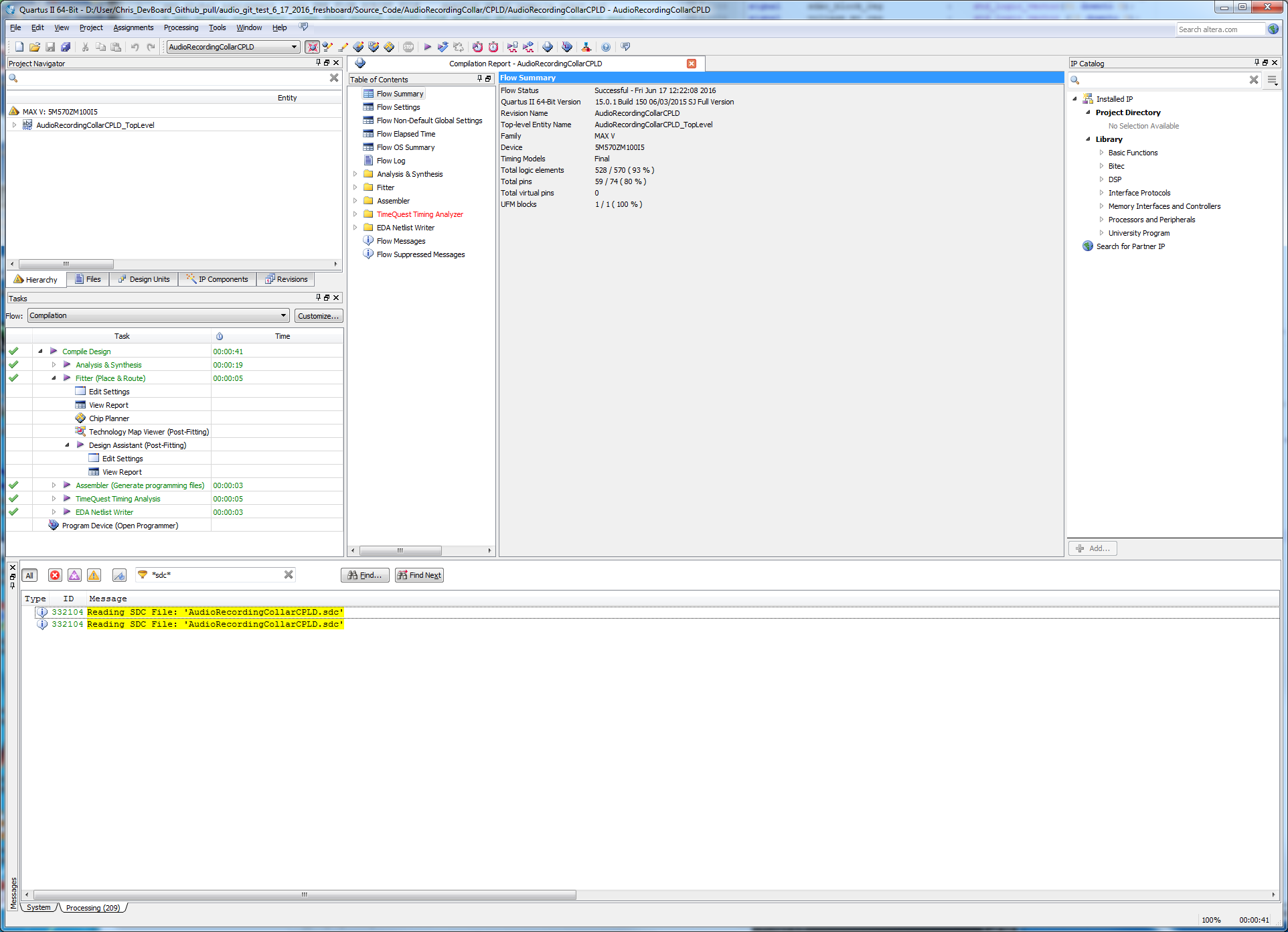
**This CPLD image is timing sensitive. Not including the SDC will result in the CPLD/FPGA booting process failing.**

The timing AudioRecordingCollarCPLD.sdc is named the same as the project file, AudioRecordingCollarCPLD.qpf. Quartus should auto analyze this SDC file be default. However, it might not be a bad idea to explicitly add it to the project as well.

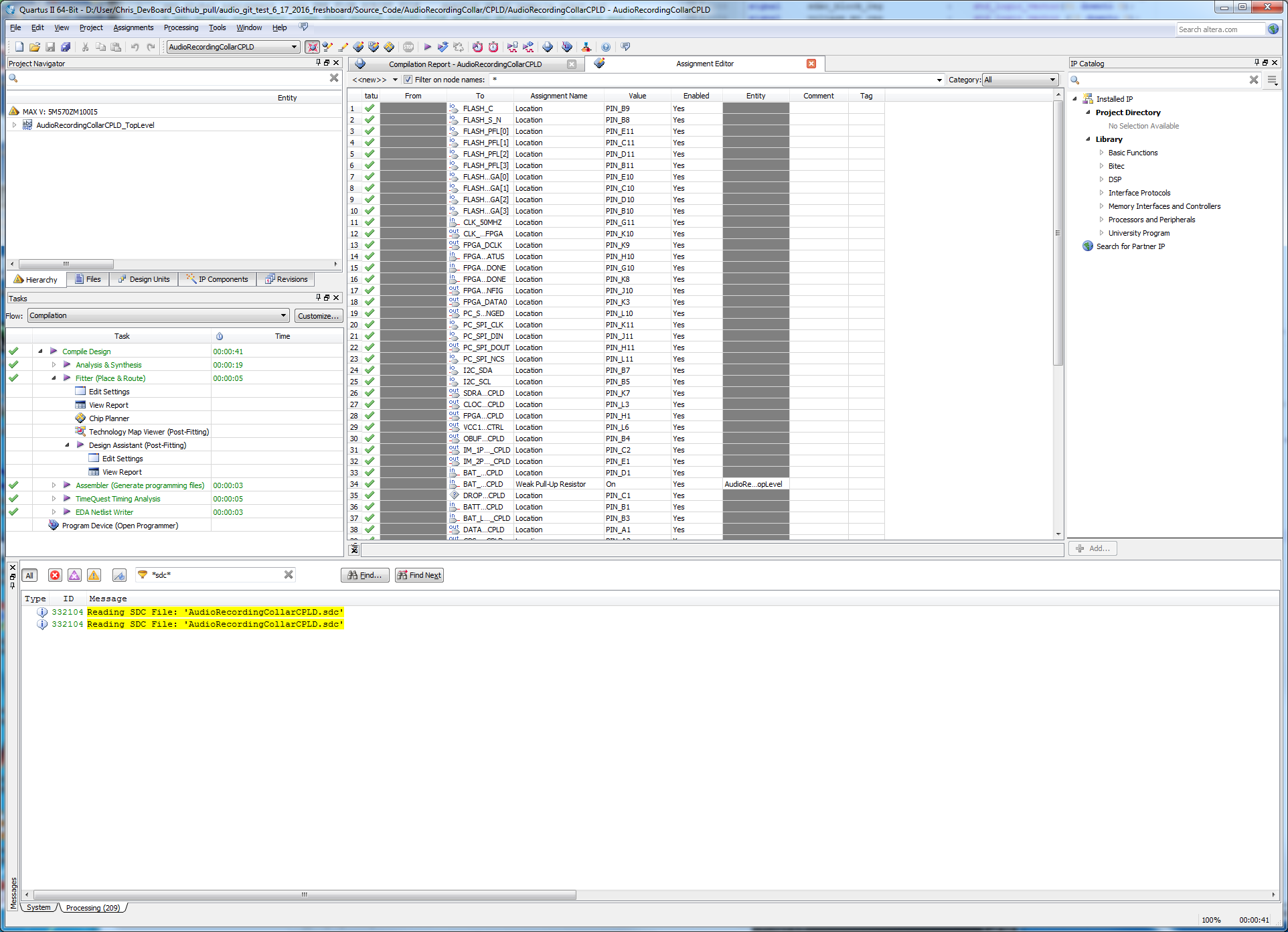
The project should compile now.

Double check a few things.

Check that the SDC was used for compile.



Check that the part number is correct and pins assignment did occur.



In the Quartus programmer, the generated POF can now be programmed.

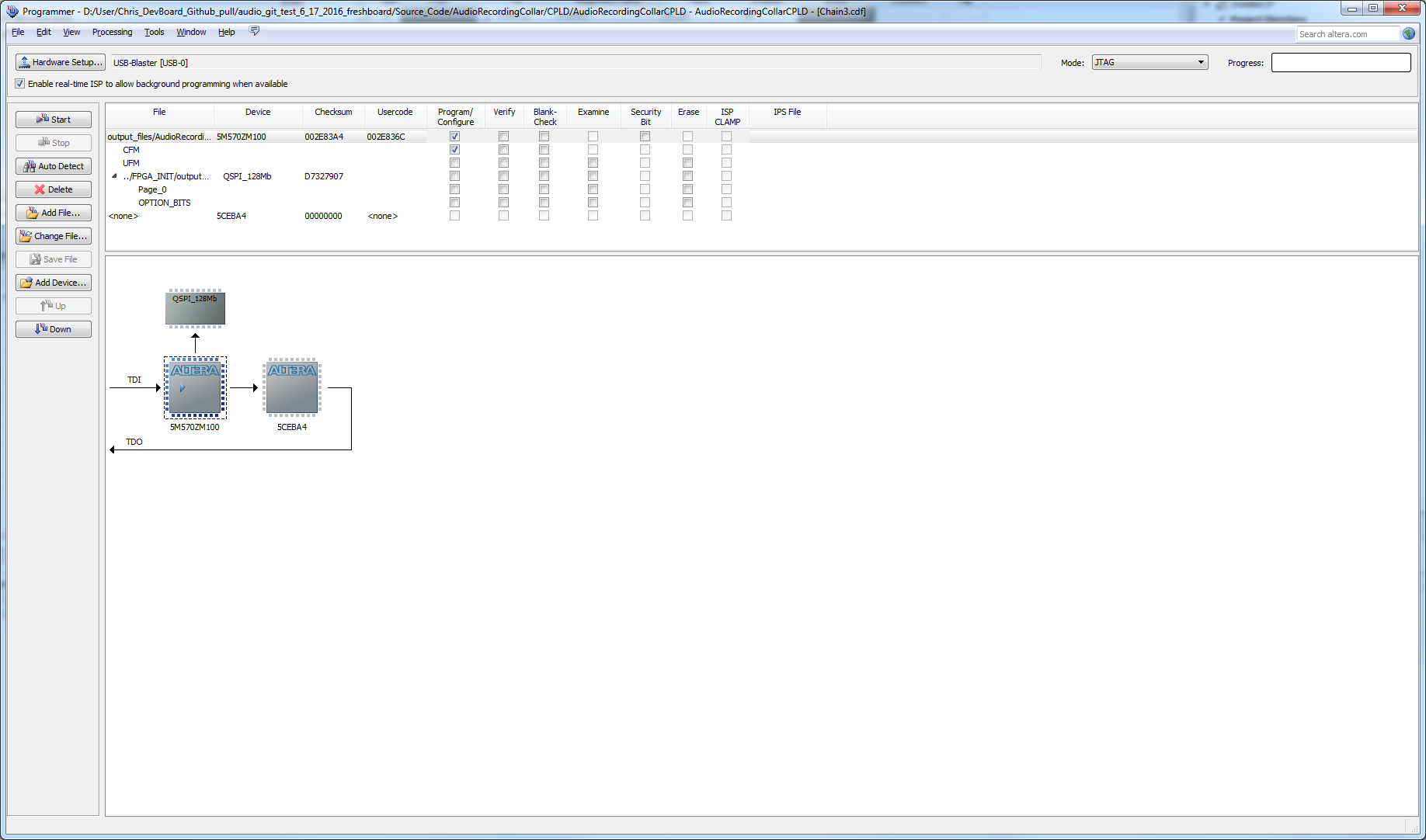


Figure 5 Programming the CPLD with runtime image

The POF is programmed to the CPLD (5M570ZM100) by checking the Program/Configure CFM boxes.

Now the CPLD needs to be rebooted. Remove and reapply power to the board.

Upon restart the CPLD will turn on and program the FPGA from the Micron Flash. The current power controller is sensitive to interrupt from the RTC/Battery Monitor/Force Startup Line. Any of these conditions will cause the FPGA to be booted. Upon start, the FPGA should boot. I see 60mA of current usage with the FPGA\_INIT image loaded and running.

One can check that the image booted okay, by checking for the programmed counter on the D+ line of the interface board at the EXT SERIAL header.

**Interpreting the Source Code Tree and Building the Full System**

The system is now ready to be programmed with another FPGA image of choice of the JTAG chain. You can add all the fancy stuff in the main FPGA directory. This includes the full collar system and the SDC timing systems. You could also just make your own prototyping FPGA images which interact with the device you choose. One can make use of the QSFs and TopLevel.vhd in the FPGA directory for a start.

A small amount of the collar system is required to supply power to devices. The CPLD must be told to turn on device X by the FPGA over an SPI bus. All device powered is switched by the CPLD.

The power switching functionality is tied into the startup\_shutdown and the StatCtlSPI\_FPGA entities. The generic Collar\_Control\_usePC\_c should be set to ‘1’ inside the Collar\_Control\_pkg.vhd. This enables the generate statement in the collar.vhd system corresponding to the FPGA CPLD SPI control. Startup\_Shutdown will signal the StatCtlSPI\_FPGA entity to send the control register with the corresponding switch enable bit, if the corresponding device is enabled in the Collar\_Control\_pkg.vhd.

For example: Collar\_Control\_useRadio\_c set to ‘1’ in Collar\_Control\_pkg.vhd will tell startup\_shutdown to switch on power to the TXRX chip during startup.

This system can be instantiated or you can just jump power to your device past the switches.

Source\_Code\DevBoard\_PowerMonitor\FPGA

Main FPGA JTAG programmed system. Top level instantiates collar.

To build a full system all .SDC files in the /Source\_Code/MainCollar and its subdirectories should be moved to the main project directory. All vhd/qip files under and in /MainCollar should be added to the project. One should not add files in test directories.

The Collar\_Control\_pkg.vhd is key. The startup shutdown entity looks to this file to decide which devices to turn on. Collar only generates entities and systems which have been turned on in Collar\_Control\_pkg.vhd. Startup Shutdown will only signal the CPLD to turn on devices upon bootup which have been set to ON in the Collar\_Control\_pkg.vhd.

The system will automatically step through the booting of all systems one after the other upon boot/program of the full MainCollar systems.

One should get boot up of IMU/SDCARD/GPS/Flashblock/SDRAM/MICROPHONES (if attached)/SYSTEM TIME/I2C/OTHERS. Plenty of bits will be flipping on both the main board and the daughter board.

**Odds and Ends**

**Flash Boot Pins of Interest**

Of interest are the pullups on the following FPGA pins. The FPGA pin guidelines for the Cyclone V E series indicates these should all be pulled up. In the audiorecording\_collar these pullups were added to the board design. In the devboard\_powermonitor these pullups are programmed into the CPLD image. These pins are all required to be pulled up and enabled for the CPLD to successfully configure the FPGA. In the audiorecording\_collar board, the pins are pulled up to 1.8V through 10k.

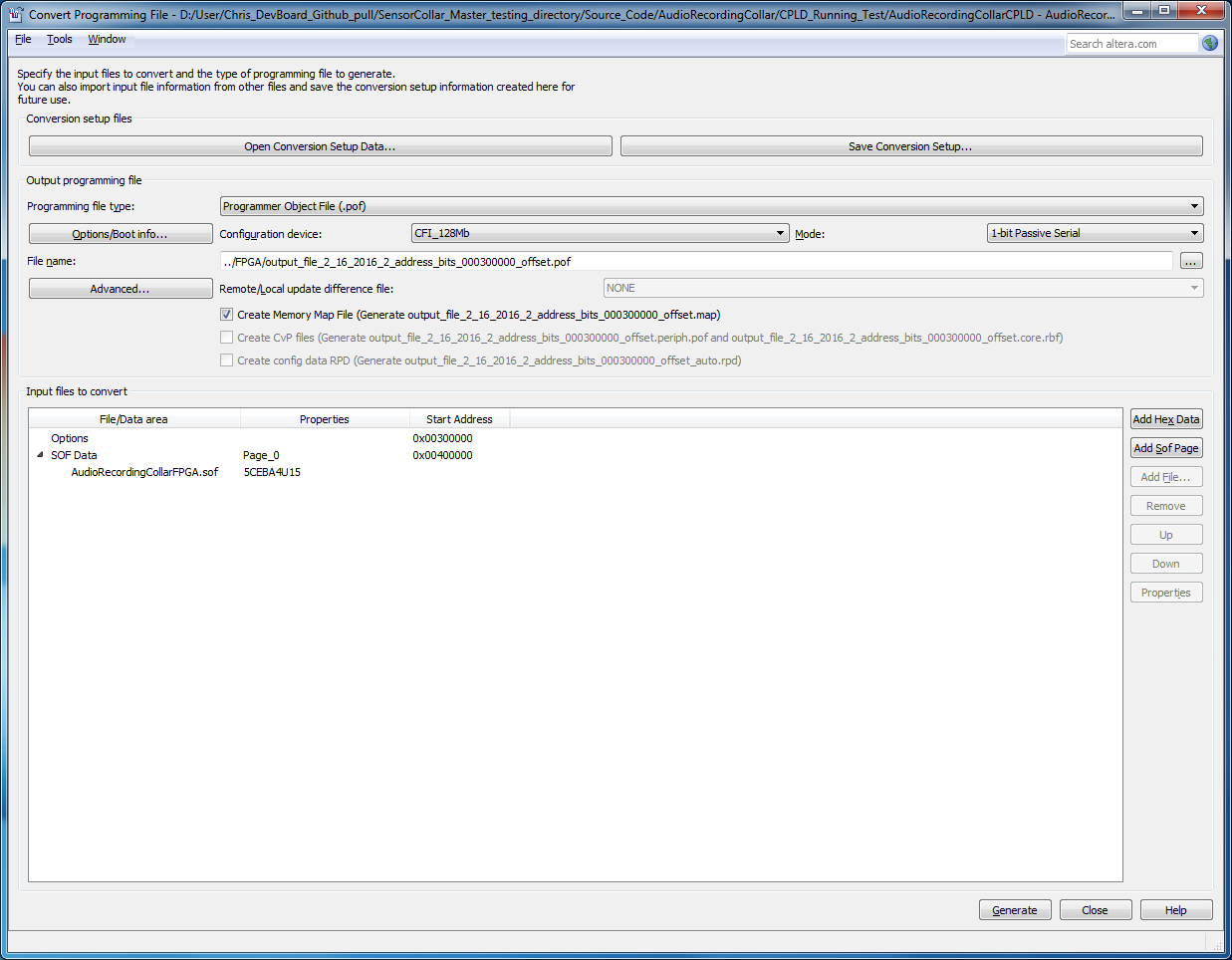
CONF\_DONE

INIT\_DONE

NSTATUS

**Use of the Convert Programming Files Menu**

Conversion of a SOF to POF is done through the Convert Programming Files section of the Quartus Application. It is located under the Files menu.



Parallel flash loader has been changed in the CPLD project to look for the FPGA image in flash from a certain offset. PFL also looks for option bits at a certain offset. This hard coding of addresses makes the system more robust. It also allows you to verify the FPGA image in the flash using the programmer.

Important check boxes.

Configuration device: CFI\_128Mb 1-bit Passive Serial.

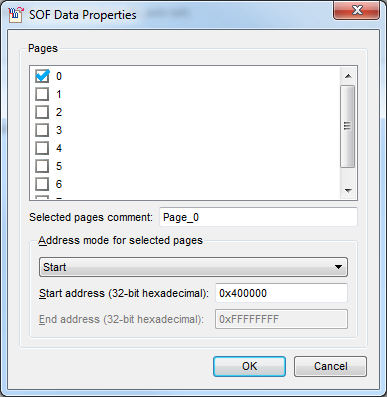
Note the odd choice of CFI\_128Mb, when the actual chip is a QSPI\_128Mb. This is explained by Altera here: <https://www.altera.com/support/support-resources/knowledge-base/solutions/rd05082012_592.html>

Check the name and location of your output.pof file.

Add the FPGA .SOF to the .POF by double clicking the SOF\_DATA Page\_0. Browse to where the .SOF image was compiled.

We now change where in the POF this image will be offset.

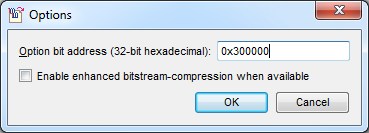
Click the SOF DATA and hit properties off to the right.



Change the address mode to “start”

Change the offset to 0x40\_0000. This location is coded into the option bits of the POF image. The PFL IP is not aware of this location. It looks to the option bits which then point to this location.

Now click the “Options/Boot Info…” in the main convert screen.



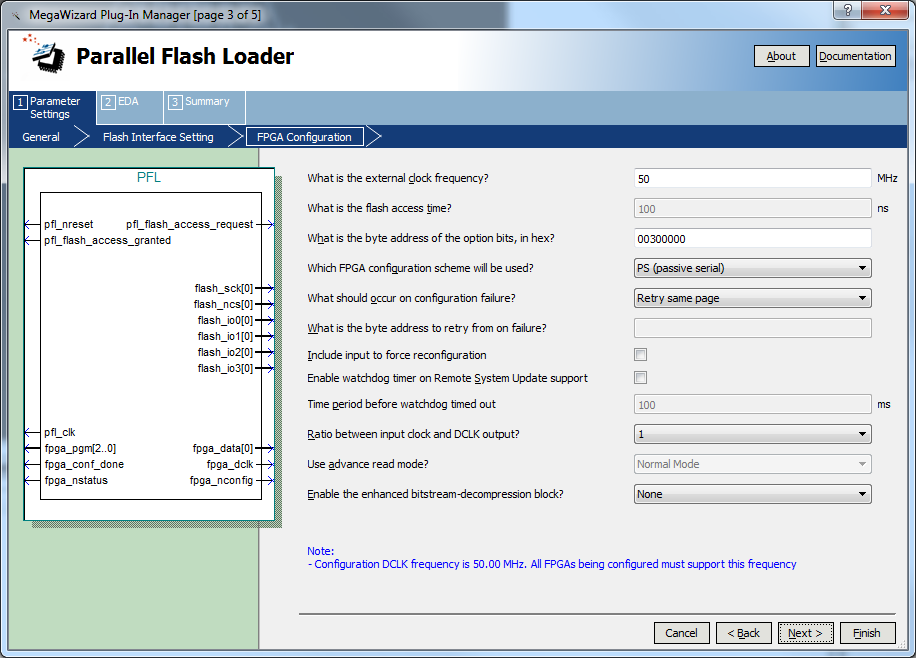
Change the option bits so that the reside at 0x30\_0000 in the POF. This address is coded into the PFL IP loaded into the CPLD image.

Hit Generate in the main window. The POF is now ready to be loaded to flash.

In the Quartus programmer, program the POF to the flash.

The PFL which is loaded into the CPLD RUNTIME image is the one which is aware of the option bits location. The PFL which is loaded to the Flashing loading CPLD doesn’t care about option bit locations.

Here is the relevant IP screen for the runtime CPLD PFL.



**If a bad FPGA image was programmed to flash, this is the recovery procedure.**

**A bad autobooted FPGA image can lock the JTAG chain as it is multiplexed on the board.**

Short not(OE) and 1.8V on the octal buffer. This will prevent booting the FPGA out of flash, and the CPLD alone will remain on the JTAG chain. The OE pin is highlighted in the image below. It is the lower pin on the board when looking at it from the front. You can either tack to the pin itself or scrape the solder mask from the filled via which is nearby and routes from the pin before going into the board.

