**Setting up the Devboard\_PowerMonitor**

This is a work in progress.

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

It is recommended you proceed with 15.0.

Sections.

1)3.3 Volt switch

2)Battery Recharge and Recharge Enable Switch

3)Powering the Device.

3)Programming the CPLD/FLASH

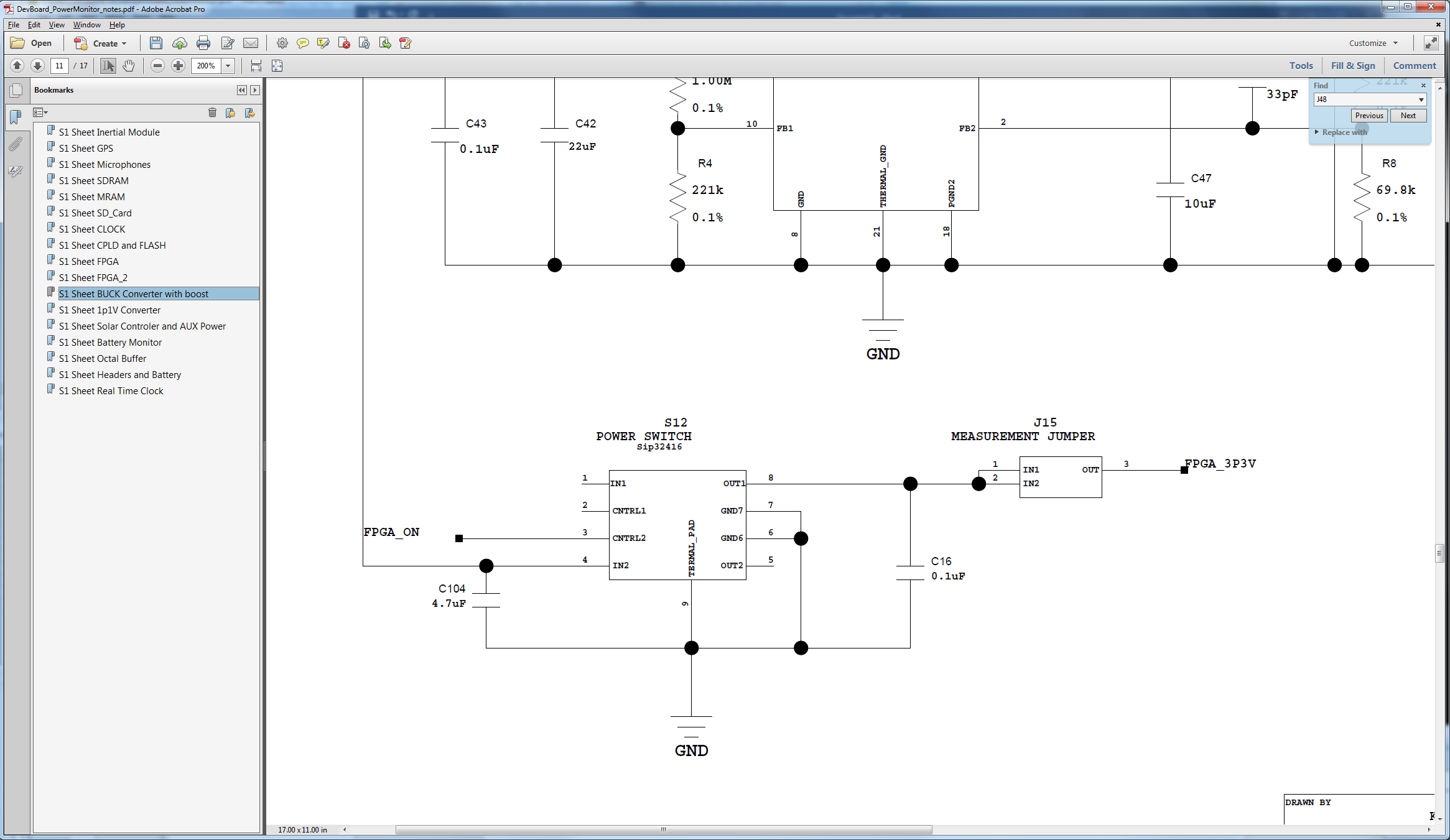
a) JTAG Jumping

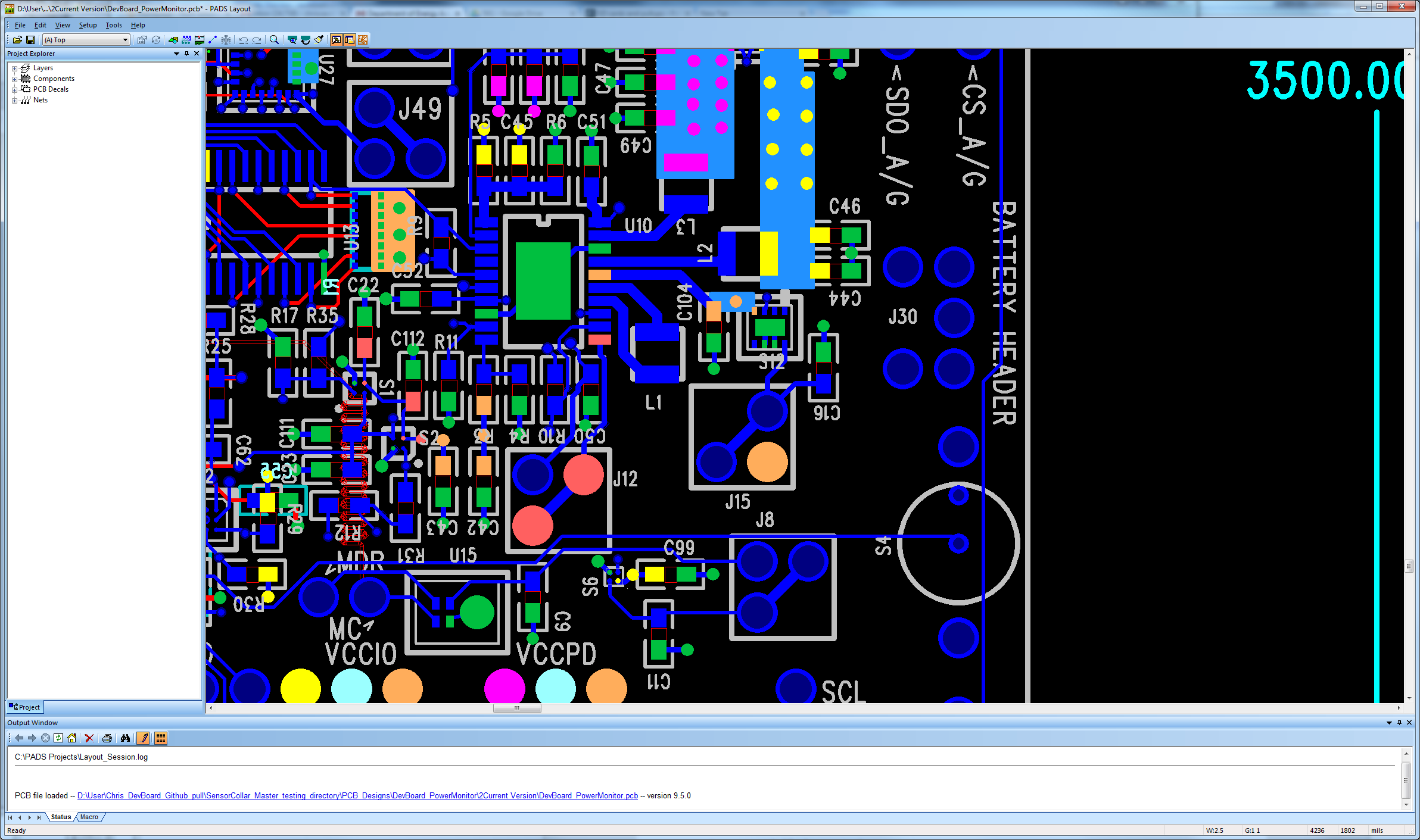
b) Programming correct images to CPLD and Flash

4)Booting and Programming the FPGA.

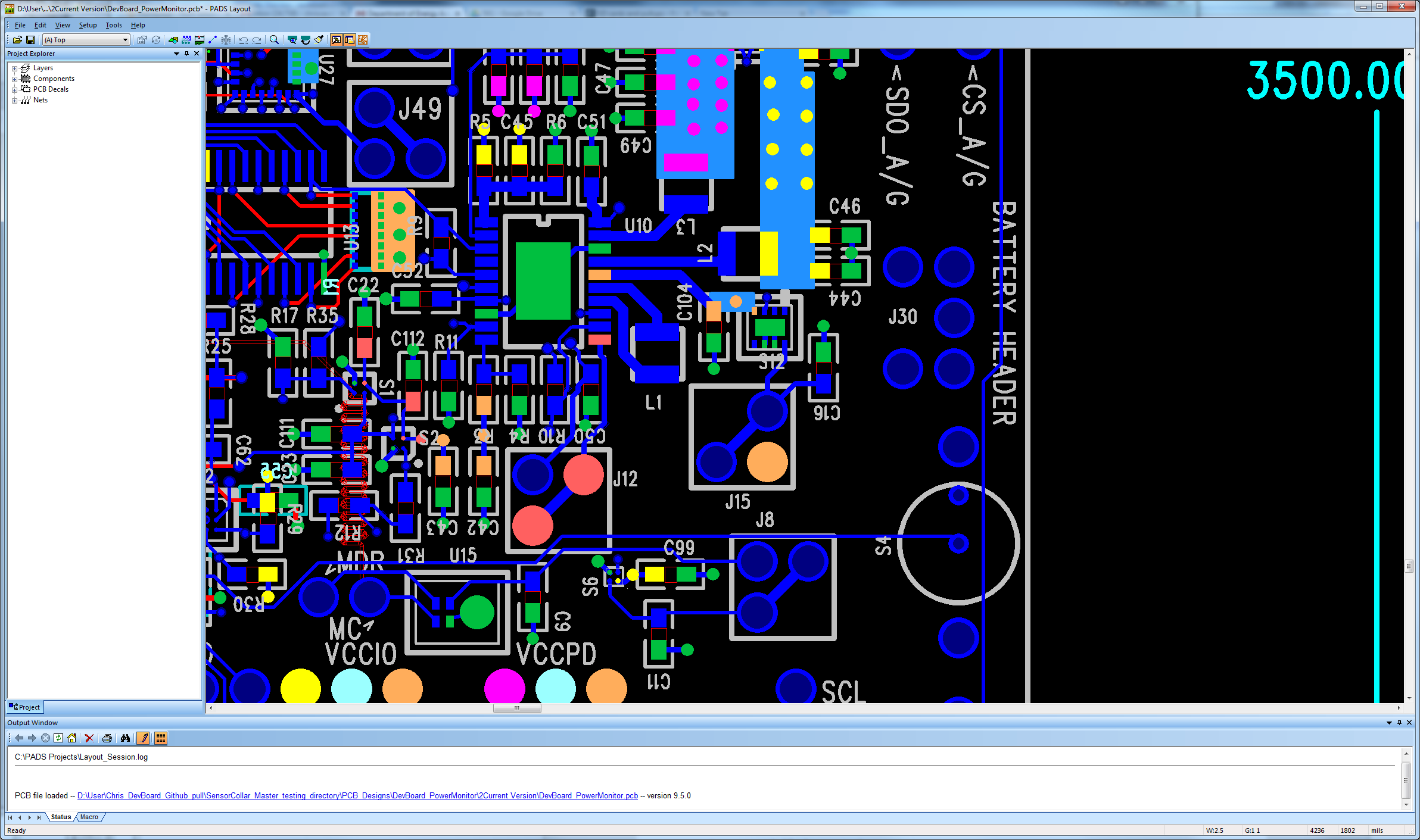
**1) 3.3 Volt Switch Fix**

The switch which controls the 3.3V rail which feeds the FPGA needs to be fixed. The switch is S12. The problem is that the control input and output do not match. Output 2 which is controlled is not tied anywhere.



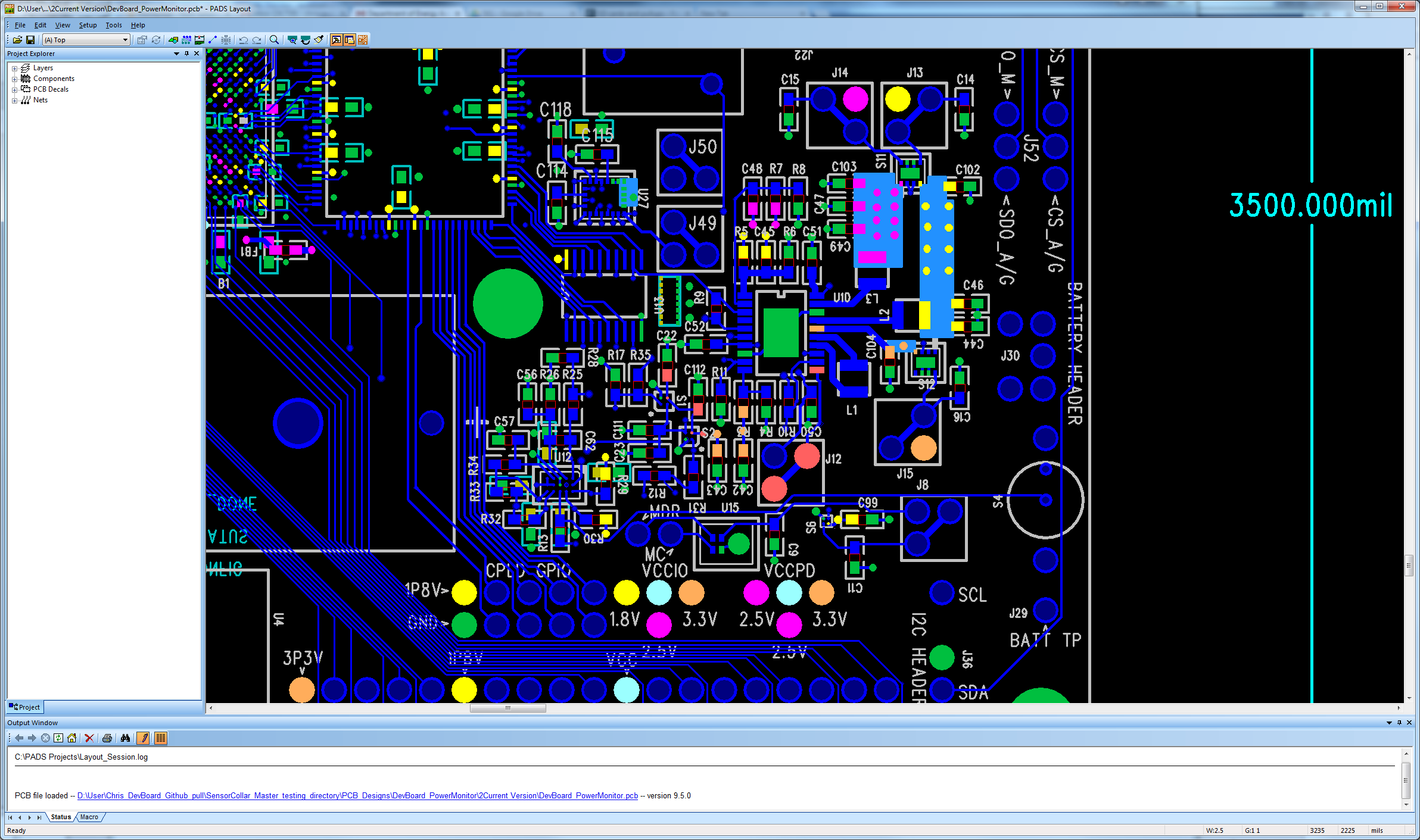


This is what needs to be physically done. Cut the trace at the dotted line. Insert a jumper wire as shown. I’m sure there are other solutions, this is what I did. You can also take out J15 completely and solder into those holes if that’s easier.



**2) S1 and S2 placed incorrectly. Pin 1 Placement Dot Incorrect. Decal Fine. Schematic Fine.**

Two switches on the board are placed incorrectly. If the switches are left in place, power is shorted to ground. You can either remove these switches or rotate them. If any battery testing is to be done on the board at a later time, these switches need to be rotated.



**Solution**

Either take the switches off with hot air or taken them off and rotate them 180 degrees. I’ve found you can do this without reapplying any paste or balls. They work after that.

**3) Powering the Device**

Jumpers needed for base operation:

J12 BuckBoost Input

J13 FPGA 1.8V

J14 FPGA 2.5V

J15 FPGA 3.3V

J21 Clock Power

J22 FPGA 1.1V

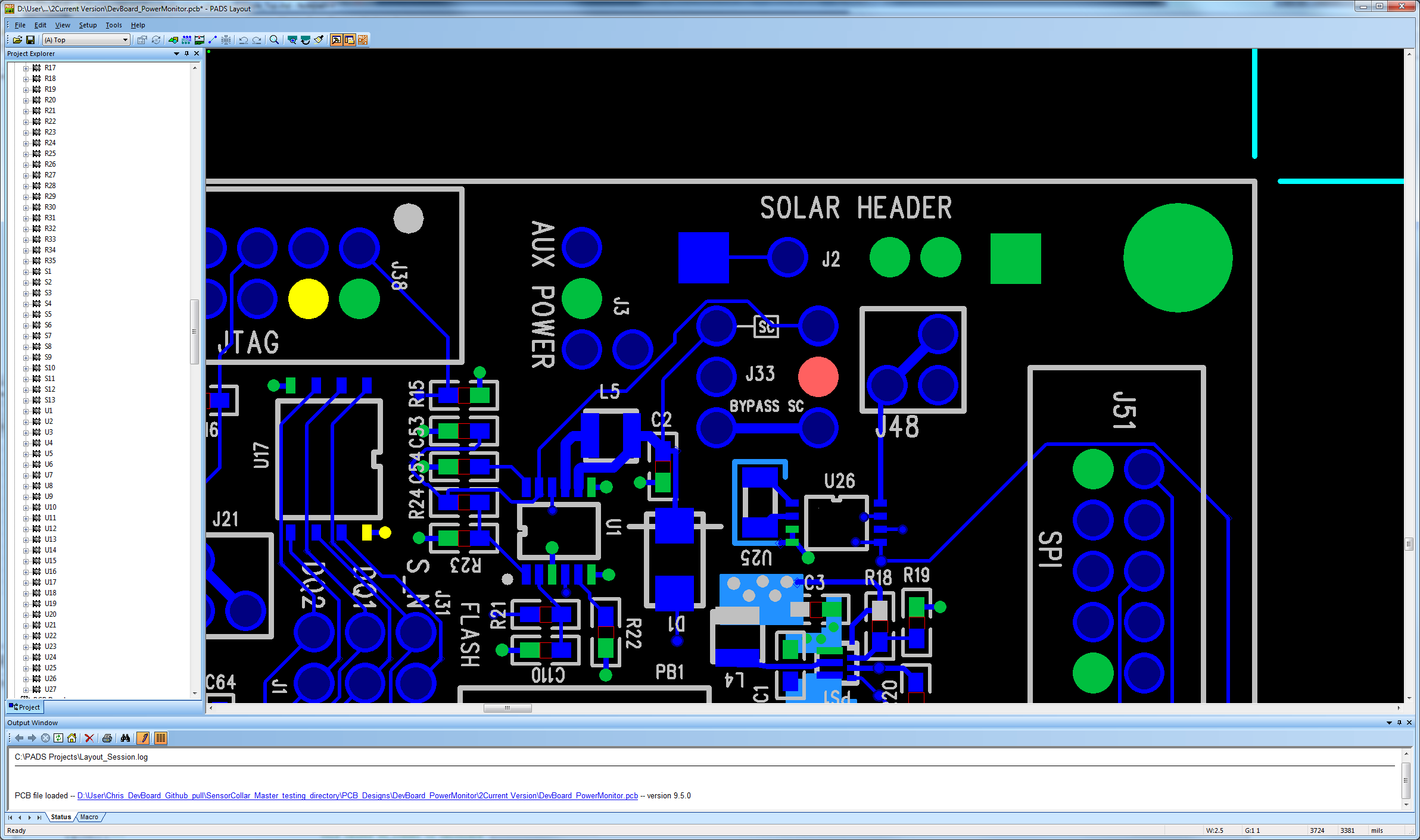
VCCIO and VCCPD Jumped 1.8V and 2.5V Respectively. Left Pin and Center Pin Jumped on both.

Bottom 2 sets of 2 at J33 Jumped. (Shown Below)

Any other devices you wish to work on need their associated power jumper.

Power should be applied as shown below at the solar header.

I strongly recommend current limiting your supply at first in case something goes wrong.

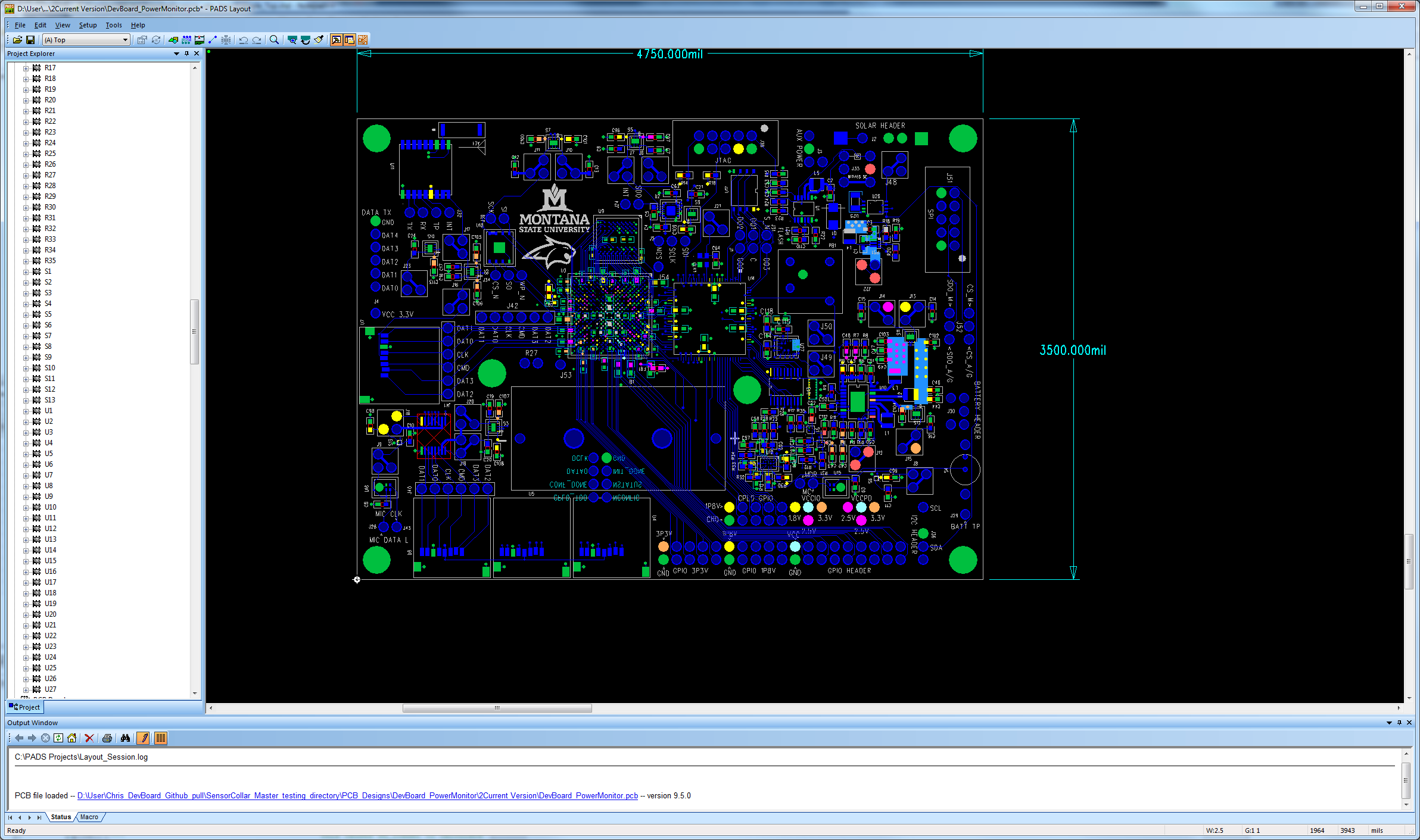


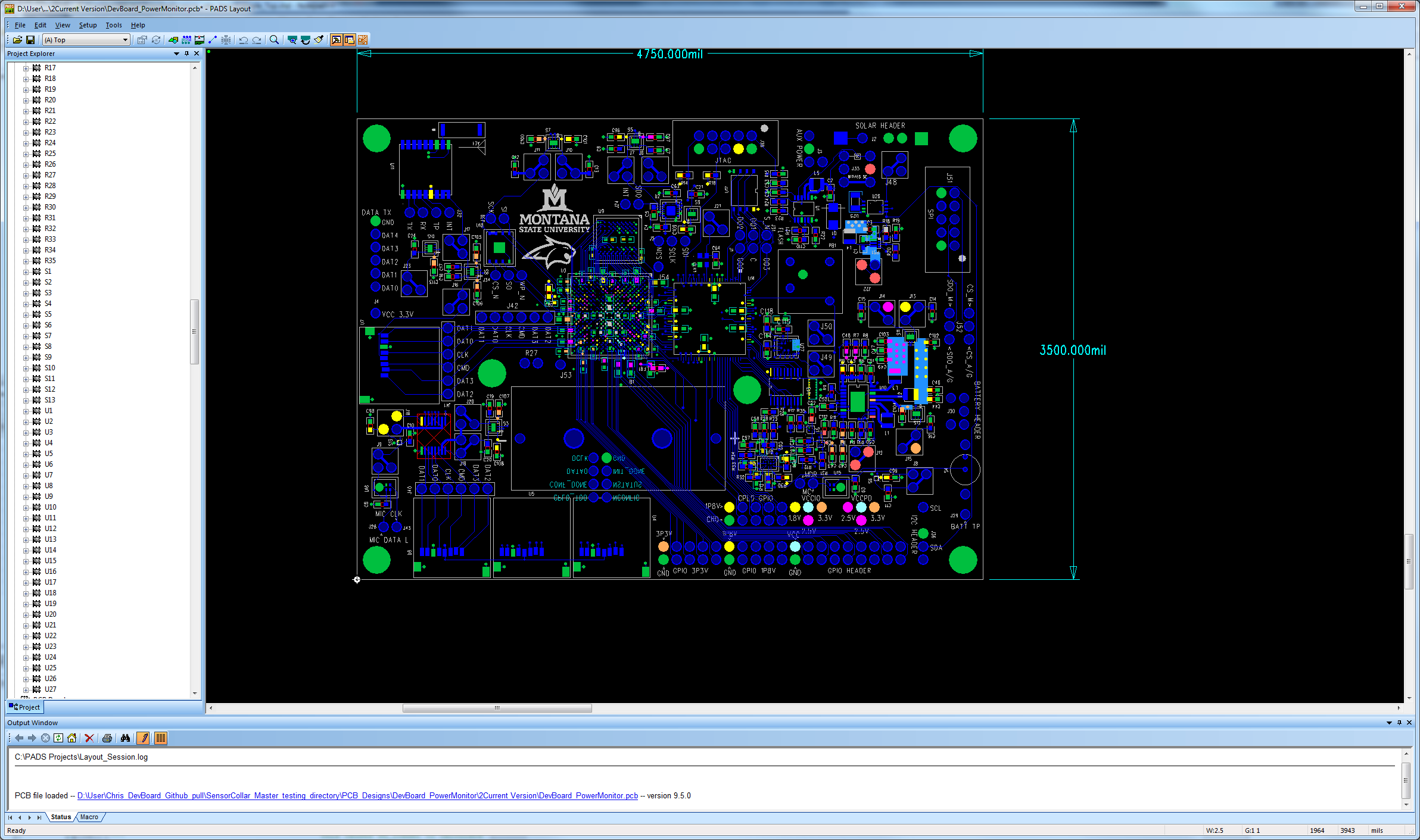
The buck boost needs >2.5V in. I recommend ~3.3V. However don’t exceed chip input max 5.5V. This configuration routes power onto power main and directly to the buck-boost regulator, bypassing the solar controller.

**3A)Programming the CPLD/FLASH**

Due to the FPGA now routing its TDI to TDO when it is shut off, the JTAG chain is broken by default. To fix this two things need to be done

1) Jump CPLD\_TDO to TDO of the JTAG header.





2) Get 1.8V Power onto the JTAG VCC line.

I recommend jumping J13 to the CPLD GPIO 1.8V (Located near the bottom of the board.)

You should now be able to see the CPLD on the JTAG chain in Quartus when a USB Blaster is connected to the JTAG header. The ribbon cable lays away from the power board on the USB Blaster. Now we will load a CPLD image which will allow us to see and load the FLASH with an FPGA image.

**3B) Programming the CPLD/FLASH**

Source\_Code\DevBoard\_PowerMonitor\CPLD\_INIT

Used for loading flash with an FPGA image.

Before opening the Quartus project. Copy DevBoard\_PowerMonitor\_CPLDInit\_IO.qsf and rename to DevBoard\_PowerMonitor\_CPLDInit.qsf

A note on the QSF files. The 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.

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

DevBoard\_PowerMonitorCPLDInit\_TopLevel.vhd

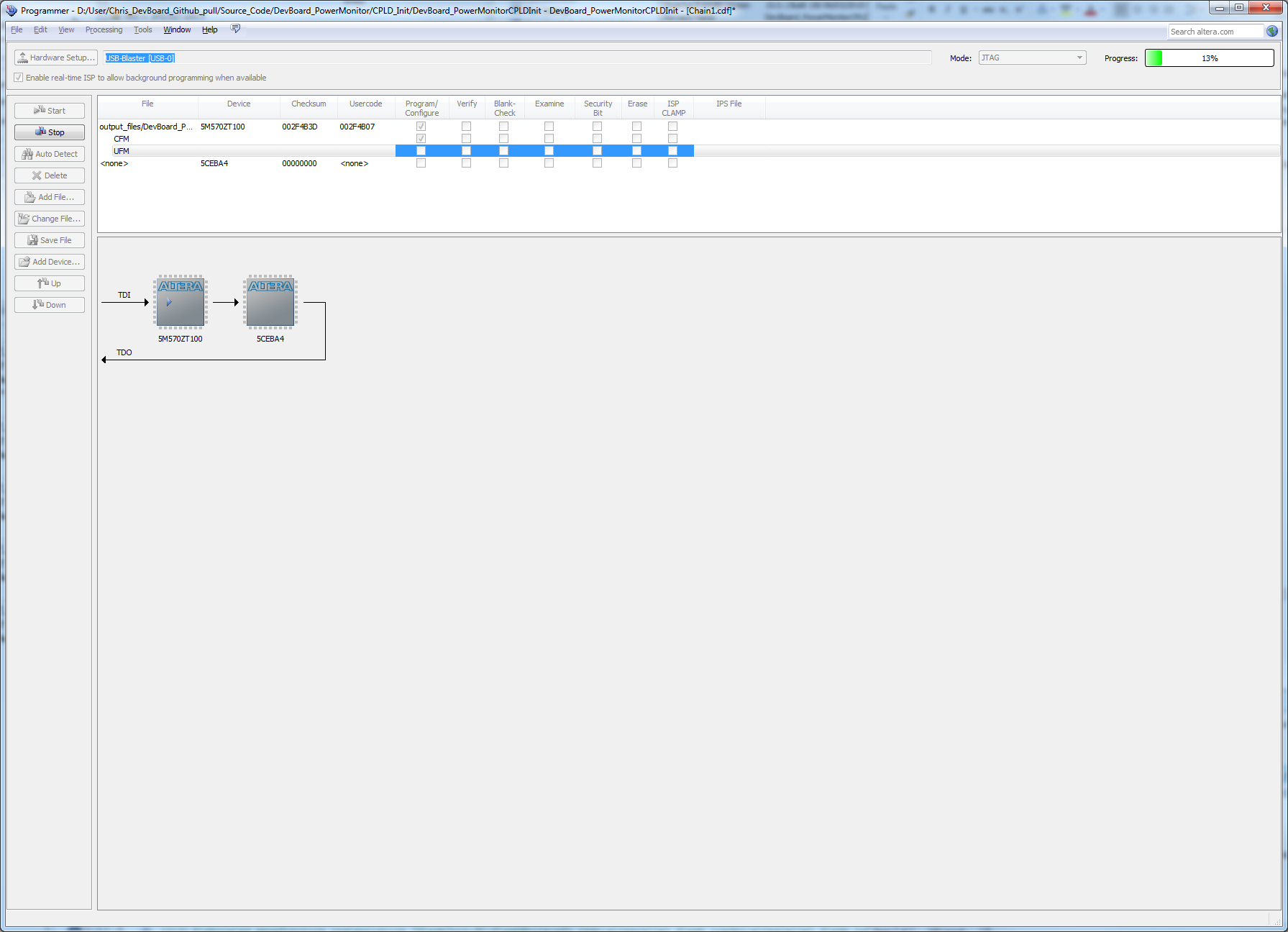
Several Project wide .TCL scripts are needed. These include at\_compile\_start, sdc\_values, set\_vhdl\_constants.

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.

Open the programmer and program the POF to the CPLD. The CPLD is the first device on the JTAG chain. Its device identifier is 5M570ZT100. 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. Enable 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.



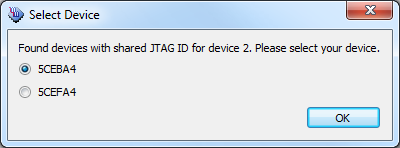
Reboot the board after flashing the CPLD. The new CPLD image is loaded after a reboot of the device. Remove and reapply power to the board.

Remove the JTAG CPLD TDO jumper. It is no longer needed, as the FPGA is turned on now. You can also remove the 1.8V jumped from the CPLD GPIO bank. The switch controlling the FPGA 1.8V rail is now turned on as well.

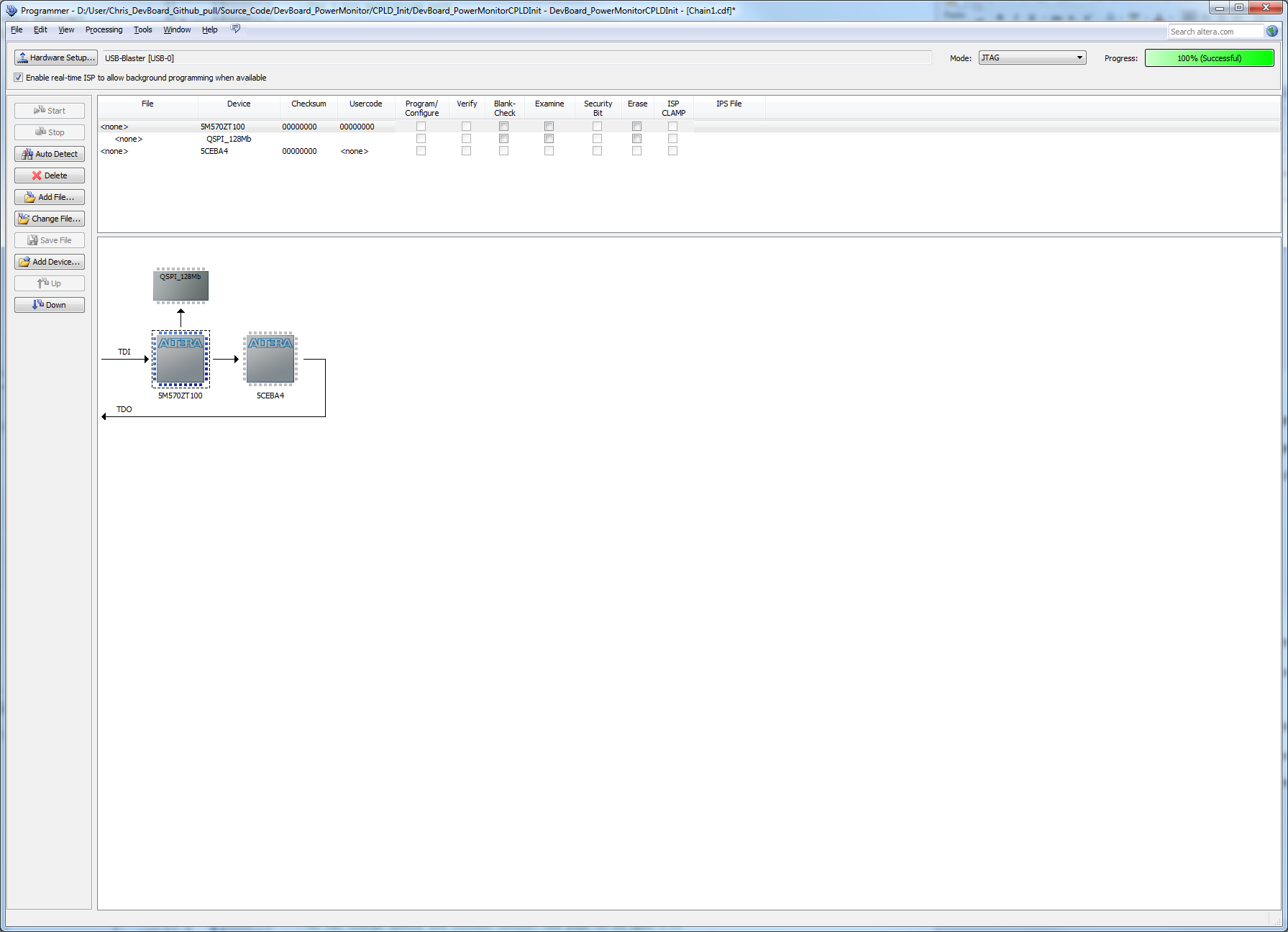
Now we can program the flash with an FPGA image.

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:



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.

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

This will involve:

Source\_Code\DevBoard\_PowerMonitor\FPGA\_INIT

Bare bones FPGA project for loading to flash.

Copy DevBoard\_PowerMonitorFPGA\_IO.qsf and rename DevBoard\_PowerMonitorFPGA.qsf

Include files:

DevBoard\_PowerMonitorFPGA\_TopLevel.vhd

sdc\_values.tcl

Copy the .tcl scripts from:

QuartusII

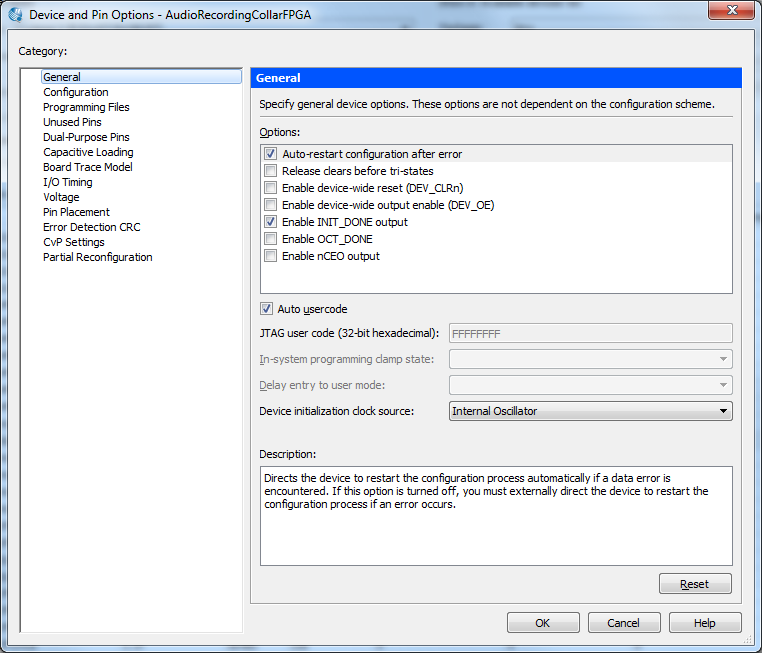
First a bare-bones bootable image of the FPGA must be compiled. The FPGA\_INIT directory exists for this purpose. It does not include any systems or the sdc timing framework. It simply displays a counter on a GPIO pin. This is a good way to check that your FPGA has booted and it booted with the image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

When inserting an FPGA POF into flash, that image 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

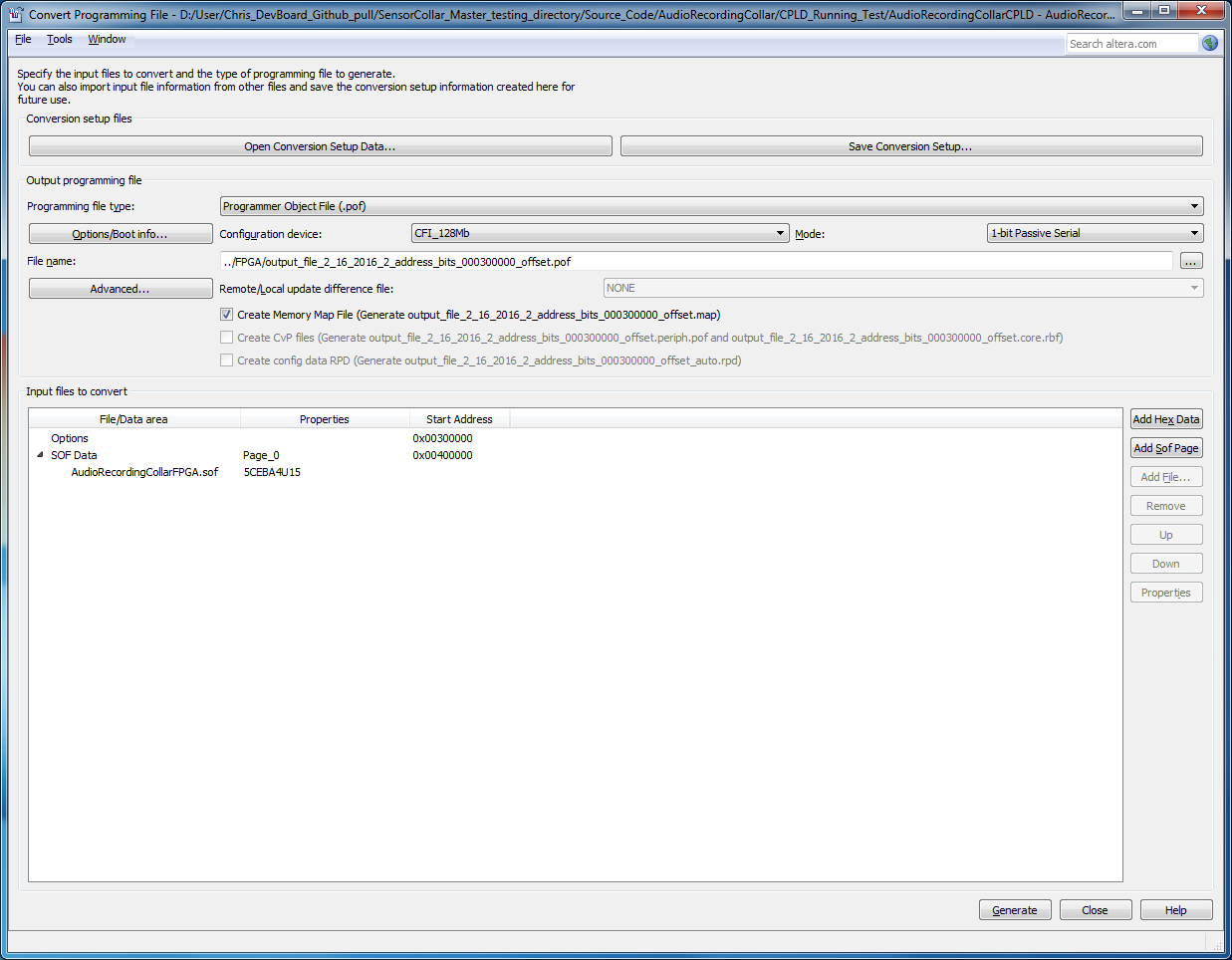
set\_global\_assignment -name DEVICE\_INITIALIZATION\_CLOCK INIT\_DCLK

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.

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.

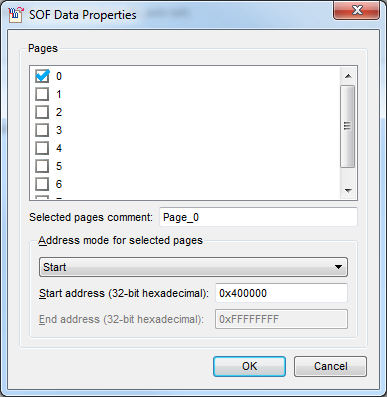
Note the odd choice of CFI\_128Mb, when the actual chip is a QSPI\_128Mb. This is explained by Altera here: <https://www.altera.com/support/support-resources/knowledge-base/solutions/rd05082012_592.html>

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.

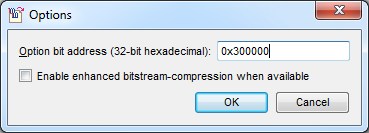
Click the SOF DATA and hit properties off to the right.



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.

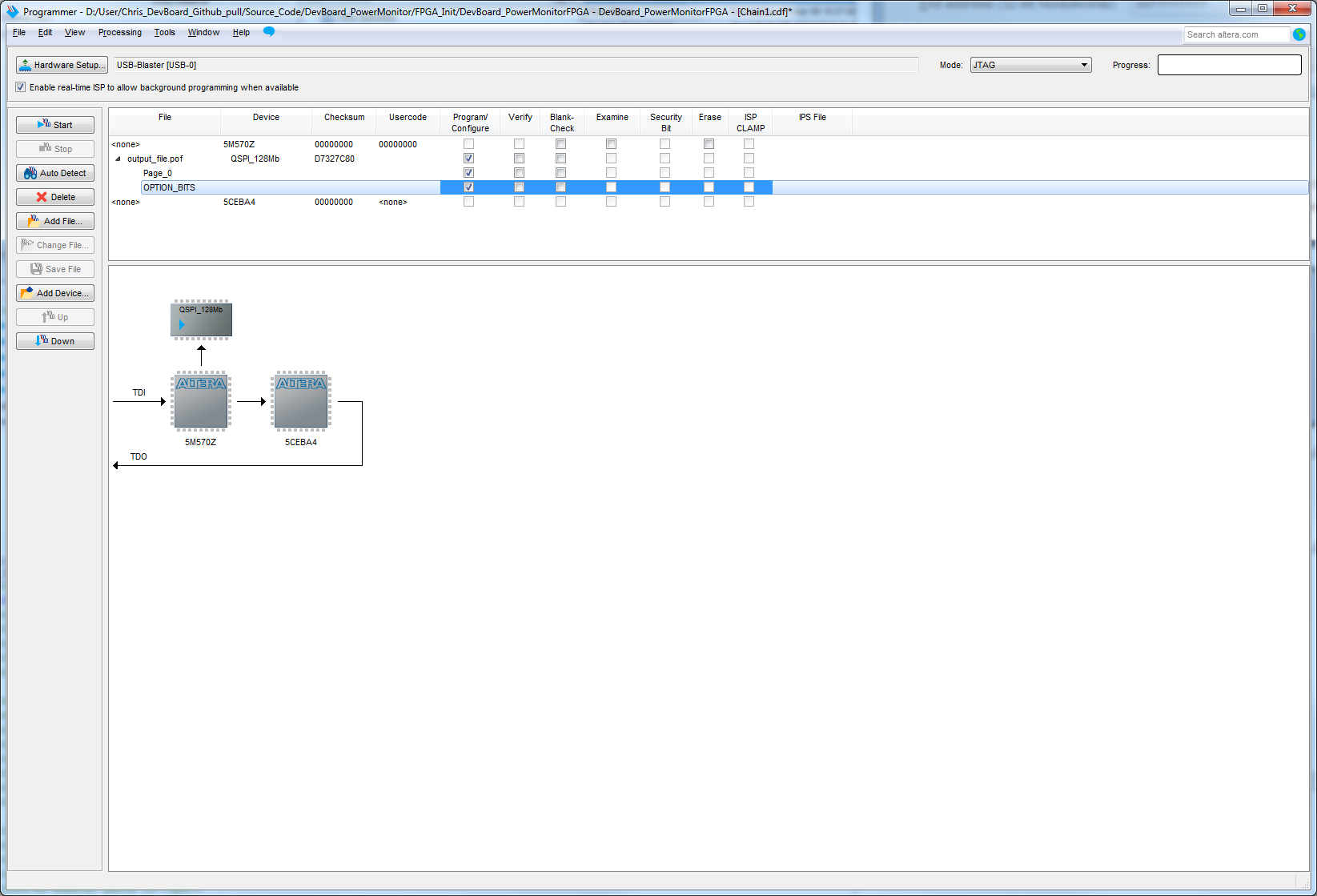
Now click the “Options/Boot Info…” in the main convert screen.



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.

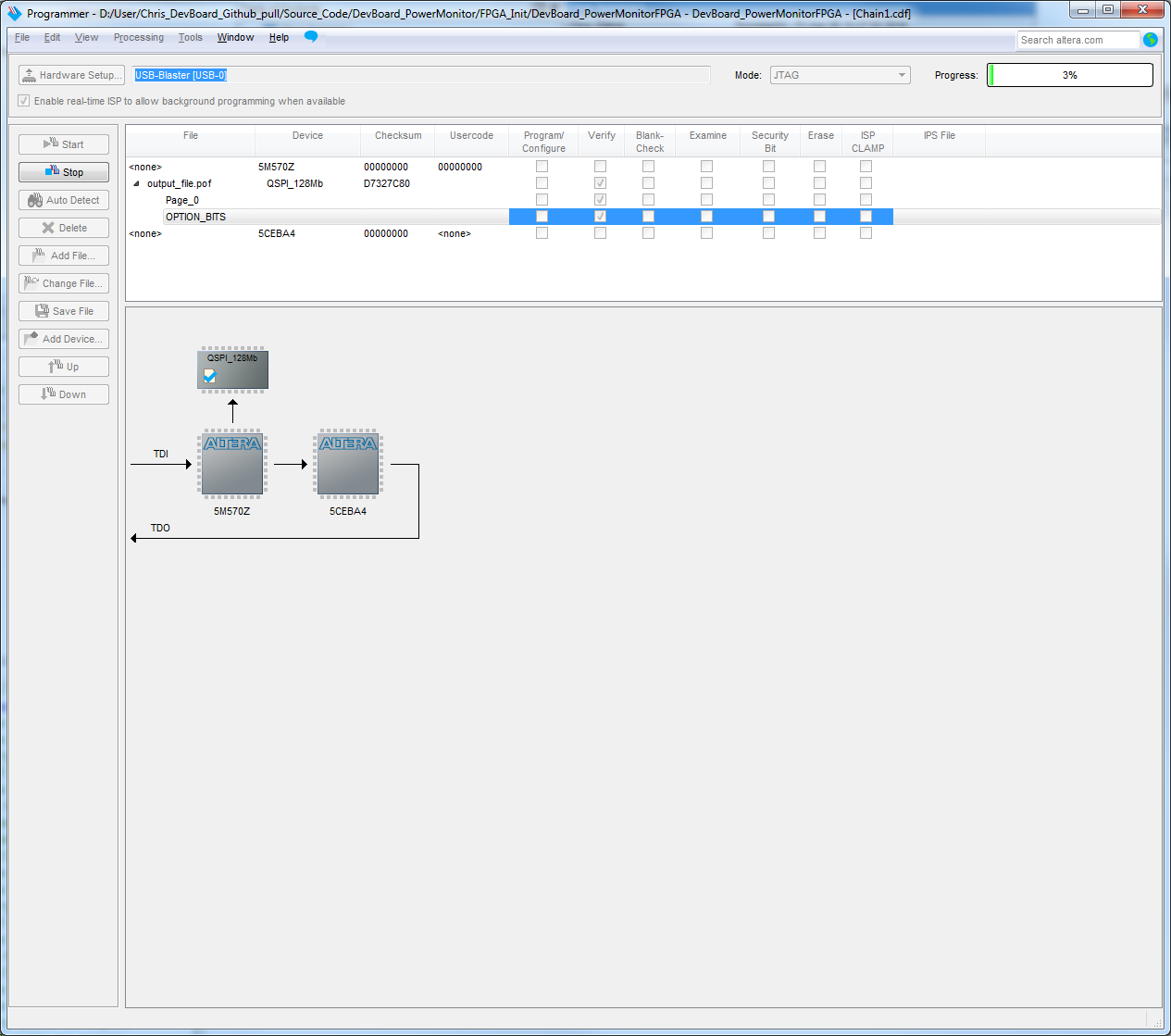
Hit Generate in the main window. The POF is now ready to be loaded to flash.

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



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

Now 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.



**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 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\DevBoard\_PowerMonitor\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.

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 VHDL\_FILE ../../MainCollar/PowerController/PowerController.vhd

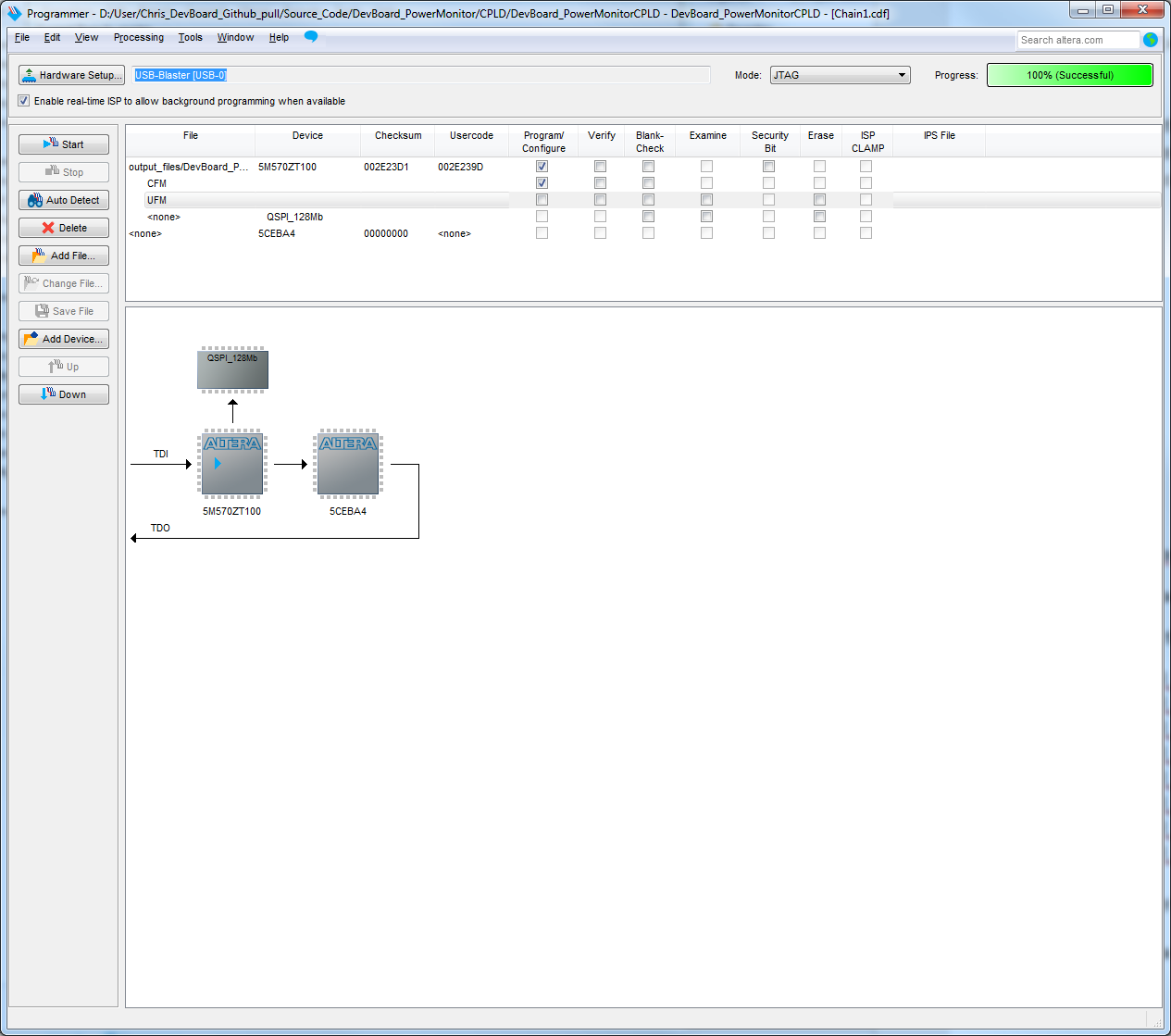
set\_global\_assignment -name QIP\_FILE ../../MainCollar/PowerController/InternalFlash.qip

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.

The project should compile now.

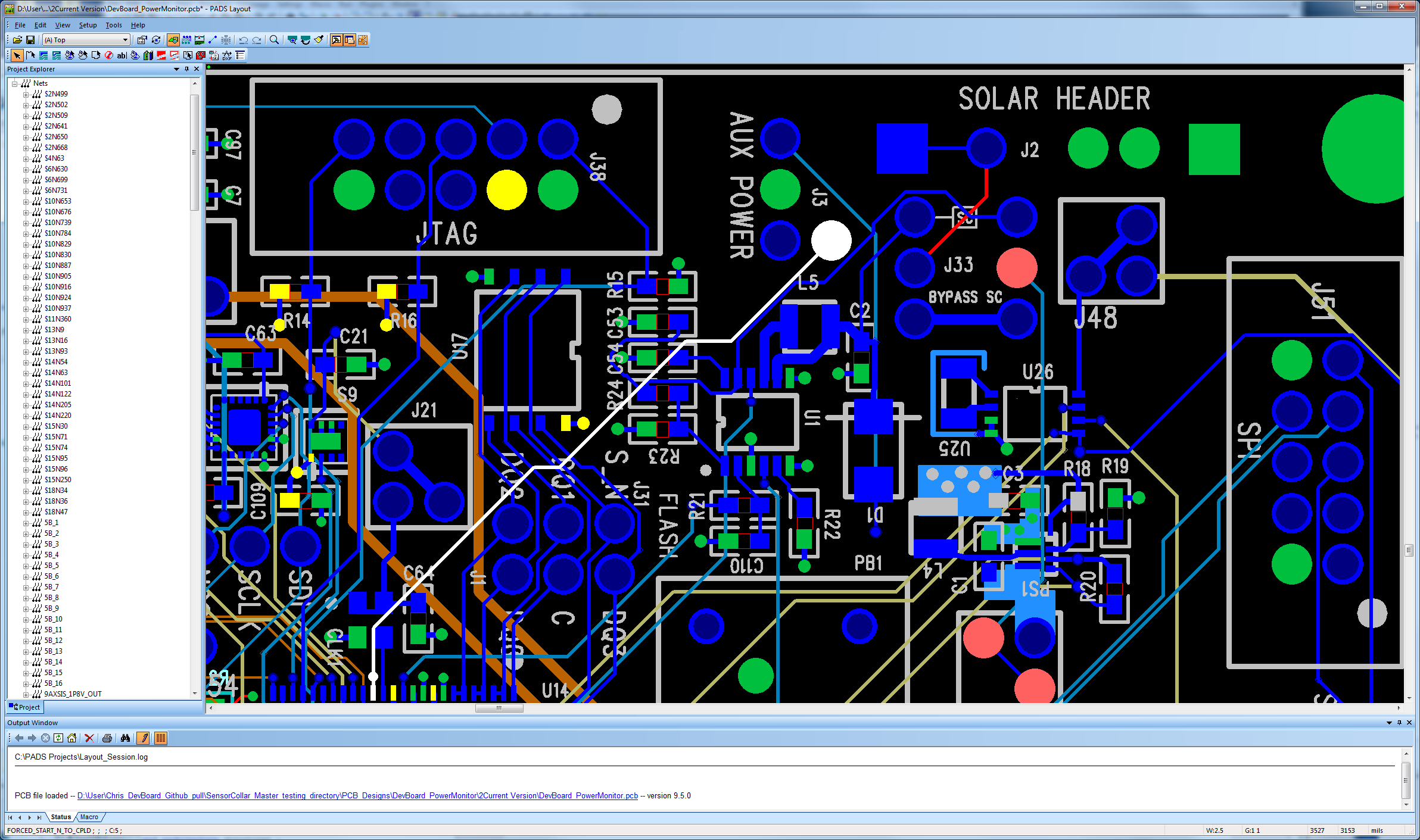
In the Quartus programmer, the generated POF can now be programmed.



The POF is programmed to the CPLD (5M570ZT100) by checking the Program/Configure CFM boxes.

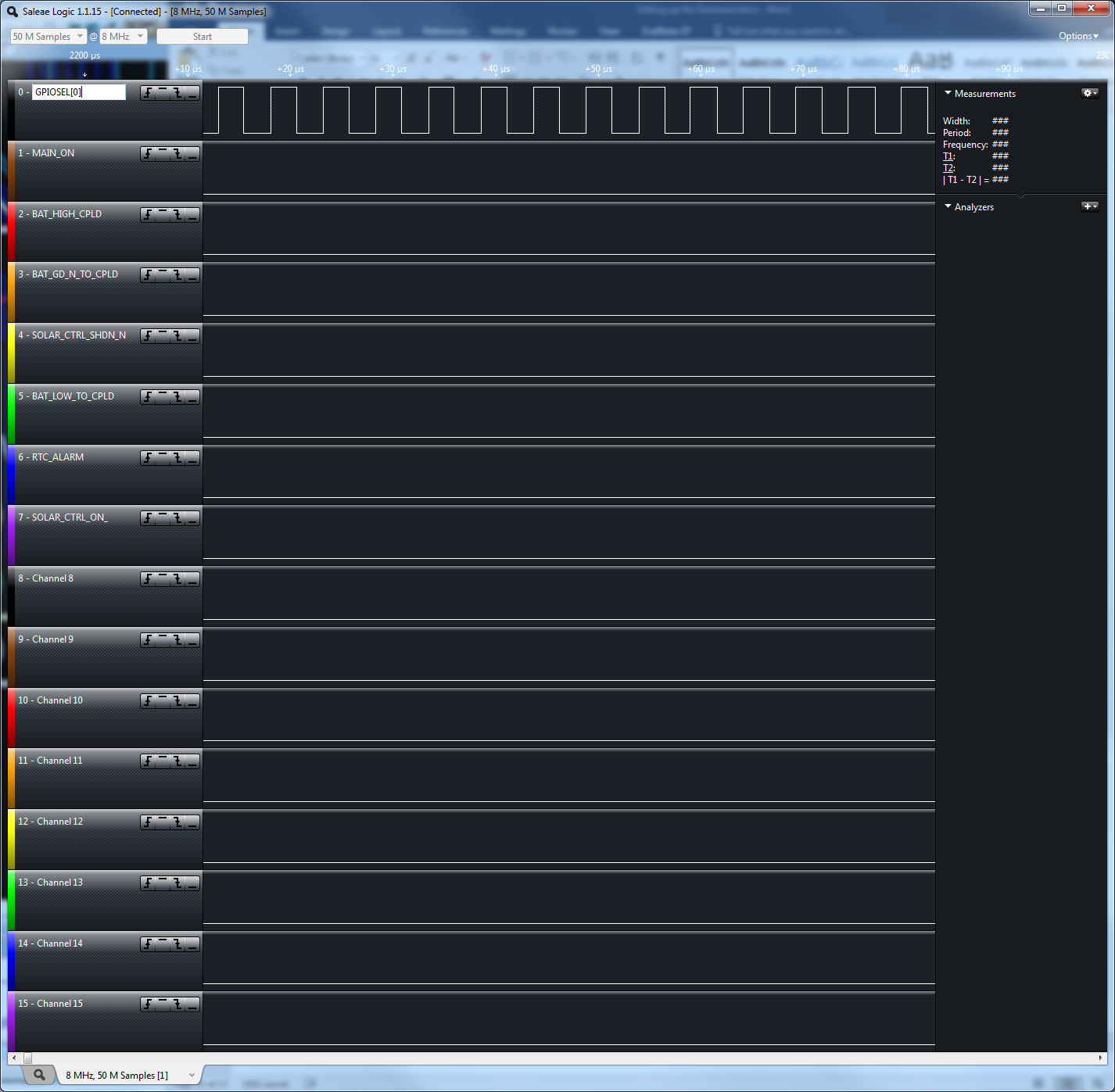
Now the CPLD needs to be rebooted. Remove and reapply power to the board.

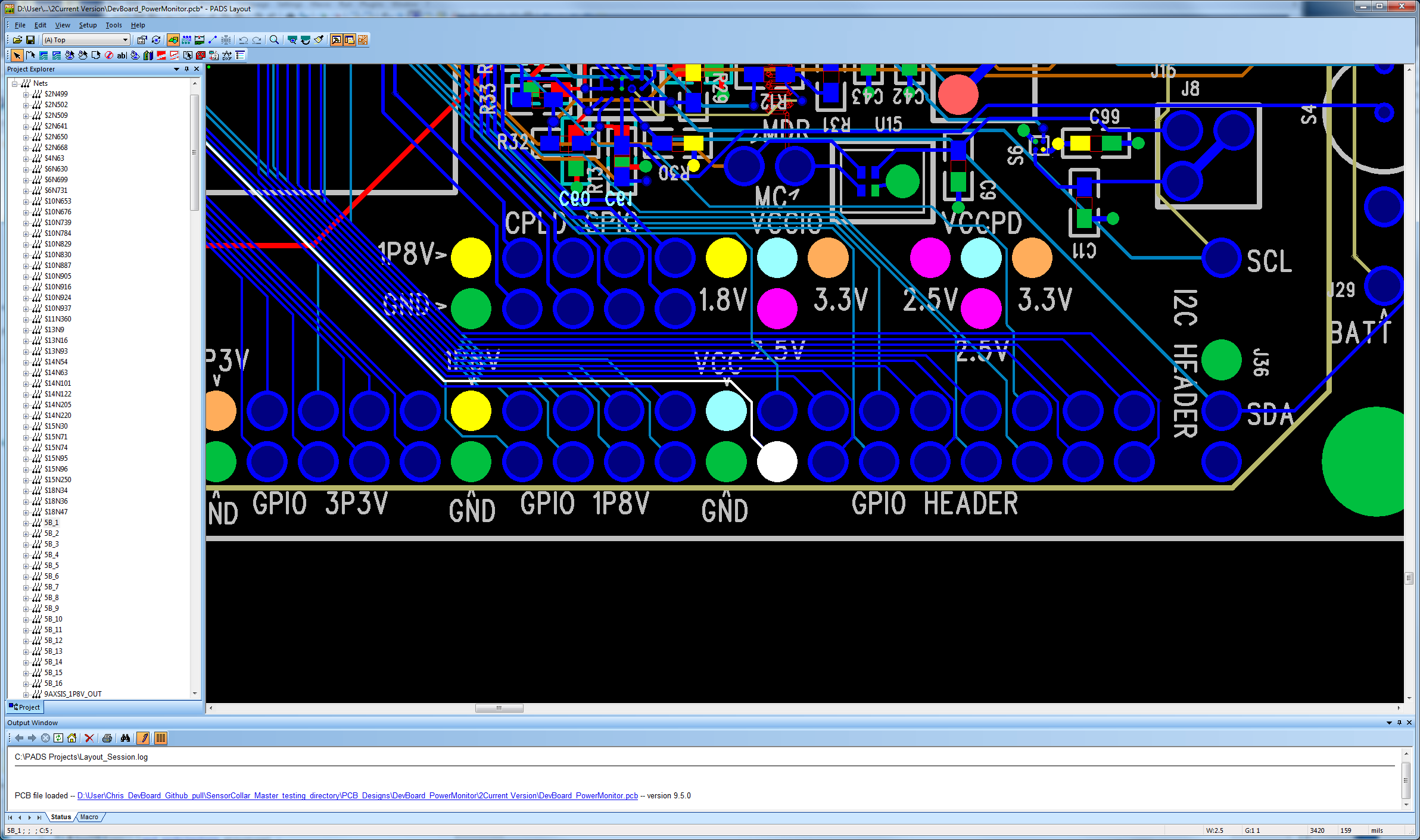
Upon restart only the CPLD will be on. A force\_startup line to the CPLD tells it to boot the FPGA. A low pulse on force\_startup can be accomplished by momentarily grounding the forced\_start\_n\_to\_cpld line.



When the CPLD alone is running current usage at the power supply should be ~.01 amps. When the FPGA\_INIT image is booted, the current will rise to ~.09 amps.

One can check that the image booted okay, by checking for the programmed counter on the GPIO\_SEL(0) line.





The system is now ready to be programmed with another FPGA image of choice. 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 (coming soon) 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.

**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