SAN DIEGO STATE
UNIVERSITY

Tutorial
Xilinx Virtex-5 FPGA ML506 Edition

Department of Electrical and Computer Engineering
Real-Time DSP and FPGA Development Lab

Mark Sison Manalo and Ashkan Ashrafi

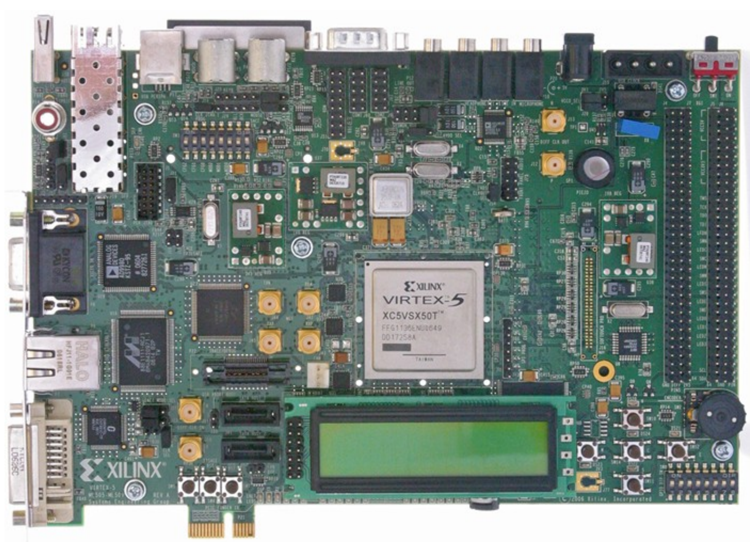


Table of Contents

Blank Page

Overview

About this tutorial

This tutorial will help you familiarize yourself with Xilinx's XtremeDSP Development Platform – Virtex-5 FPGA ML506 Edition. Familiarity of C, VHDL and MATLAB/Simulink would help but it is not required. The ML505/506/507 are the same boards, only the FPGA is different so this tutorial will apply to all of them. Parts of this tutorial are taken from Xilinx tutorials available here:

http://www.xilinx.com/ml506

http://www.xilinx.com/support/documentation/ml506.htm

http://www.xilinx.com/products/boards/ml506/reference_designs.htm

In addition to the Xilinx documents, the following websites and books were also referenced:

http://www.fpgadeveloper.com/

http://myfpgablog.blogspot.com/2009/12/sysgen-create-new-hwcosim-target-with.html

http://academic.csuohio.edu/chu_p/rtl/fpga_vhdl.html

Software needed

- Xilinx ISE Design Suite (version 12.1 and 13.2 is used here)
- Xilinx Platform Studio
- Xilinx Software Development Kit
- Xilinx System Generator
- MATLAB/SIMULINK
- Hyper Terminal / Tera Term / Putty or similar software (search it on Google and install)
- msdosfs.exe with Windows XP (used for reformatting CF in case of file system corruption)

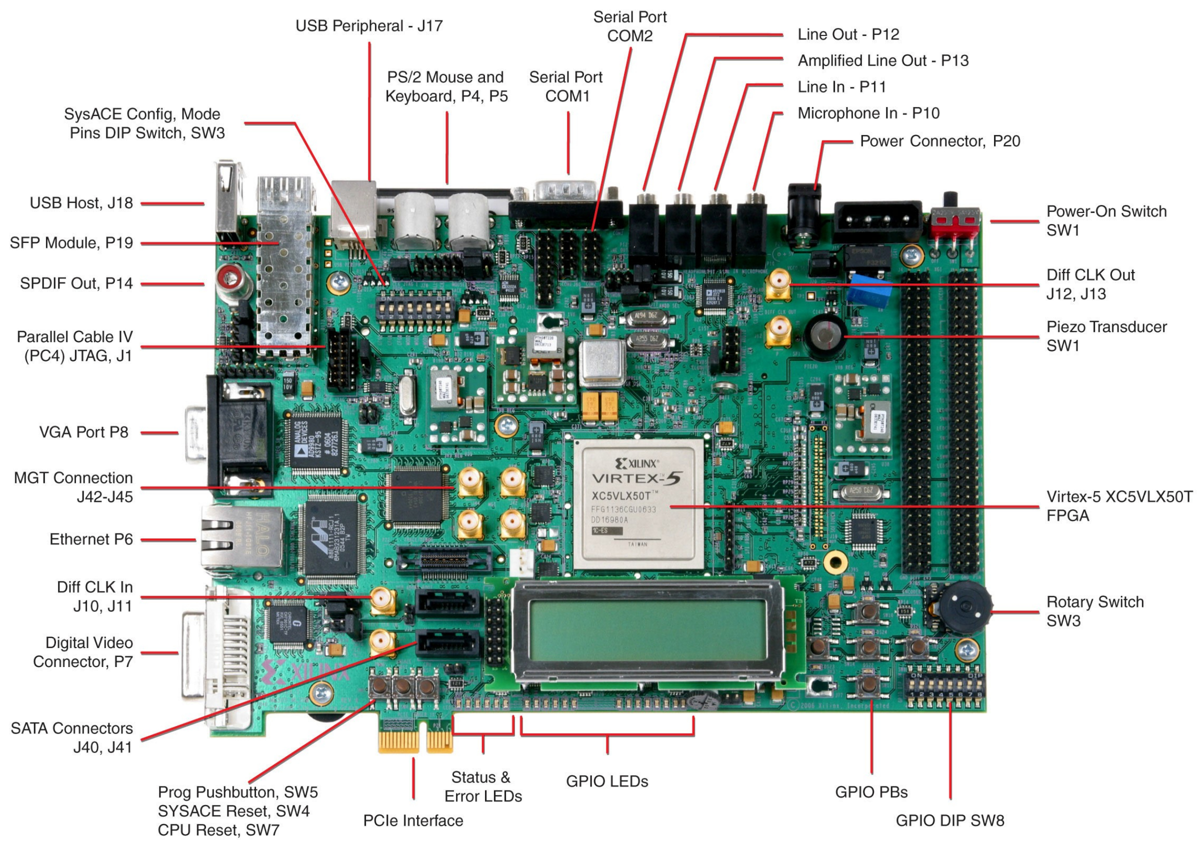
Hardware needed

- DVI cable or VGA cable with DVI adapter
- RS232 cable for UART
- Xilinx Platform USB cable
- Speakers or headphones
- Compact Flash reader (to load programs into the compact flash)

Setting up the ML506 board

The ML506

Here is a picture of the ML506 evaluation board with its components labeled.

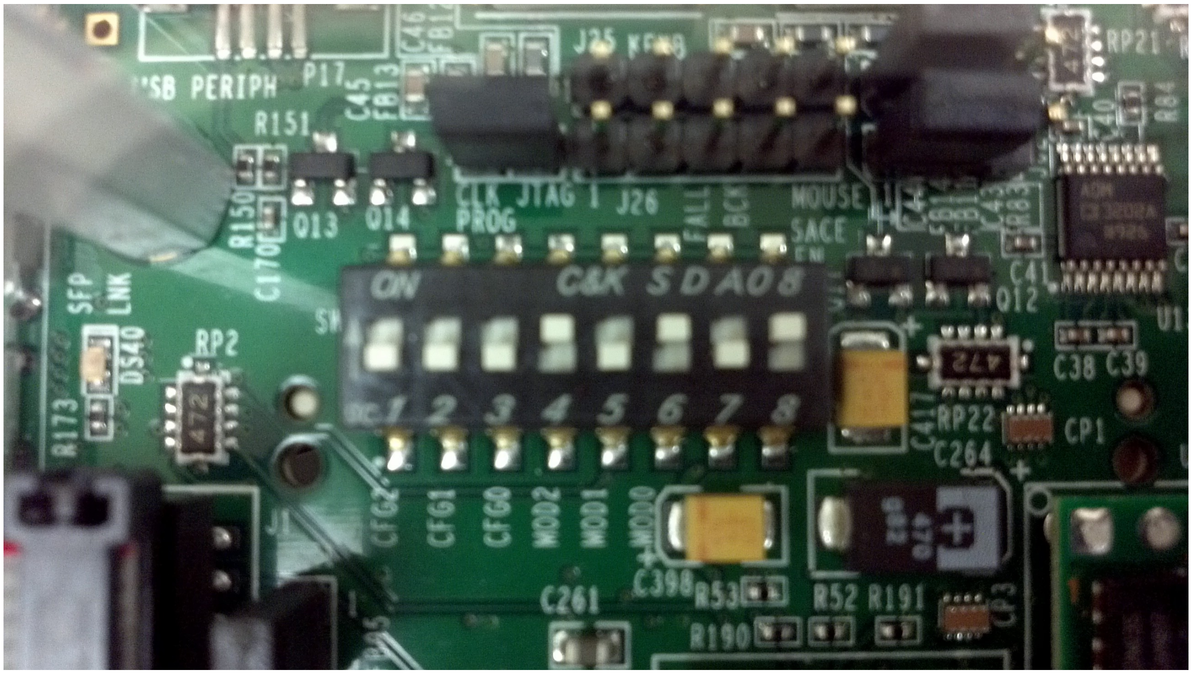


Connecting the board

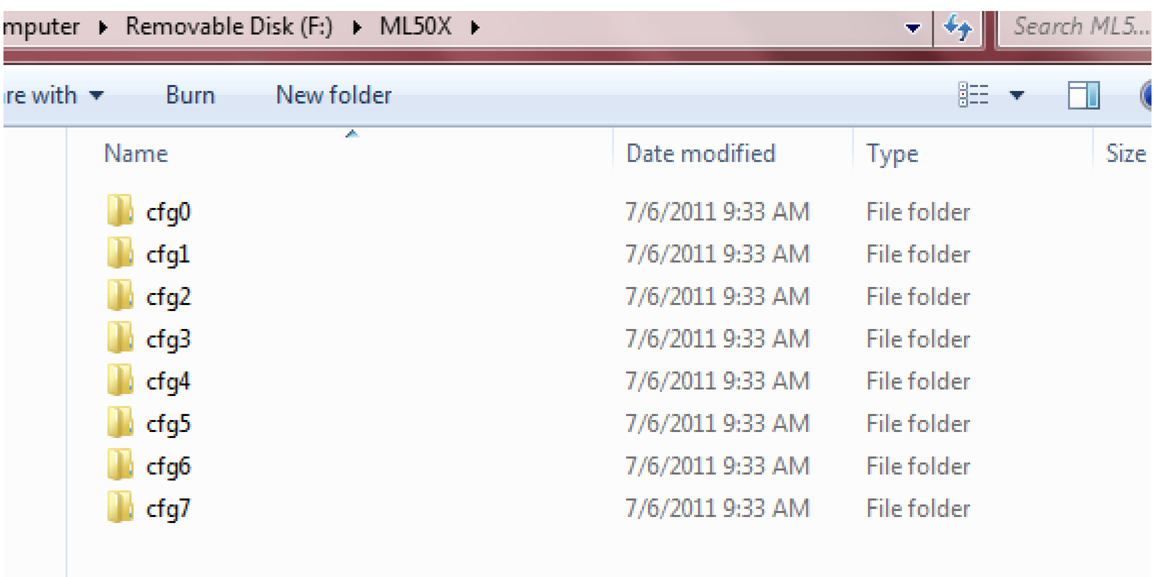
The board should already be connected in the FPGA lab. If not, please refer to the Xilinx document titled: ml505_overview_setup.pdf

For convenience, I will summarize it here.

- Set the SW3 switches to 00010101 as shown in the picture:



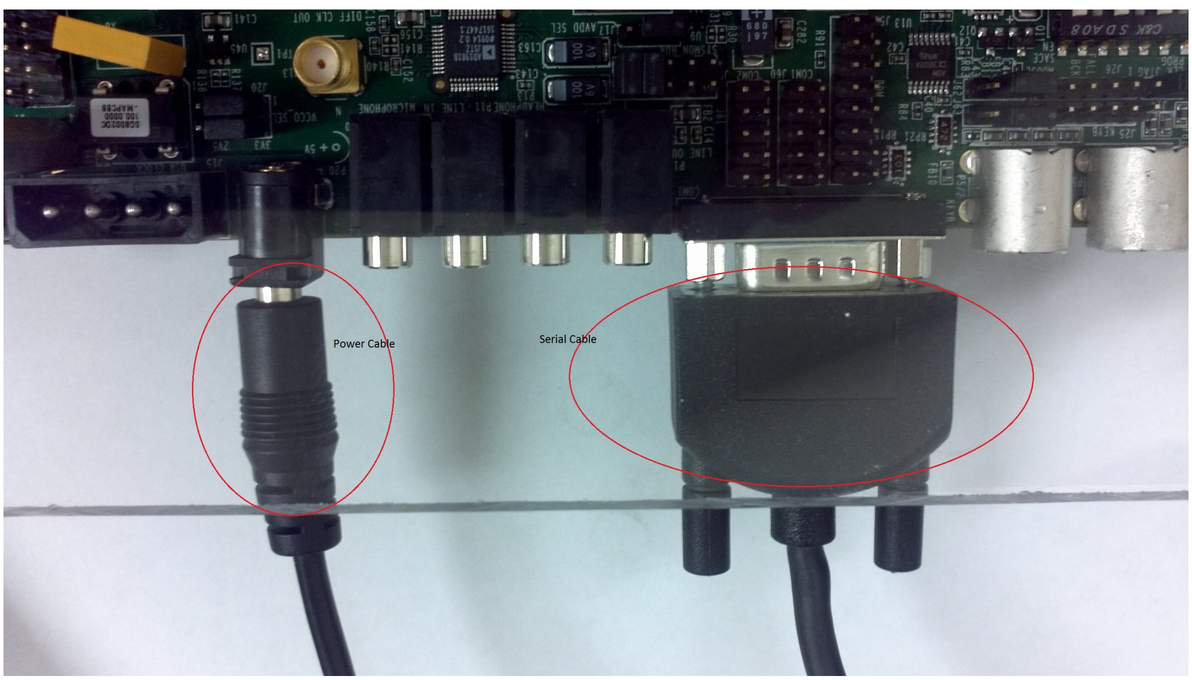
Bits 4-8 (10101) means that on power up, the ML506 board will program the FPGA with the ACE file pointed to by bits 1-3 (in this case: 000). These ACE files are located on the compact flash. 000 means that the ACE file located in cfg0 of the compact flash will be programed into the FPGA. The factory compact flash files has the cfg0 folder containing a system_bootload .ace file. So if you want to load your own ACE file into folder cfg6 (say). Then you would set SW3 to read 110 10101 if you want cfg7 then SW3 = 111 10101 etc... (Search http://www.fpgadeveloper.com/ for a tutorial on creating your own ACE file). But for now lets stick the with factory default files.



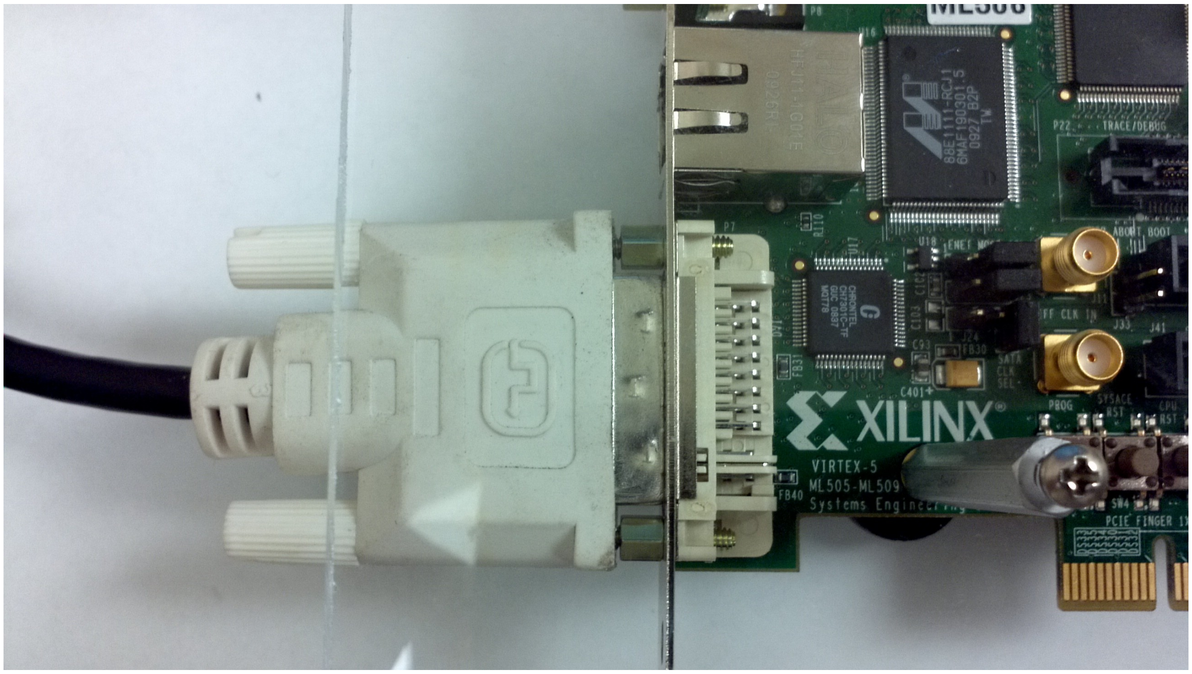
-

- Set SW6 (on the back side of the board) to 11001010
- Connect the Xilinx Platform Cable to PC4 JTAG

- Connect the Serial cable (RS232) to the PC either directly or with a serial to USB adapter
- Connect the Power cable to the board

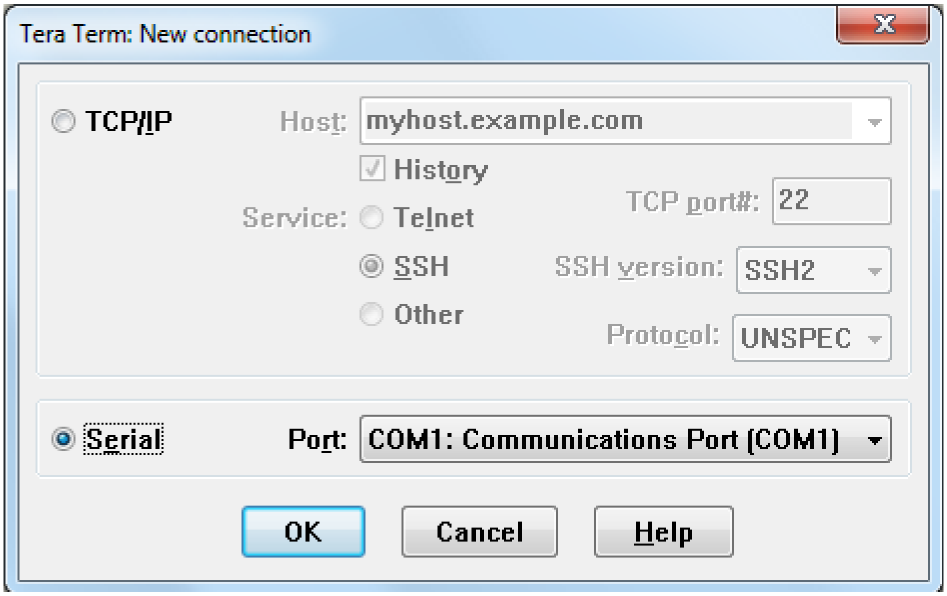


• Connect the DVI cable from the ML506 to a monitor.

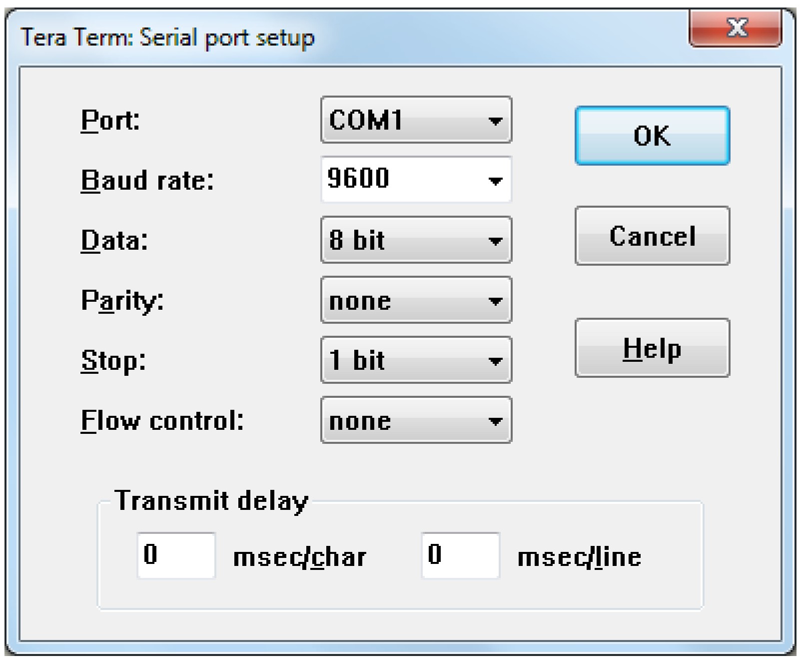


• Insert the compact flash into the board.

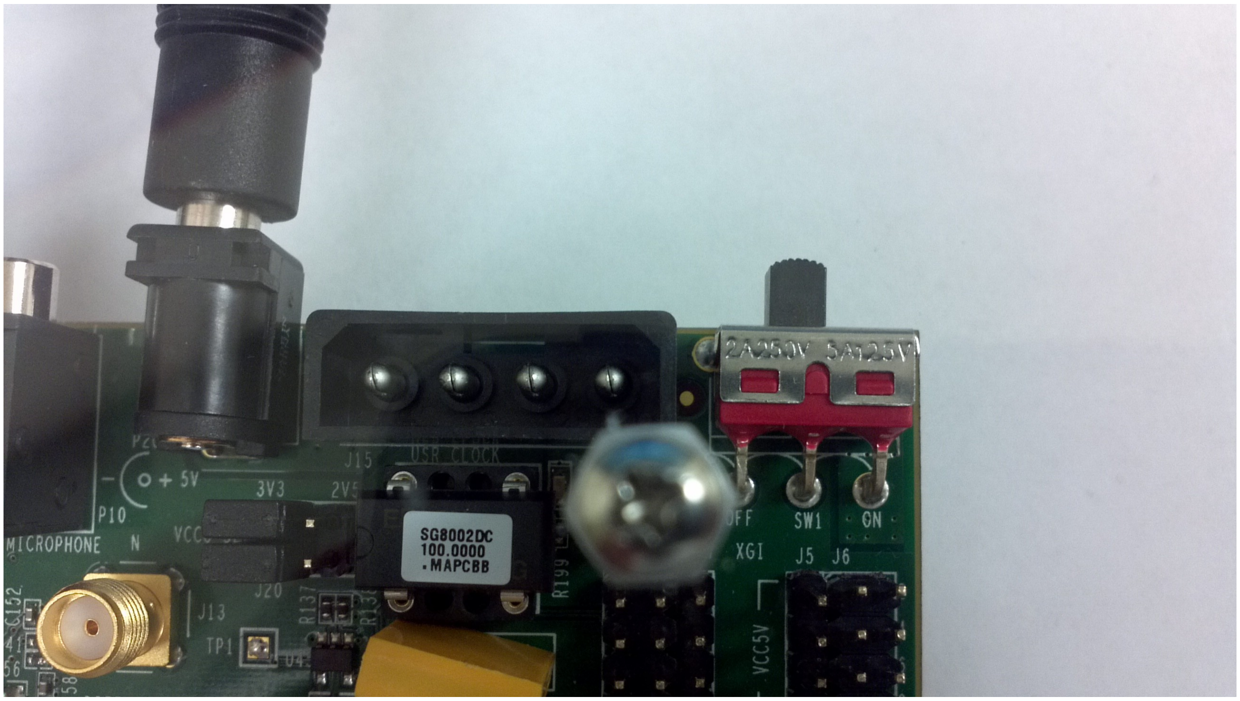
• Install a terminal program such as Tera Term Hyper Terminal or Putty and set it up with the following settings. (On my computer the Serial cable is connected to COM1, yours may differ)



- Go to setup → serial port, and use these settings.



Now your board should be ready to go. Don't forget to power it on when your ready.



Preparing the Compact Flash (Only when necessary)

The compact flash should already be formatted correctly as FAT12 or FAT16. After connecting the board insert the compact flash and power up the board. If the CF is not formatted correctly the LED labeled: SACE ERR would be a solid red color as shown in illustration 1.

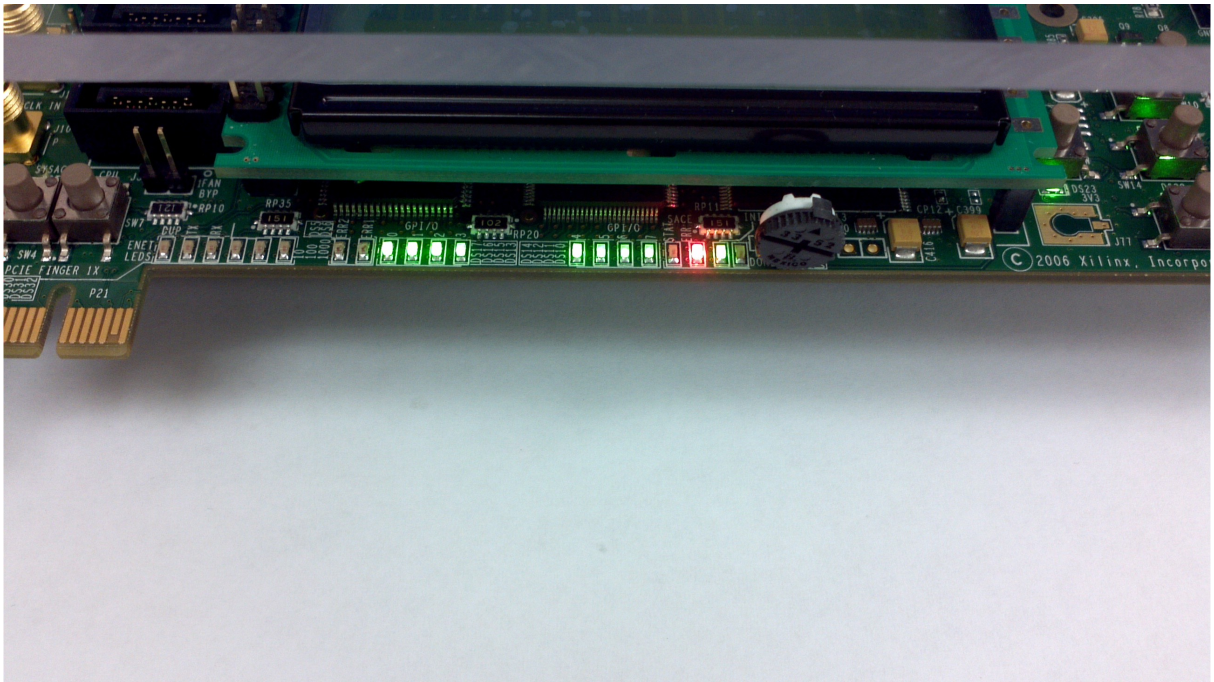


Illustration 1: SACE ERR, CF card is not being read correctly

In this case, one would need to reformat the CF and place the factory default files back into it.

Xilinx provides a tutorial on how to do this:

http://www.xilinx.com/products/boards/ml506/ml506_12.1/images.htm

However, my attempts at their tutorial using the dd tool did not work. You can try for yourself if you want. (For the records, I tried this using Windows 7 and I am getting either a non-existent file error or a permissions error, even with administrative privileges)

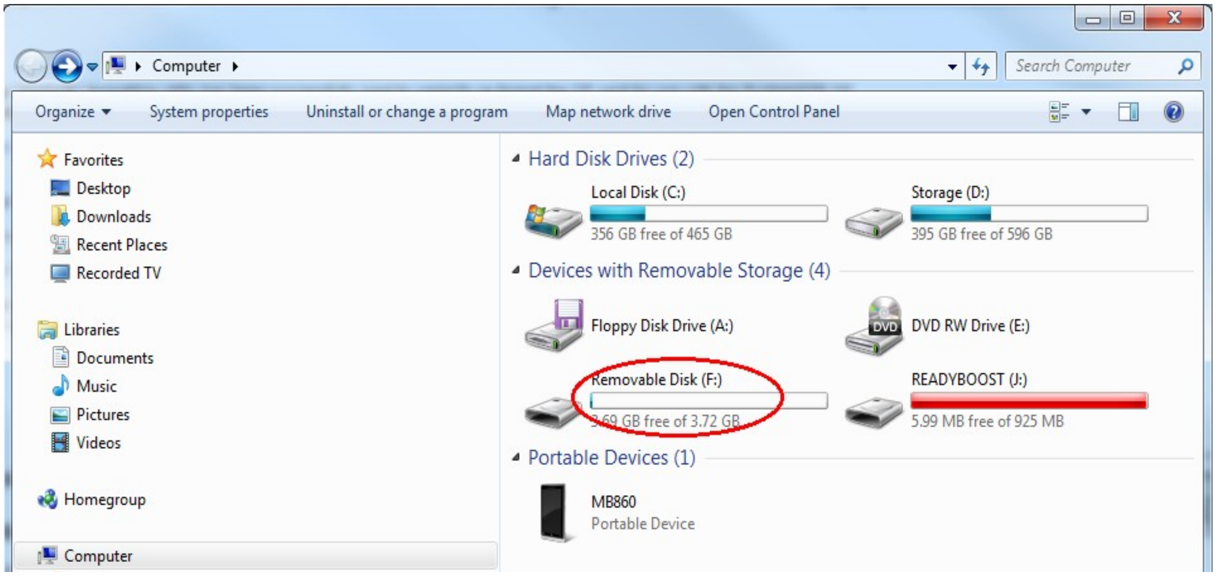
I managed to reformat the CF by another method, described below. (with the help of this: http://www.xilinx.com/support/answers/14456.htm)

If you are going to reformat the CF, follow the directions carefully. Failure to do so might cause the primary hard drive to be erased .

The following items are required:

- Windows XP (I have not confirmed that it works with Windows 7, you can try though)
- mkdosfs.exe (http://www.plunder.com/MKDOSFS-for-Windows-Format-drives-larger-than-32GB-as-FAT-or-FAT32-download-f738e8fe97.htm)
- Compact flash reader (order at Newegg.com, or some other electronics store)
- ML506 factory CF files, go here: http://www.xilinx.com/products/boards/ml506/ml506_12.1/images.htm

After inserting your CF card into your computer note its drive letter by clicking on Computer.



In my case the drive letter is F: , note that the System ACE controller can only support up to 2GB, the figure above is only for illustrative purposes. The 4GB CF that I have will not work.

Add mkdosfs to your system path: Click start, right-click Computer → Properties → Advanced system settings → Environment Variables. Then edit the 'Path' variable to include the location of where you downloaded mkdosfs.exe. Google how to add to system path for more info.

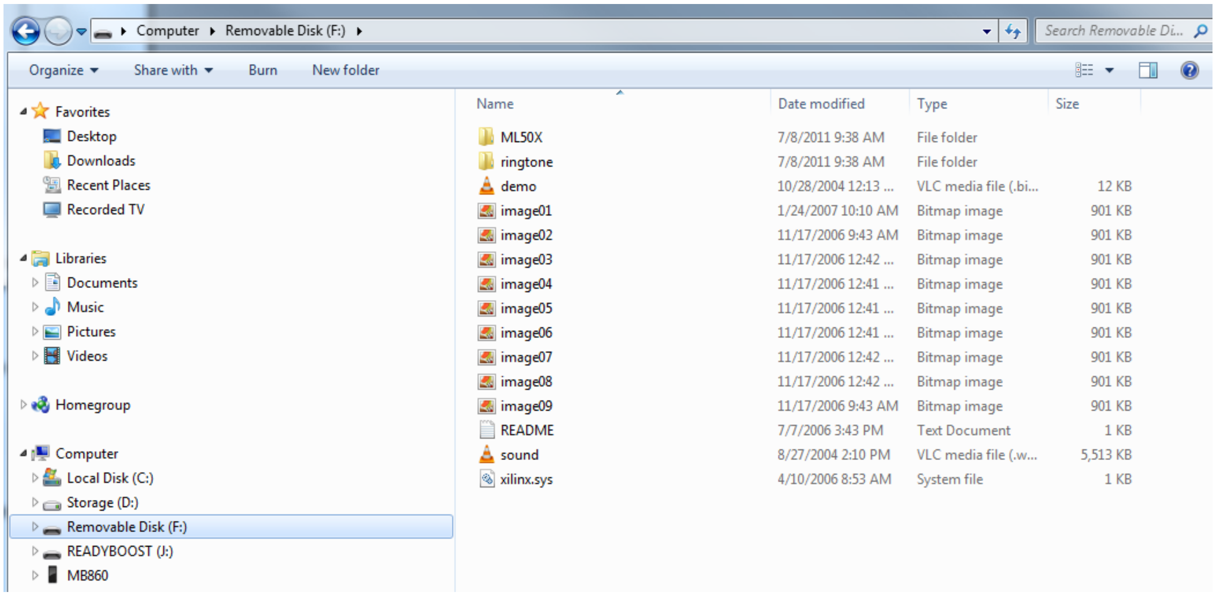
Then open up a command prompt in windows, preferably with administrative privileges (right click command prompt –> run as administrator).

Type the following command:

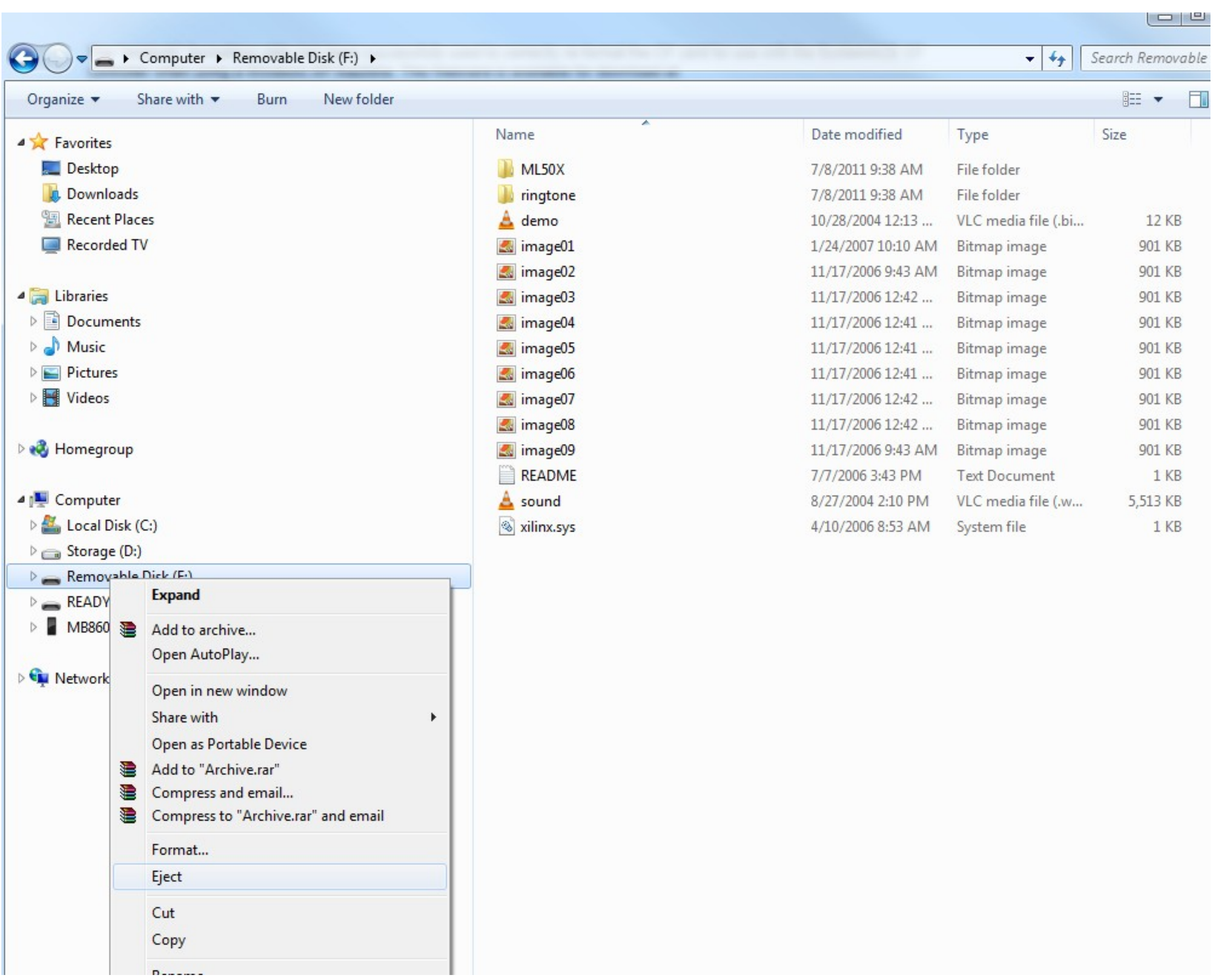
    mkdosfs -v -F 16 F:

Where 'F:' would be changed to your CF drive letter. Leave the first F as '-F'

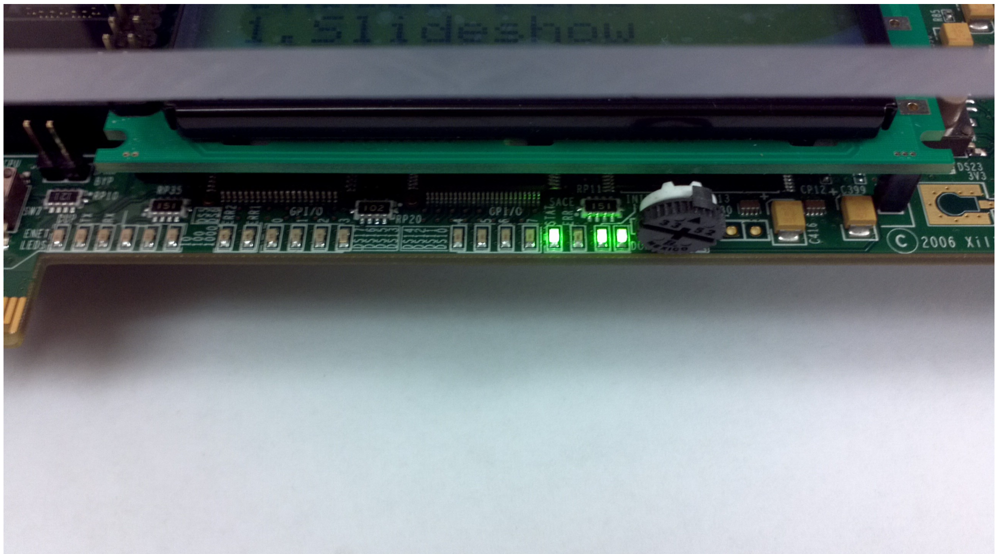
After this completes, copy and paste the CF factory files into the compact flash as shown:



Once this is complete, right-click the compact flash and eject it.



Insert the CF back into the ML506 and turn on the power. Hopefully, you will have no red SACE ERR.



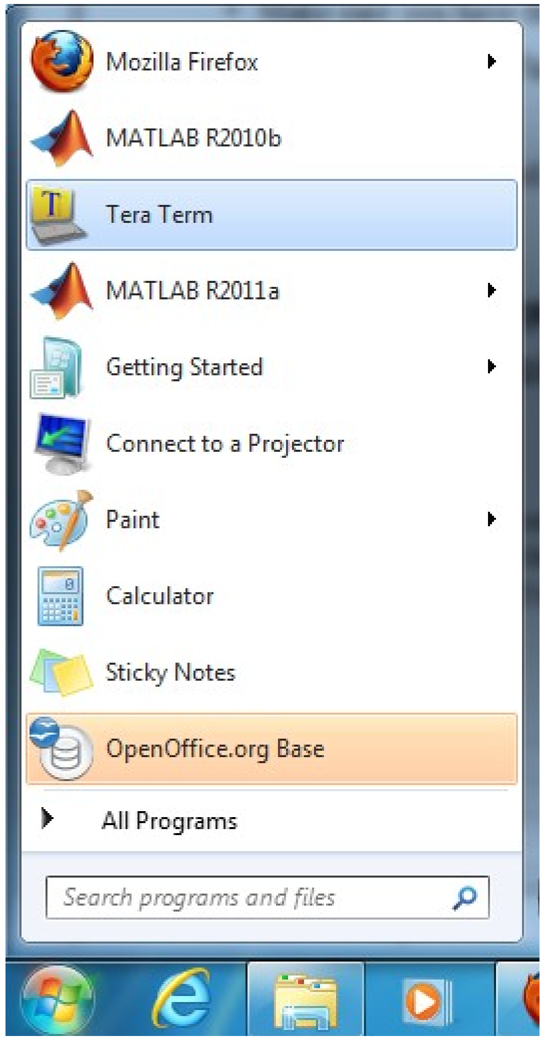
No SACE ERR

Running the Xilinx Demo

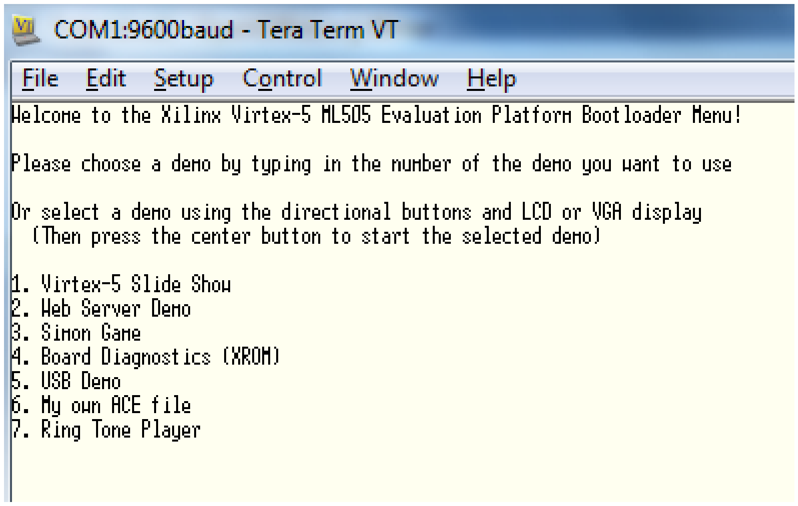
Prerequisites

- Make sure you have the board connected to the computer (see—connecting the board)
- Insert the Compact Flash with the factory files loaded into it.
- Connect some headphones or speakers to Line out.

After connecting your board. Run Tera Term/HyperTerminal/Putty with the setting described in the section 'Connecting the Board '.



Power up the board and it should start displaying text to the Tera Term window as shown below.



You can run the demo programs by pressing a number on your PC keyboard that corresponds to your demo of choice.

Also, this demo makes use of the monitor connected to your ML506 via DVI or VGA adapter. Make sure you switch your screen input (press on the buttons in front of you monitor to access the menu) to the ML506 so you can see the video output.

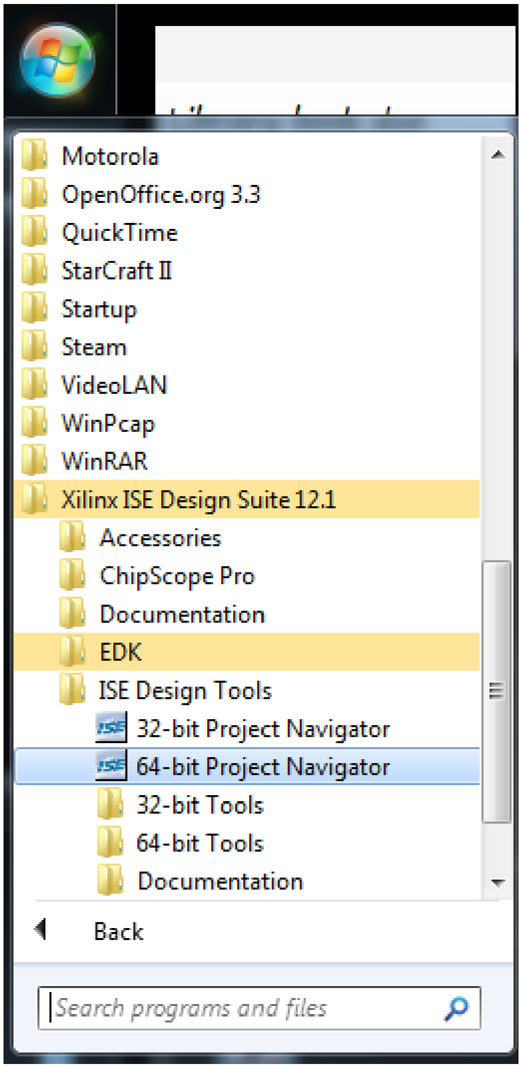
Programming the FPGA with a VHDL Design Using iMPACT – 2 bit greater than circuit

Summary

This tutorial will show you how to program the FPGA with a VHDL design created in ISE. Then we will use 4 switches and 1 led on the ML506 board to simulate the design. We will not go over the VHDL code as it is beyond the scope of this tutorial. After finishing this tutorial, you should be able to program the FPGA with your own circuit design and connect it to some GPIO (general purpose input/output) pins.

The VHDL Design

Open up Xilinx ISE.



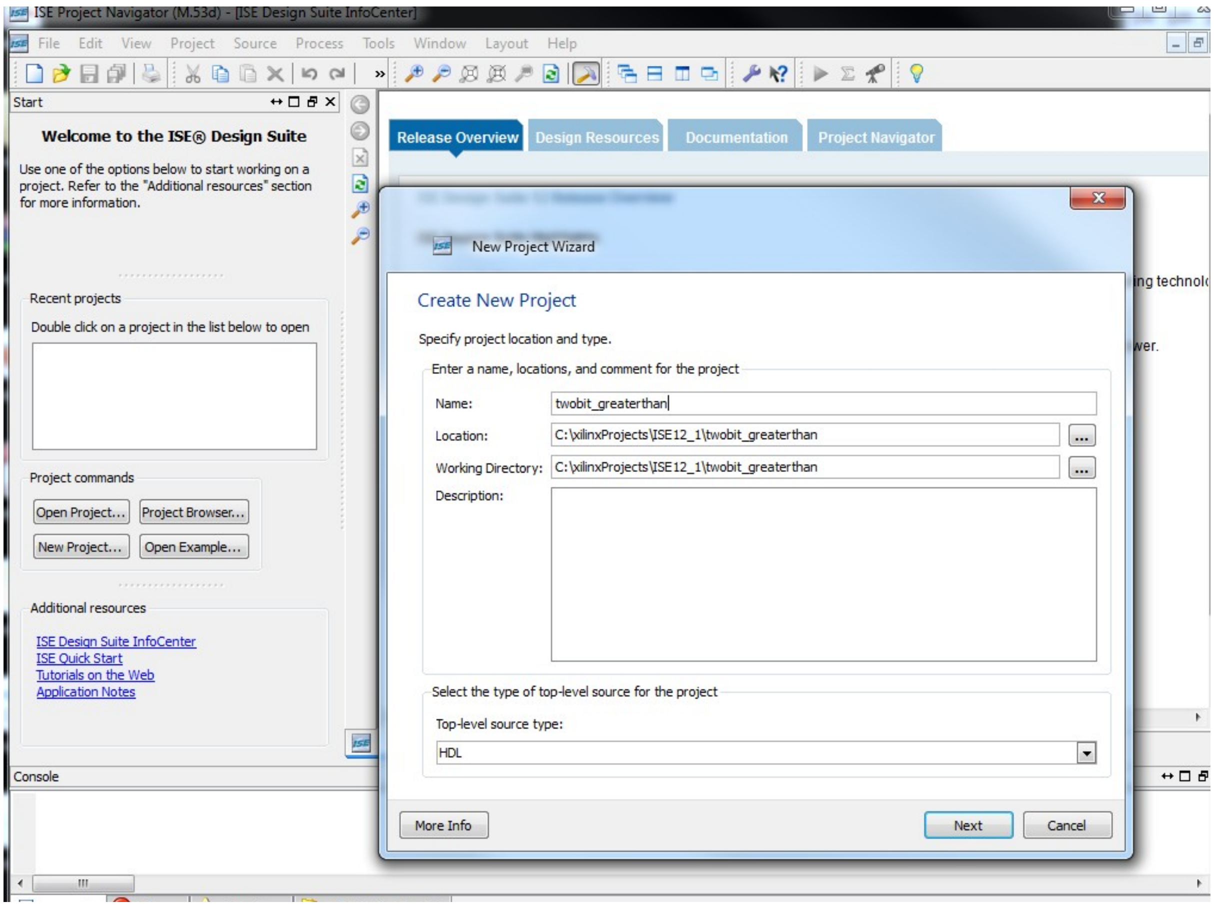
Click File--> New Project.

Set the location of your project file, to avoid any errors later on, make sure there are no spaces in the file path you choose.

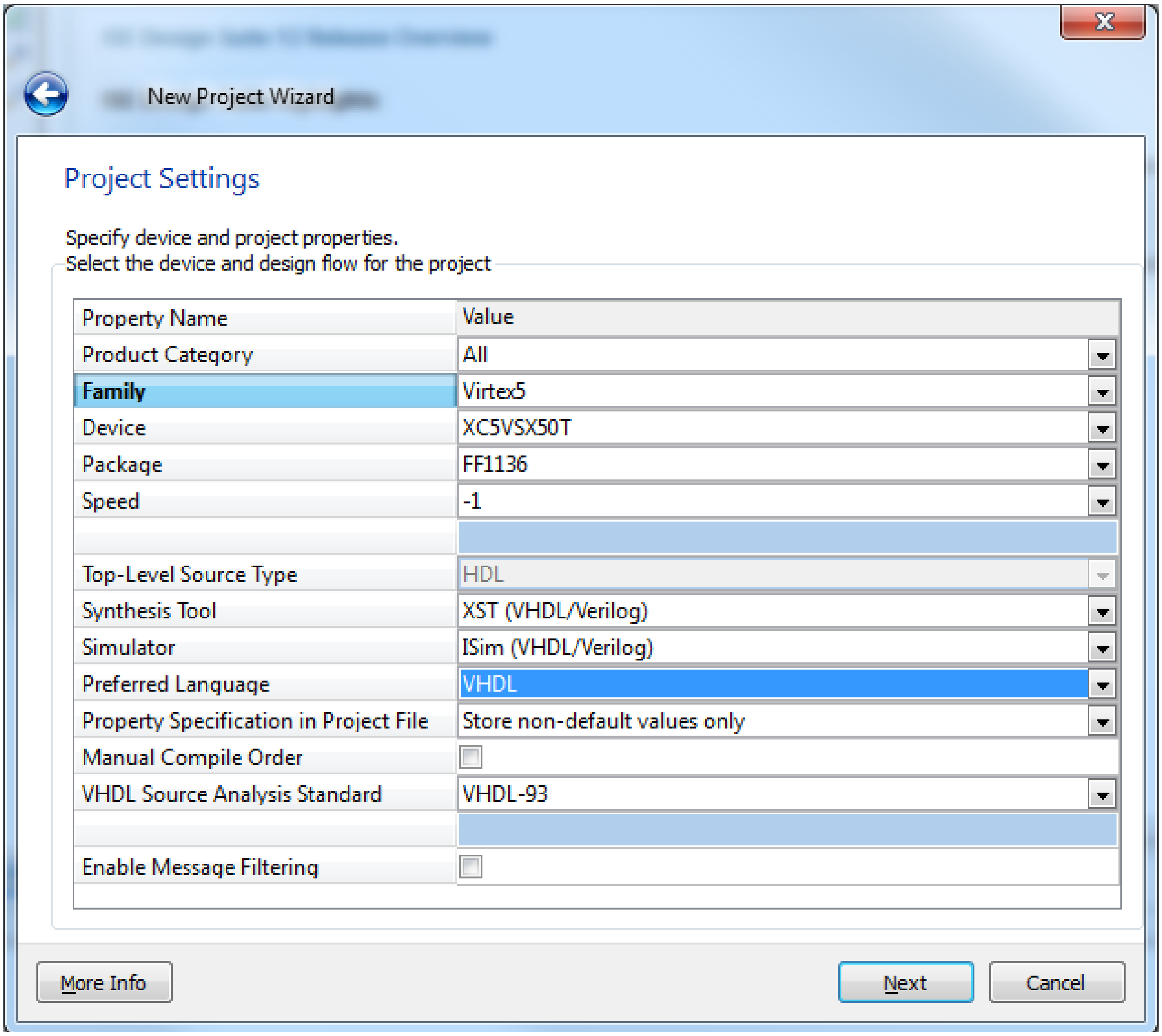
Then name your project something descriptive like : twobit_greaterthan.

Make sure the top-level source is HDL.

Click Next.



Now setup the project settings for the ML506 as shown below and click Next and Finish.



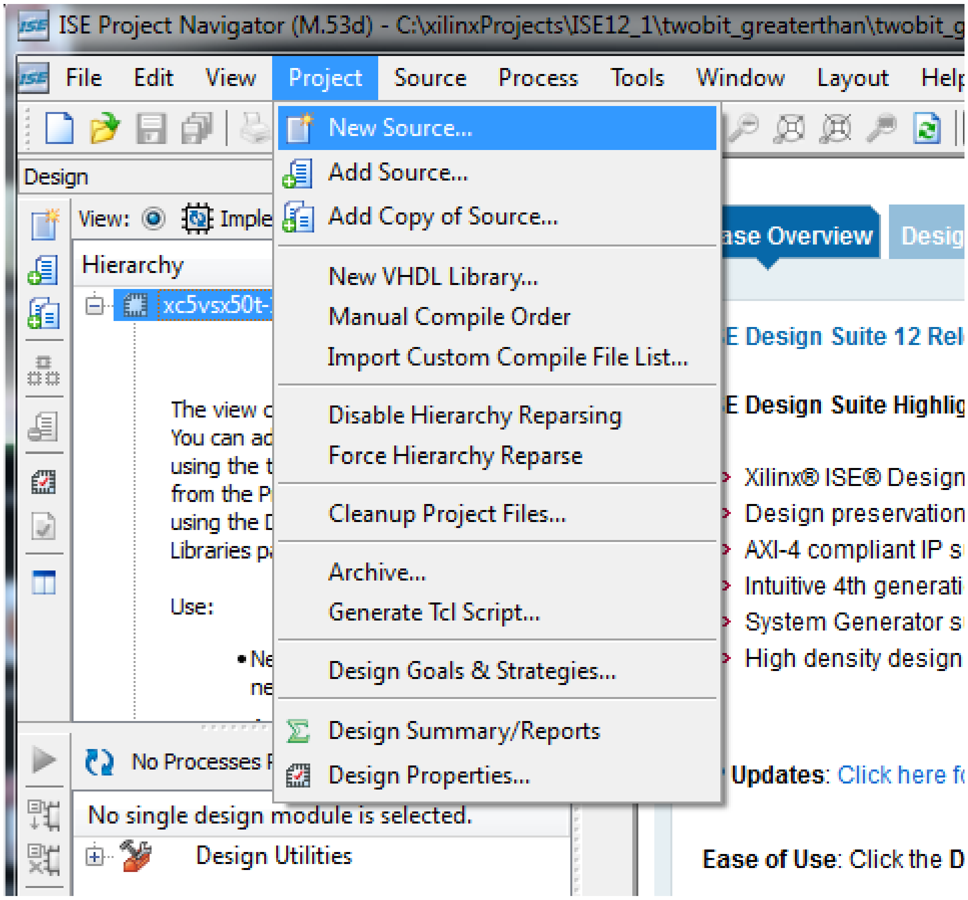
A 2-bit greater-than circuit can be realized as a sum of products, namely,

agreatb = a(1)b(1)' + a(0)b(1)'b(0)' + a(1)a(0)b(0)'

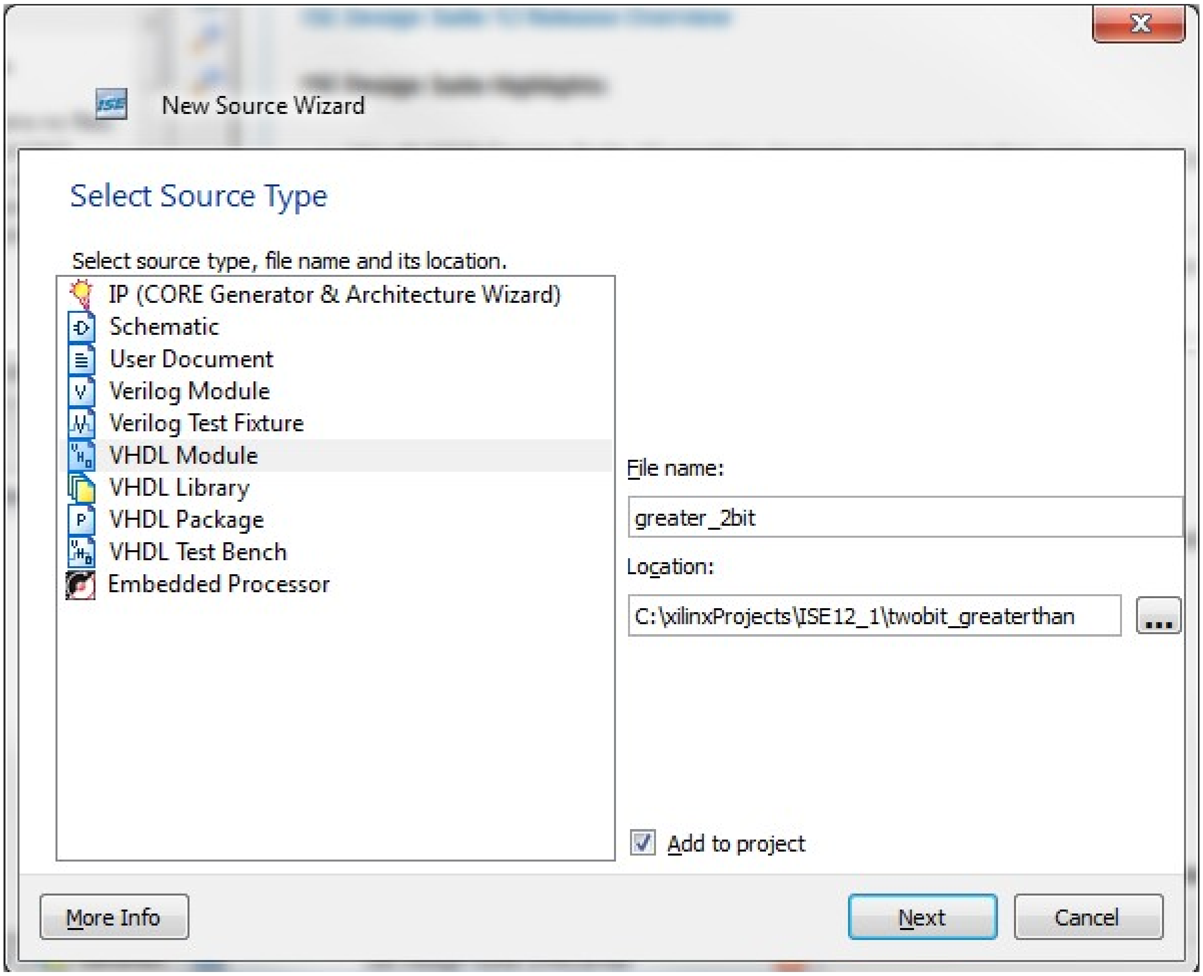
where *a* and *b* are the 2-bit inputs and *agreatb* is the output.

First we will write the VHDL file that implements the circuit, then we will connect inputs a and b to 4 switches and the output agreatb to an led using a universal constraints file.

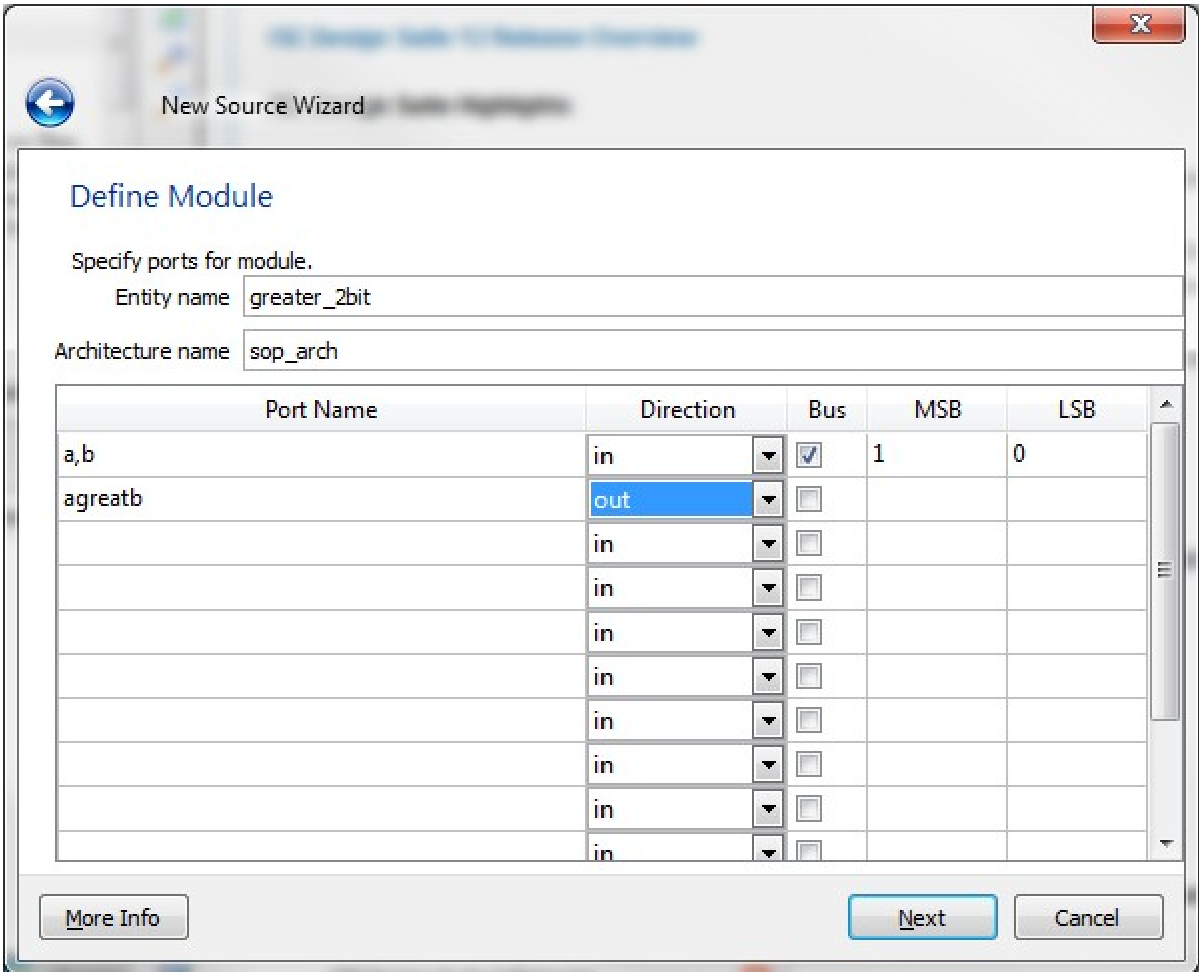
Highlight the xc5vsx50t in the Hierarchy window, then at the top menu click Project → New Source.



Select VHDL module and name the file 'greater_2bit', click Next.



In the define module screen change the architecture name to 'sop_arch' for sum-of-products architecture. We have two inputs, a and b, each two bits wide and one output 'agreatb'. Click Next and Finish.



Xilinx should now have created a VHDL template for use to write our code in. If you don't see it double click on the file 'greater_2bit – sop_arch' in the hierarchy window.

Change the architecture definition to this:

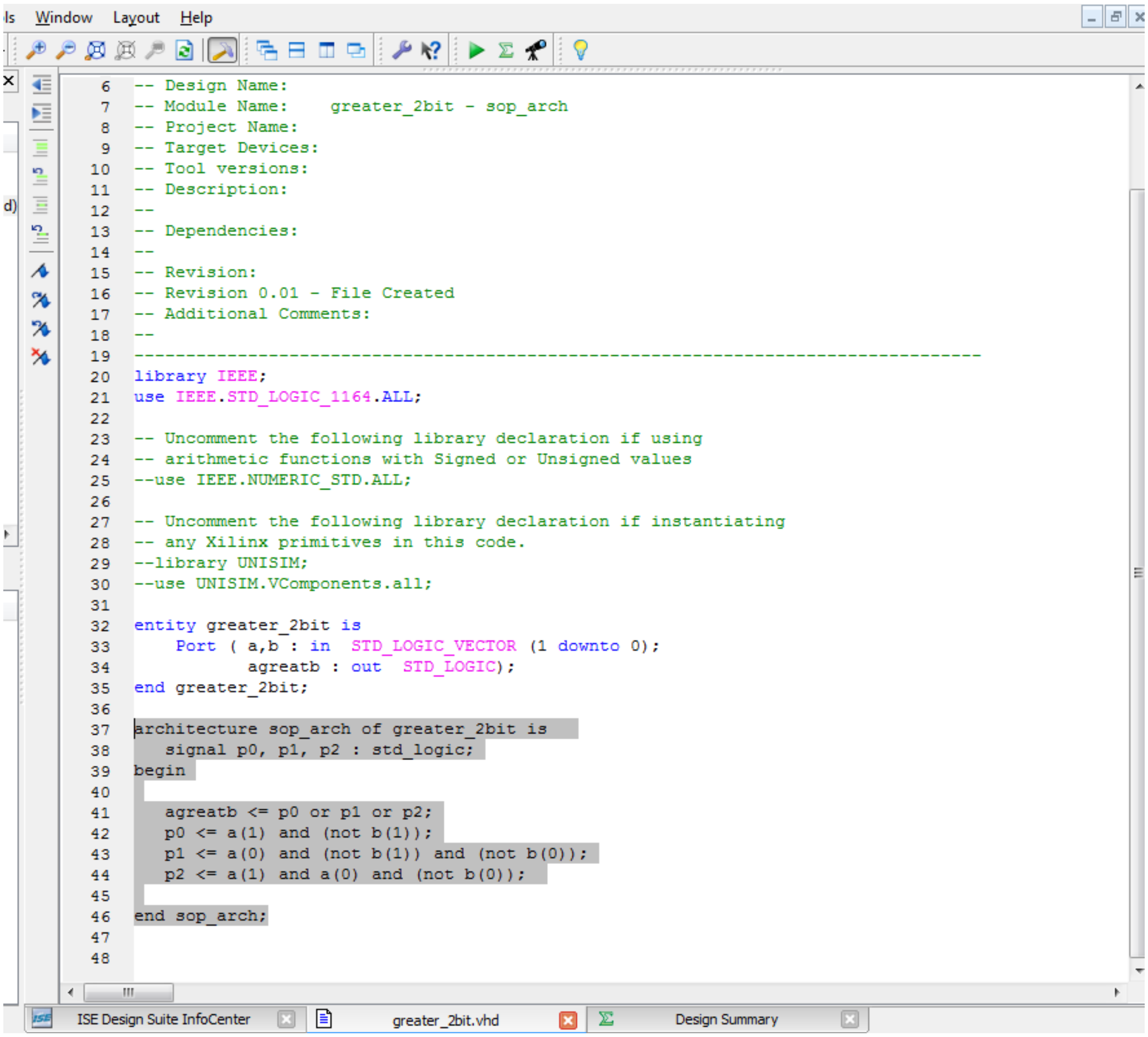
```
architecture sop_arch of greater_2bit is
        signal p0, p1, p2 : std_logic;
begin


        agreatb <= p0 or p1 or p2;
        p0 <= a(1) and (not b(1));
```

p1 <= a(0) and (not b(1)) and (not b(0));

p2 <= a(1) and a(0) and (not b(0));

end sop_arch;

Click save.



Now that we have our circuit realized in code, we will connect the inputs and outputs ,a,b,agreatb, to some switches and an led.

To do this we must know the pin numbers of the FPGA. They can be found here:

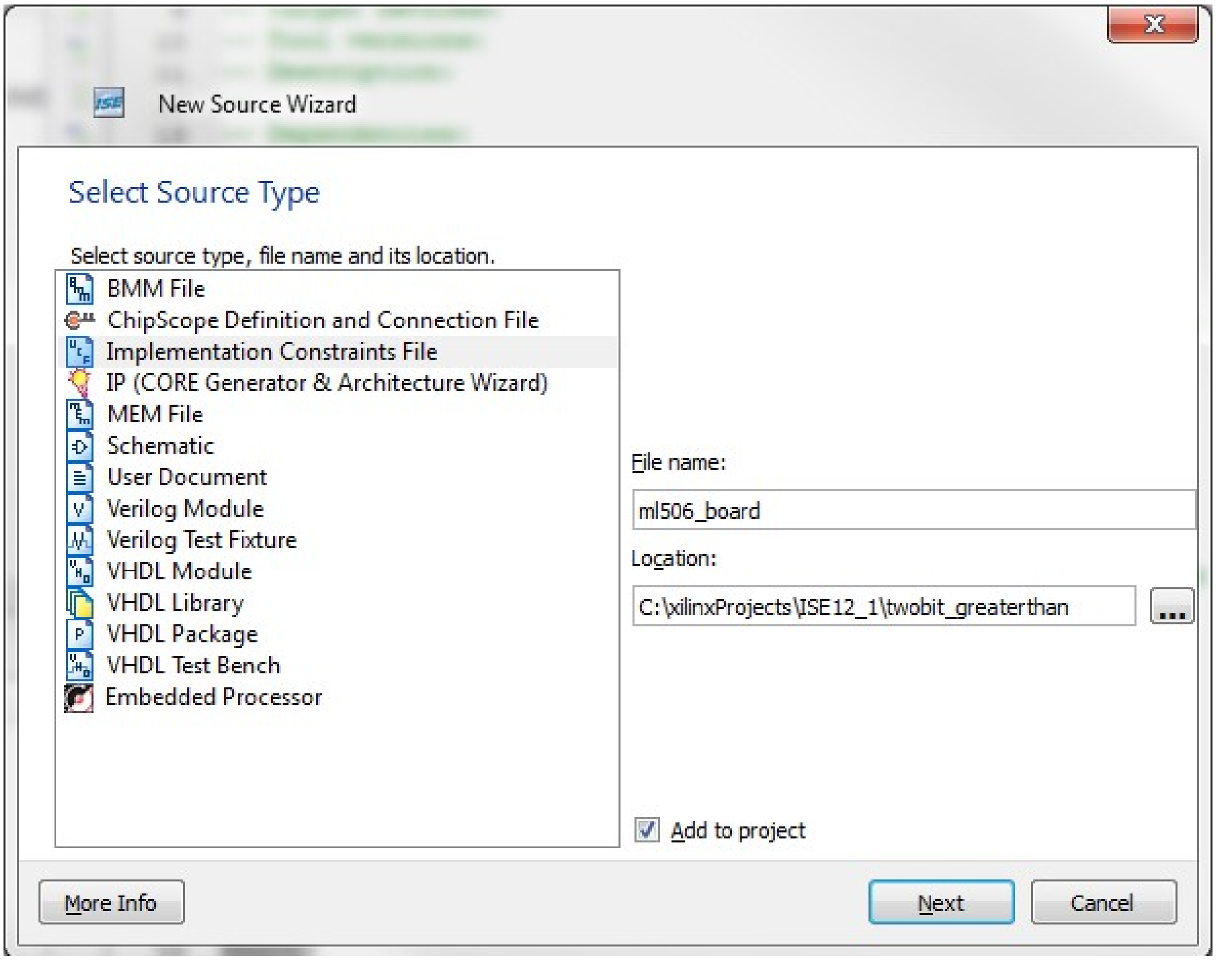
http://www.xilinx.com/products/boards/ml505/ml505_12.1/docs/ml50x_U1_fpga.ucf

We are interested in these pin locations:

```
NET   FPGA_VRN_B21          LOC="AJ25";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   FPGA_VRN_B22          LOC="AF8";   # Bank 22, Vcco=3.3V, DCI using 49.9 ohm resistors
NET   FPGA_VRP_B11          LOC="M33";   # Bank 11, Vcco=2.5V or 3.3V user selectable by J20
NET   FPGA_VRP_B13          LOC="AH33";  # Bank 13, Vcco=2.5V or 3.3V user selectable by J20
NET   FPGA_VRP_B17          LOC="AE31";  # Bank 17, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   FPGA_VRP_B19          LOC="M27";   # Bank 19, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   FPGA_VRP_B20          LOC="L11";   # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET   FPGA_VRP_B21          LOC="AH25";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   FPGA_VRP_B22          LOC="AE9";   # Bank 22, Vcco=3.3V, DCI using 49.9 ohm resistors
NET   GPIO_DIP_SW1          LOC="U25";   # Bank 15, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   GPIO_DIP_SW2          LOC="AG27";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   GPIO_DIP_SW3          LOC="AF25";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   GPIO_DIP_SW4          LOC="AF26";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   GPIO_DIP_SW5          LOC="AE27";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   GPIO_DIP_SW6          LOC="AE26";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   GPIO_DIP_SW7          LOC="AC25";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   GPIO_DIP_SW8          LOC="AC24";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   GPIO_LED_0            LOC="H18";   # Bank 3, Vcco=2.5V, No DCI
NET   GPIO_LED_1            LOC="L18";   # Bank 3, Vcco=2.5V, No DCI
NET   GPIO_LED_2            LOC="G15";   # Bank 3, Vcco=2.5V, No DCI
NET   GPIO_LED_3            LOC="AD26";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   GPIO_LED_4            LOC="G16";   # Bank 3, Vcco=2.5V, No DCI
NET   GPIO_LED_5            LOC="AD25";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   GPIO_LED_6            LOC="AD24";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   GPIO_LED_7            LOC="AE24";  # Bank 21, Vcco=1.8V, DCI using 49.9 ohm resistors
NET   GPIO_LED_C            LOC="E8";    # Bank 20, Vcco=3.3V, DCI using 49.9 ohm resistors
NET   GPIO_LED_E            LOC="AG23";  # Bank 2, Vcco=3.3V
NET   GPIO_LED_N            LOC="AF13";  # Bank 2, Vcco=3.3V
```

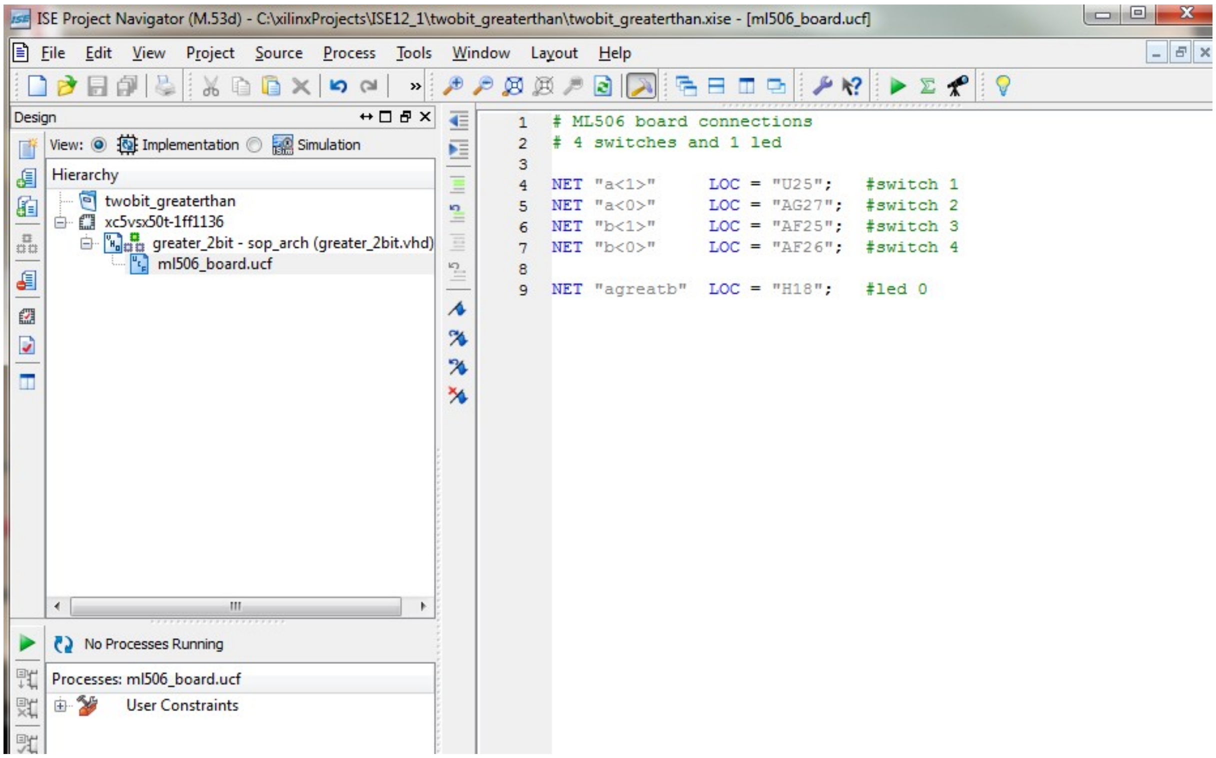
At the top click on Project → New Source.

Select Implementation Constraints File and name the file 'ml506_board'. Click Next and finish.

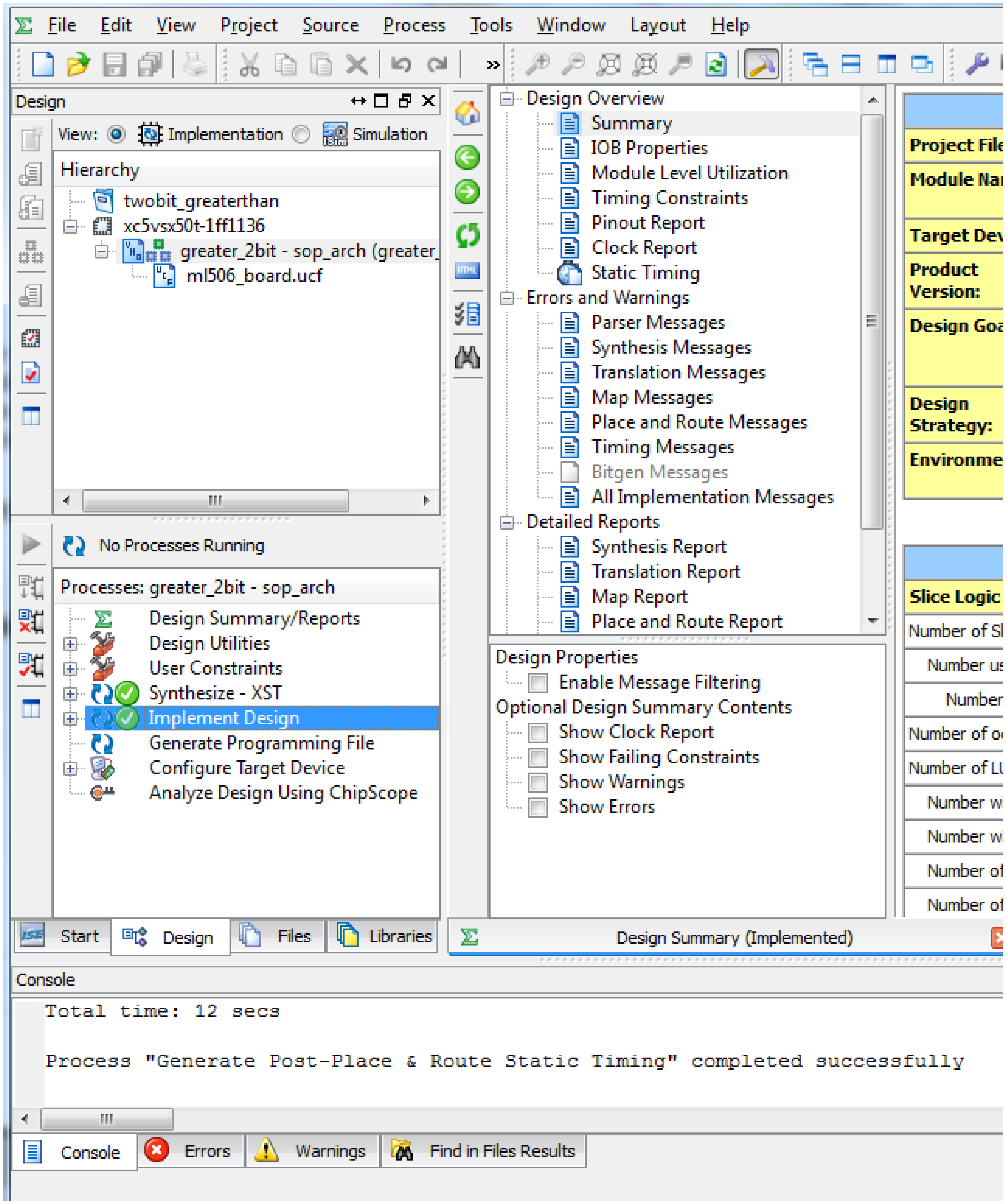


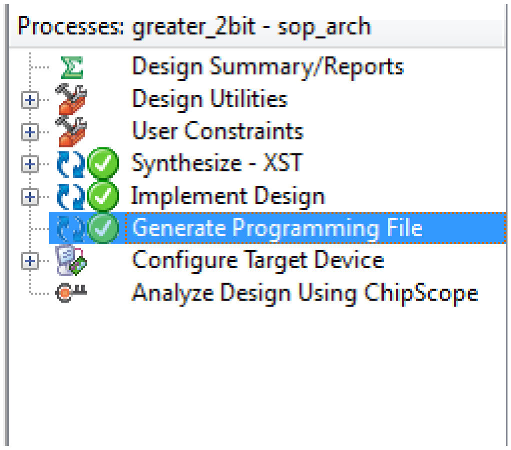
Now you should see a blank file, if not, double-click on 'ml506_board.ucf' in the hierarchy window.

Enter in the following code and save.

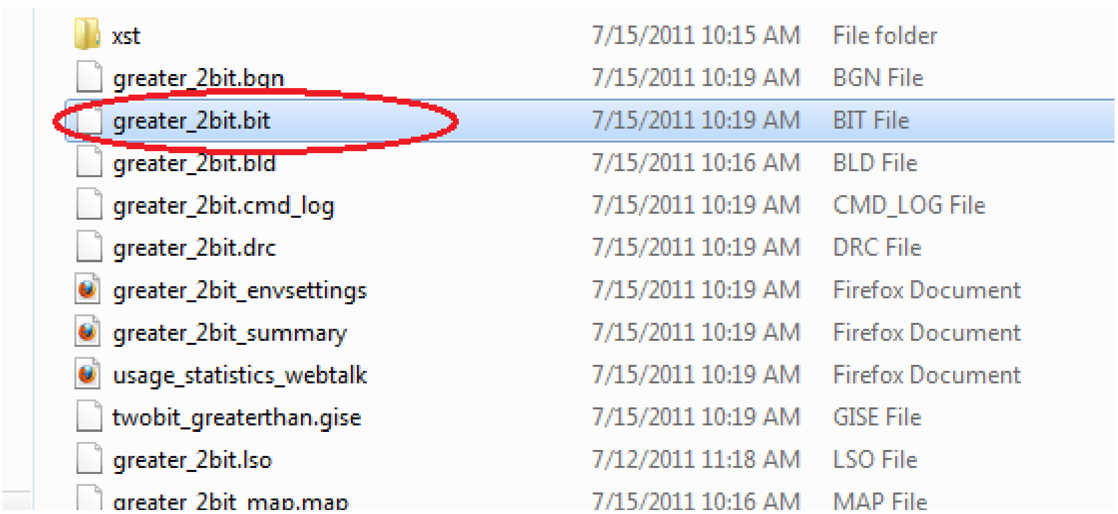


Now that our code is finished we have to implement the design and generate the bitstream. Highlight 'greater_2bit -sop_arch' in the Hierarchy window, then in the processes window below it, double-click 'Implement Design' after it finishes double-click 'Generate Programming File' , this will generate the bistream needed to program the FPGA.



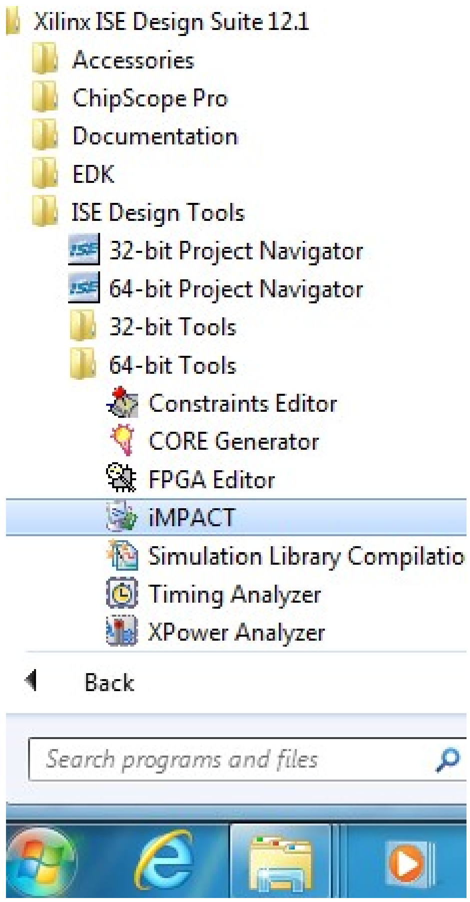


After Xilinx finishes generating the programming file, you can find it inside your project directory.



Programming the FPGA using iMPACT

Now its time to program the FPGA. Close Xilinx ISE and open up iMPACT.

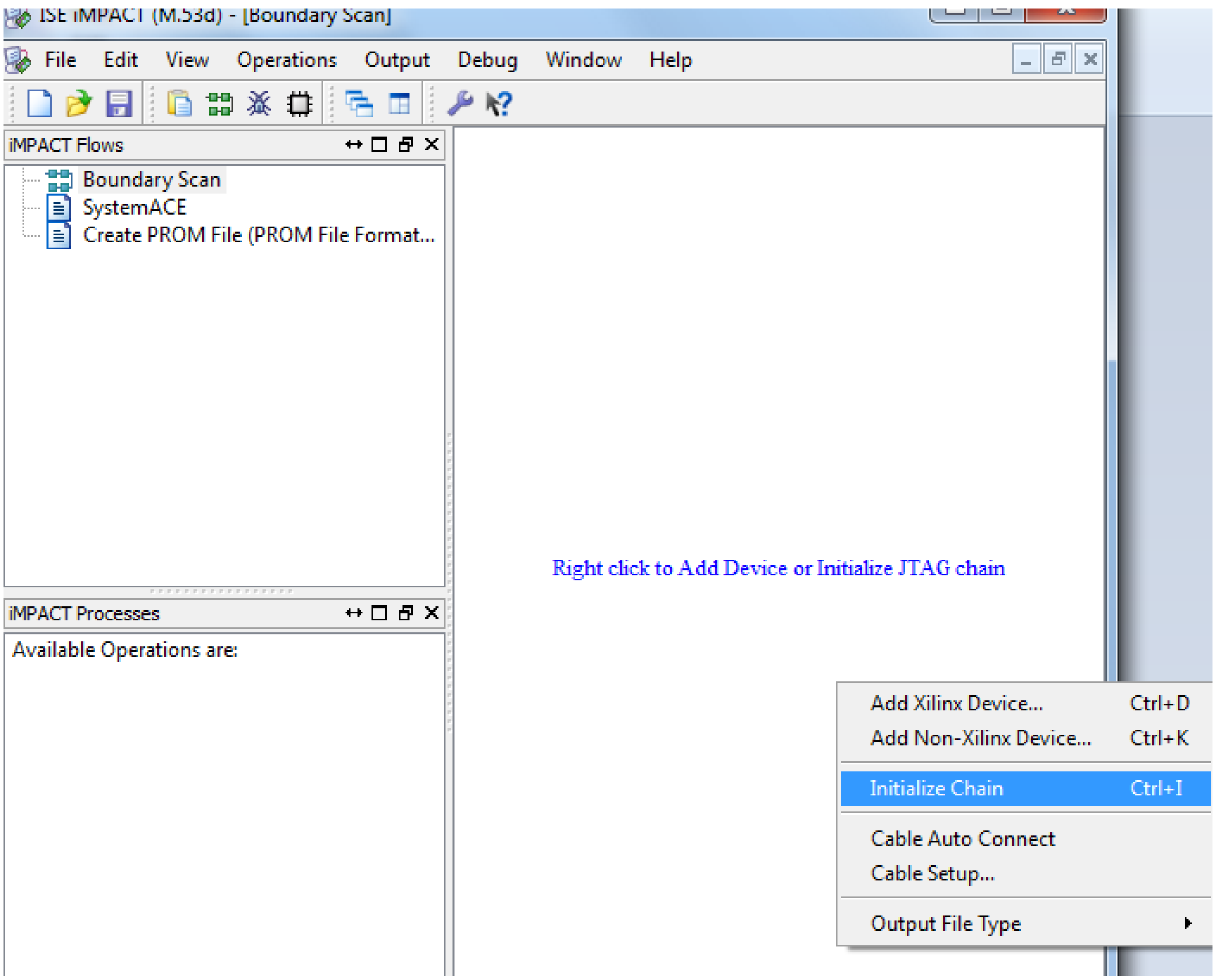


Cancel all the initial prompts that come up.

Turn on the ML506. If this is your first time turning it on, Windows might automatically install some drivers (allow it to, a restart maybe required).

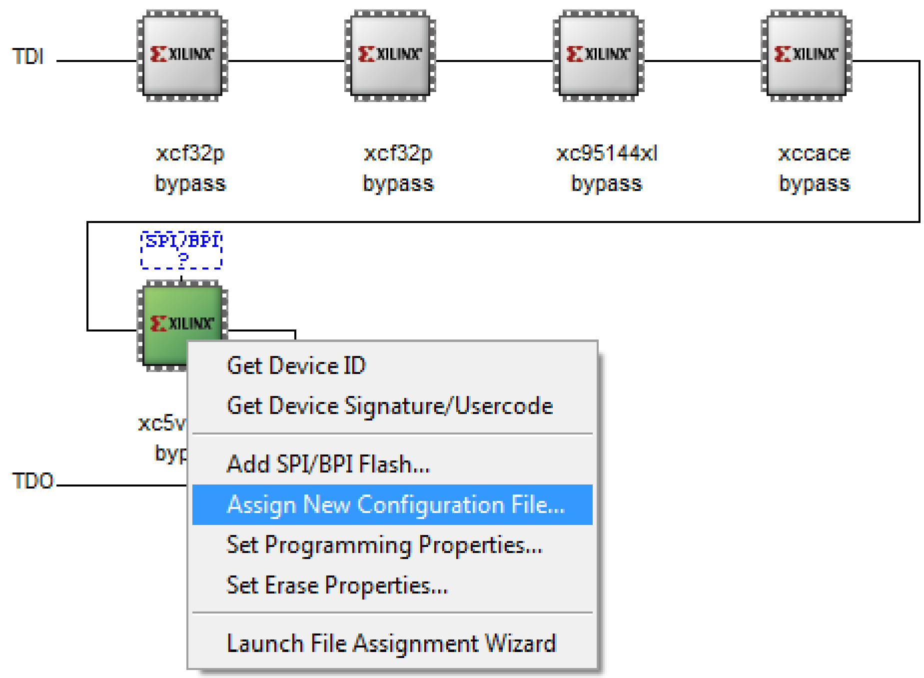
Double-click on 'Boundary Scan', then right-click on the empty plane and click 'Initialize Chain'. If you are getting errors about iMPACT unable to see your board, unplug the USB cable from your ML506 and replug it to initiate some windows auto driver install.

Cancel all the automatic prompts that come up.

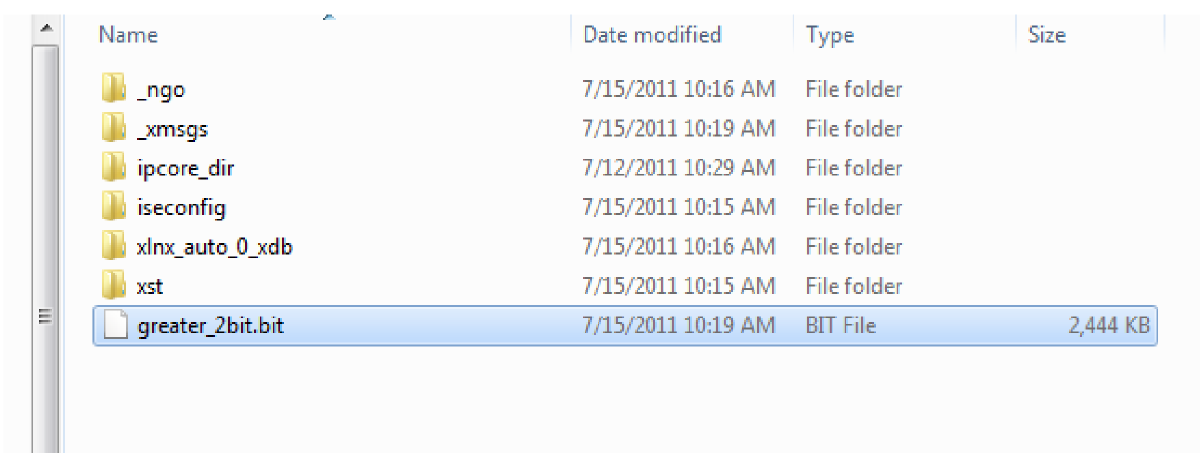


If all is well, you should see the JTAG chain with the Virtex 5 FPGA as well as some PROMs, ACE controller and CPLDs.

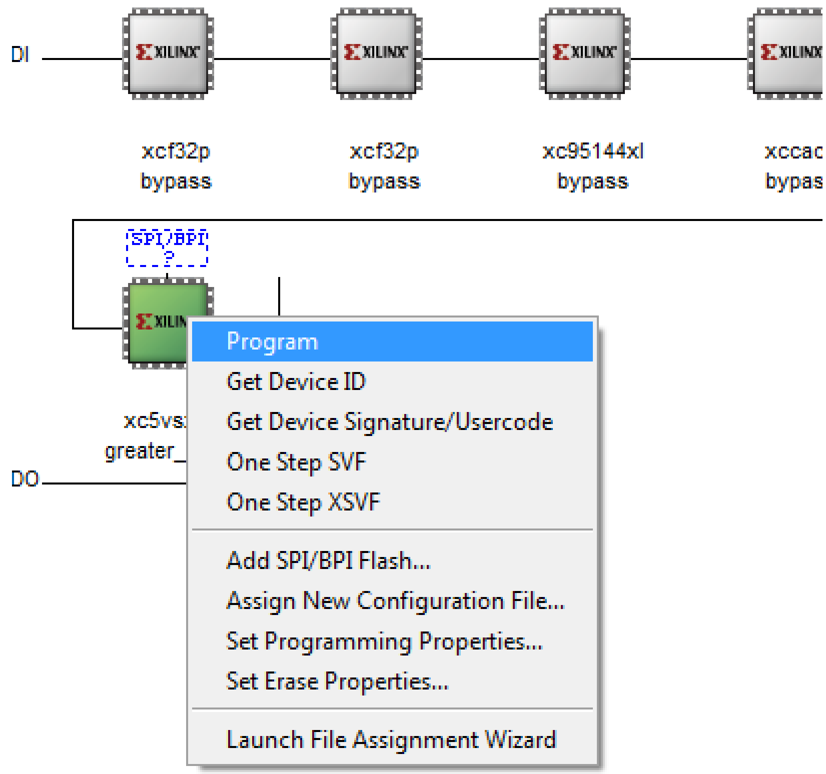
Right-click on the FPGA (xc5vsx50t) and click 'Assign New Configuration File'



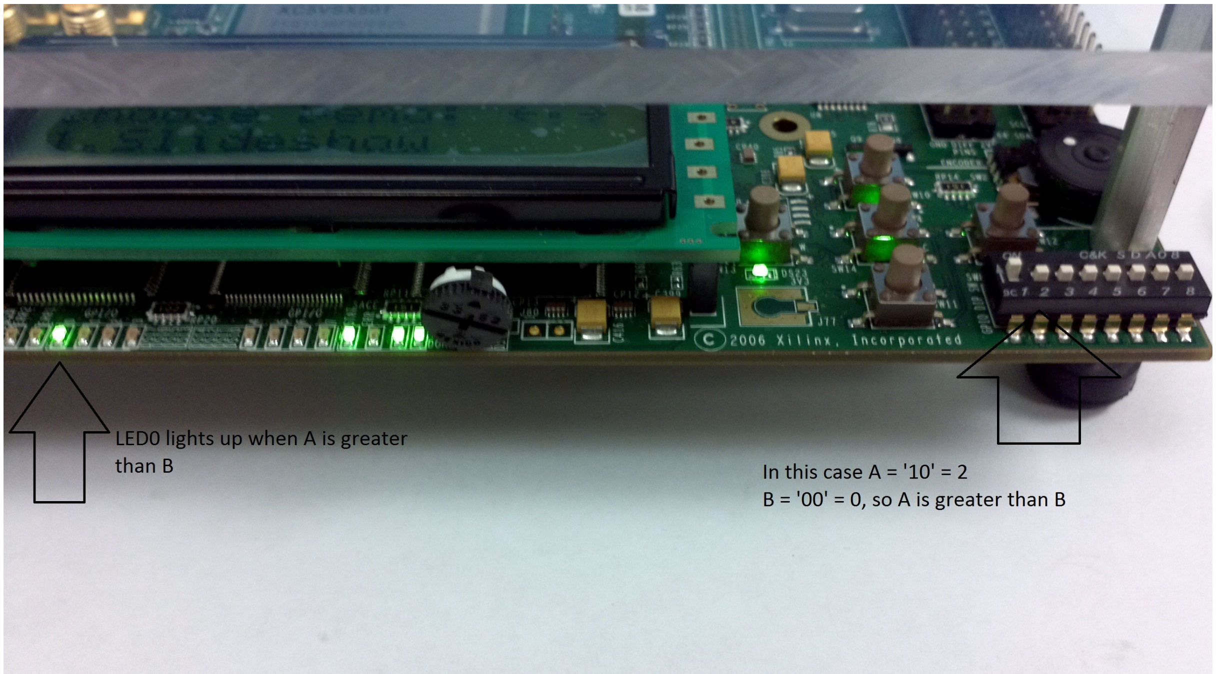
then browse to your .bit file.

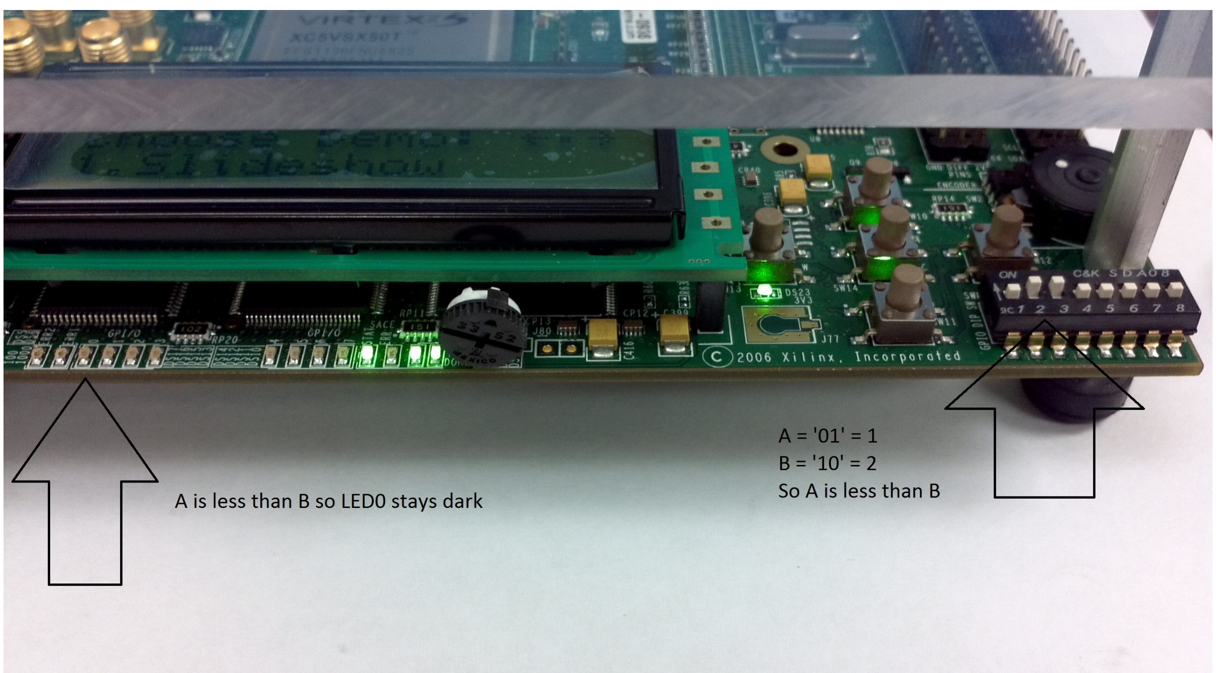


It might prompt you to attach an SPI or BPI PROM, say no. Now right-click the FPGA and click 'Program'



That's it! Your 2-bit greater-than circuit should now be operational. Input a is on switch 1 and 2 of the GPIO DIP switch and input b is on switch 3 and 4. Toggle the switches (the tip of a mechanical pencil would help if the switches are too small for your fingers) and if a is greater than b (in binary) then the GPIO led0 will light up, else it will stay dark.





Hardware Co-Sim with a System Generator created Design Using a Black Box – 2-bit greater-than from above

Summary

This builds off of a tutorial I found here:

http://myfpgablog.blogspot.com/2009/12/sysgen-create-new-hwcosim-target-with.html

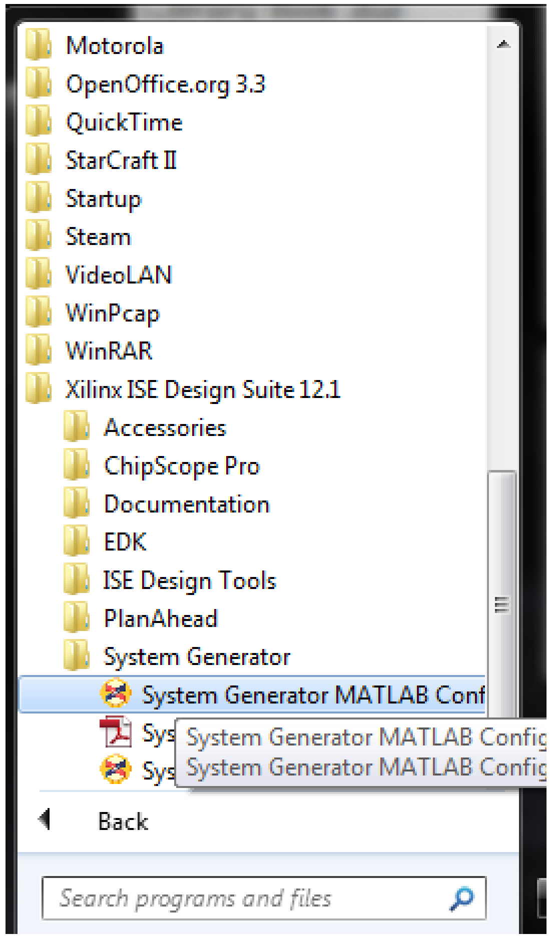
We will use the 2-bit greater-than circuit that was designed in the previous topic 'Programming the FPGA using a VHDL design...' Also the Xilinx blockset in Simulink will be used to define our circuit inputs and outputs. The logic will be fulfilled by using the 'Black Box' block containing our greater-than VHDL code. The bitsream will be created with System Generator and the design will be simulated via hardware co-simulation.

After finishing this tutorial you will be able to create a design using the Xilinx Blockset in Simulink and add a pre-made hdl design using a Black Box, then simulate the design on hardware within Simulink via hardware co-sim.

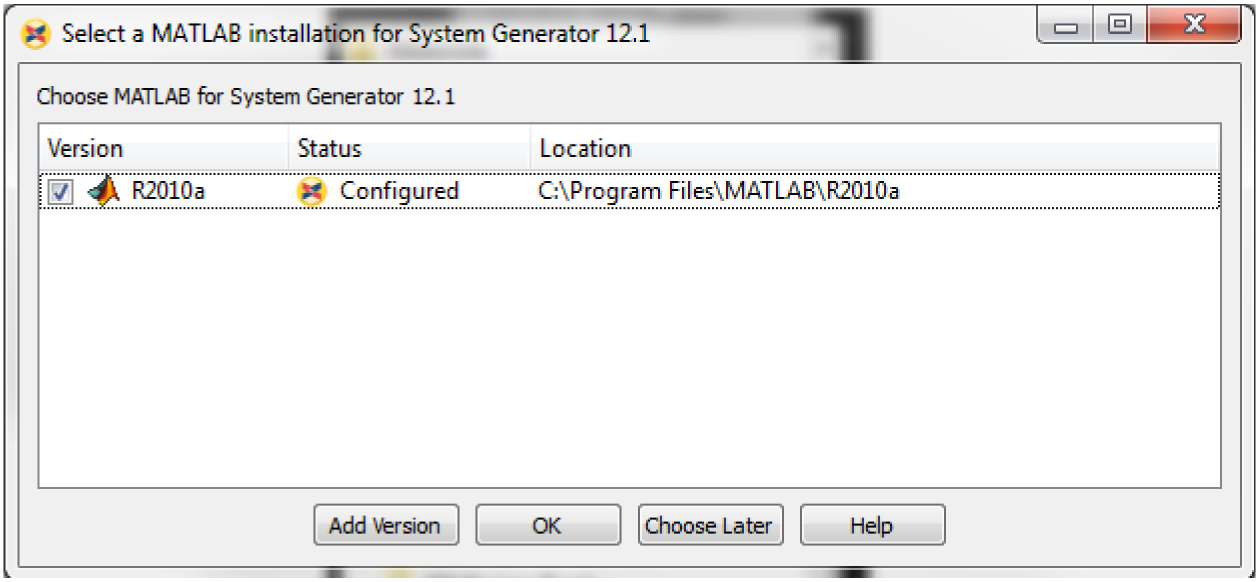
Creating the Design

First make sure that Xilinx System Generator is configured to use your version of MATLAB.

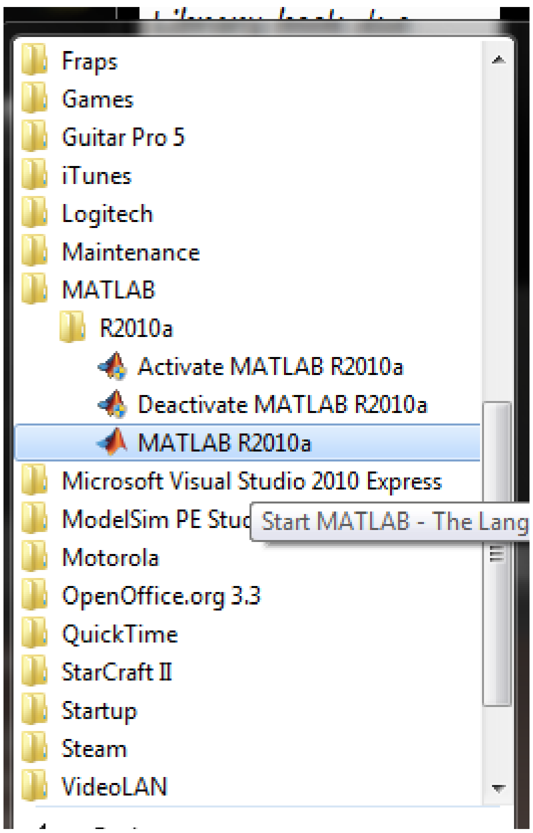
Run the System Generator MATLAB configurator as administrator (right-click → run as administrator):

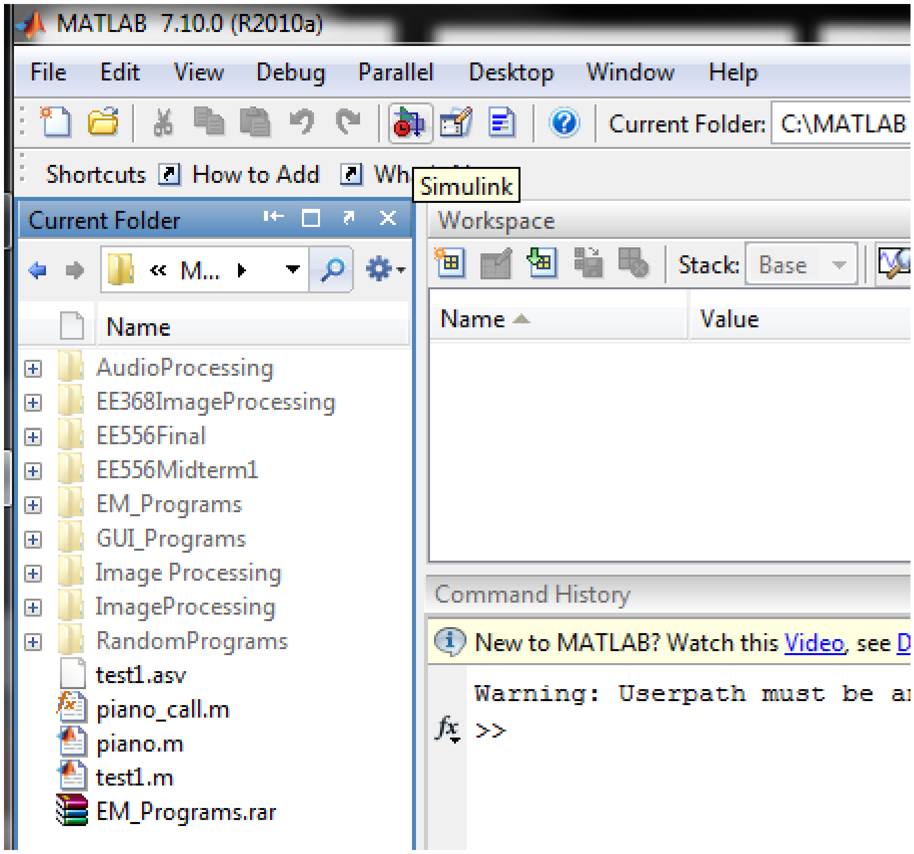


Select the version of MATLAB you will be using and click ok:

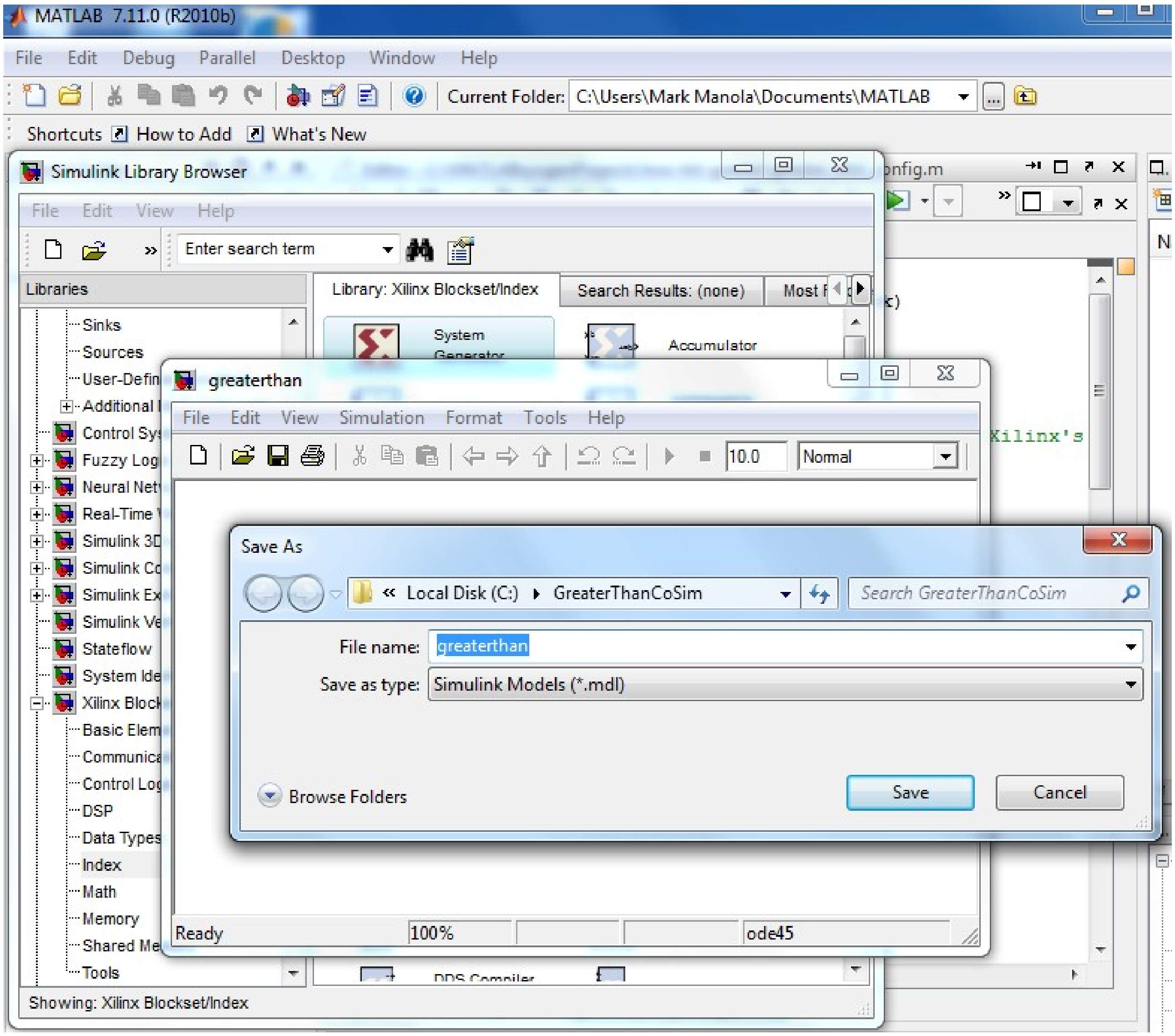


Now open up MATLAB and then Simulink.

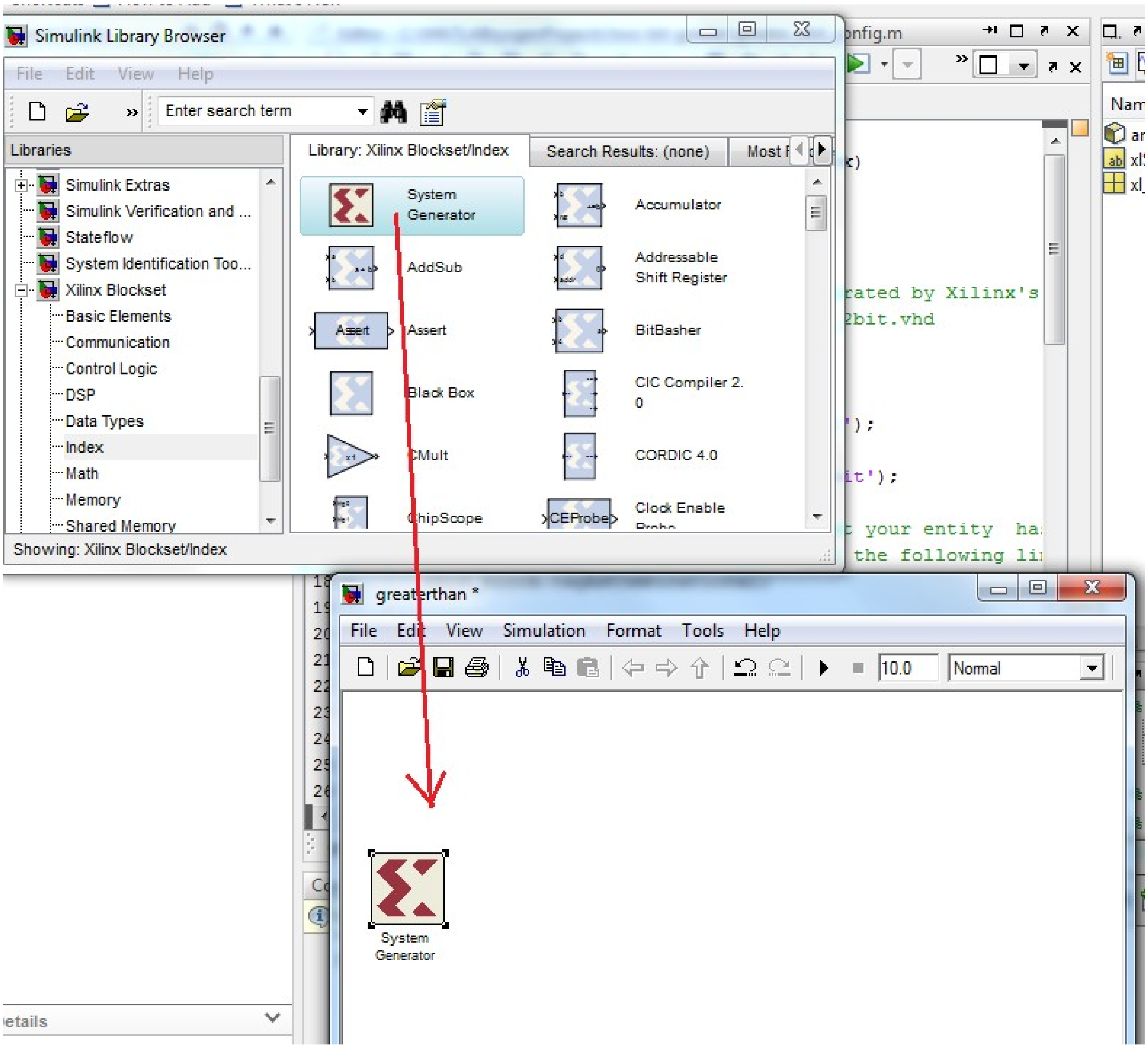



MATLAB 7.10.0 (R2010a)
File Edit View Debug Parallel Desktop Window Help
Current Folder: C:\MATLAB
Shortcuts How to Add Wh
Simulink
Current Folder
« M...
Name
AudioProcessing
EE368ImageProcessing
EE556Final
EE556Midterm1
EM_Programs
GUI_Programs
Image Processing
ImageProcessing
RandomPrograms
test1.asv
piano_call.m
piano.m
test1.m
EM_Programs.rar
Workspace
Stack: Base
Name
Value
Command History
New to MATLAB? Watch this Video, see D
Warning: Userpath must be a
>>

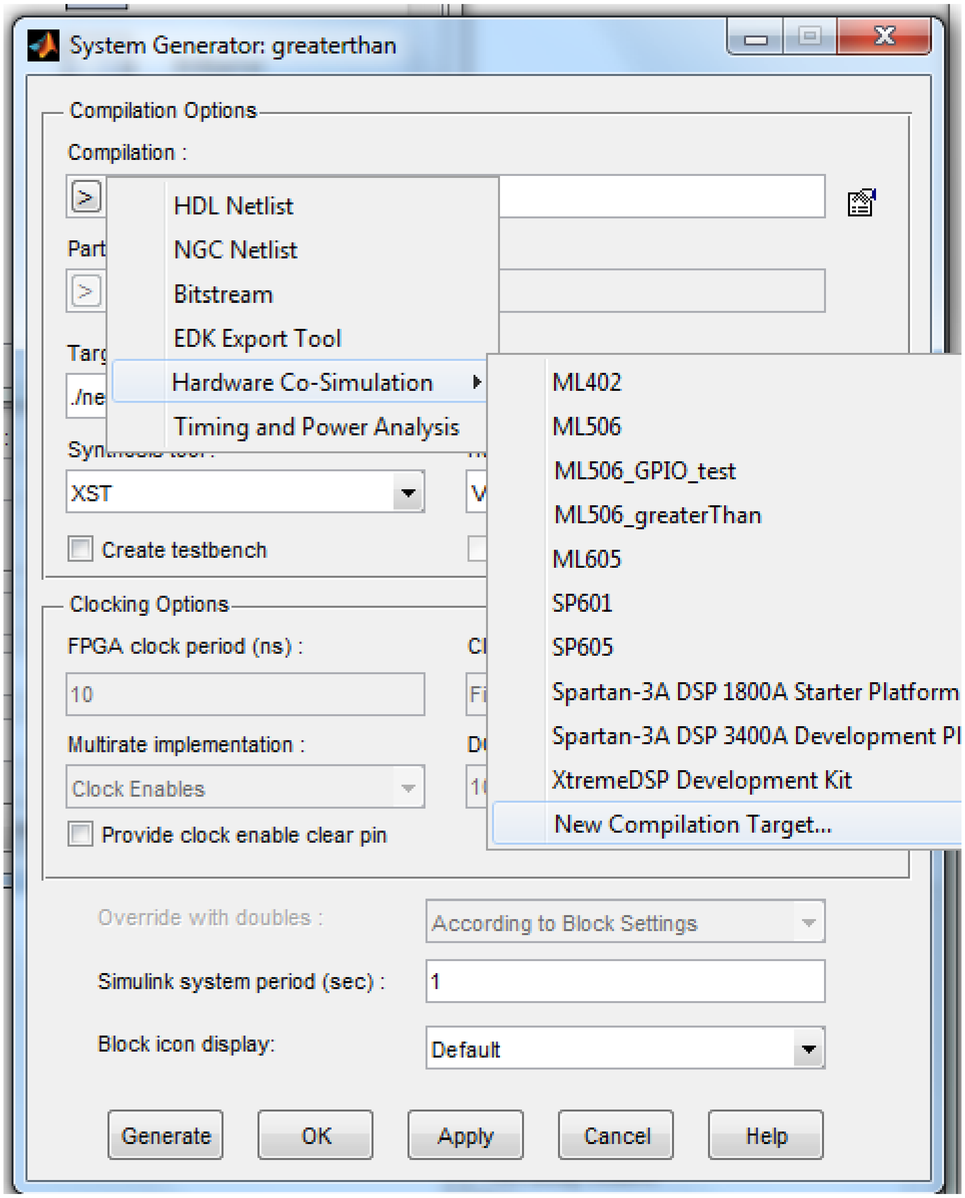
Once Simulink is open make a new model by clicking file → New → Model. Make a new folder on your local drive called 'GreaterThanCoSim' then save the model as 'greaterthan' into your newly created folder.



In the Simulink Library Browser scroll down to Xilinx Blockset and double-click it, then double click on the Index tab. You should see all the available Xilinx blocks to create your design. Click and drag the block 'System Generator' into your model editor.

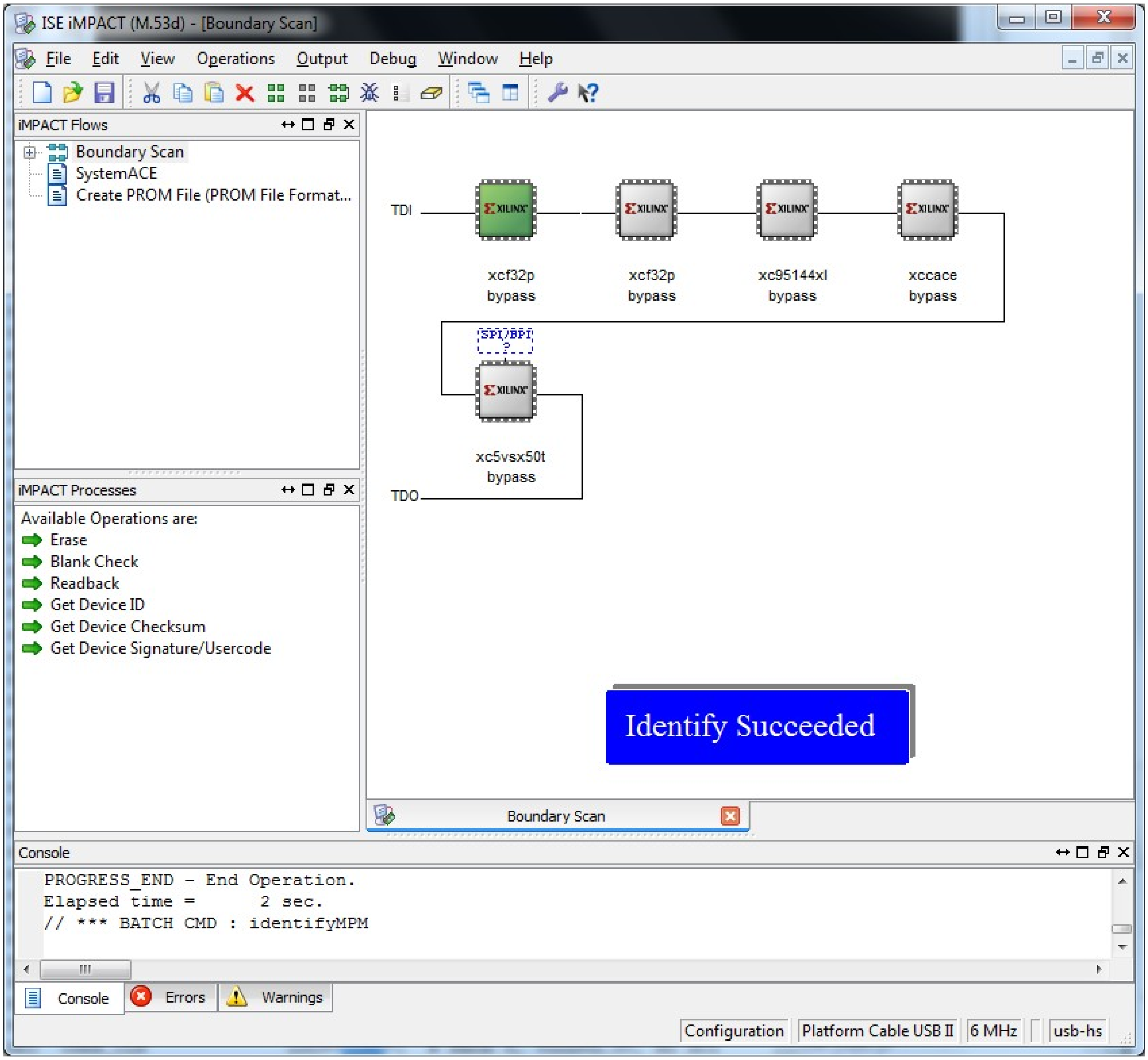


Now we will create a new compilation target so we co-simulate with the ML506. In your model editor window, double-click on the 'System Generator' block. Under Compilation click Hardware Co-Simulation → New Compilation Target.



Now the System Generator Board Description Builder should open up. Under Board Name type 'ML506_twobitGreater'. Set the system clock Frequency to 100MHz and the pin location to AH15 which is the USER_CLK pin.

To find the Boundary Scan Position turn on your board and open up iMPACT (see previous tutorial) after initializing the JTAG chain you will see this:

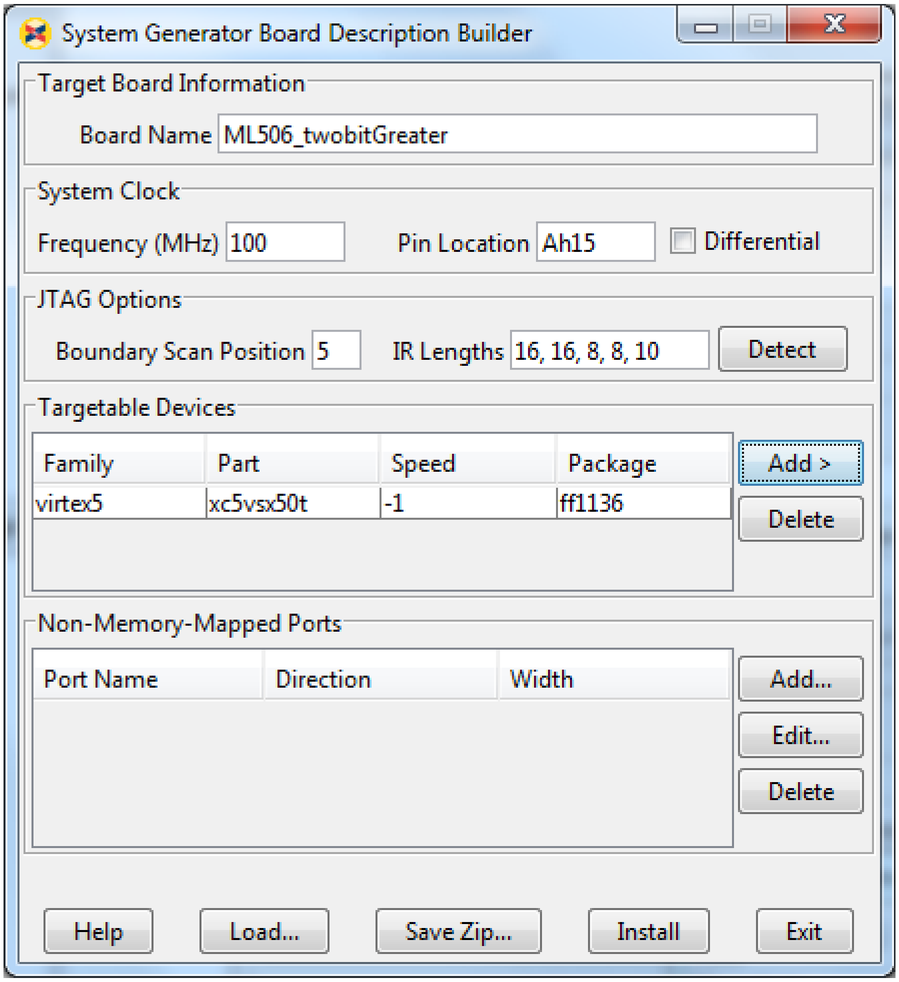


From here we see that the FPGA is in position 5 in the chain.

Leave the board on and click 'Detect' to get the IR Lengths.

On the Targetable Devices section, click Add → virtex5 → xc5vsx50t → -1 → ff1136

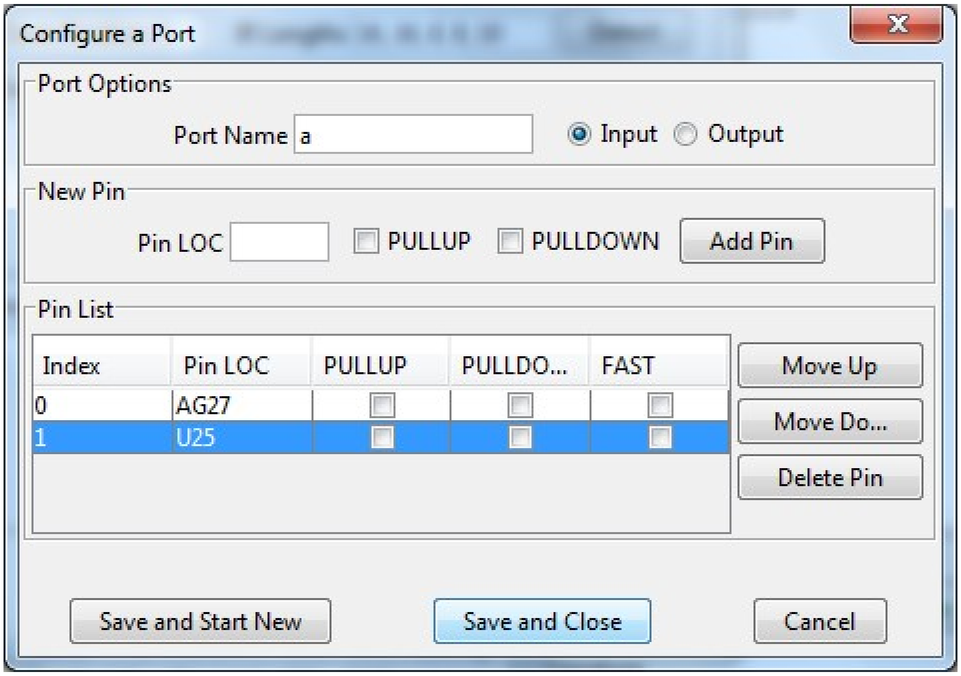
It should now look similar to this:



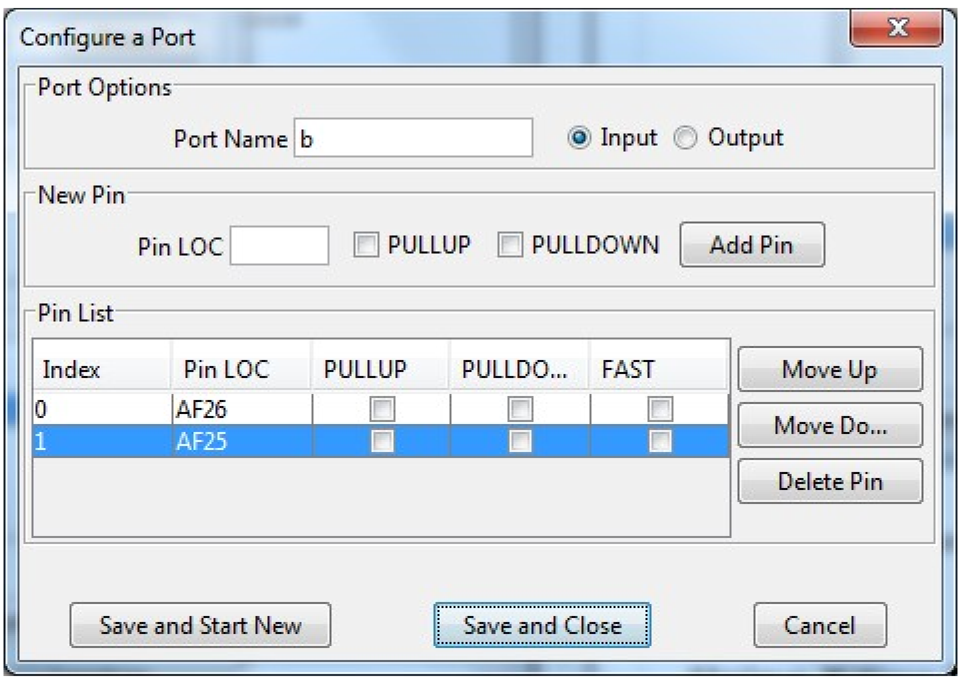
Now it is time to add our inputs and outputs. We will have two inputs each 2-bits wide and one output that is 1-bit wide.

Under the 'Non-Memory-Mapped Ports' section click Add..

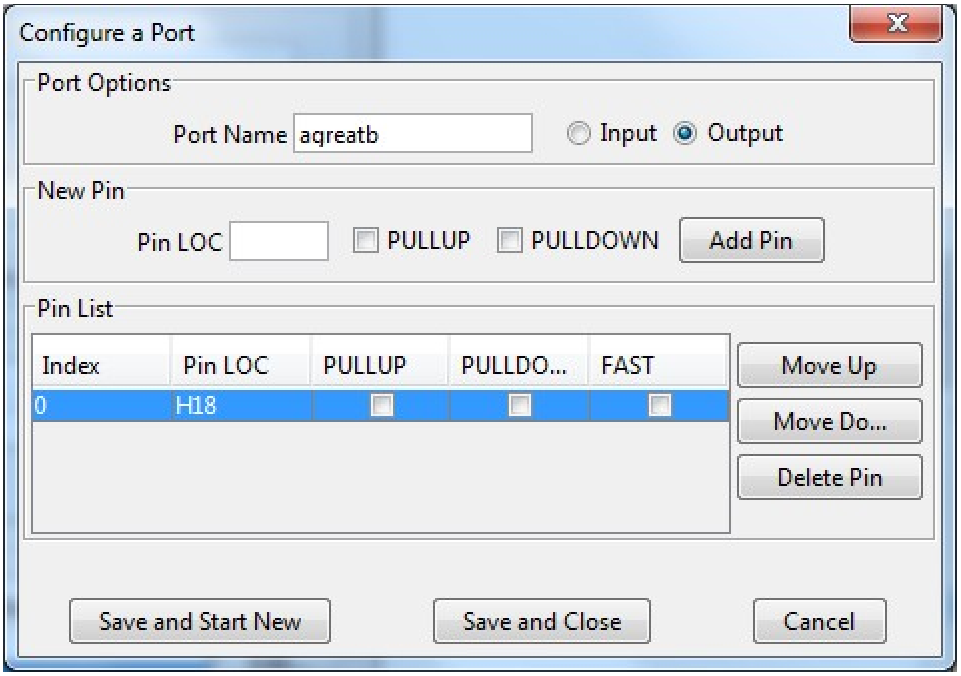
Name the Port Name 'a' and select it as an input. Then for Pin LOC type: AG27 and click Add Pin. This will be the first bit of input a. Do the same for the second bit which is located on pin U25. After adding the second bit, click Save and Close.



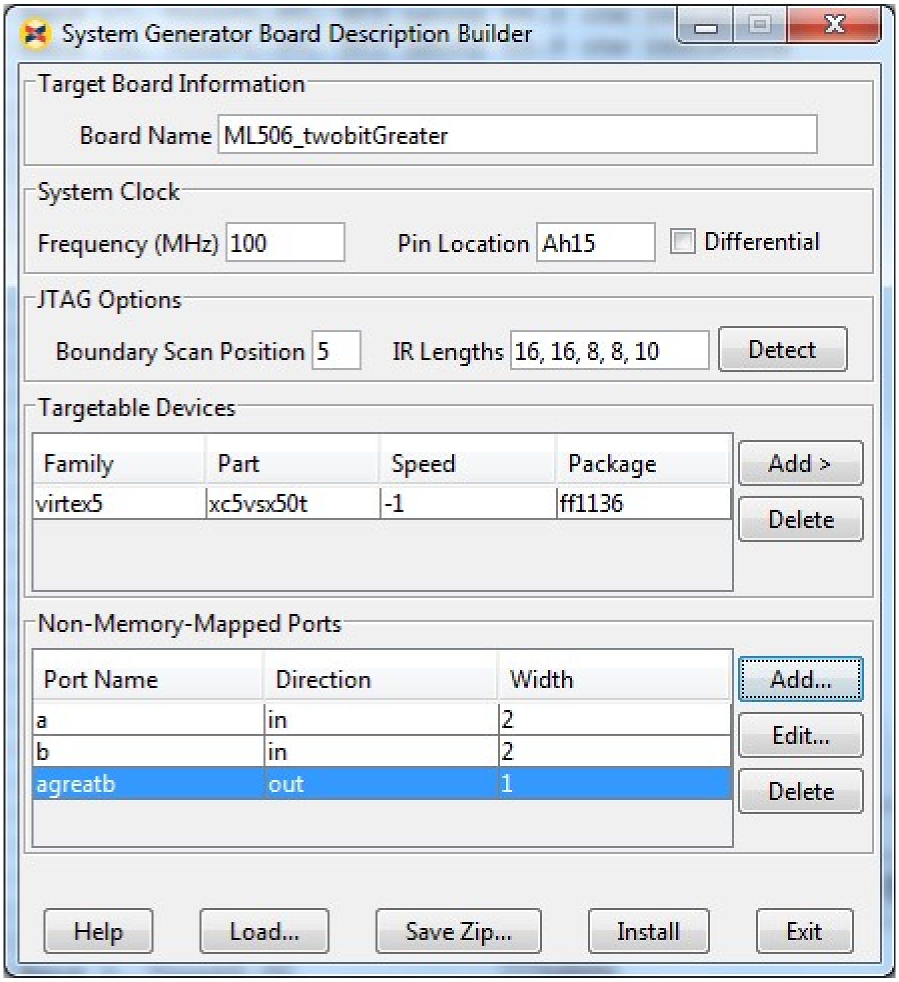
Now we do the same for input b.



Click Save and Close, then click 'Add...' to add the output which will be sent to the GPIO led 0.

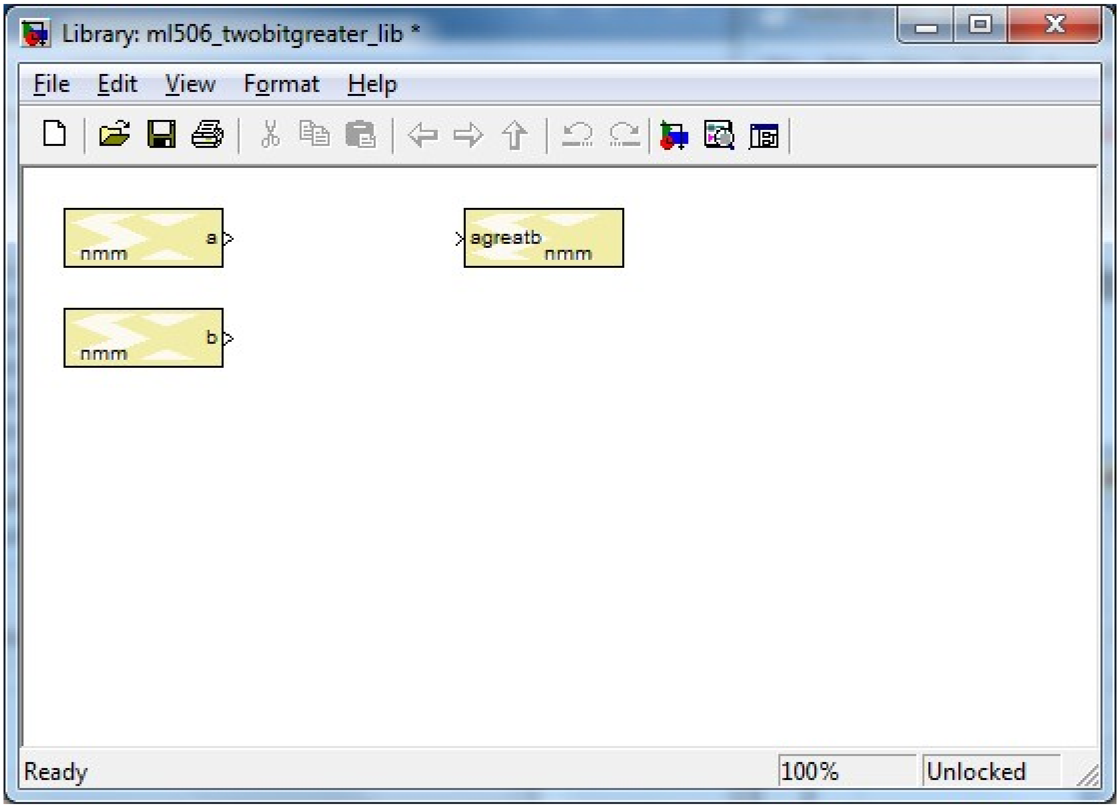


Your screen should now look like this:



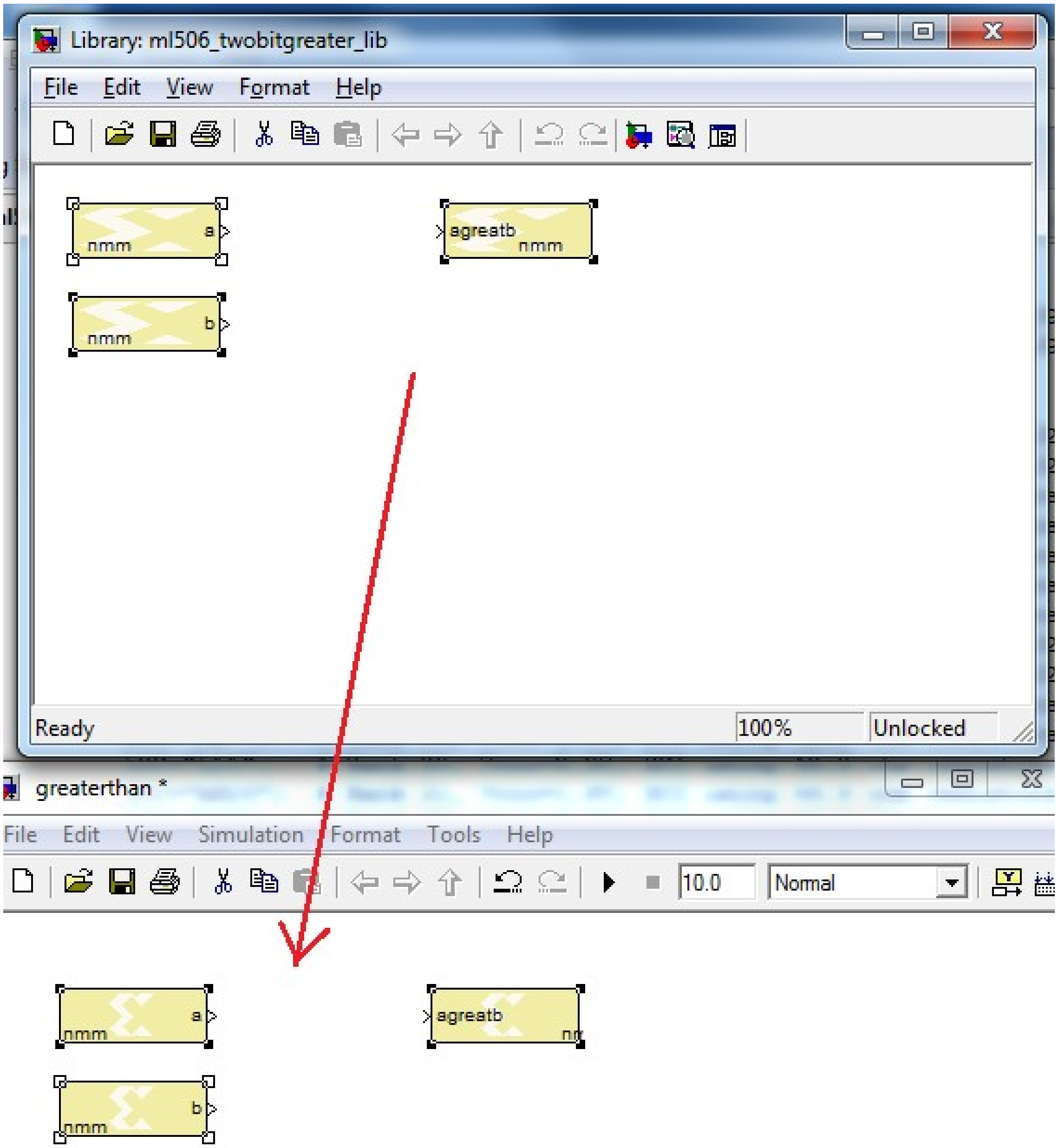
Click Install to finish up the board description.

Once the installation is complete, a new model editor will open up with this model (you can now close the 'System Generator Board Description Window'):



In order to use this template we have to save it first. Click File → Save as and save it in the 'GreaterThanCoSim' folder.

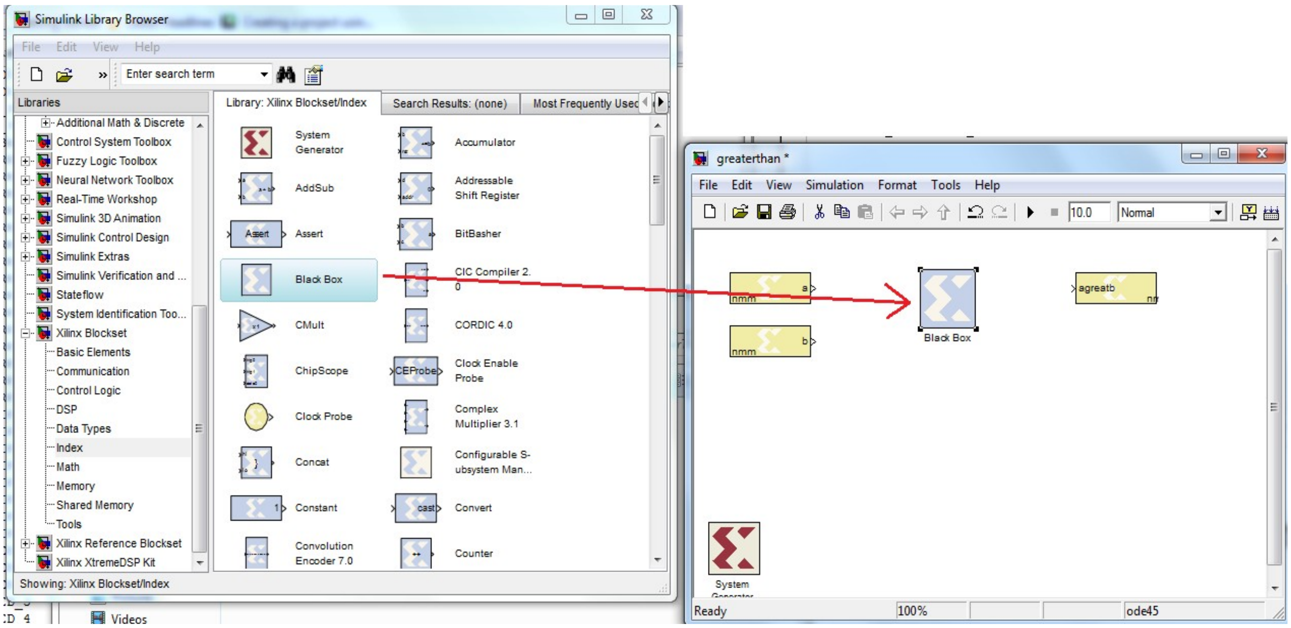
Now highlight the template and drag it into your 'greaterthan' model.


Library: ml506_twobitgreater_lib
File Edit View Format Help
a
nmm
b
nmm
agreatb
nmm
Ready
100%
Unlocked
greaterthan *
File Edit View Simulation Format Tools Help
10.0
Normal
a
nmm
b
nmm
agreatb

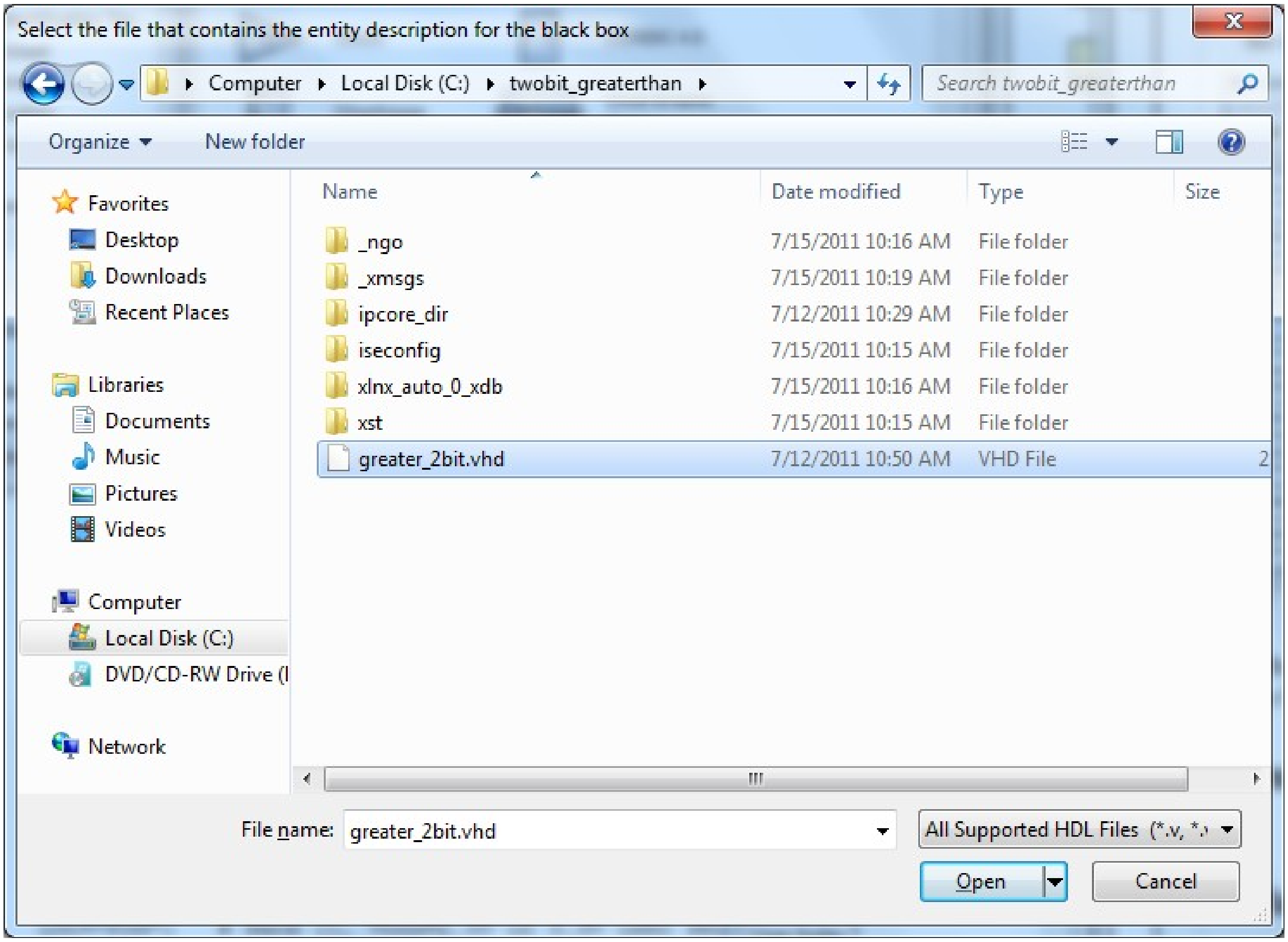
Next we will add the logic to our circuit model. You can do this using blocks by going to the 'Simulink Library Browser' window and selecting the blocks you need. But since we already have our greater-than logic (in VHDL) we will use this to save some time.

In order to use a VHDL design in Simulink, we need the 'Black Box' block located under Xilinx Blockset → Index in the Simulink Library Broswer.

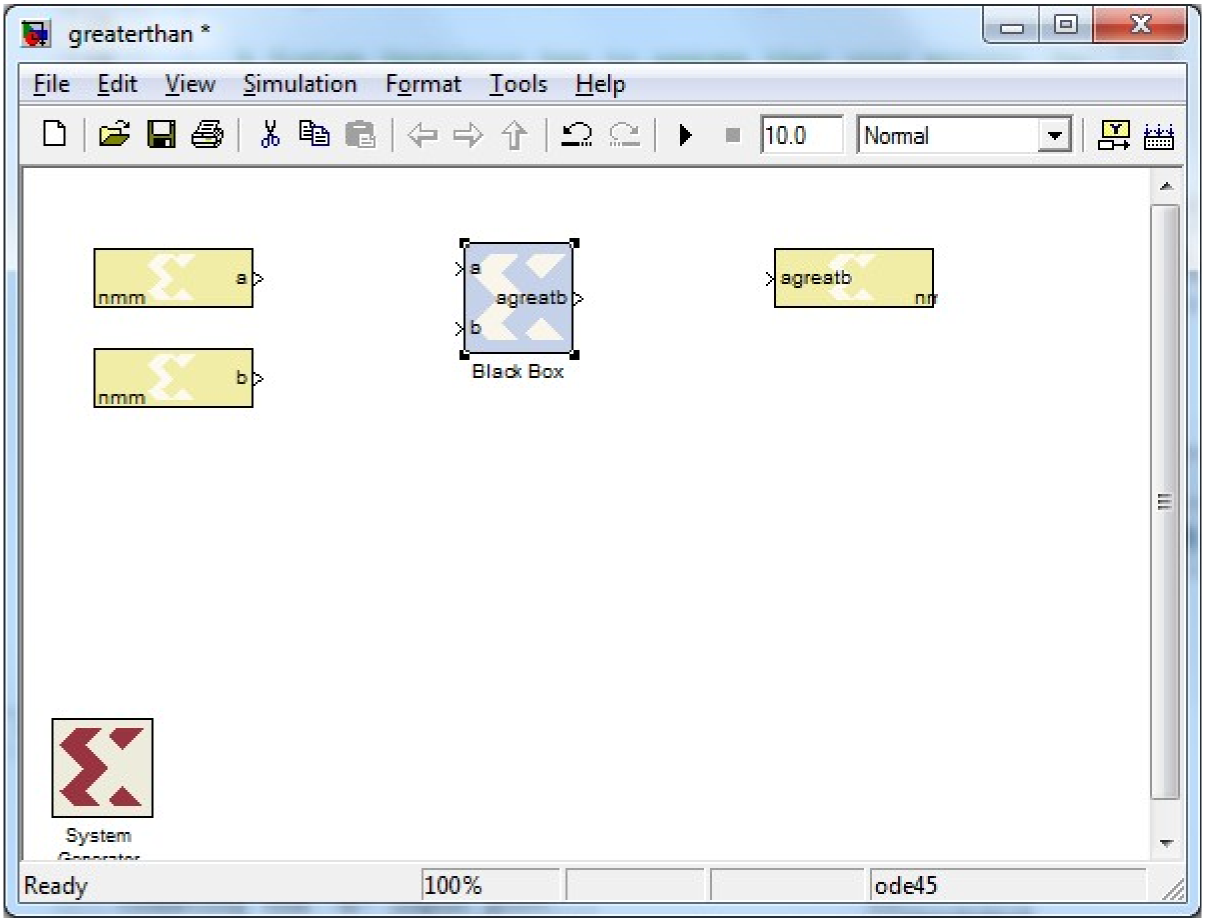
So drag the Black Box into your model editor.



After dragging in the box, a window will automatically appear requesting the entity description for the black box. Locate the VHDL file for the greater-than circuit and click open:

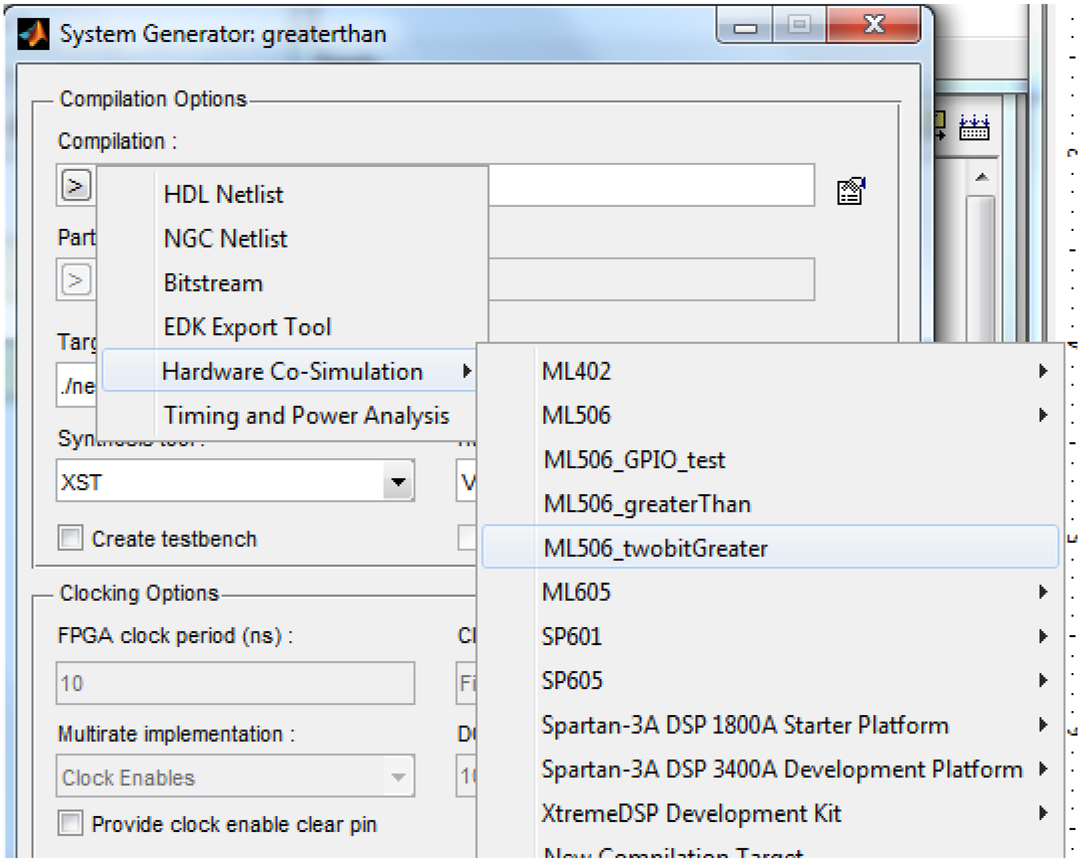


Now your Black Box should have inputs and outputs as described by our VHDL file.



Wire up your inputs and outputs to the Black Box by clicking and holding down the left mouse button.

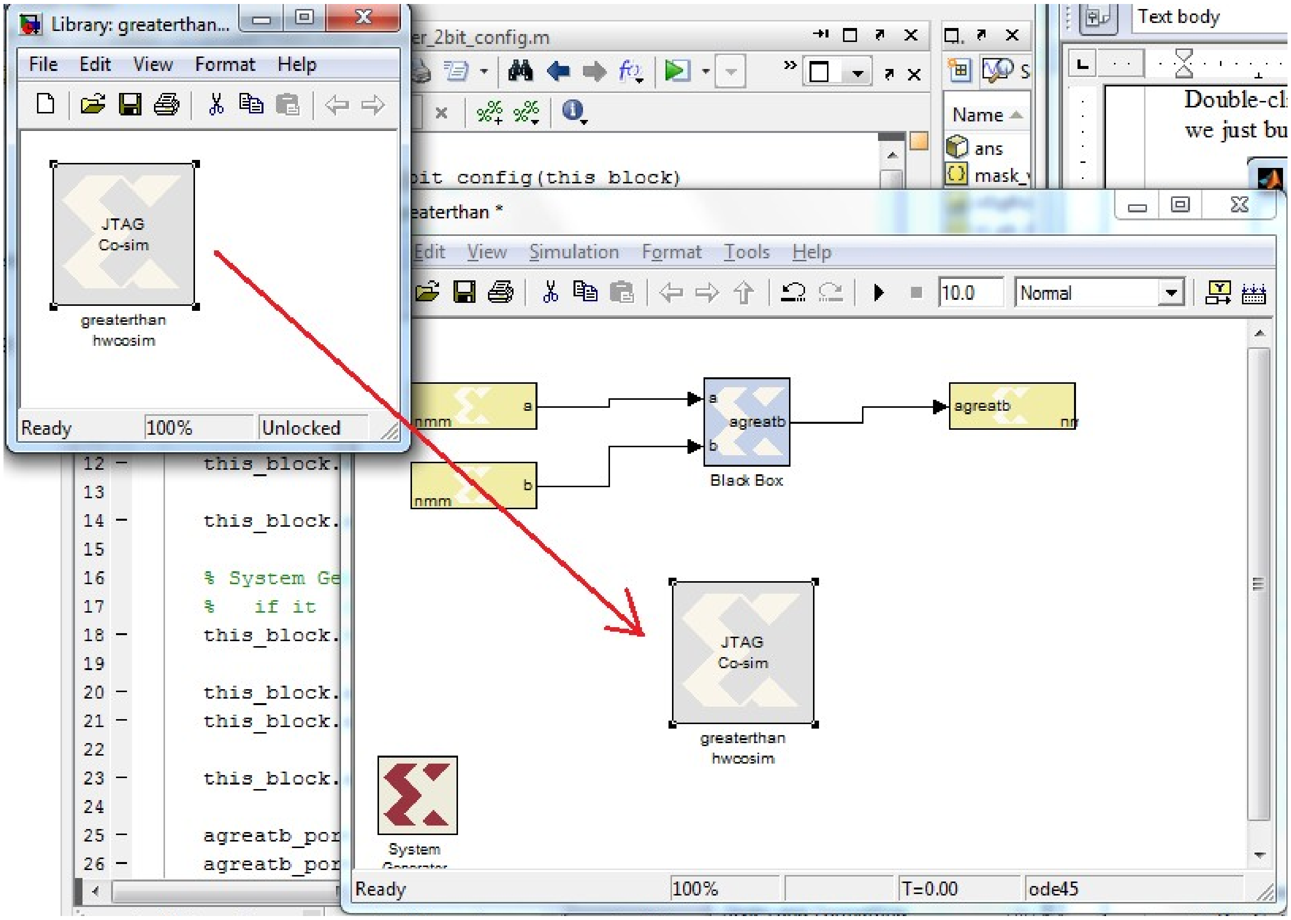
Double-click on the System Generator icon in your model editor and select the board description we just built under the Compilation tab.



If you don't see it, close the System Generator window and save your model file. Then double-click on the Sytsem Generator icon again.

After selecting 'ML506_twobitGreater' click Generate. This may take a while (5-10 mins).

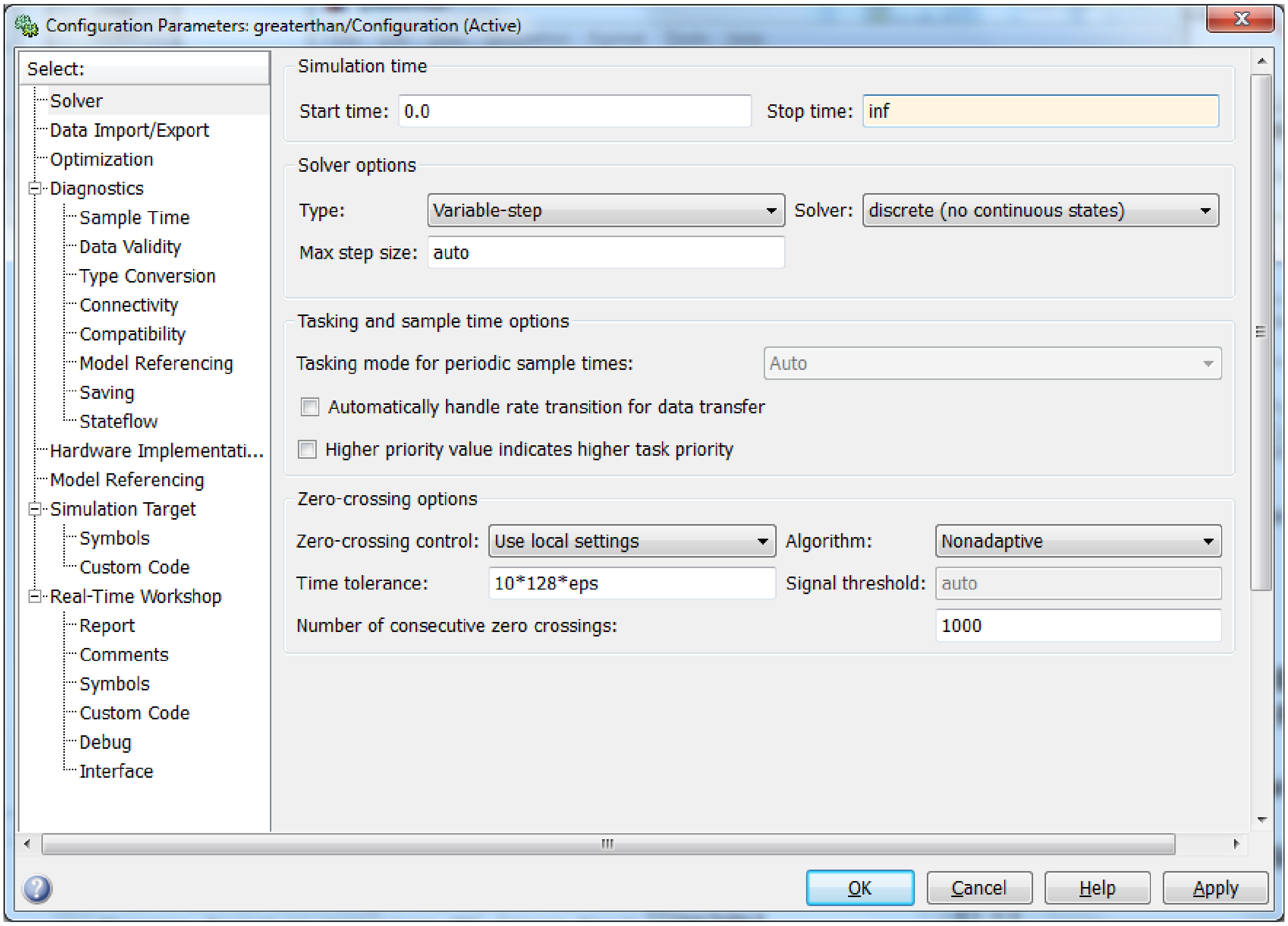
Once completed, System Generator will show a Co-Sim block, drag this block into your model editor.



We can now simulate our design. In your model editor, click Simulation → Configuration Parameters.

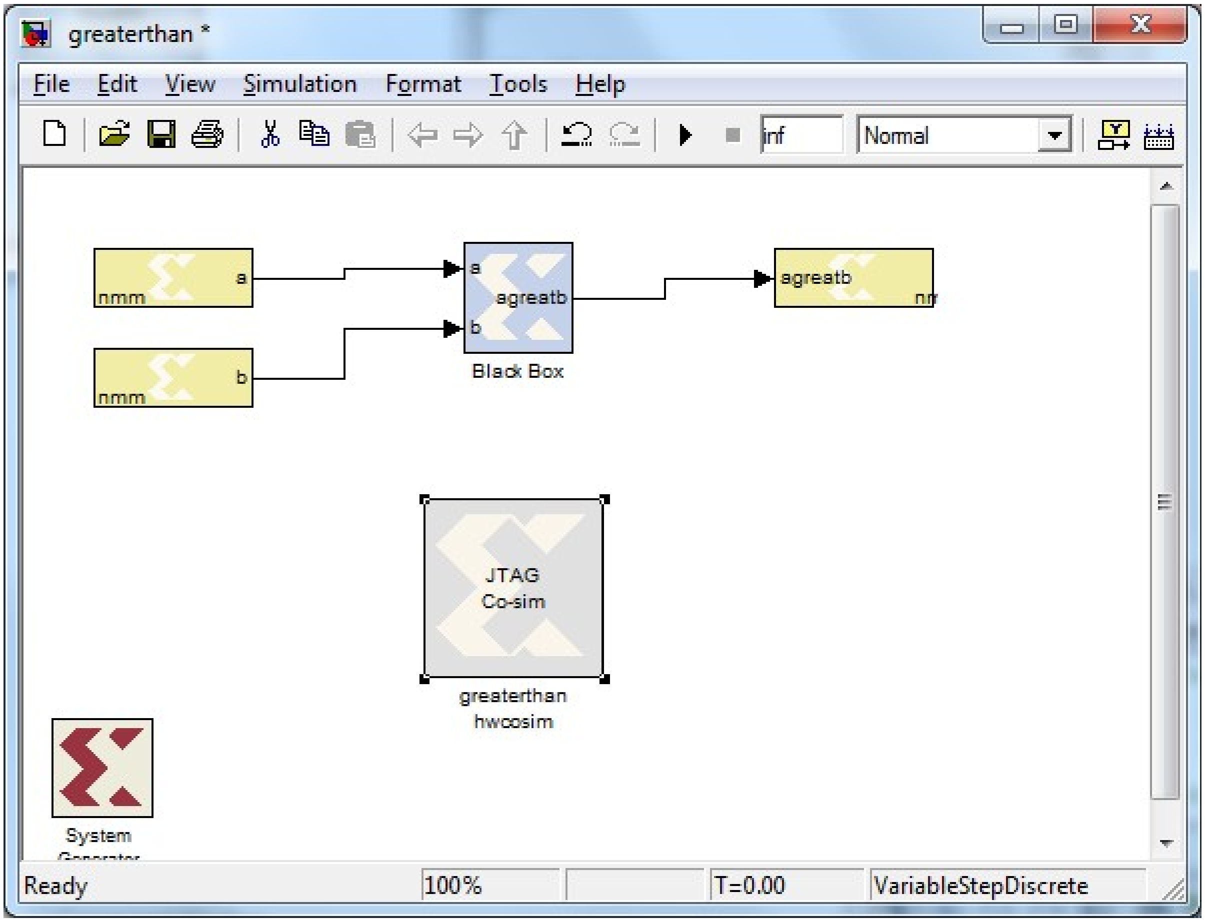
Set the stop time to: inf

Set Solver to : discrete

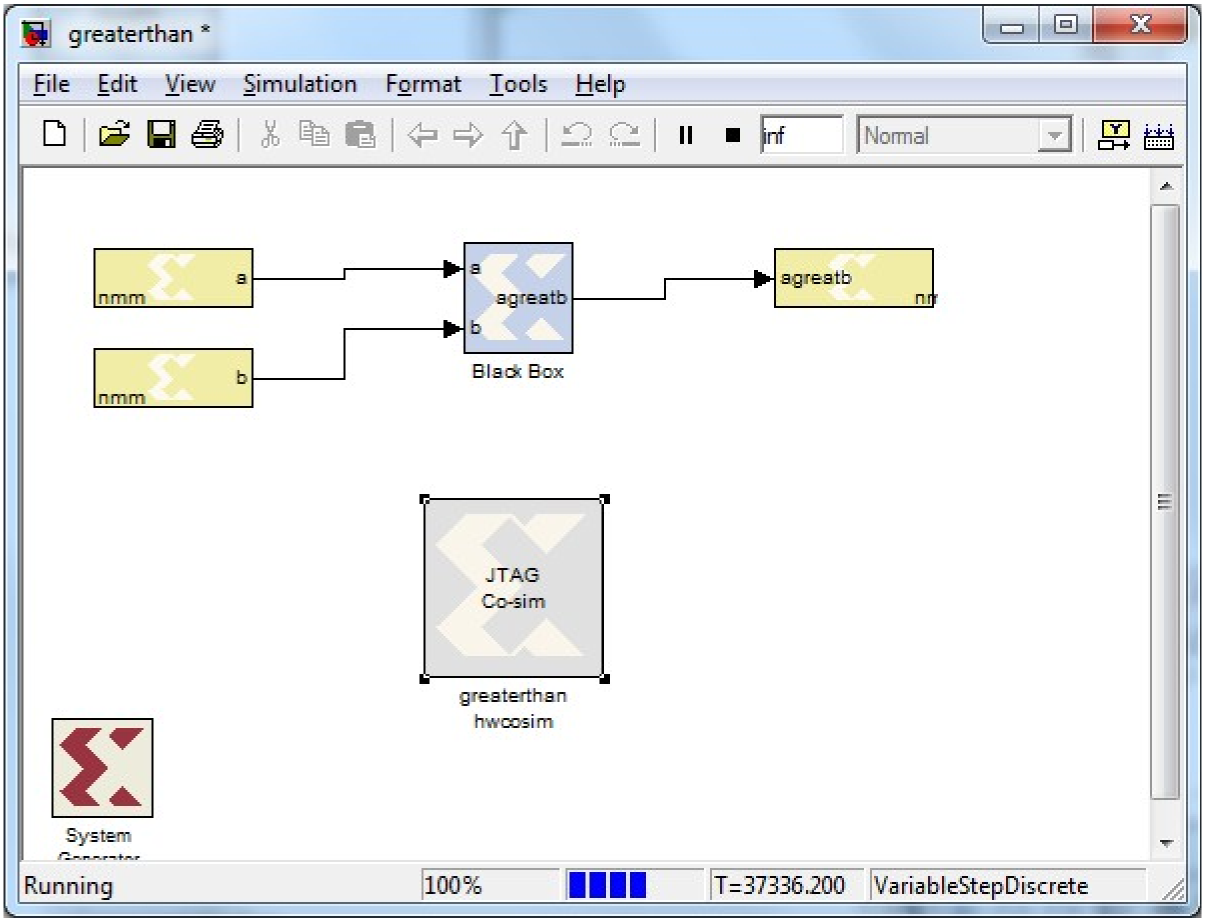

Configuration Parameters: greaterthan/Configuration (Active)
Select:
Solver
Data Import/Export
Optimization
Diagnostics
Sample Time
Data Validity
Type Conversion
Connectivity
Compatibility
Model Referencing
Saving
Stateflow
Hardware Implementati...
Model Referencing
Simulation Target
Symbols
Custom Code
Real-Time Workshop
Report
Comments
Symbols
Custom Code
Debug
Interface
Simulation time
Start time: 0.0
Stop time: inf
Solver options
Type: Variable-step
Solver: discrete (no continuous states)
Max step size: auto
Tasking and sample time options
Tasking mode for periodic sample times: Auto
Automatically handle rate transition for data transfer
Higher priority value indicates higher task priority
Zero-crossing options
Zero-crossing control: Use local settings
Algorithm: Nonadaptive
Time tolerance: 10*128*eps
Signal threshold: auto
Number of consecutive zero crossings: 1000
OK
Cancel
Help
Apply

Click ok.

Make sure your board is on then in your model editor click the play button.



Simulink will now program your FPGA (this may take a while). Wait until the model editor screen says running..



Then you can toggle the GPIO switches to test the greater-than circuit, a picture is shown below:

Illustration 2: a = '10' = 2 ; b = '01' = 1 ; thus a > b

You can further customize this by adding more blocks to manipulate the signal. Here are some more complex design tutorials using a MAC FIR from Xilinx:

http://www.xilinx.com/products/boards/ml506/ml506_12.1/dsp.htm

Xilinx also provides a set of System Generator tutorials located in the installation directory. Mine is in

C:\\Xilinx\12.1\ISE_DS\ISE\sysgen\examples

Using Xilinx XPS and SDK to implement serial communication using the RS232 cable and a terminal program

Summary

This tutorial will show you how to use Xilinx XPS base system builder to create a base system package that connects the FPGA to its board peripherals such as the UART serial interface, GPIO LEDs, etc. After building the base system we will export it to Xilinx XPS where we can write the software to test the UART interface.

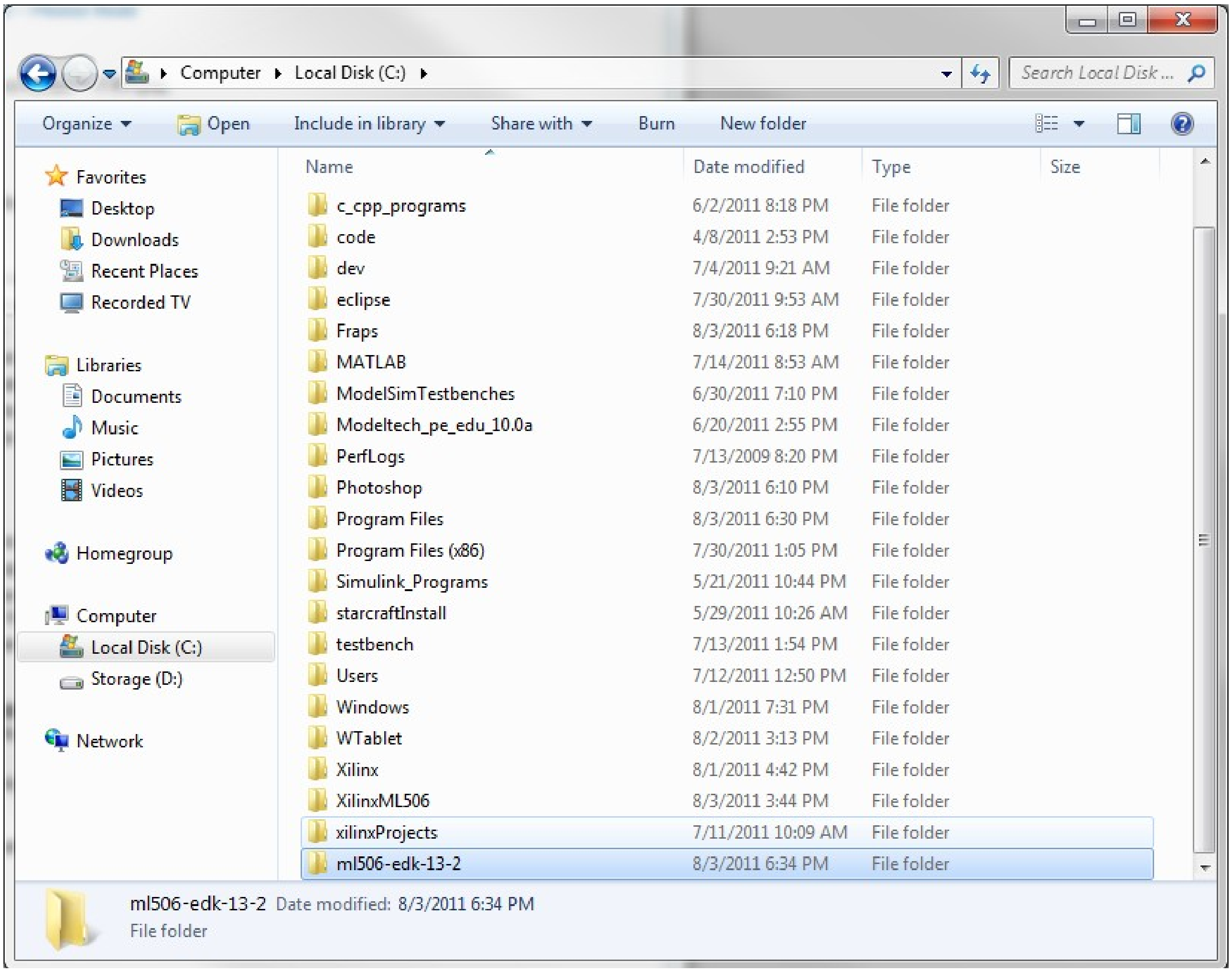
Note: I have been experiencing some crashes with the Xilinx SDK version 12.1 in Windows 7. Version 13.1 and later is supposed to support Windows 7, so for this tutorial I will use Xilinx EDK version 13.2.

Xilinx XPS

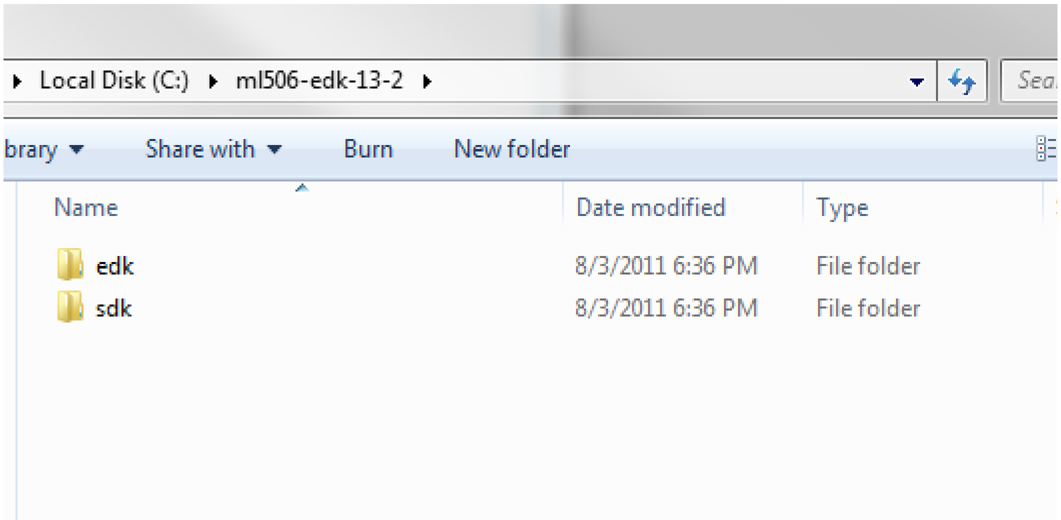
First we will use XPS and the system builder wizard to add peripherals to our FPGA. This is done by creating a soft microprocessor to handle peripheral interfacing and communicate with the FPGA.

First we will create a file system to organize our project files. Create a folder (I put mine in the local drive) and name it (make sure there are no spaces):

ml506-edk-13-2

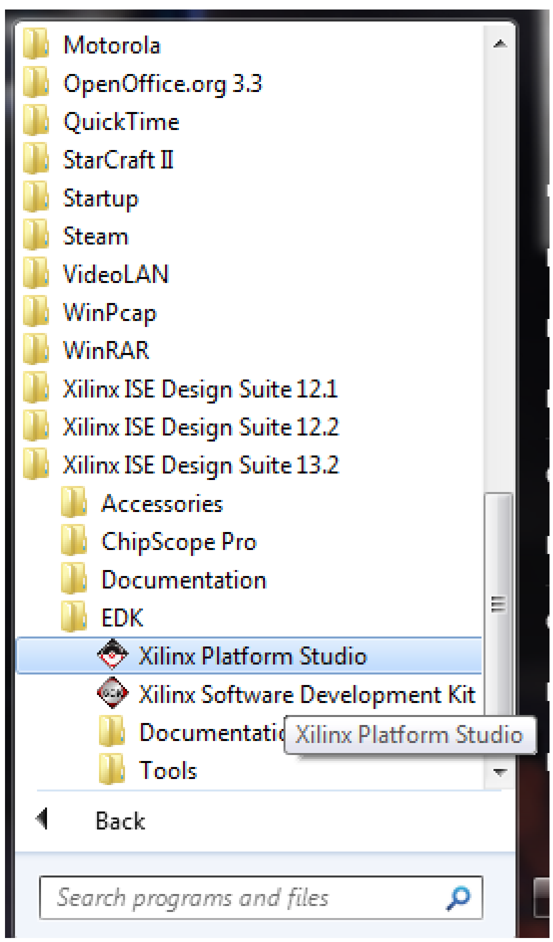


Now within this folder make two empty folders name: edk, sdk

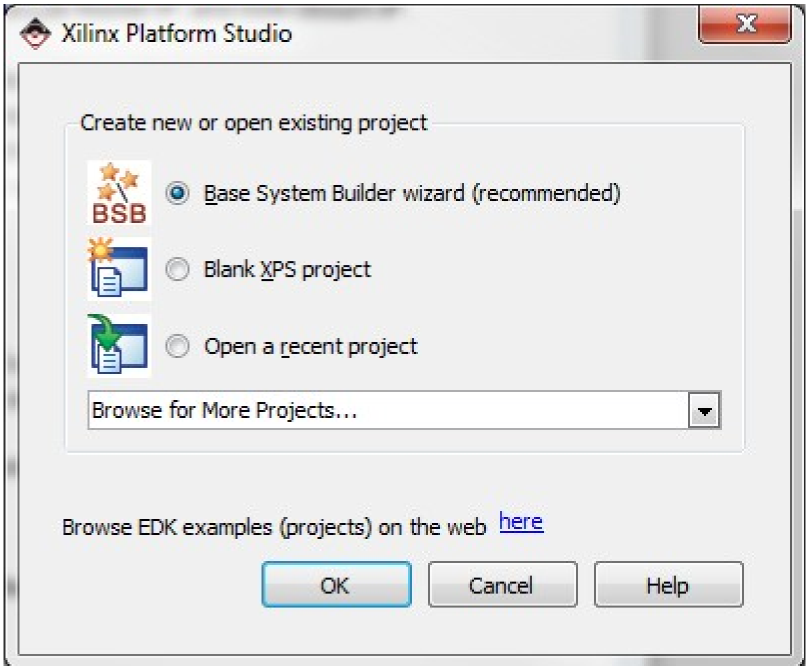


We will store the board configuration files in the edk folder and the software files in the sdk folder.

Open up Xilinx XPS (if you are using Windows 7, use Version 13.1 or later)

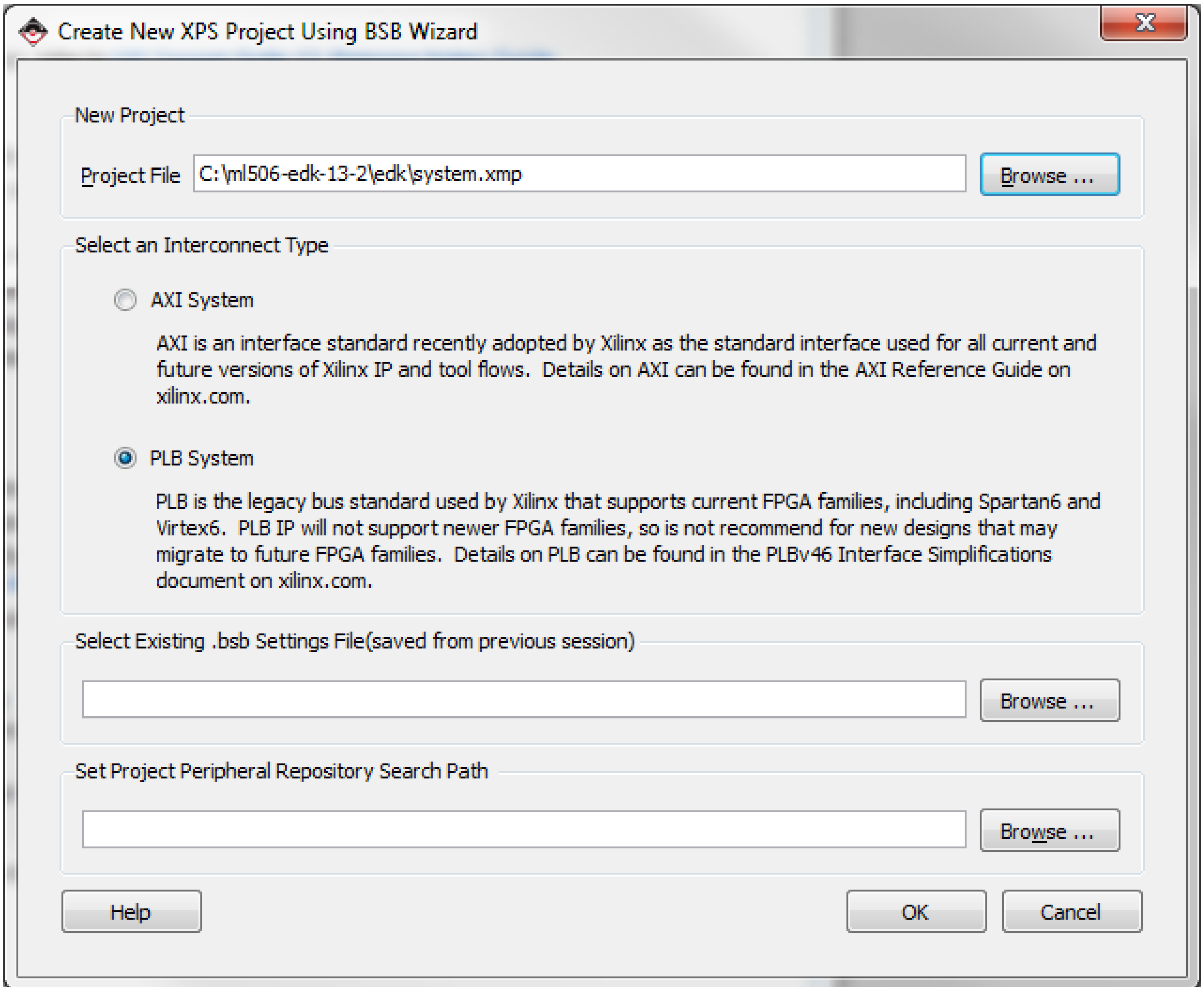


Choose Base System Builder wizard:

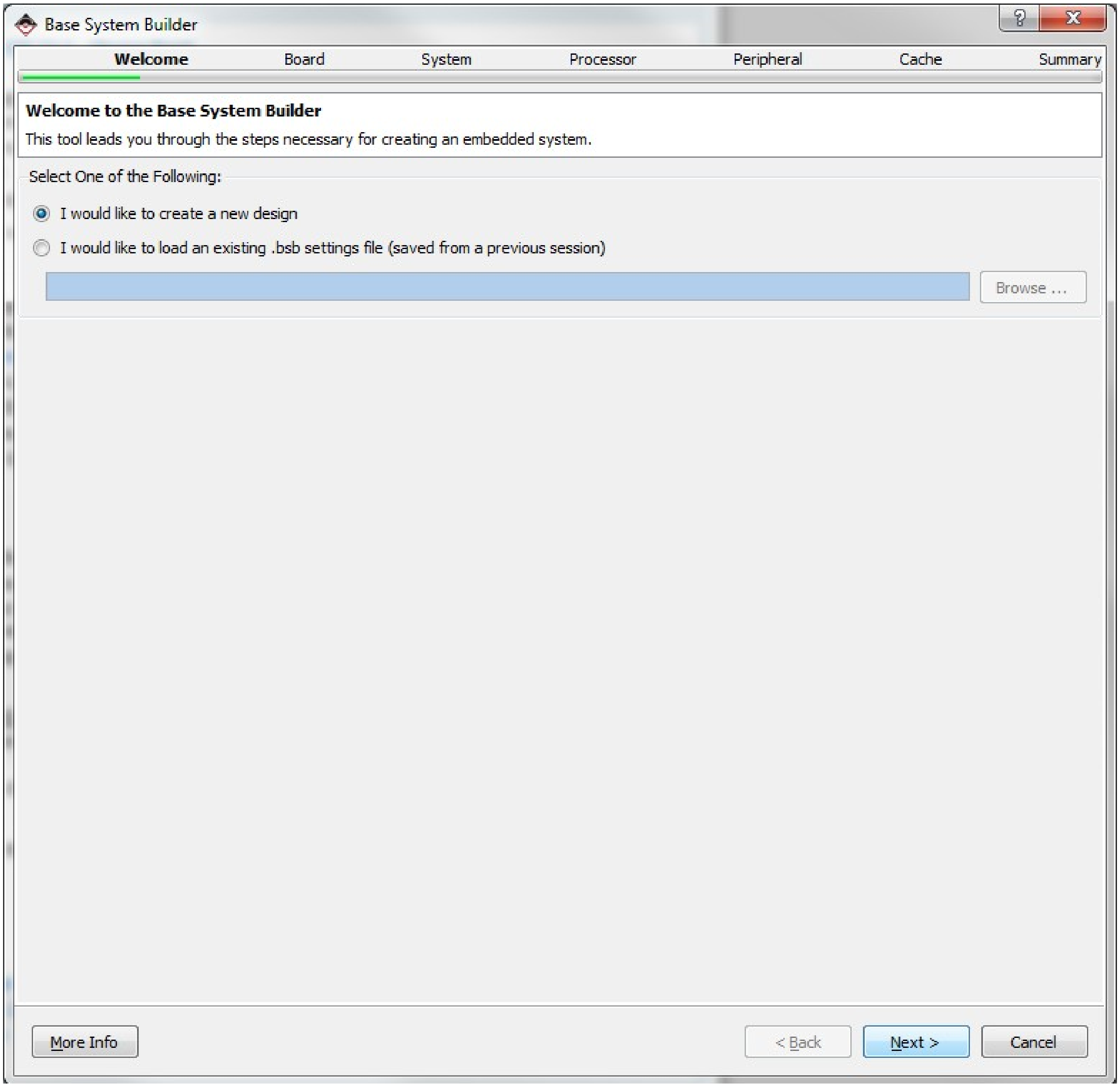


In the Project File section point it to the edk folder we created earlier and for the interconnect type choose PLB. For newer models of FPGA choose the AXI.

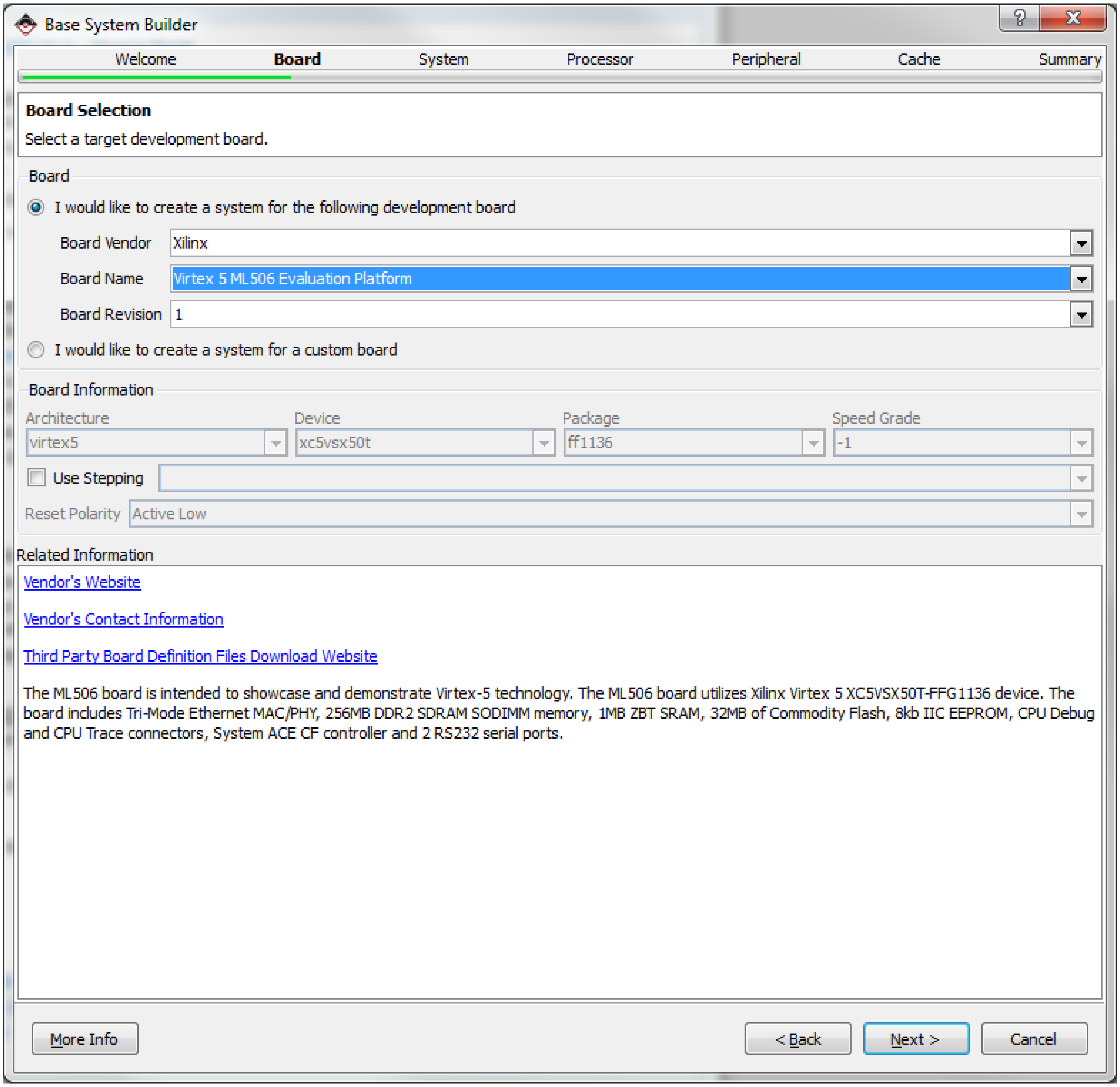
Click OK.



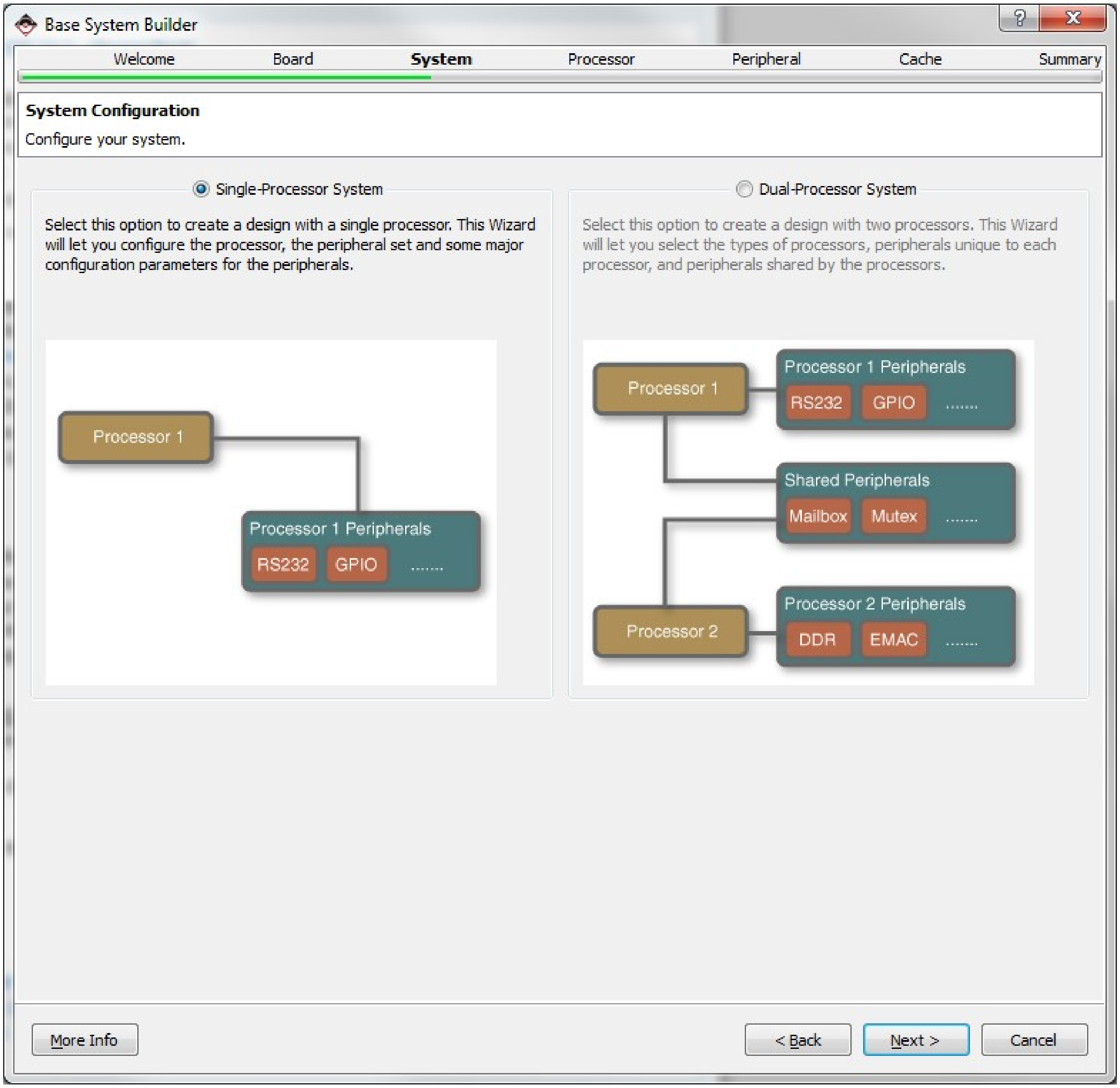
Choose new design and click Next.



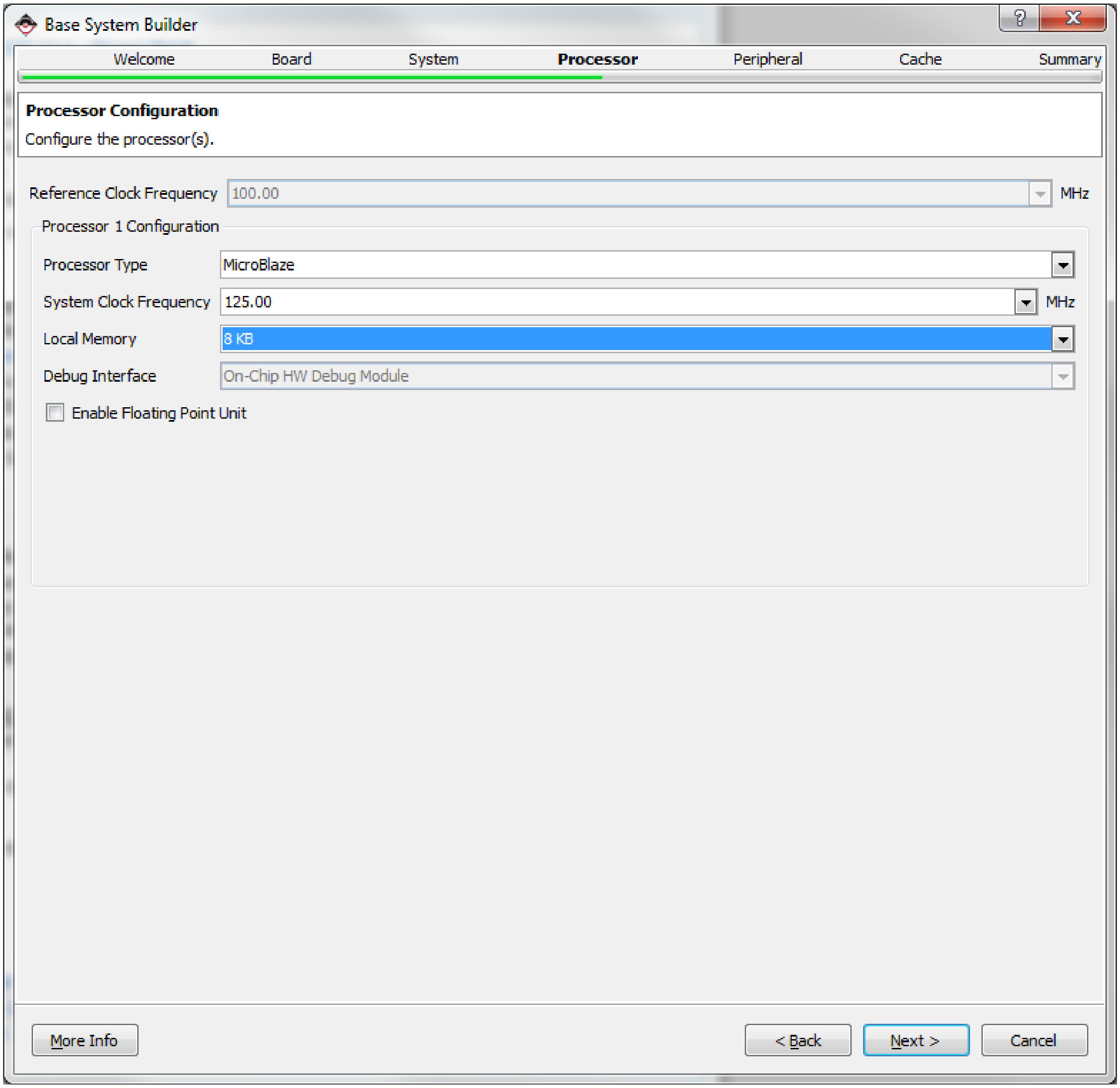
Choose the development board, in our case it is a Virtex 5 ML506 Evaluation Platform and click next.



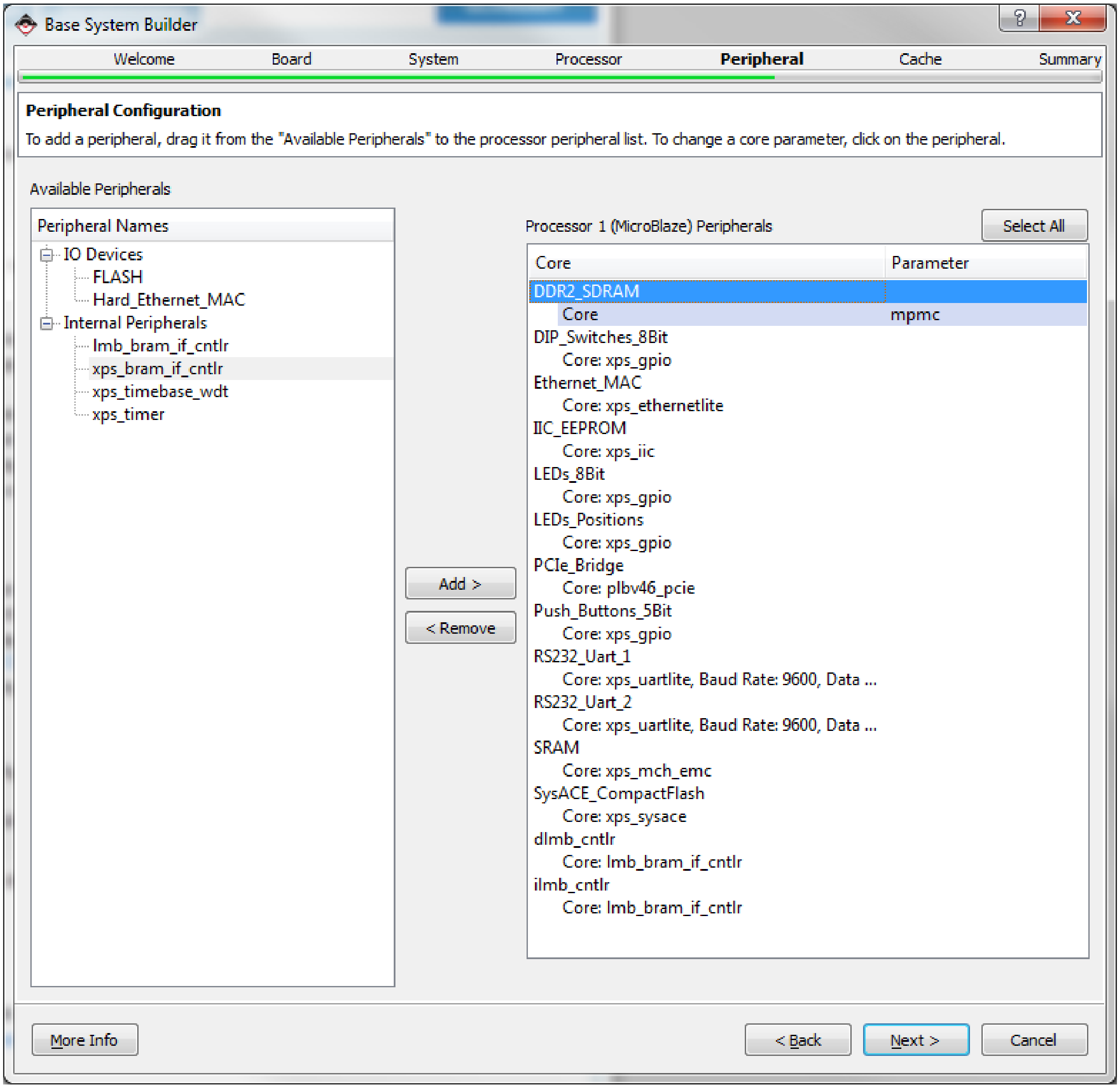
The Base System Builder will create a soft processor for interfacing with the peripherals, choose Single-Processor System and click Next.



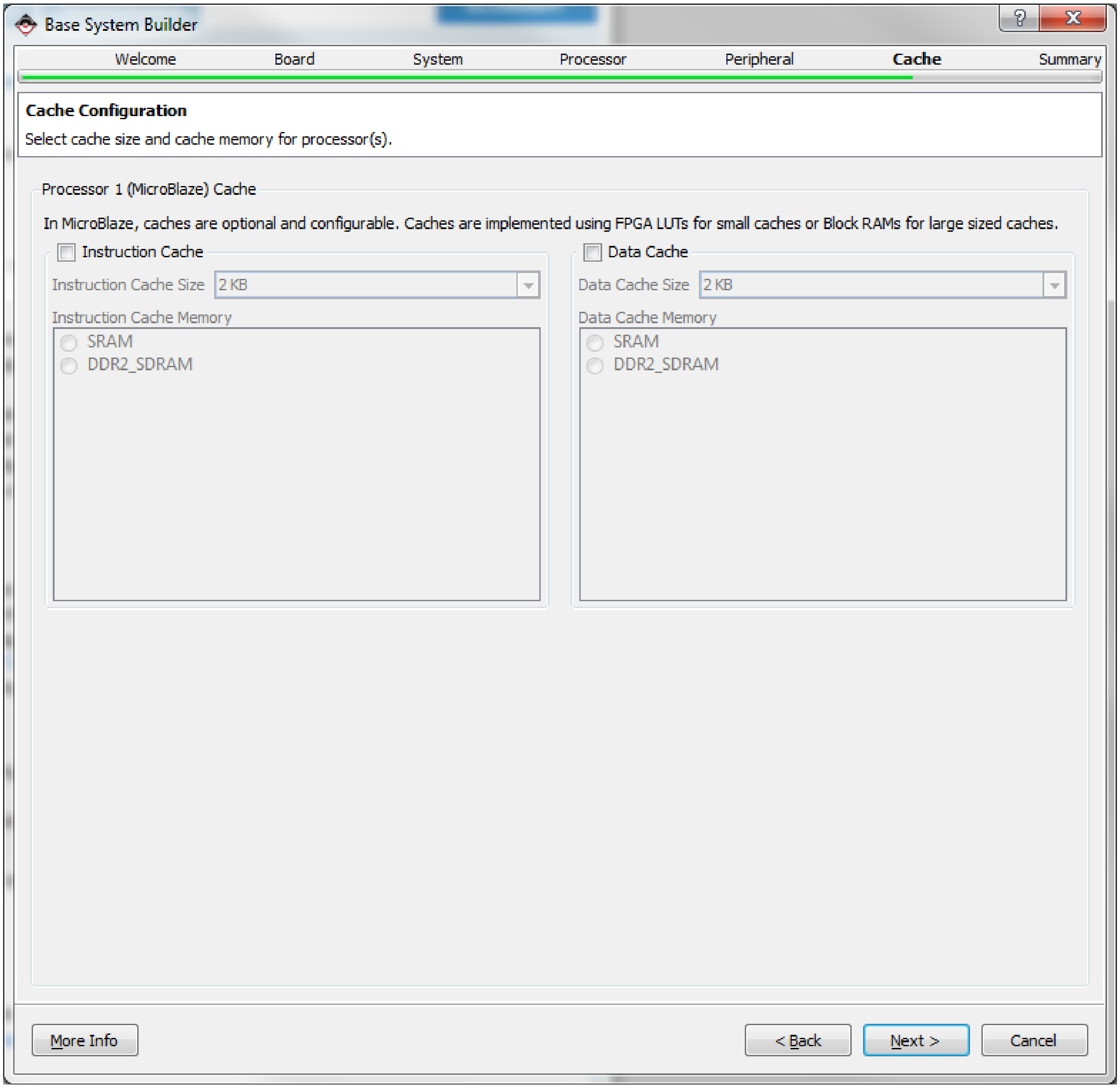
Choose these default values for the processor specs and click Next.



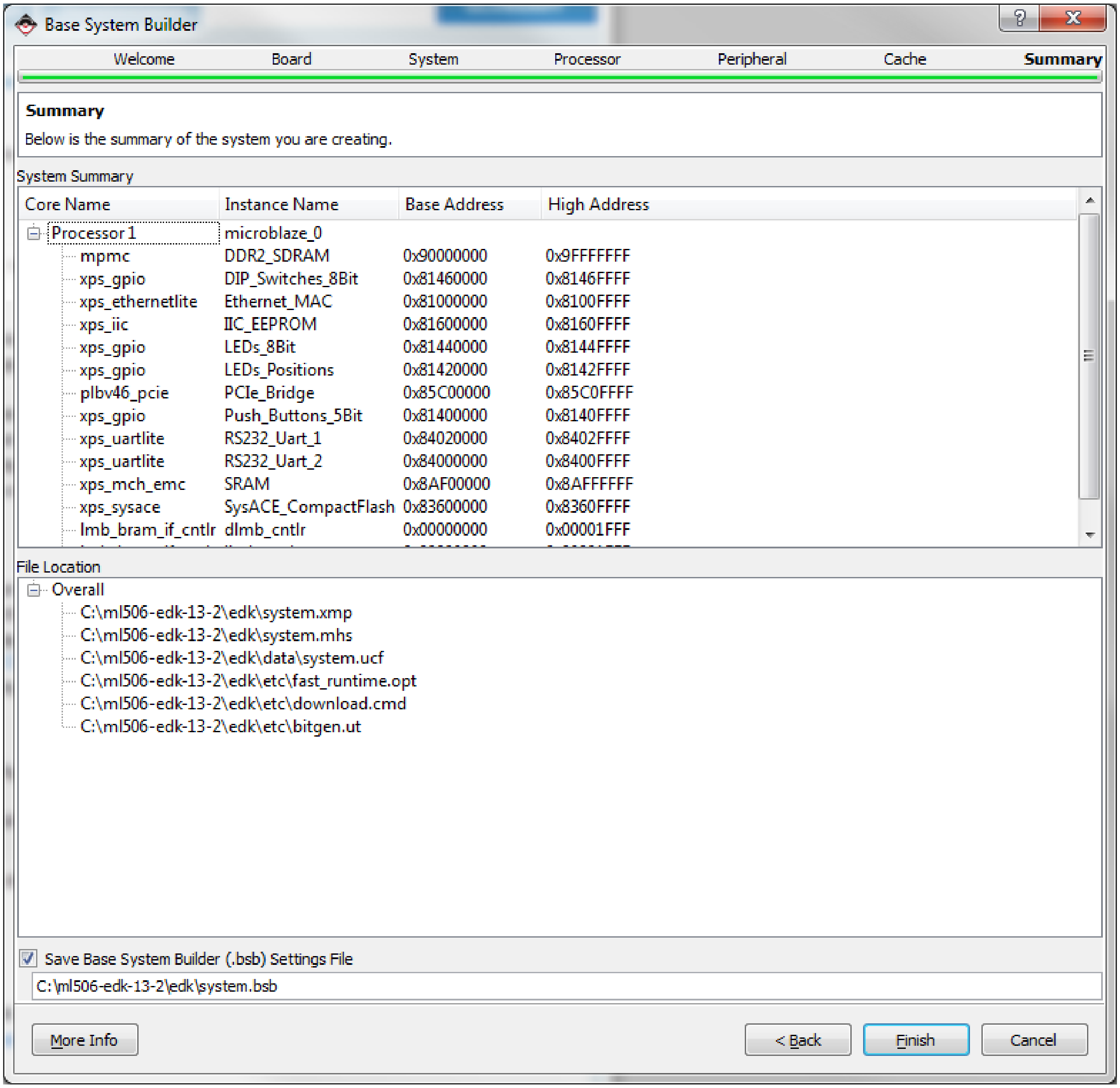
In the Peripheral window, leave it default. As you can see there are many peripherals connected to the MicroBlaze soft processor. We only need to create a base system once and we can reuse it for different projects so we might as well connect all the peripherals in case we need to use one for another project. Click Next.



Click Next on the Cache window as we won't use any.

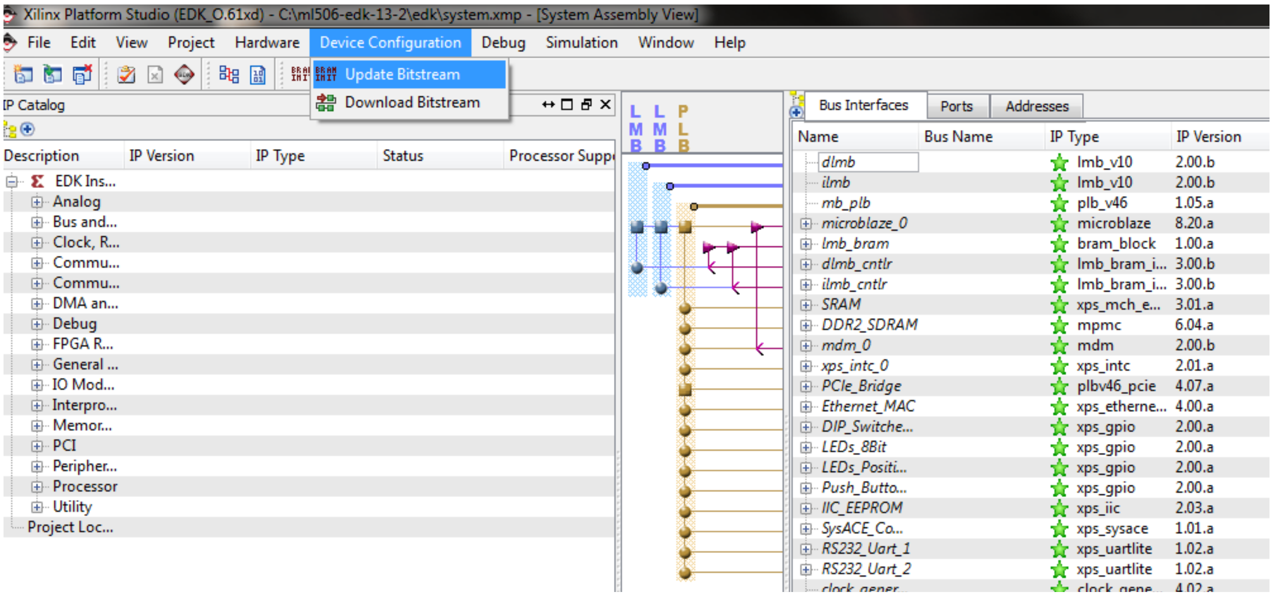


The next page is the summary of our design, review it if you like and click Finish. (Make sure to check the Save Base System Builder Settings File box).

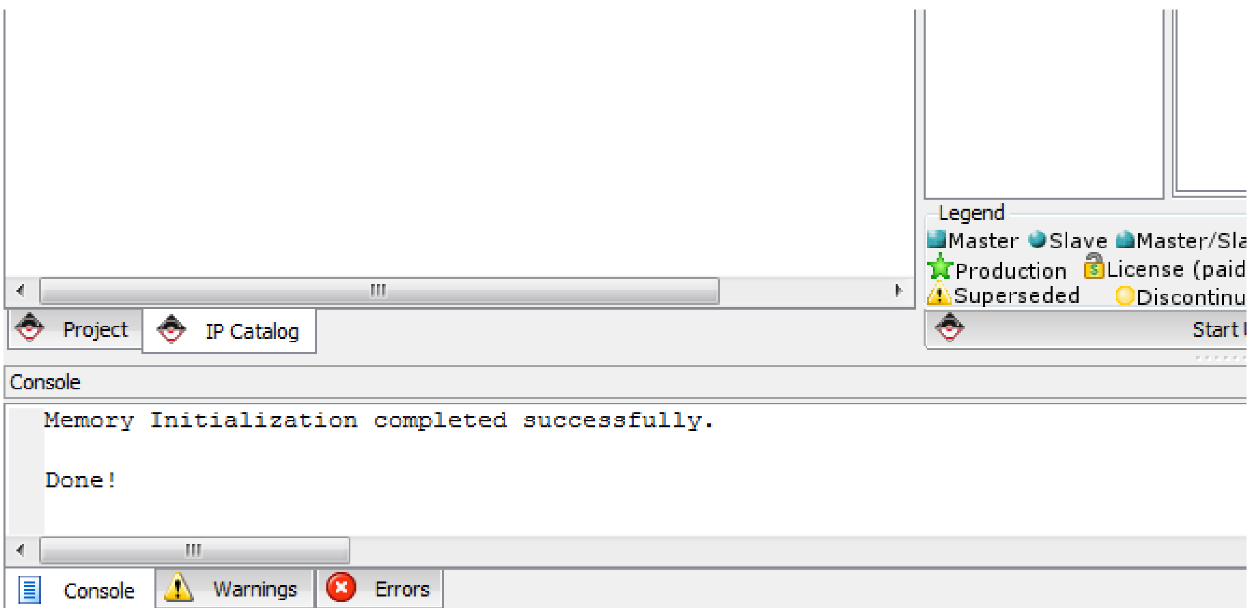


The base system is now finished but before we can write some software for it we have to update the bitstream and export the board configurations to Xilinx SDK.

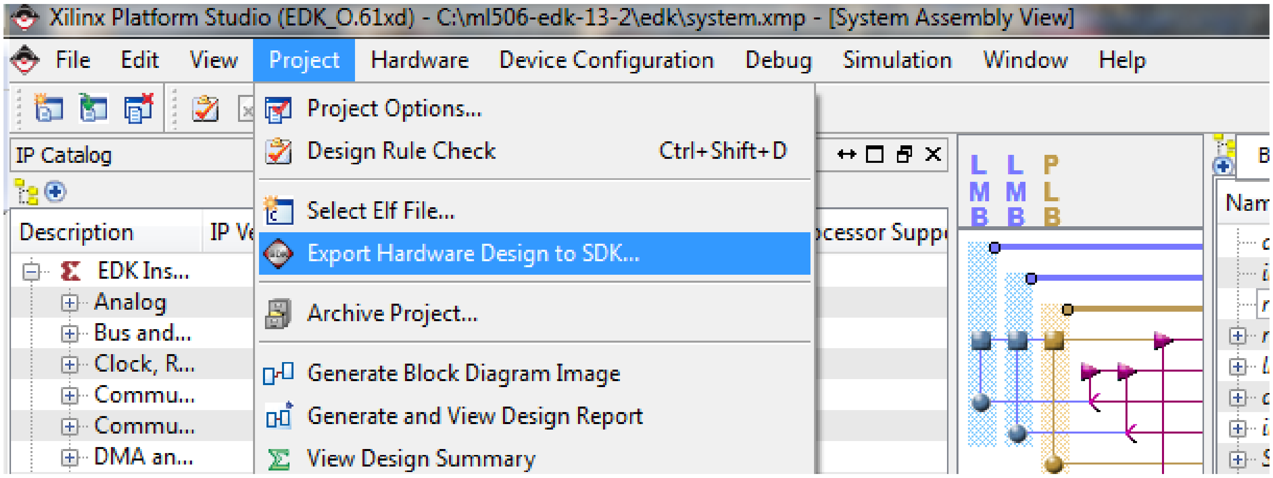
Under Device Configuration click Update Bitstream, this process takes a long time (around half an hour).



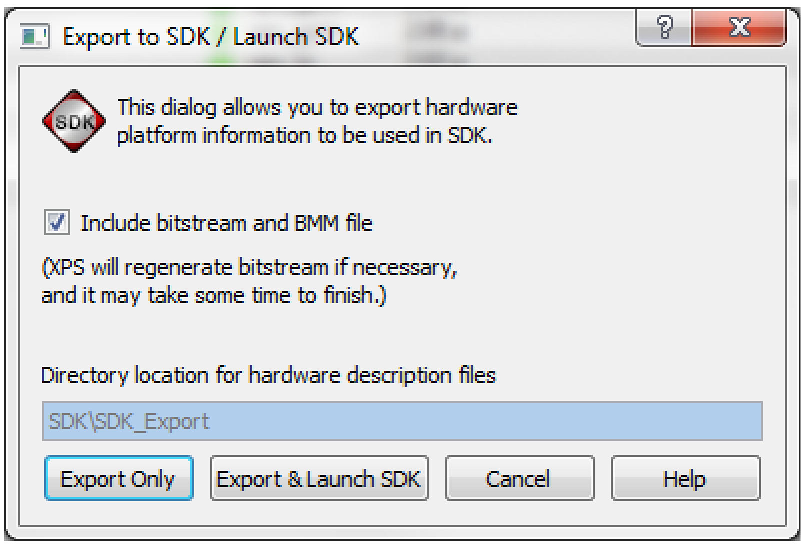
The console will say done when the bitstream update is complete:



To export this project to the SDK, click Project → Export Hardware Design to SDK..



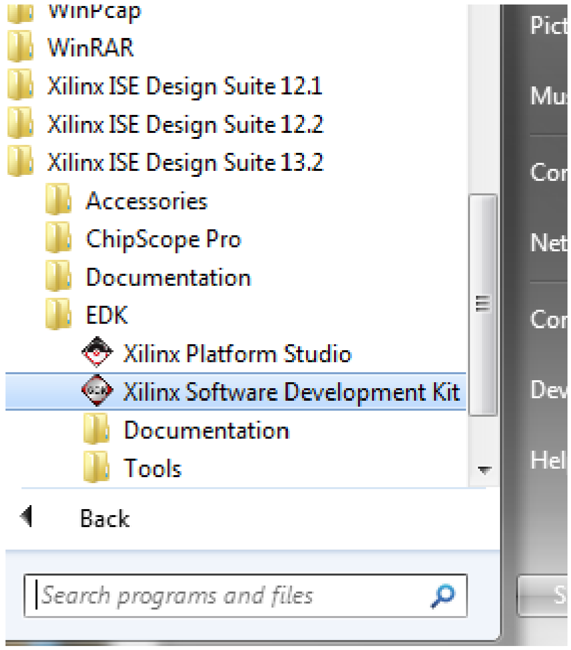
Check the box to include the bitstream file we just generated and the BMM file. Then click Export Only: (This will created some files located in C:\ml506-edk-13-2\edk\SDK\SDK_Export\hw)



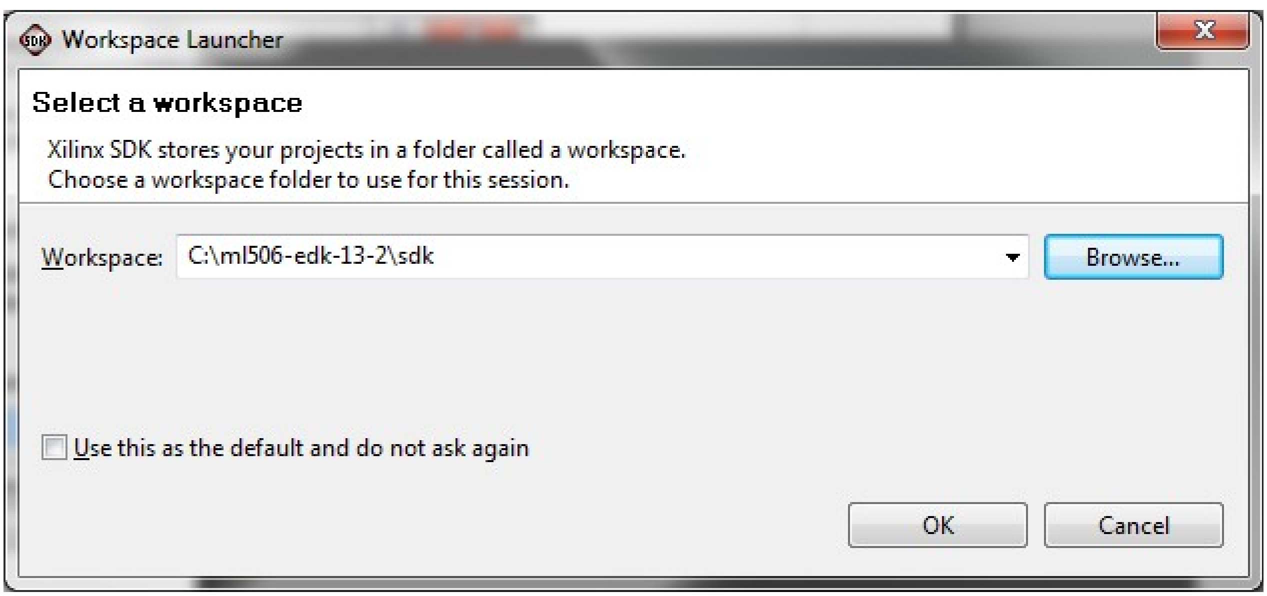
This will enable the Xilinx SDK to use the hardware configuration we just described and create software to utilize them.

Xilinx SDK

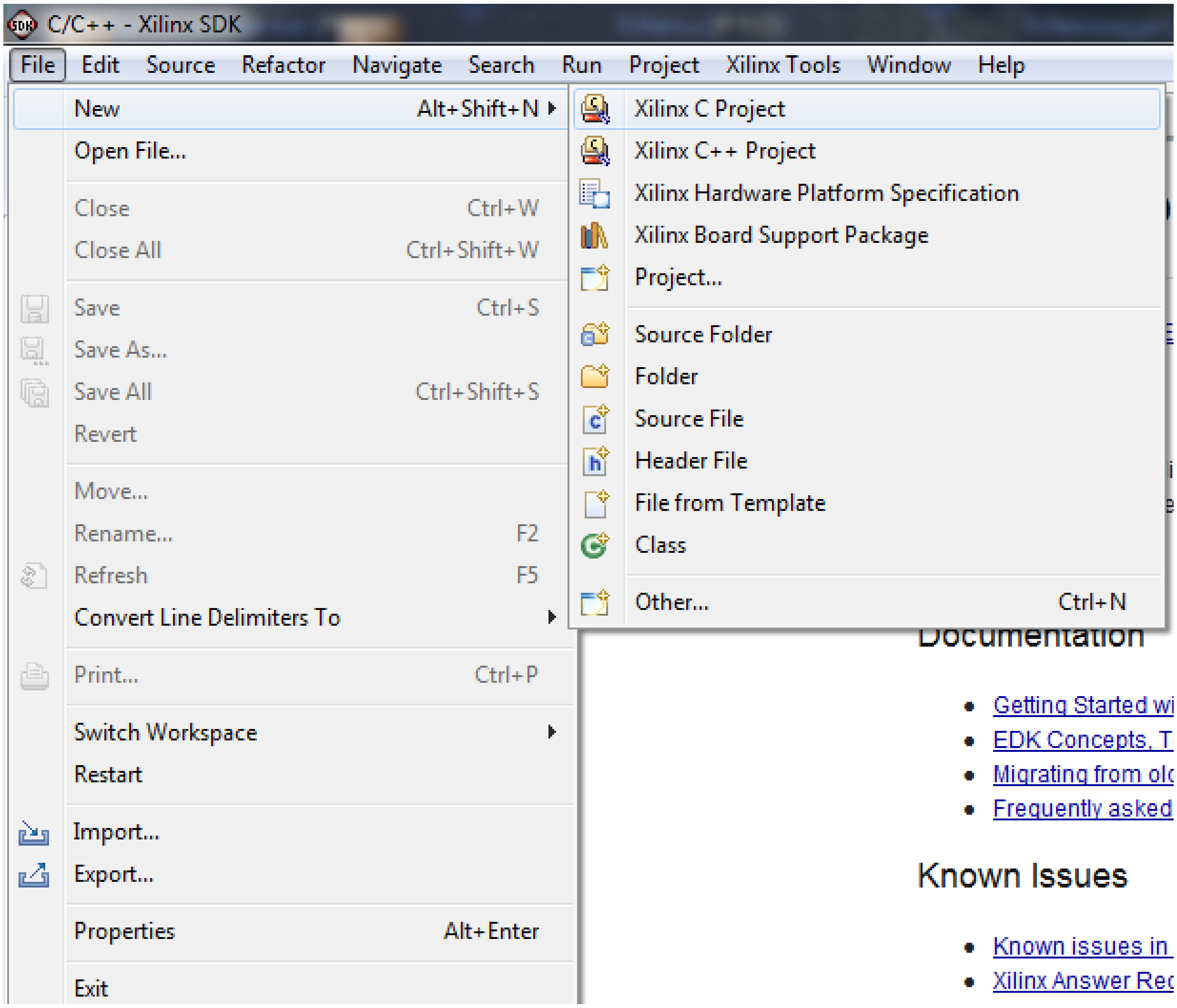
Close Xilinx XPS and open Xilinx SDK:



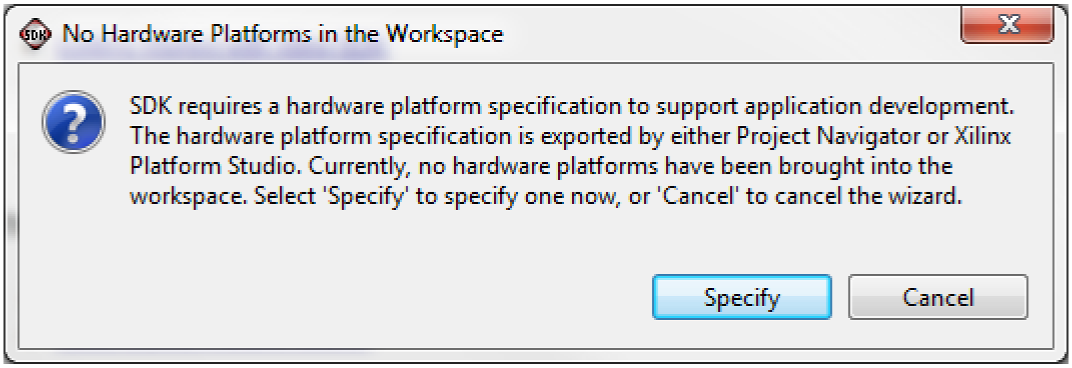
Point your workspace to the sdk folder we created at the beginning of this tutorial and click OK:



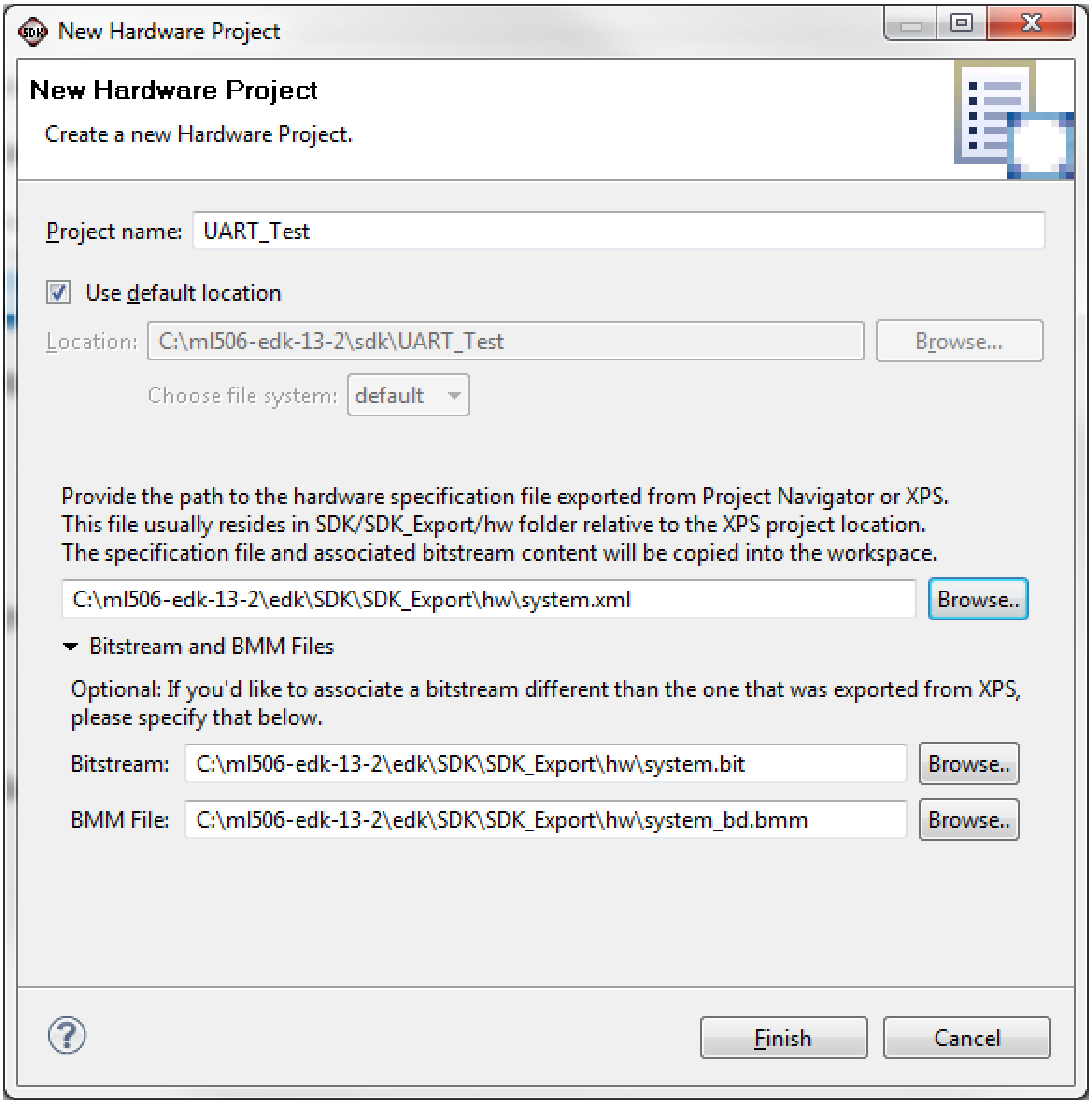
Click File → New → Xilinx C Project:



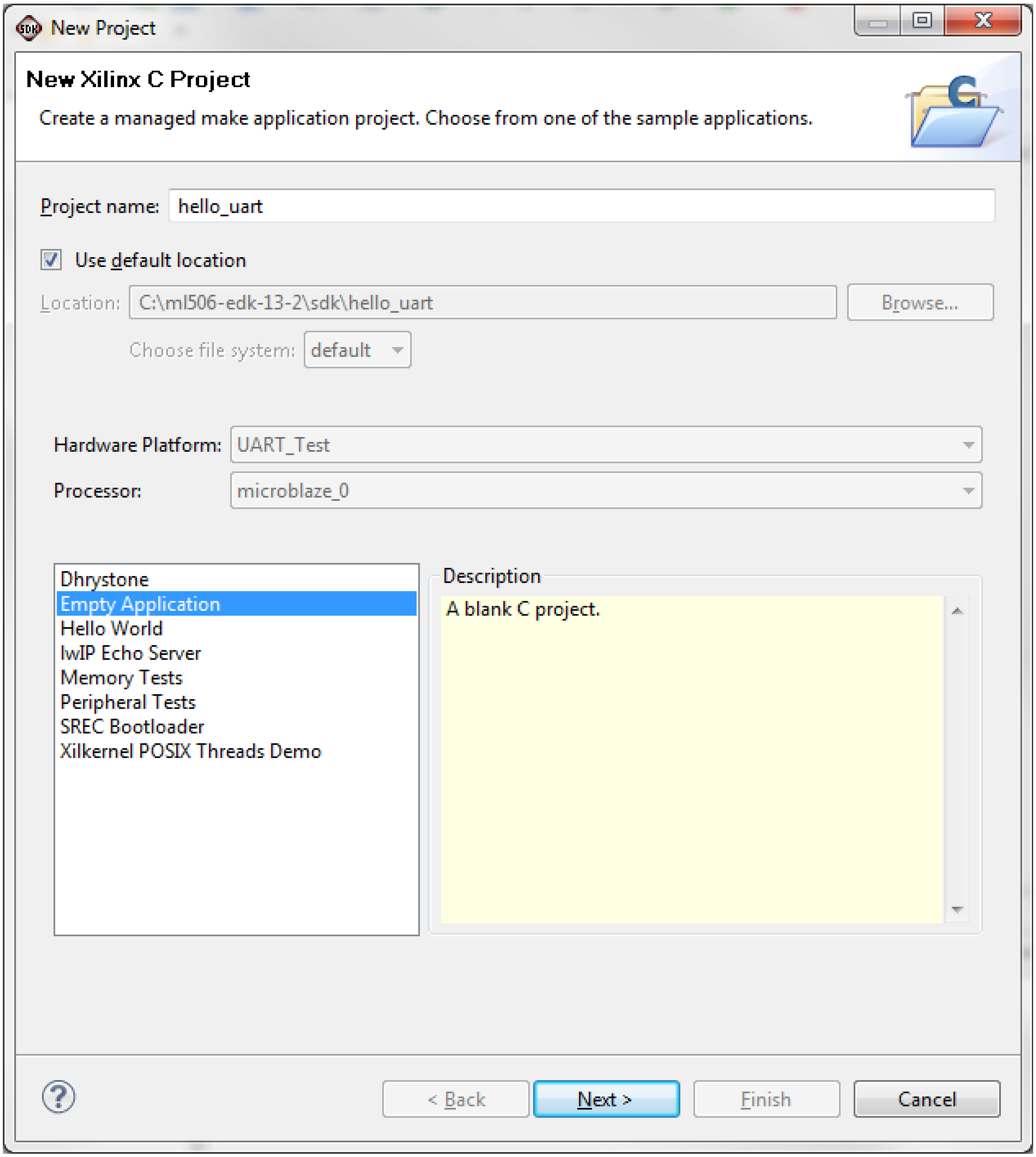
The Xilinx SDK will prompt you to specify a hardware platform. This is the file we exported in XPS and it contains all the hardware info of our board. Click Specify:



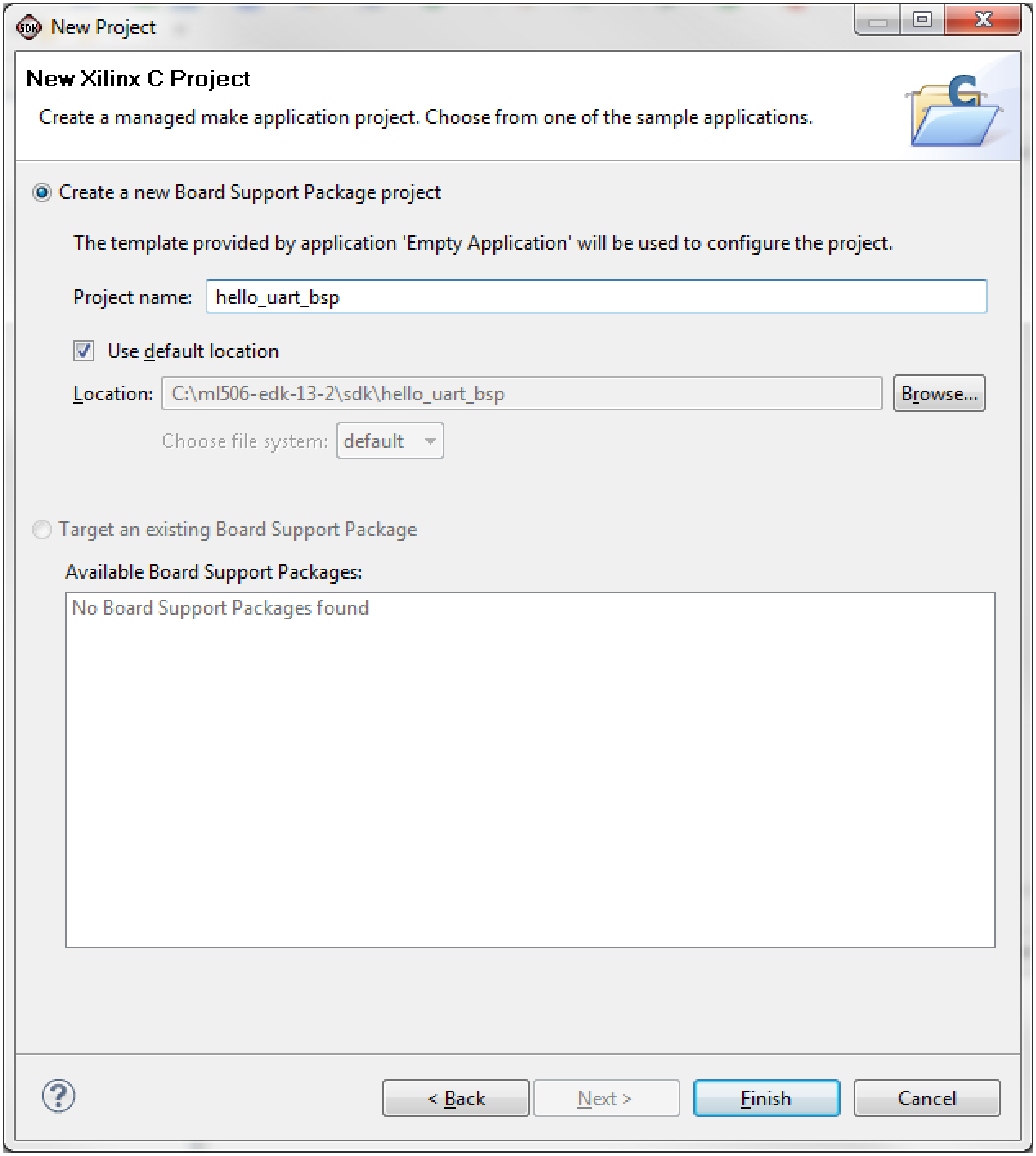
Name the project UART_Test and point the hardware specification entry to the system.xml file that we exported. The bitstream and BMM entries should automatically fill after setting the system.xml entry: Click Finish



In the next window, name the project hello_uart and select the Empty Application template, then click Next:



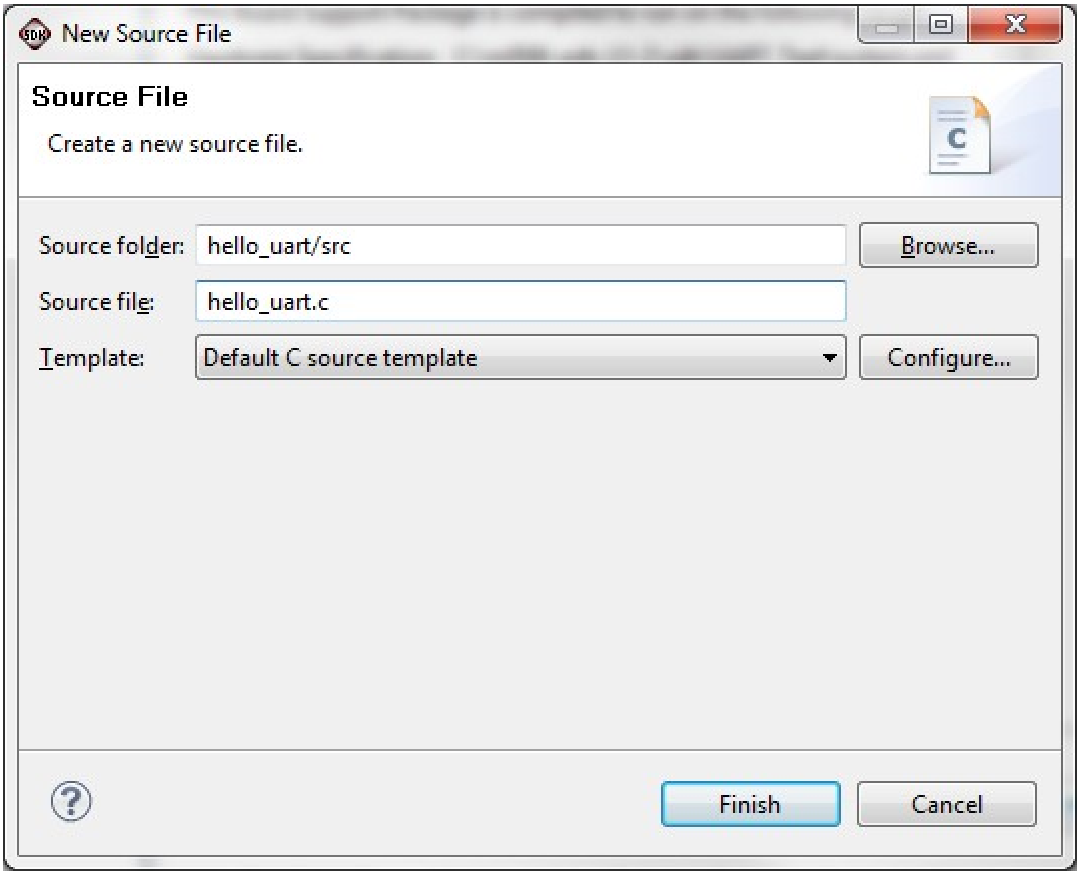
Then name the board support package hello_uart_bsp and click Finish:



Xilinx SDK will create a template for your new C project.

Expand the 'src' folder under the 'hello_uart' project folder, then right-click on 'src' and click New → Source File.

Name the source file hello_uart.c and click Finish:

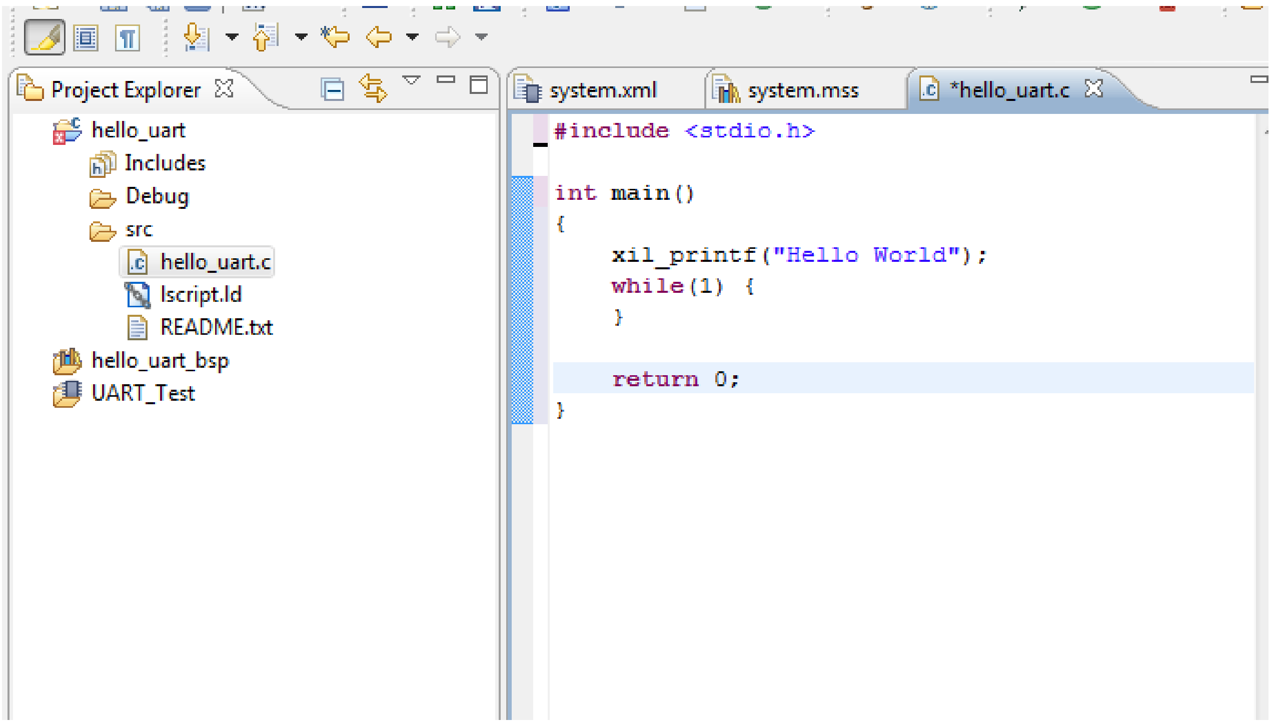


A Xilinx text editor should come up (if it doesn't, double click hello_uart.c in the Project Explorer windows), delete all the auto generated comments and type in this code:

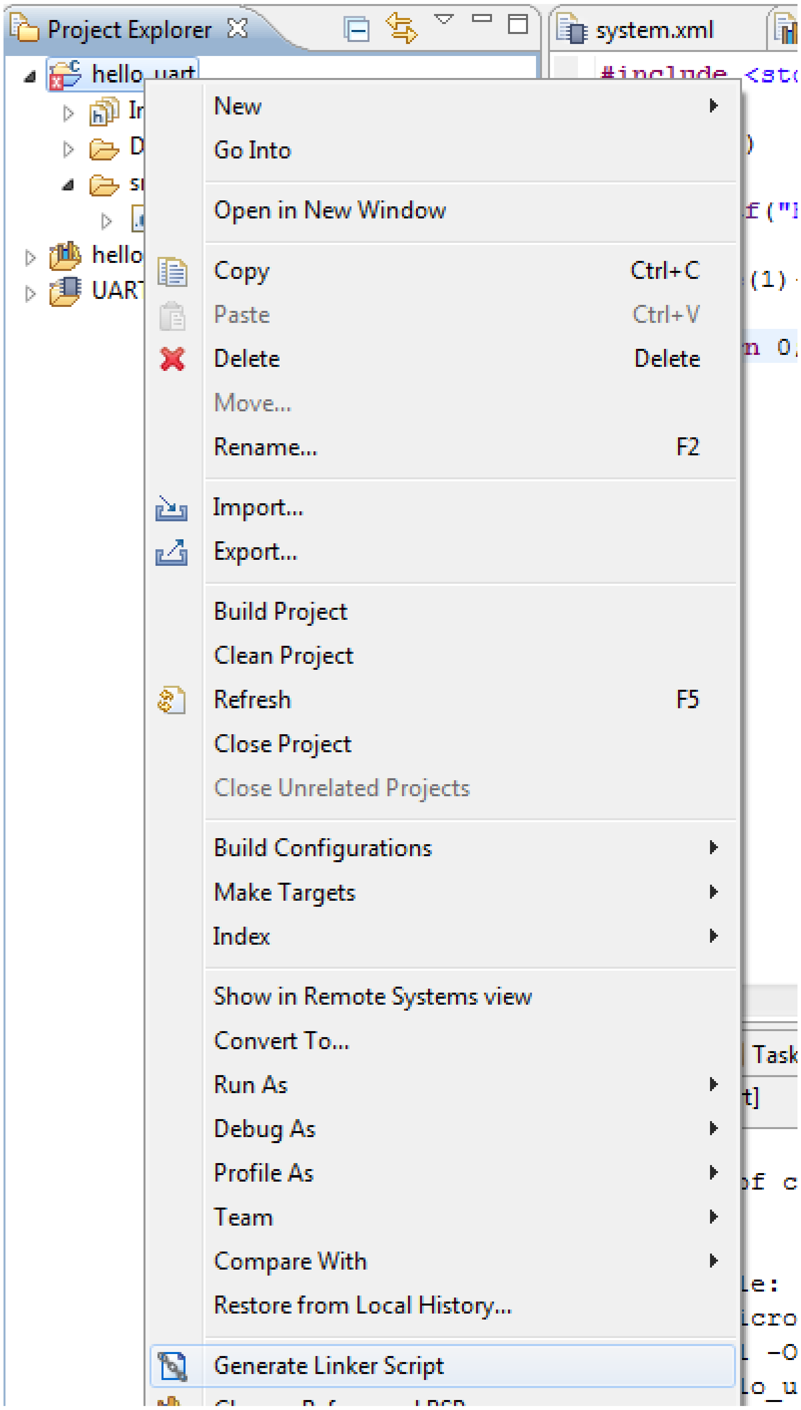
```c
#include <stdio.h>

 int main()
 {
        xil_printf("Hello World");
        while(1) {
        }

        return 0;
 }
```

Note: use xil_printf() when you can, printf () takes up too much memory space.

Click save and Xilinx SDK should automatically build and compile your code and create a file called hello_uart.elf which you can use to program the FPGA.

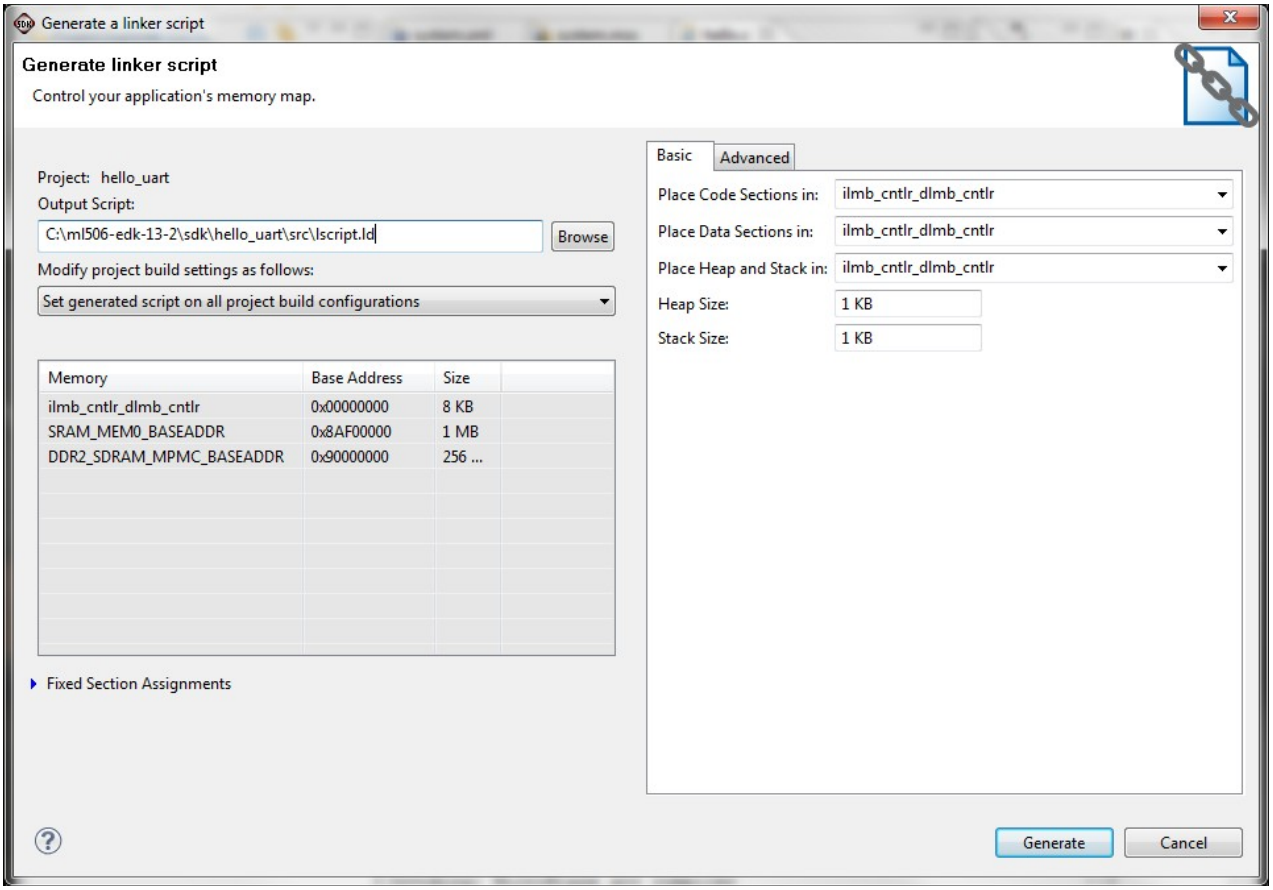


Now right-click on the hello_uart project and click Generate Linker Script



Leave all the default settings and click Generate

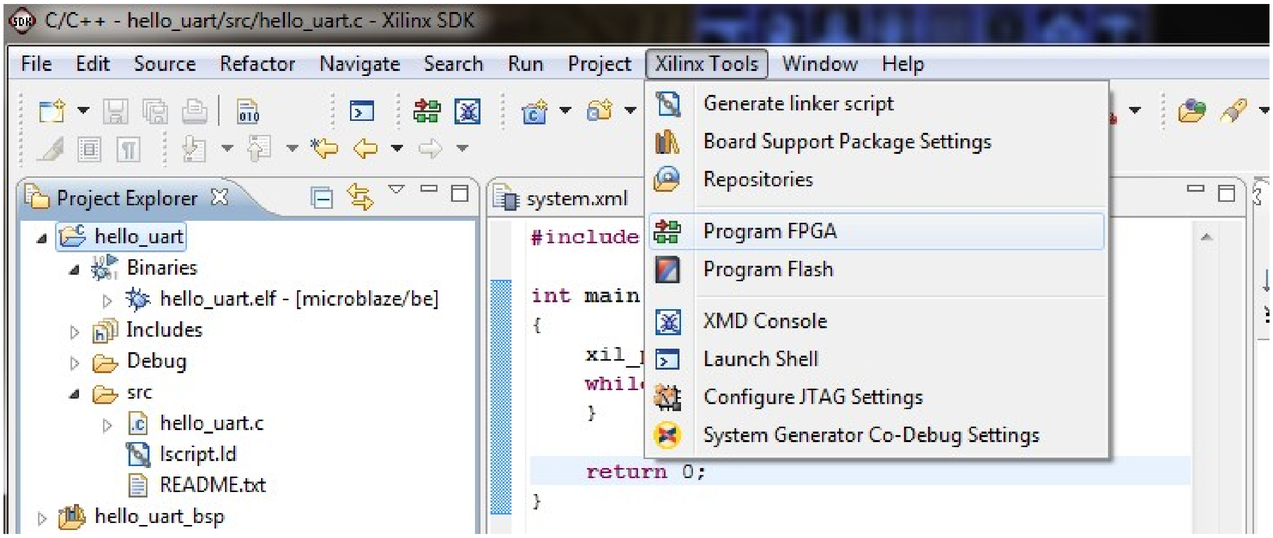
Note that if your .elf file is too big, you might have to increase the Heap and Stack size to accommodate your code:



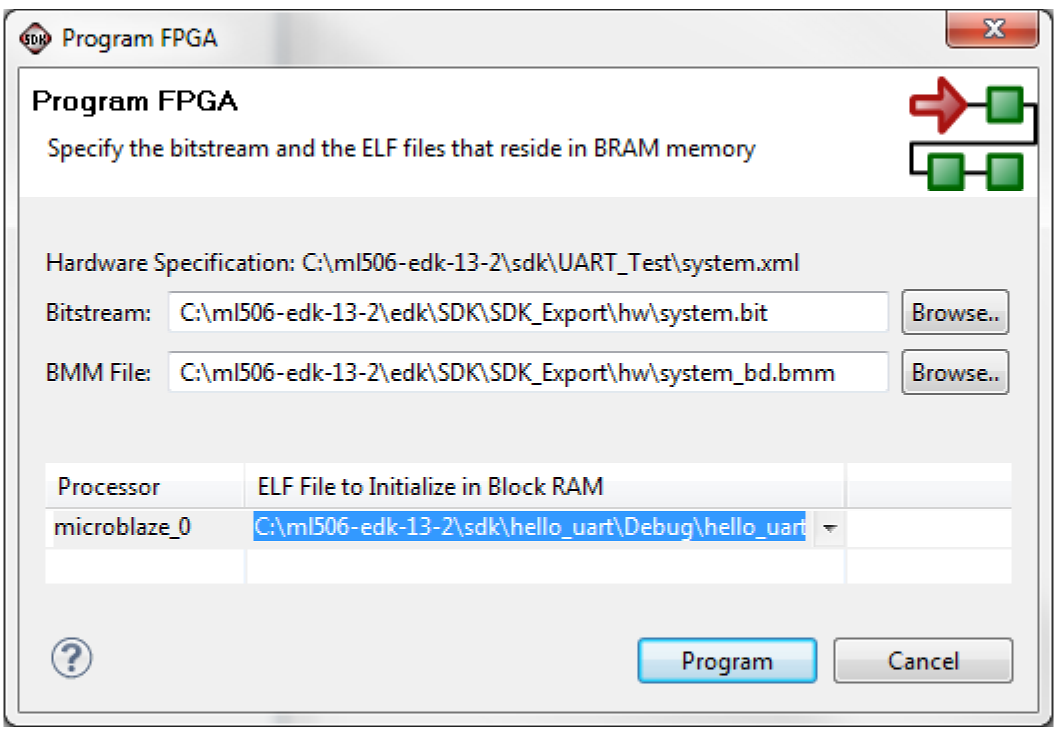
This .elf file, along with some system configuration bitstreams, is what you need to program the FPGA and run your software.

Turn on the ML506 and open up a serial terminal program such as HyperTerminal, TeraTerm, Putty etc.. Here I use TeraTerm, and set it up for serial communication on COM1 with 9600 baude:

Click on Xilinx Tools → Program FPGA:



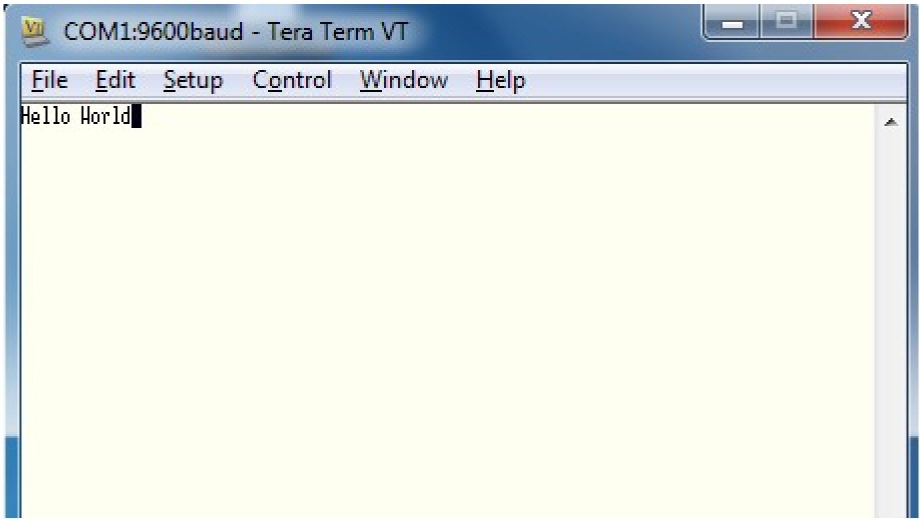
Point the Bitstream and BMM File entries to the files exported by Xilin XPS and under the microblaze_0 processor entry choose the hello_uart.elf file:



Click Program and Xilinx SDK will combine the .bit, .bmm, .elf, files into a single bitstream called 'download.bit' located in C:\ml506-edk-13-2\sdk\UART_Test.

I suggest you rename this file to something more meaningful like hellouart.bit.

The FPGA is now programmed and you can see the results in the terminal window:



Play sound with a sine wave using the AC97 codec

Summary

This tutorial makes use of the AC97 codec, LCD, UART (for text display) to play some sine waves through the line-in port of the AC97 interface. User instructions will be displayed to a COM serial window such as HyperTerminal or TeraTerm. Also some simple instruction will be displayed to the user via the on-board LCD screen.

To avoid reinventing the wheel, we will use some Xilinx pre-made board configuration files and software.

Xilinx XPS

In order to use the on-board AC97 codec we would have to read the datasheet for the codec available here:

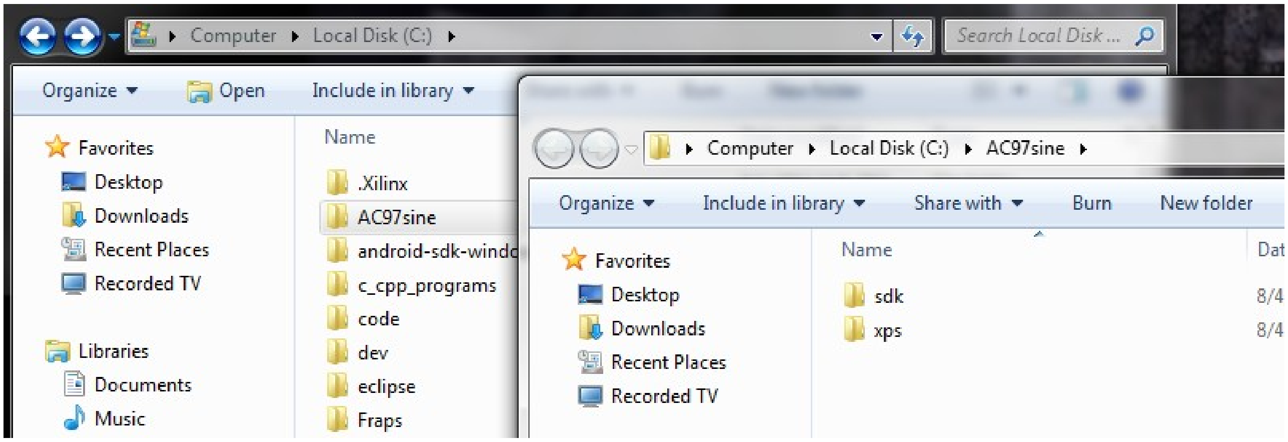
http://www.xilinx.com/products/boards/ml505/datasheets/87560554AD1981B_c.pdf

Then we would have to use our MicroBlaze softprocessor to setup AC97 according to its datasheet.

But Xilinx has already done this for use so we can just use theirs and tailor it to our needs.

First lets setup our file system to organize our project.

Create a folder on the local drive called: AC97sine then within this folder create two more named sdk and xps.



The Xilinx XPS files will be stored in the xps folder and the Xilinx SDK workspace will point to the sdk folder.

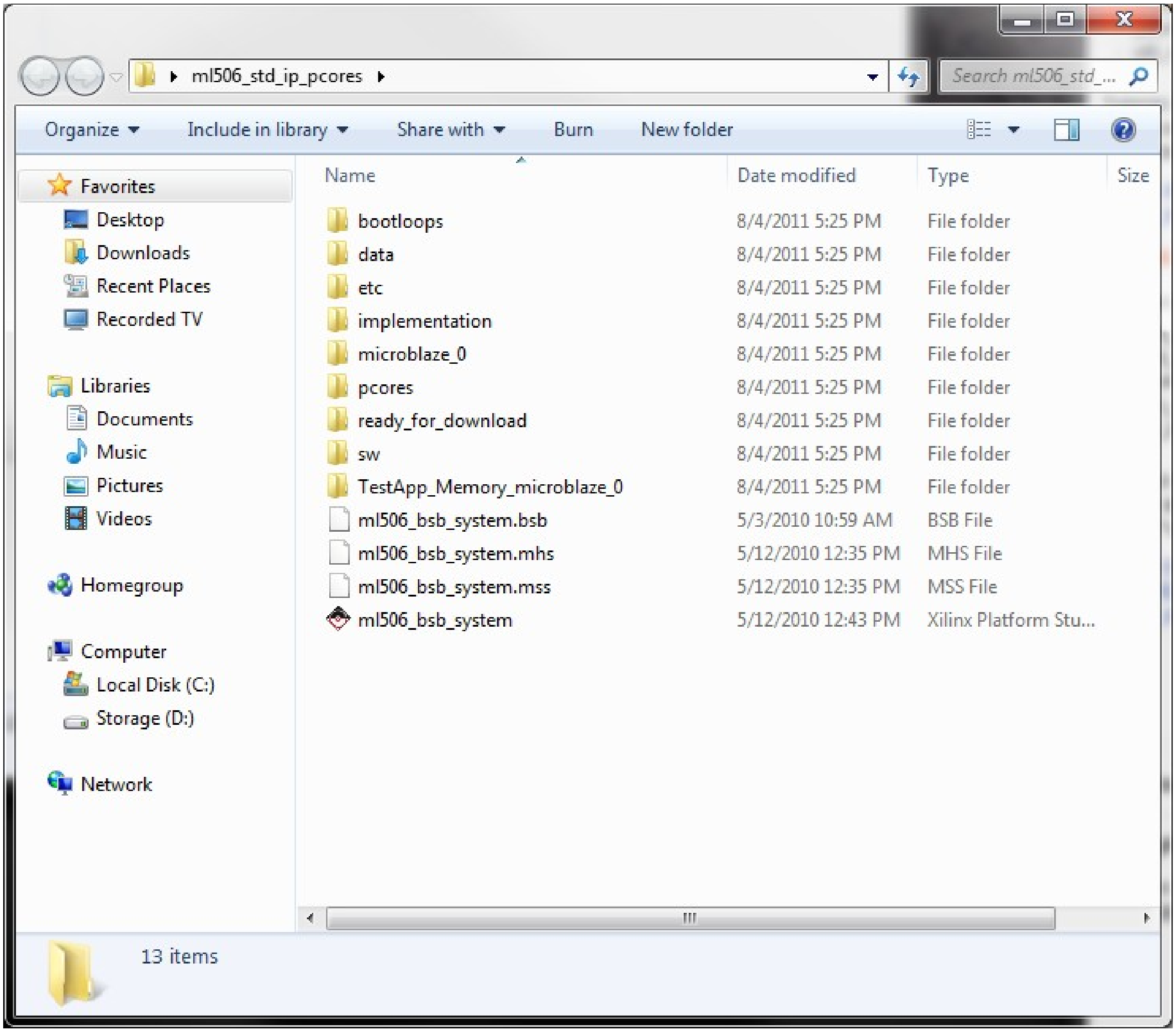
First we need to download the pre-made Xilinx files.

Get them here:

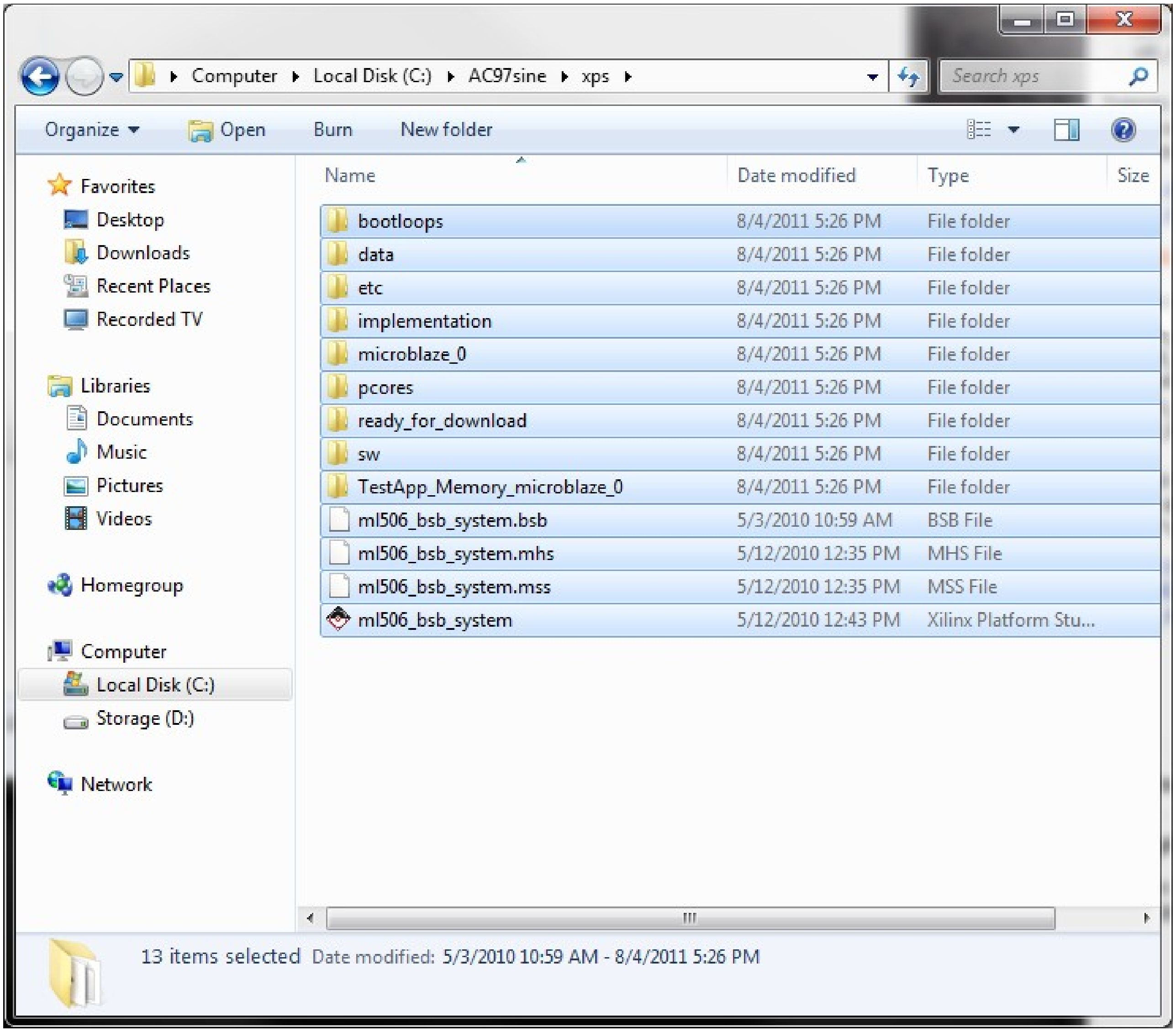
http://www.xilinx.com/products/boards/ml506/ml506_12.1/bsb.htm

Choose the ML506 EDK Standard IP Design with Pcores Addition .zip file

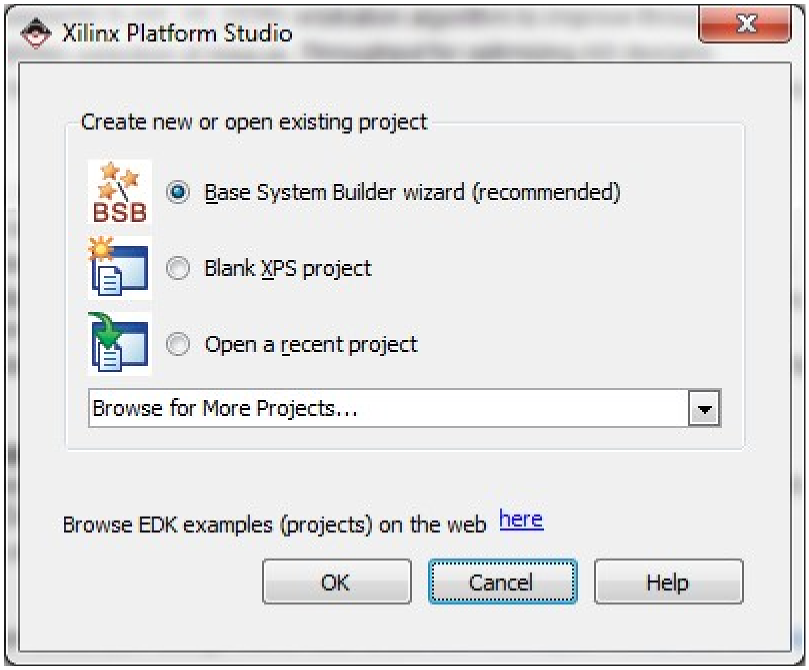
Extract the files somewhere like the desktop or a temp directory:



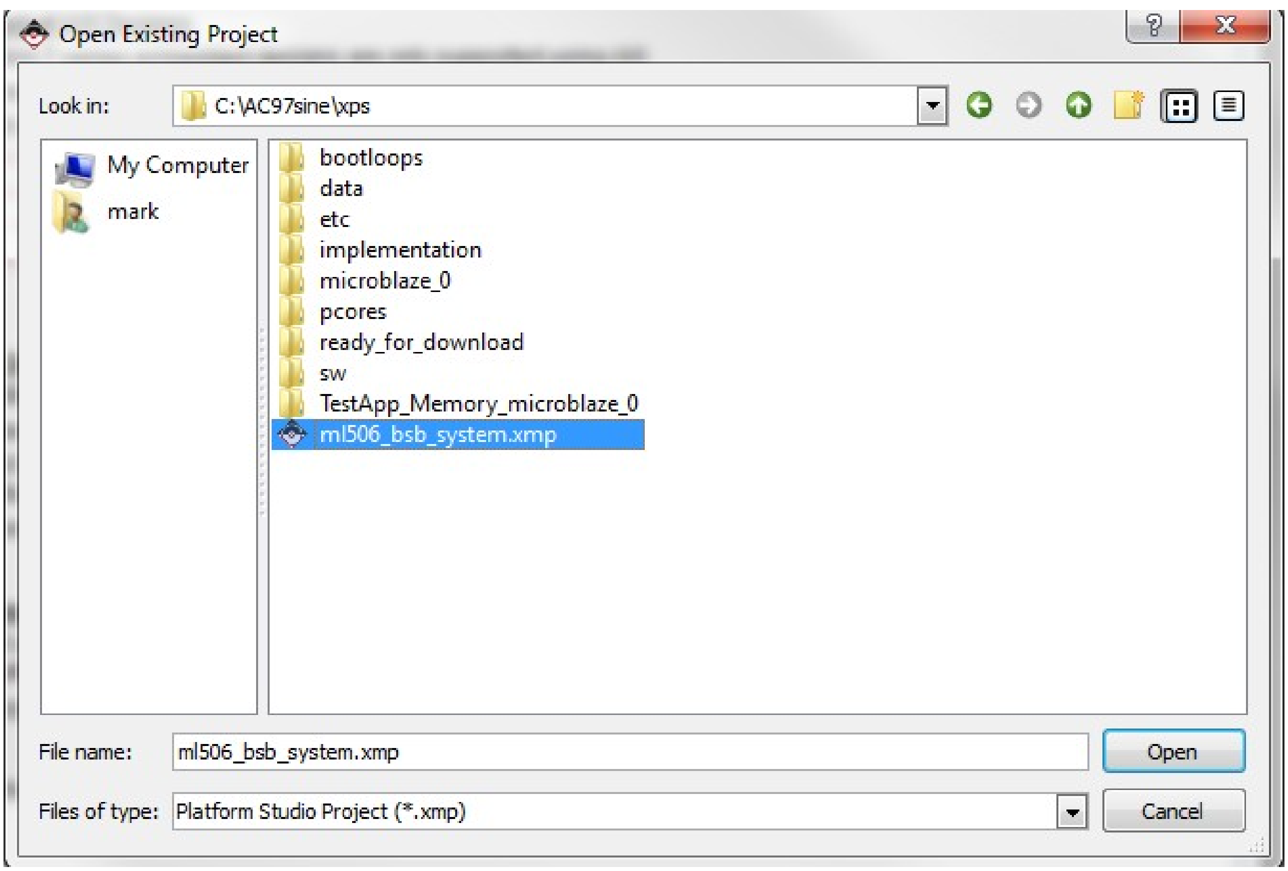
Then copy these files and put them in your xps folder created earlier:



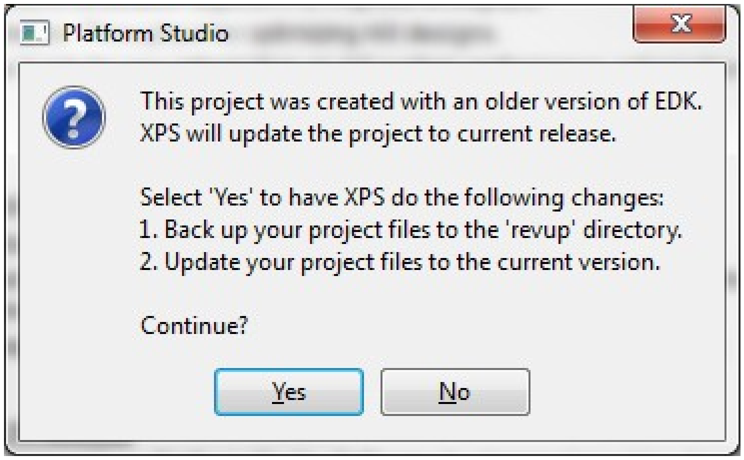
Open up Xilinx XPS and cancel the first prompt:



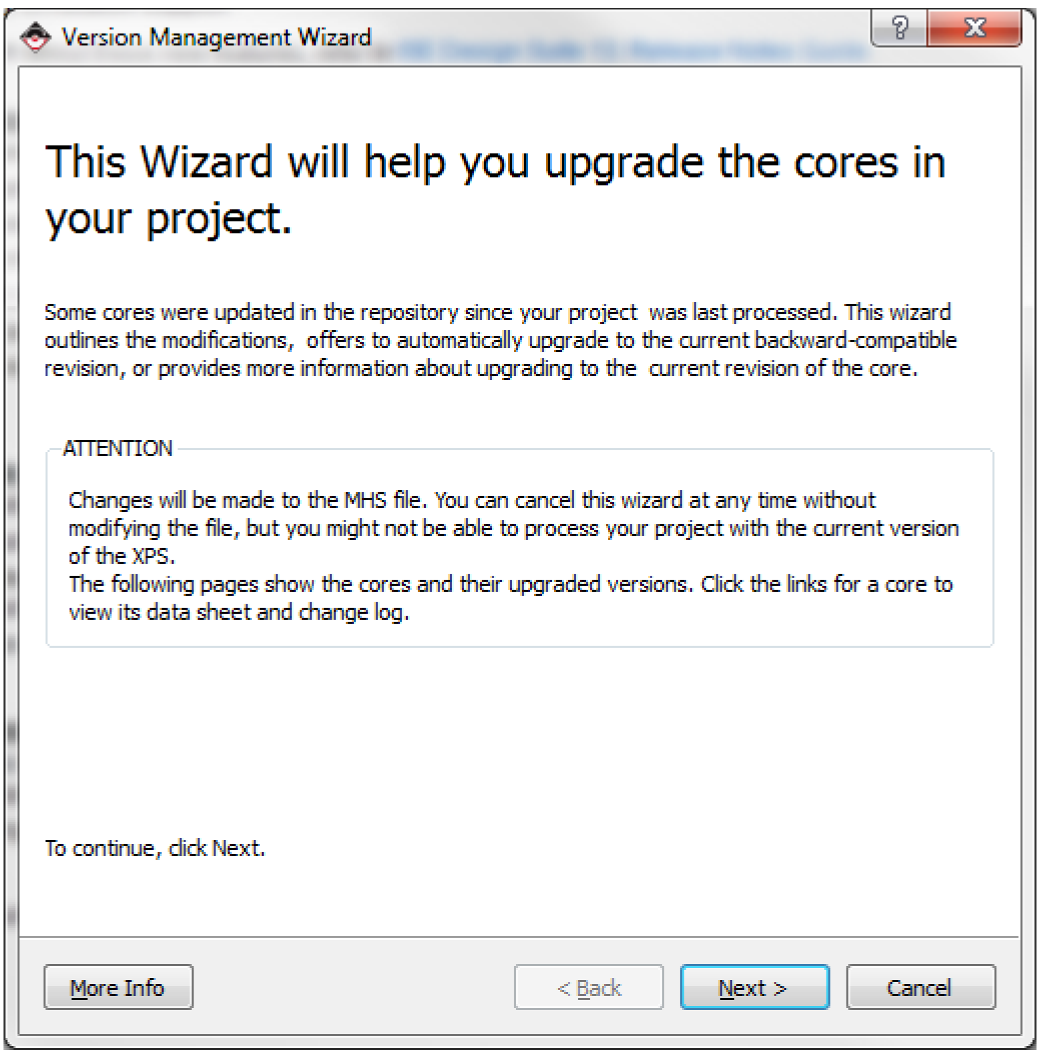
Click File → Open Project.. and browse to the ml506_bsb_system.xmp file in the xps folder and click Open:



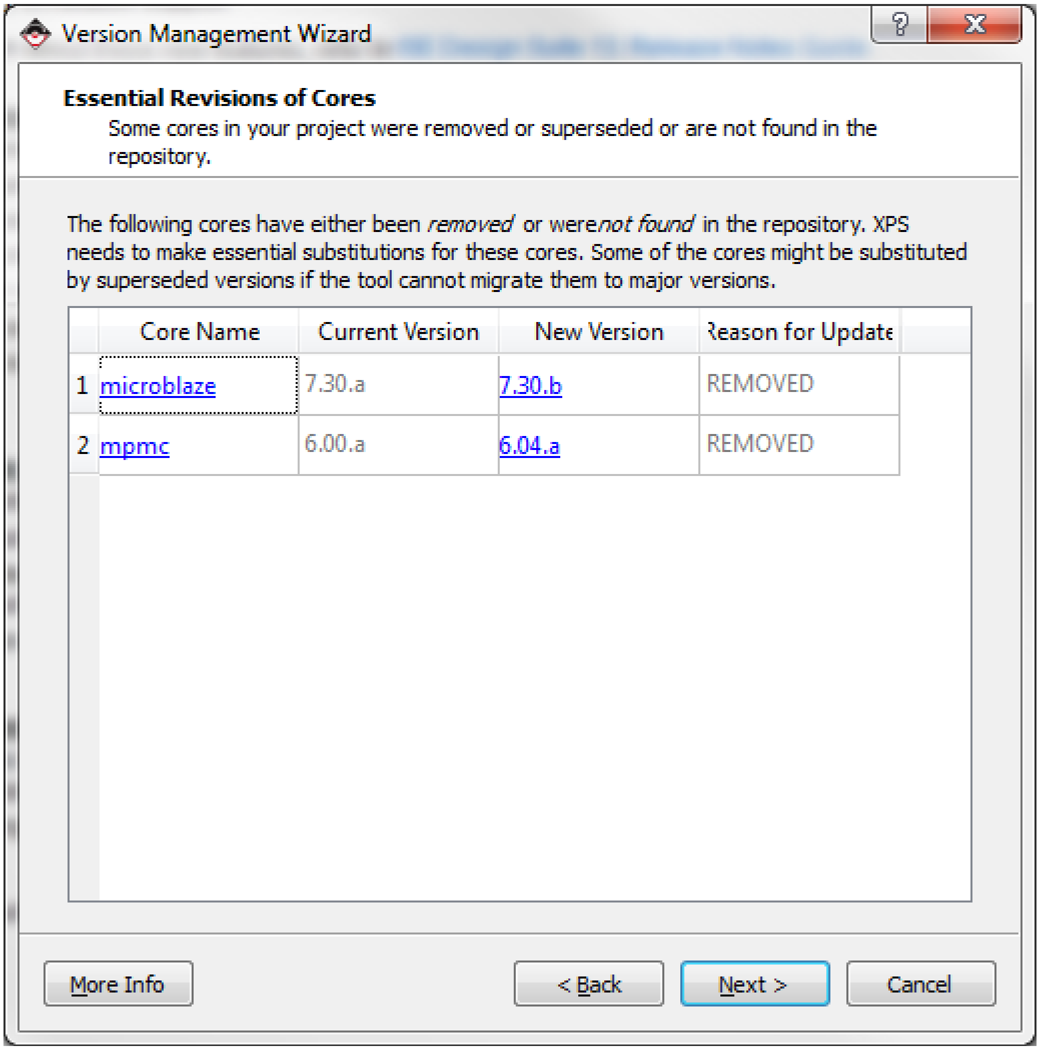
Since this Xilinx project was create with EDK version 12.1 and we are using version 13.2, Xilinx will ask us to update the project. Click Yes:



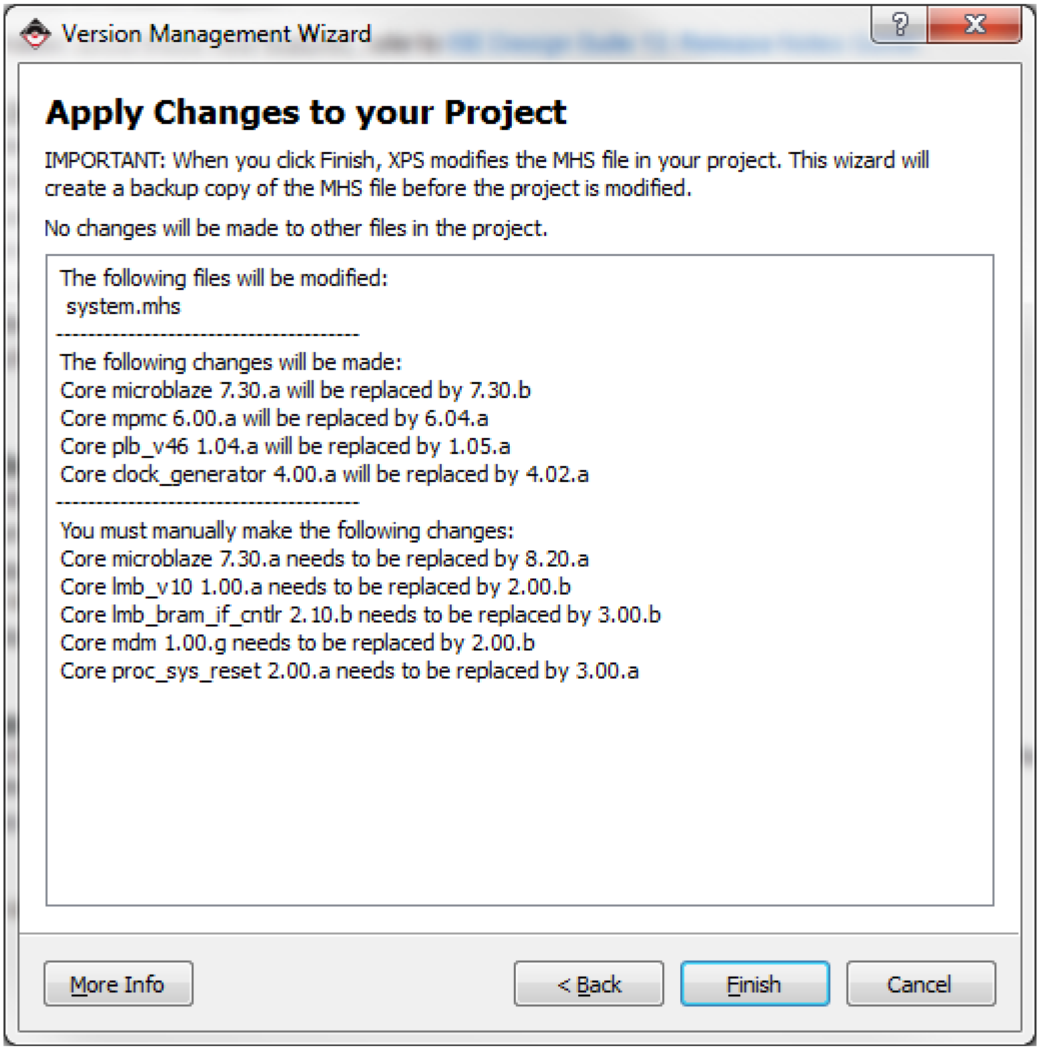
Click next on this first screen:



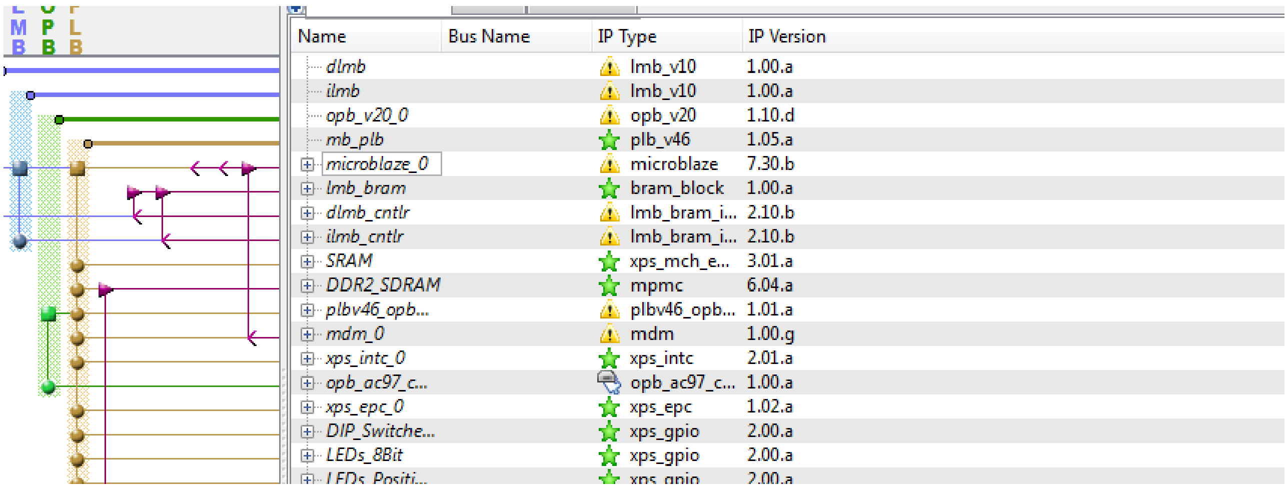
Click Next on this window and the rest that follow:



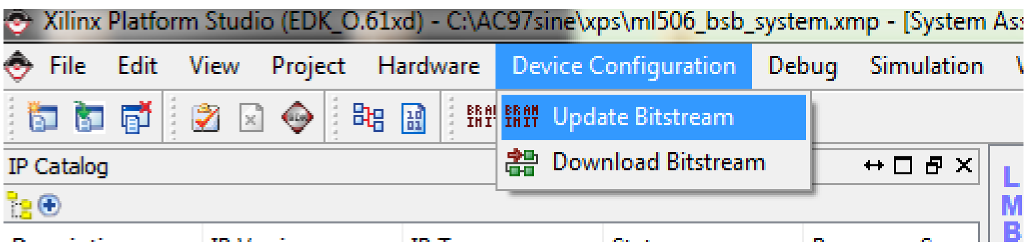
Finally, click Finish:



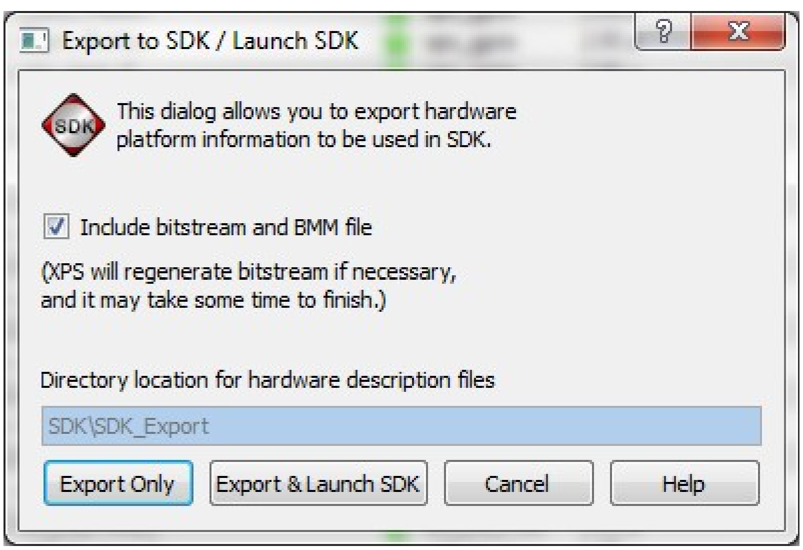
After it finishes loading the files take a look at the peripherals. Notice that it has an ac97 controller:



Click on Device Configuration → Update Bitstream (this can take up to 30min):



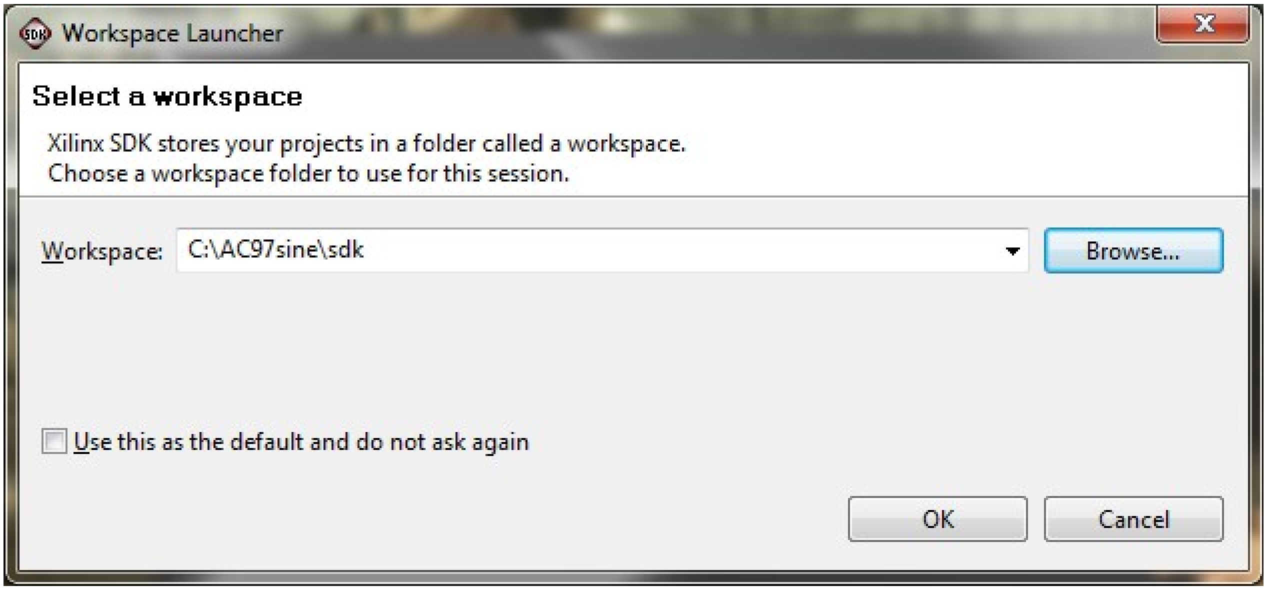
After this finishes click Project → Export Hardware Design to SDK and click Export Only:



Xilinx SDK

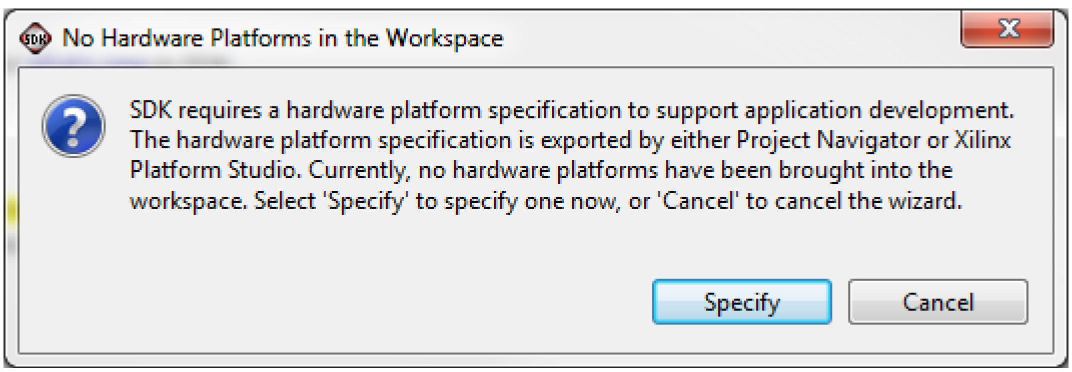
Close Xilinx XPS and open Xilinx SDK.

Choose the workspace to be the sdk folder we created earlier and click OK

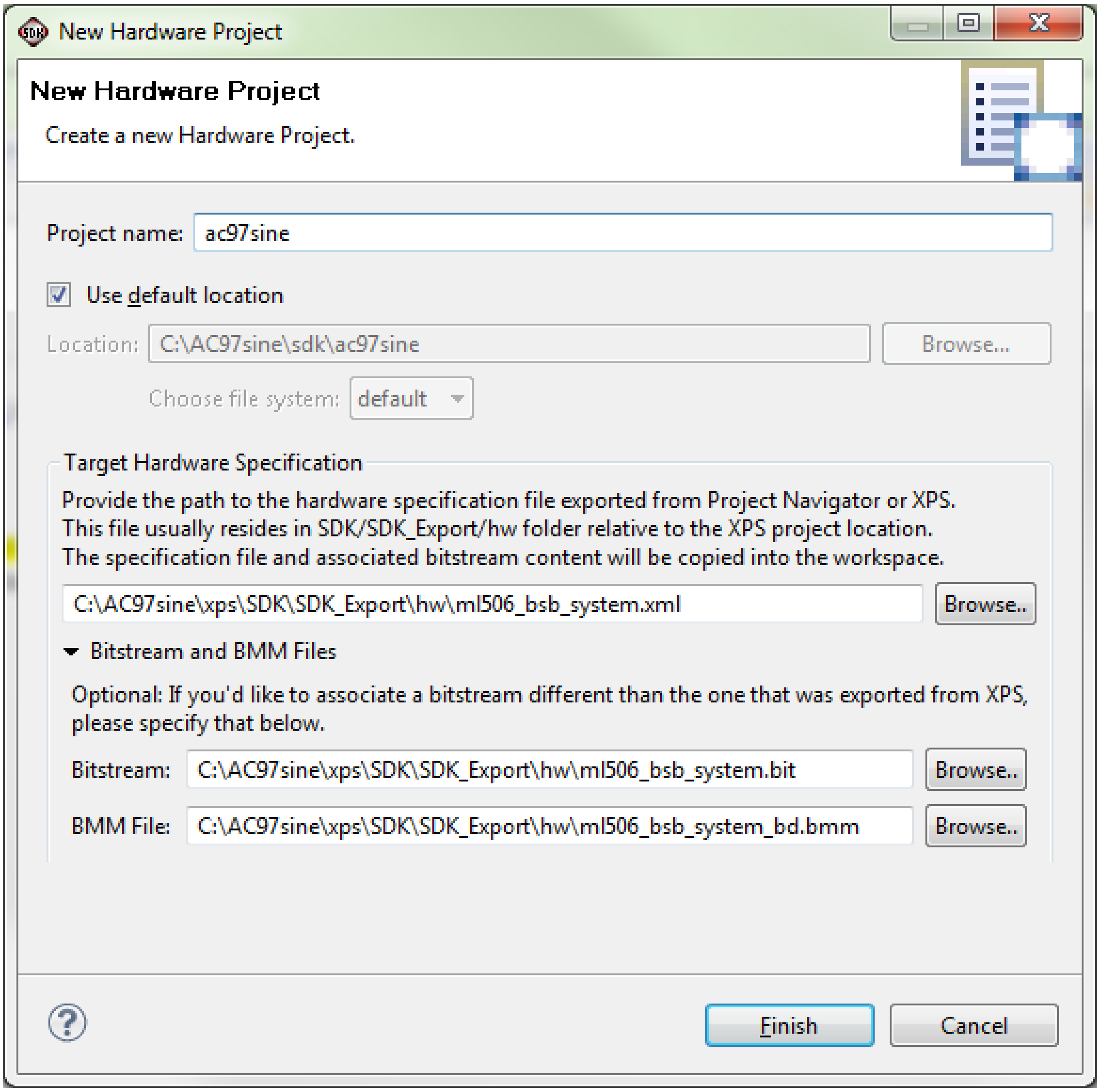


Click File → New → Xilinx C Project.

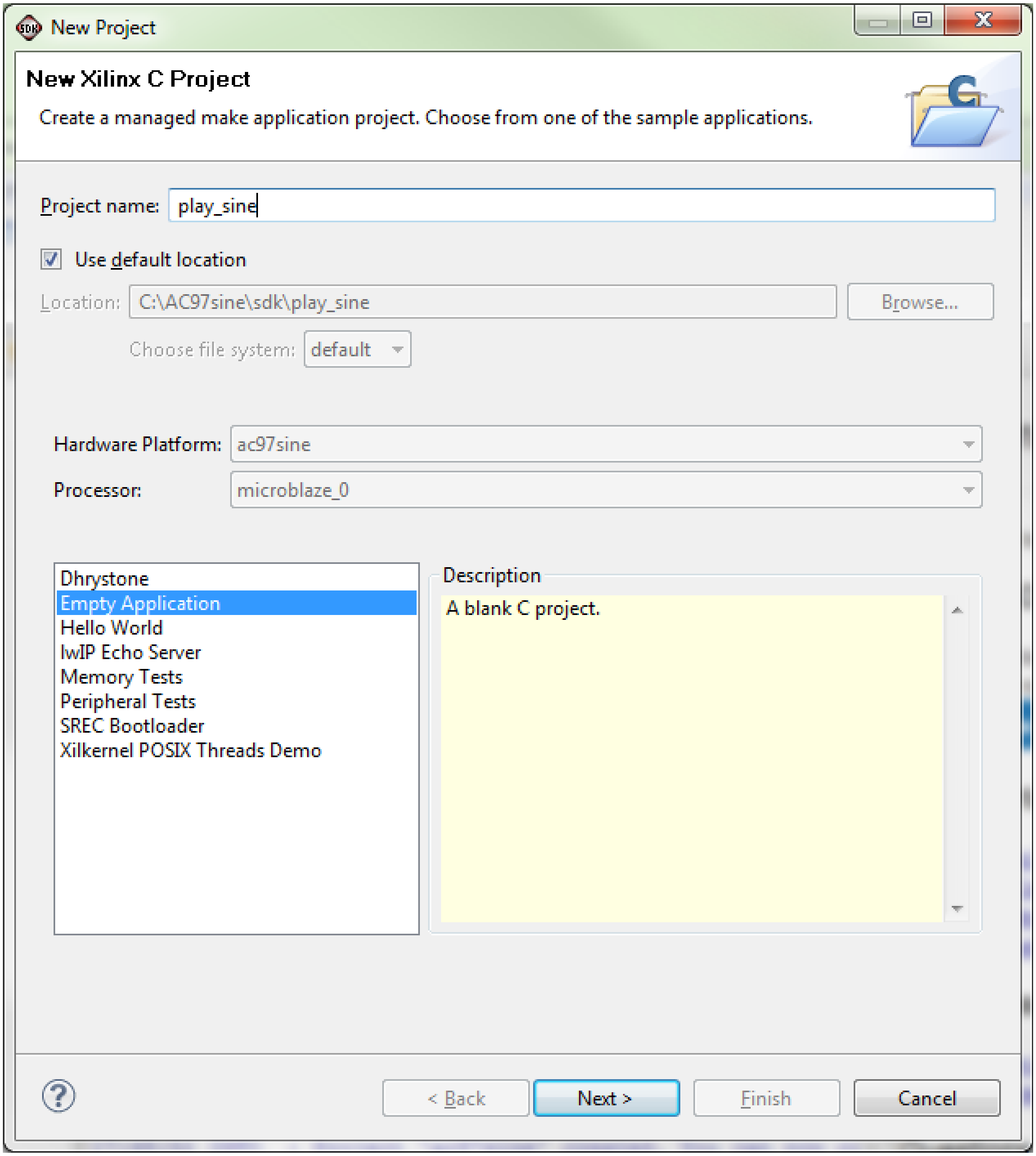
Xilinx will ask you to Specify a hardware platform, click Specify:



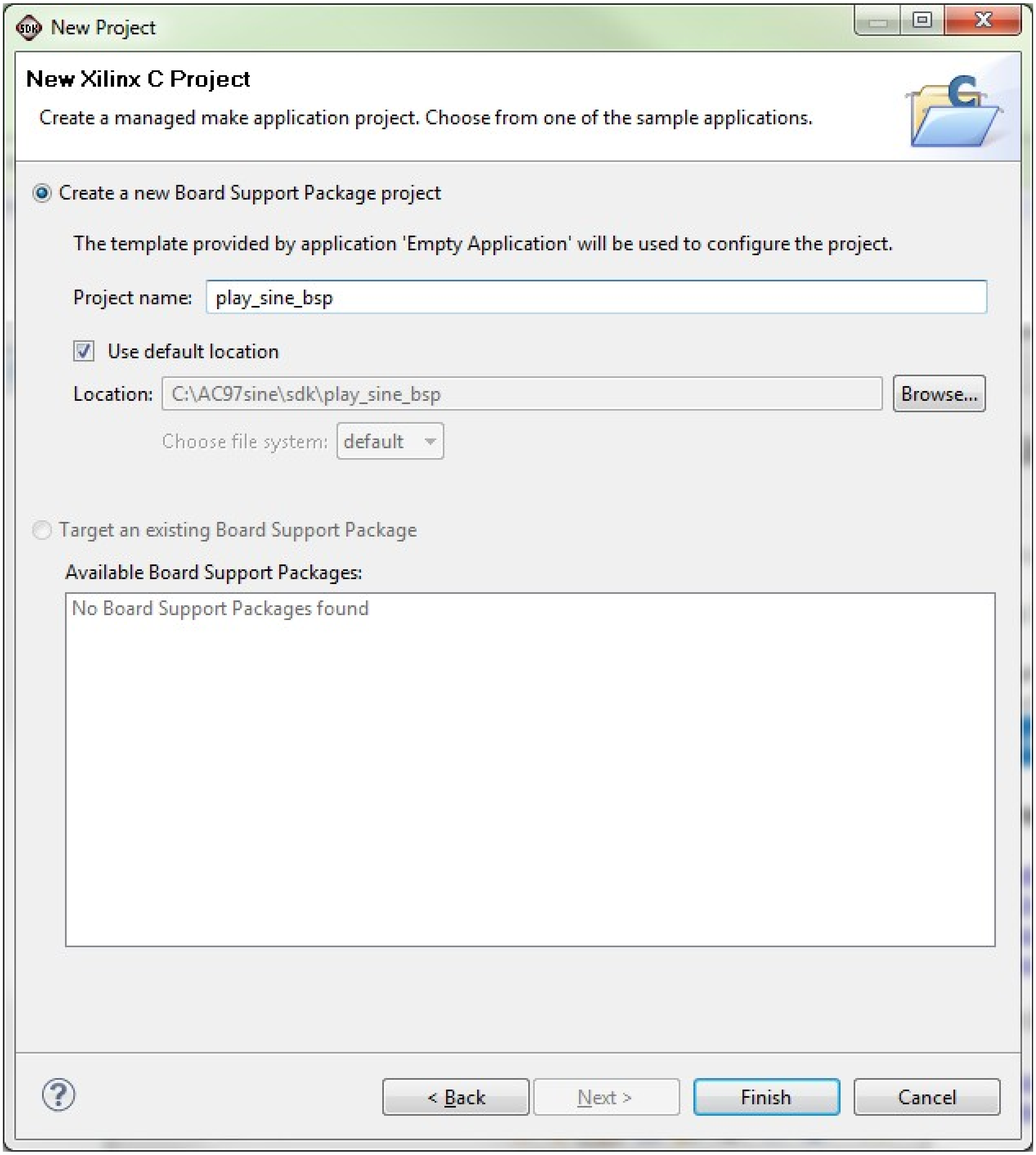
Name the project ac97sine and browse to the exported files we created from Xilinx XPS and click Finish:



Name the C project 'play_sine' and choose the Empty Application Template, then click Next:



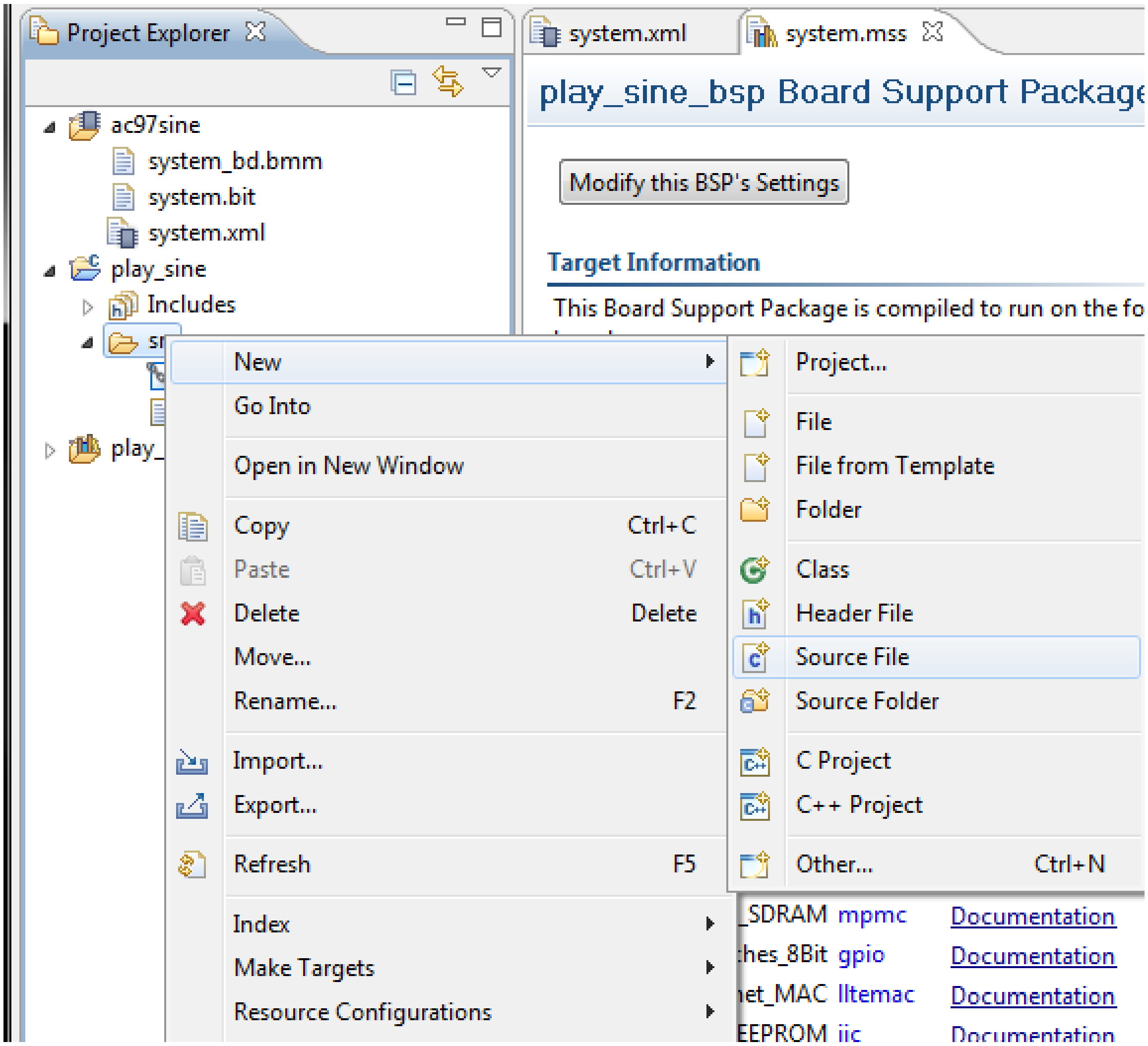
Name the board support package 'play_sine_bsp' and click Finish:



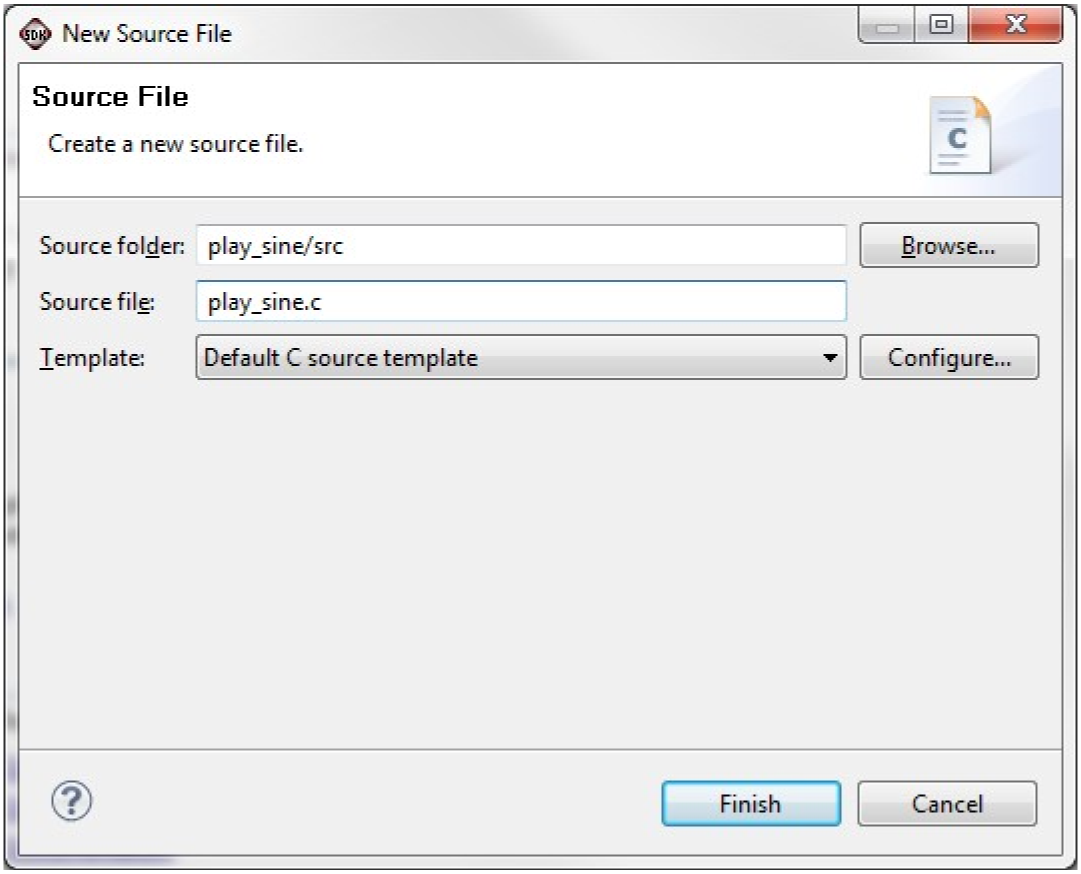
The files I will be using can be found in:

ml506_std_ip_pcores\sw\standalone

Right-click the 'src' folder under the play_sine project and click New → Source File:



Name the file 'play_sine.c' and click Finish:



Delete all the auto-generated comments and put this in (copy and paste this code). The original version can be found in ml506_sdt_ip_pcores\sw\standalone\test_ac97. I cleaned it up and added the ability to choose different frequencies and use the LCD screen (its about 4 ½ pages long):

```
#include <stdio.h>
#include <math.h>

#include "xio.h"
#include "xuartns550_l.h"
#include "sleep.h"
#include "xparameters.h"
#include "lcd.h"

#include "memory_map.h"

#define UART_CLOCK     XPAR_XUARTNS550_CLOCK_HZ

#if !SIM
#define UART_BAUDRATE 9600                          /* real hardware */
#else
#define UART_BAUDRATE (UART_CLOCK / 16 / 3)         /* simulation */
#endif
```

```c
/* local prototypes */
volatile int data_in_buf_len;
int data_in_buf[19];
volatile int dummy;

volatile int cur_sound_len;
volatile unsigned char *cur_sound_ptr;

unsigned int sound_ptr;
int c;
int freq = 22;

#define MY_AC97_BASEADDR         XPAR_OPB_AC97_CONTROLLER_REF_0_BASEADDR
#define AC97_InFIFO              MY_AC97_BASEADDR
#define AC97_OutFIFO             MY_AC97_BASEADDR + 0x4
#define AC97_FIFO_Status         MY_AC97_BASEADDR + 0x8
#define AC97_Control             MY_AC97_BASEADDR + 0xC
#define AC97_RegAddr             MY_AC97_BASEADDR + 0x10
#define AC97_RegRead             MY_AC97_BASEADDR + 0x14
#define AC97_RegWrite            MY_AC97_BASEADDR + 0x18
#define AC97_InFIFO_Full         0x01
#define AC97_InFIFO_Half_Full    0x02
#define AC97_OutFIFO_Full        0x04
#define AC97_OutFIFO_Empty       0x08
#define AC97_Reg_Access_Finished 0x10
#define AC97_CODEC_RDY           0x20
#define AC97_REG_ACCESS          0x40
#define AC97_Enable_In_Intr      0x01

// AC97 CODEC Registers
#define AC97_Reset              0x00
#define AC97_MasterVol          0x02
#define AC97_HeadphoneVol       0x04
#define AC97_MasterVolMono      0x06
#define AC97_Reserved0x08       0x08
#define AC97_PCBeepVol          0x0A
#define AC97_PhoneInVol         0x0C
#define AC97_MicVol             0x0E
#define AC97_LineInVol          0x10
#define AC97_CDVol              0x12
#define AC97_VideoVol           0x14
#define AC97_AuxVol             0x16
#define AC97_PCMOutVol          0x18
#define AC97_RecordSelect       0x1A
#define AC97_RecordGain         0x1C
#define AC97_Reserved0x1E       0x1E
#define AC97_GeneralPurpose     0x20
#define AC97_3DControl          0x22
#define AC97_PowerDown          0x26
#define AC97_ExtendedAudioID    0x28
#define AC97_ExtendedAudioStat  0x2A
#define AC97_PCM_DAC_Rate0      0x78
#define AC97_PCM_DAC_Rate1      0x7A
#define AC97_Reserved0x34       0x34
#define AC97_JackSense          0x72
#define AC97_SerialConfig       0x74
```

```c
#define AC97_MiscControlBits    0x76
#define AC97_VendorID1          0x7C
#define AC97_VendorID2          0x7E

// Volume Constants
#define AC97_VolMute 0x8000
#define AC97_VolMax  0x0000
#define AC97_VolMin  0x3F3F
#define AC97_VolMid  0x1010

void WriteAC97Reg( int reg_addr, int value) {
  XIo_Out32 (AC97_RegWrite, value);
  XIo_Out32 (AC97_RegAddr, reg_addr);
  usleep (10000);
}

int ReadAC97Reg( int reg_addr) {
  XIo_Out32 (AC97_RegAddr, reg_addr | 0x80);
  usleep (10000);
  return XIo_In32(AC97_RegRead);
}

void init_sound() {

  printf("Initializing AC97 CODEC...\r\n");
  // reset all reg's to known states (cause MUTE to all)
  WriteAC97Reg(AC97_Reset,0);
  usleep (1000);
  while (!(XIo_In32(AC97_FIFO_Status) & AC97_CODEC_RDY)) {};

  xil_printf ("PowerDown Reg State (Should be 0x000F) = %x \n\r", ReadAC97Reg
(AC97_PowerDown));

  // turn on external amp
  xil_printf ("Turning on External Power Amp \n\r");
  WriteAC97Reg(AC97_JackSense,0x3F00); // set Jack Sense Pins

  // powerdown DAC and ADC temporarily
  WriteAC97Reg(AC97_PowerDown,0x0300);
  usleep (1000000);

  // initialize LCD and write to it
  LCDOn();
  LCDInit();
  LCDPrintString ("1. 400Hz 2. 1kHz", "3. 5kHz");

  xil_printf ("\n\r\n\rPick the frequency:\n\r");
  xil_printf ("1. 400Hz\n\r2. 1kHz\n\r3. 5kHz\n\r\n\r\n\r");

  while (!(XUartNs550_GetLineStatusReg(UART_BASEADDR) & 0x1));

  do {freq = XUartNs550_RecvByte(UART_BASEADDR); }
  // check for invalid choices
  while (((char) freq != '1') && ((char) freq != '2') && ((char) freq != '3'));
  if ((char) freq == '1') freq = 110;
  if ((char) freq == '2') freq = 44;
  if ((char) freq == '3') freq = 9;
```

```
  WriteAC97Reg(AC97_ExtendedAudioStat,1);  // Enabling VRA mode
  WriteAC97Reg(AC97_PCM_DAC_Rate1, 48000); // sampling rate PLAY
  WriteAC97Reg(AC97_PCM_DAC_Rate0, 48000); // sampling rate rec

  xil_printf ("DAC sample rate= %d hz\n\r", ReadAC97Reg (AC97_PCM_DAC_Rate1));
  xil_printf ("ADC sample rate= %d hz\n\r", ReadAC97Reg (AC97_PCM_DAC_Rate0));
  xil_printf ("vendor id 1= %x\n\r", ReadAC97Reg (AC97_VendorID1));
  xil_printf ("vendor id 2= %x\n\r", ReadAC97Reg (AC97_VendorID2));

  // Turn back on power to ADC and DAC
  WriteAC97Reg(AC97_PowerDown,0x0000);
  usleep (100000);
  xil_printf ("PowerDown Reg State (Should be 0x000F) = %x \n\r", ReadAC97Reg
(AC97_PowerDown));

  xil_printf ("Misc Control Bits = %x\n\r", ReadAC97Reg (AC97_MiscControlBits));


  XIo_Out32(AC97_Control, 0x00000003); // clear FIFOs

  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);
  XIo_Out32(AC97_InFIFO, 0);

  // turn off digital loopback
  WriteAC97Reg(AC97_GeneralPurpose,0x0000);
  xil_printf ("General Purpose reg state = %x\n\r", ReadAC97Reg
(AC97_GeneralPurpose));

  WriteAC97Reg(AC97_SerialConfig,0x7000);
  xil_printf ("config reg state = %x\n\r", ReadAC97Reg (AC97_SerialConfig));

  WriteAC97Reg(AC97_MasterVol,     AC97_VolMid);
  WriteAC97Reg(AC97_HeadphoneVol,  AC97_VolMid);
  WriteAC97Reg(AC97_MasterVolMono, AC97_VolMid);
  WriteAC97Reg(AC97_PCBeepVol,     AC97_VolMute);
  WriteAC97Reg(AC97_PhoneInVol,    AC97_VolMute);
  WriteAC97Reg(AC97_CDVol,         AC97_VolMute);
  WriteAC97Reg(AC97_VideoVol,      AC97_VolMute);
  WriteAC97Reg(AC97_AuxVol,        AC97_VolMute);
  WriteAC97Reg(AC97_PCMOutVol,     AC97_VolMid);
  WriteAC97Reg(AC97_RecordSelect,  0x0000);
```

```
  WriteAC97Reg(AC97_MicVol,         0x0040);
  WriteAC97Reg(AC97_LineInVol,      AC97_VolMid);
}

void play_sound(){
  int i;
  int j;
  xil_printf ("Play Start\n\r");
  sound_ptr = DDR_BASEADDR;
  WriteAC97Reg(AC97_RecordGain,  AC97_VolMute);
  WriteAC97Reg(AC97_PowerDown,   0x0100);
  WriteAC97Reg(AC97_LineInVol,   AC97_VolMute);
  i = 0;
  do {
    XIo_Out32(AC97_Control, 0x00000003); // clear FIFOs
    XIo_Out32(AC97_InFIFO, 0);
    XIo_Out32(AC97_InFIFO, 0);
    XIo_Out32(AC97_InFIFO, 0);
    XIo_Out32(AC97_InFIFO, 0);
    XIo_Out32(AC97_InFIFO, 0);
    XIo_Out32(AC97_InFIFO, 0);
    XIo_Out32(AC97_InFIFO, 0);
    XIo_Out32(AC97_InFIFO, 0);
  } while (XIo_In32(AC97_FIFO_Status) & 0x0040);

  while (1) {
      if (XIo_In32(AC97_FIFO_Status) & 0x0040) xil_printf ("play underrun\n\r");
      while (XIo_In32(AC97_FIFO_Status) & AC97_InFIFO_Half_Full) {};

    // The frequency is 44.1kHz so if we divide 44.1kHz by 22 we
    // get 2004.54 ~ 2kHz sine wave
    switch ((i) % freq) {
       case 0: j = 0 ; break;
       case 1: j = 9232; break;
       case 2: j = 17715; break;
       case 3: j = 24764; break;
       case 4: j = 29806; break;
       case 5: j = 32434; break;
       case 6: j = 32435; break;
       case 7: j = 29808; break;
       case 8: j = 24766; break;
       case 9: j = 17718; break;
       case 10: j = 9234; break;
       case 11: j = 3; break;
       case 12: j = -9229; break;
       case 13: j = -17713; break;
       case 14: j = -24762; break;
       case 15: j = -29805; break;
       case 16: j = -32434; break;
       case 17: j = -32435; break;
       case 18: j = -29809; break;
       case 19: j = -24768; break;
       case 20: j = -17720; break;
       case 21: j = -9237; break;
     }
   i = i+1;
```

```
      XIo_Out32(AC97_InFIFO, j);

  }
}

int main ()
{
  XUartNs550_SetBaud(UART_BASEADDR, UART_CLOCK, UART_BAUDRATE);
  XUartNs550_SetLineControlReg(UART_BASEADDR, XUN_LCR_8_DATA_BITS);
  init_sound();
  while (1) {
    play_sound();
  }
  return 0;
}
```

After pasting in the above code, your window should look something like this:

play_sine.c

```
#include <stdio.h>
#include <math.h>

#include "xio.h"
#include "xuartns550_l.h"
#include "sleep.h"
#include "xparameters.h"
#include "lcd.h"

#include "memory_map.h"

#define UART_CLOCK    XPAR_XUARTNS550_CLOCK_HZ

#if !SIM
#define UART_BAUDRATE 9600                      /* real h
#else
#define UART_BAUDRATE (UART_CLOCK / 16 / 3)     /* simula
#endif

/* local prototypes */
volatile int data_in_buf_len;
int data_in_buf[19];
volatile int dummy;

volatile int cur_sound_len;
volatile unsigned char *cur_sound_ptr;

unsigned int sound_ptr;
int c;
int freq = 22;

#define MY_AC97_BASEADDR         XPAR_OPB_AC97_CONTROLLER
#define AC97_InFIFO              MY_AC97_BASEADDR
#define AC97_OutFIFO             MY_AC97_BASEADDR + 0x4
#define AC97_FIFO_Status         MY_AC97_BASEADDR + 0x8
#define AC97_Control             MY_AC97_BASEADDR + 0xC
#define AC97_RegAddr             MY_AC97_BASEADDR + 0x10
#define AC97_RegRead             MY_AC97_BASEADDR + 0x14
#define AC97_RegWrite            MY_AC97_BASEADDR + 0x18
#define AC97_InFIFO_Full         0x01
#define AC97_InFIFO_Half_Full    0x02
```

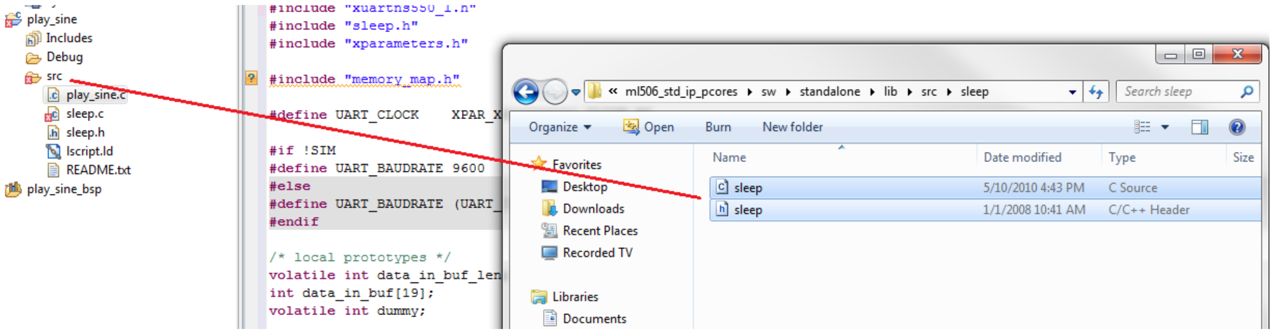
Next we need to add some header files.

Go to the standalone folder:

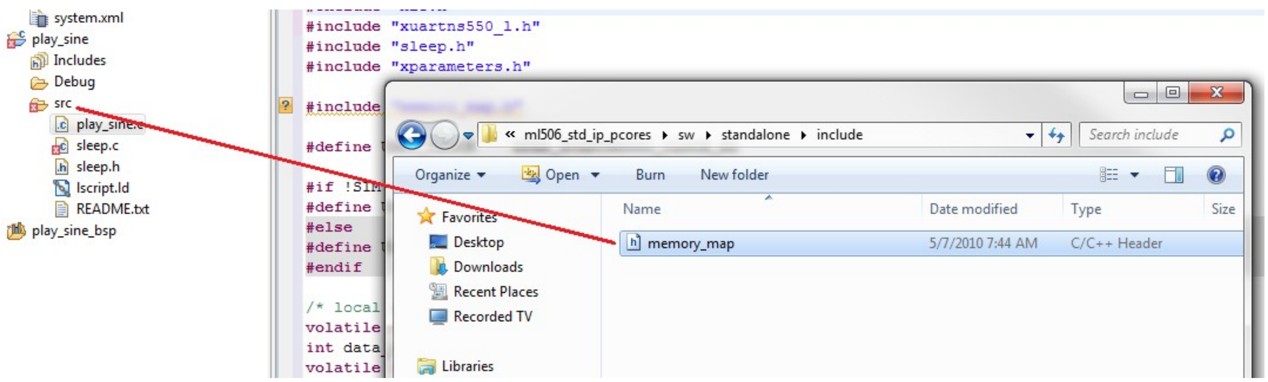
ml506_std_ip_pcores\sw\standalone

and open the ml506_std_ip_pcores\sw\standalone\lib\src\sleep file.

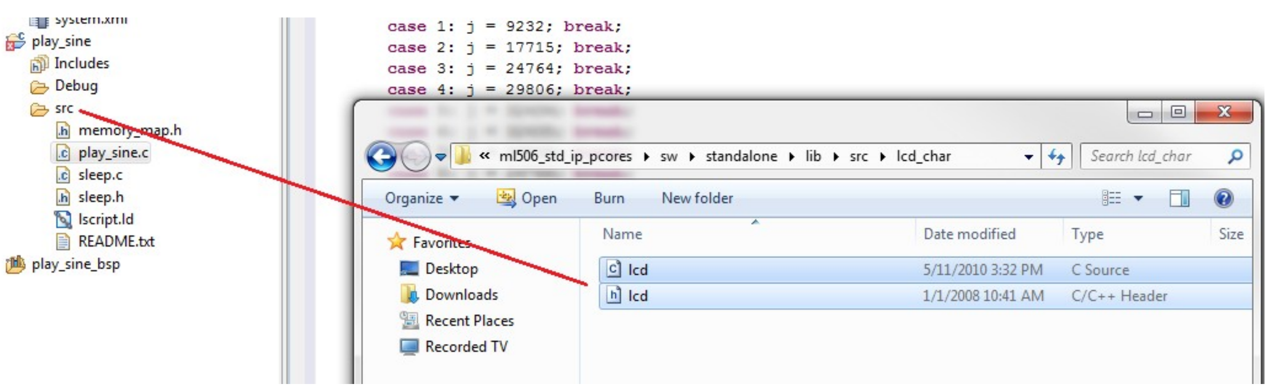
Highlight sleep.c and sleep.h and drag it into the src folder:



Next, go to the ml506_std_ip_pcores\sw\standalone\include folder and drag the memory_map.h file into the src folder:

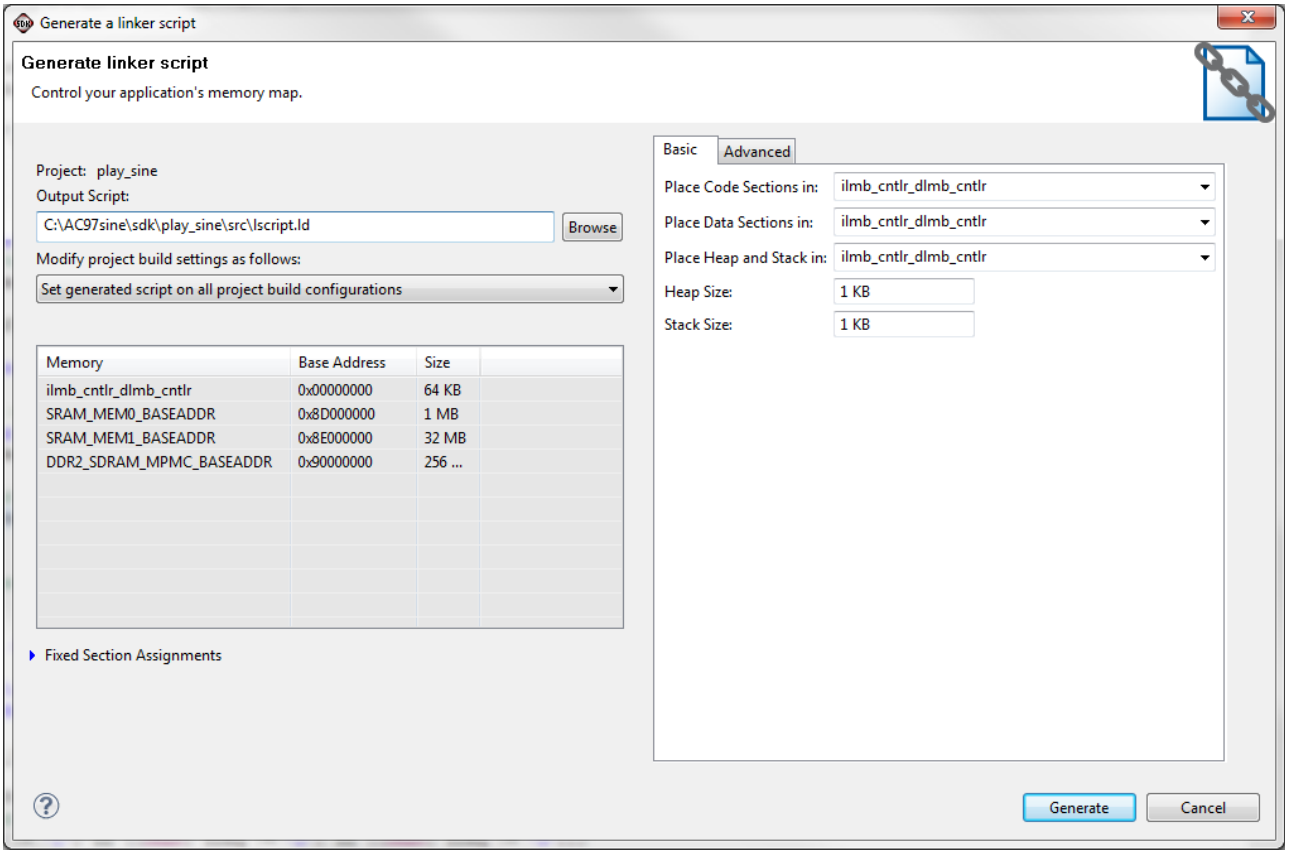


Then, go to the ml506_std_ip_pcores\sw\standalone\lib\src\lcd_char folder and drage lcd.h and lcd.c into the src folder:



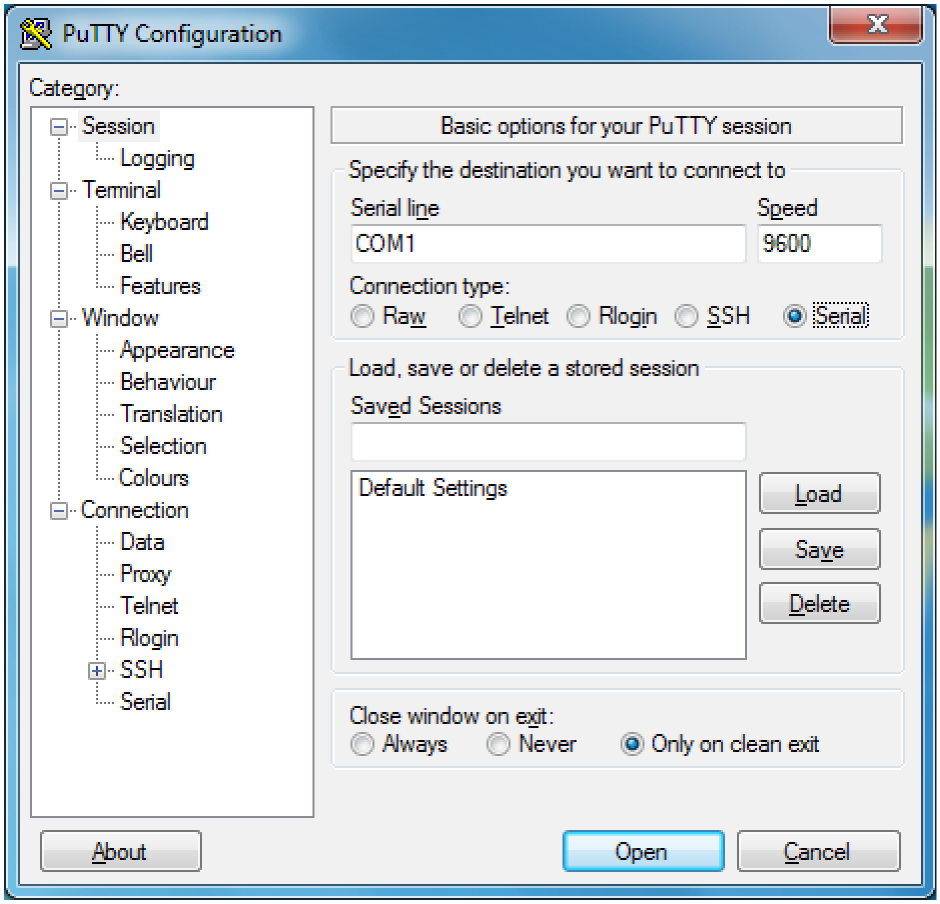
Click Save and your project should start building and compiling.

Right-click on the play_sine project and click Generate Linker Script, accept the default settings and click Generate overwriting existing files if asked:

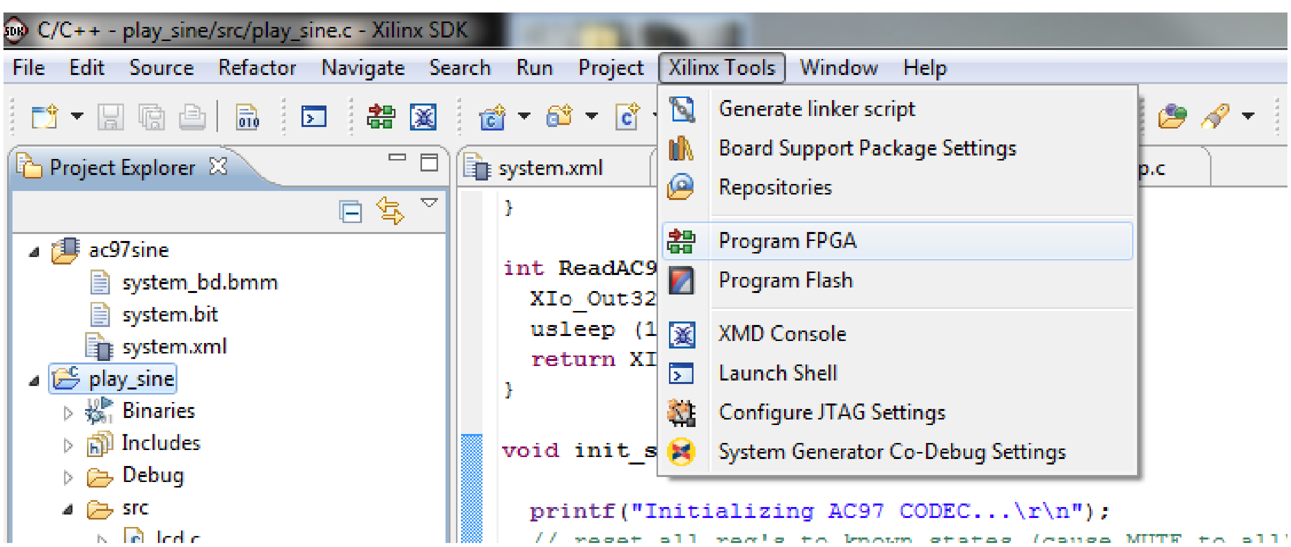


Program the FPGA

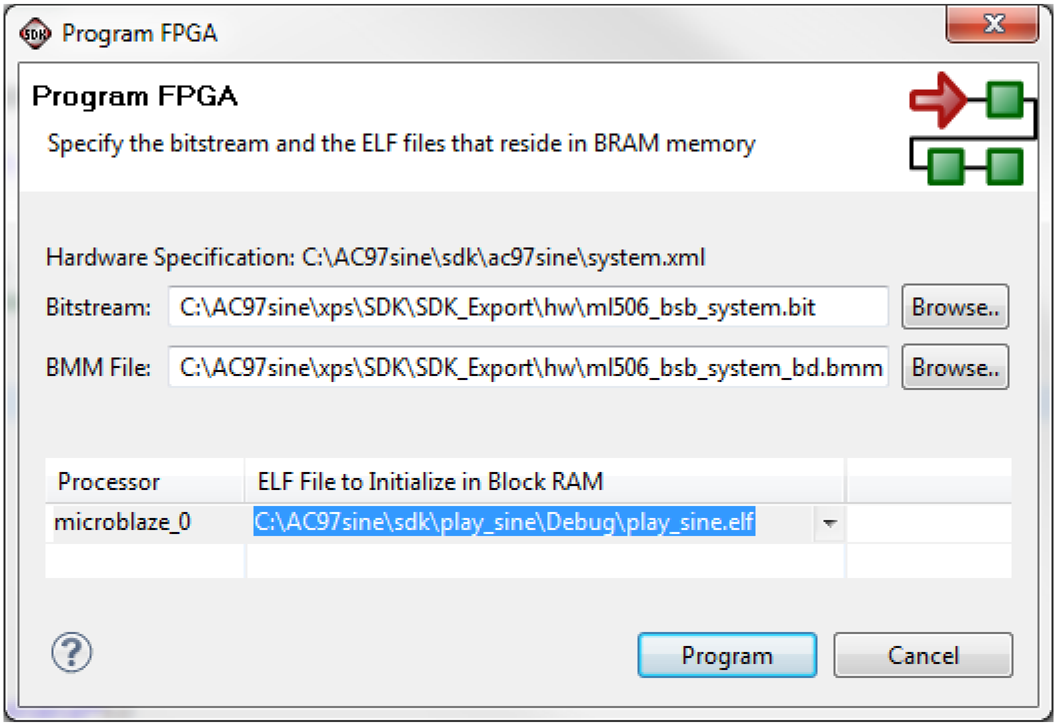
Turn on the ML506 and open up a HyperTerminal or TeraTerm etc... I use putty here:



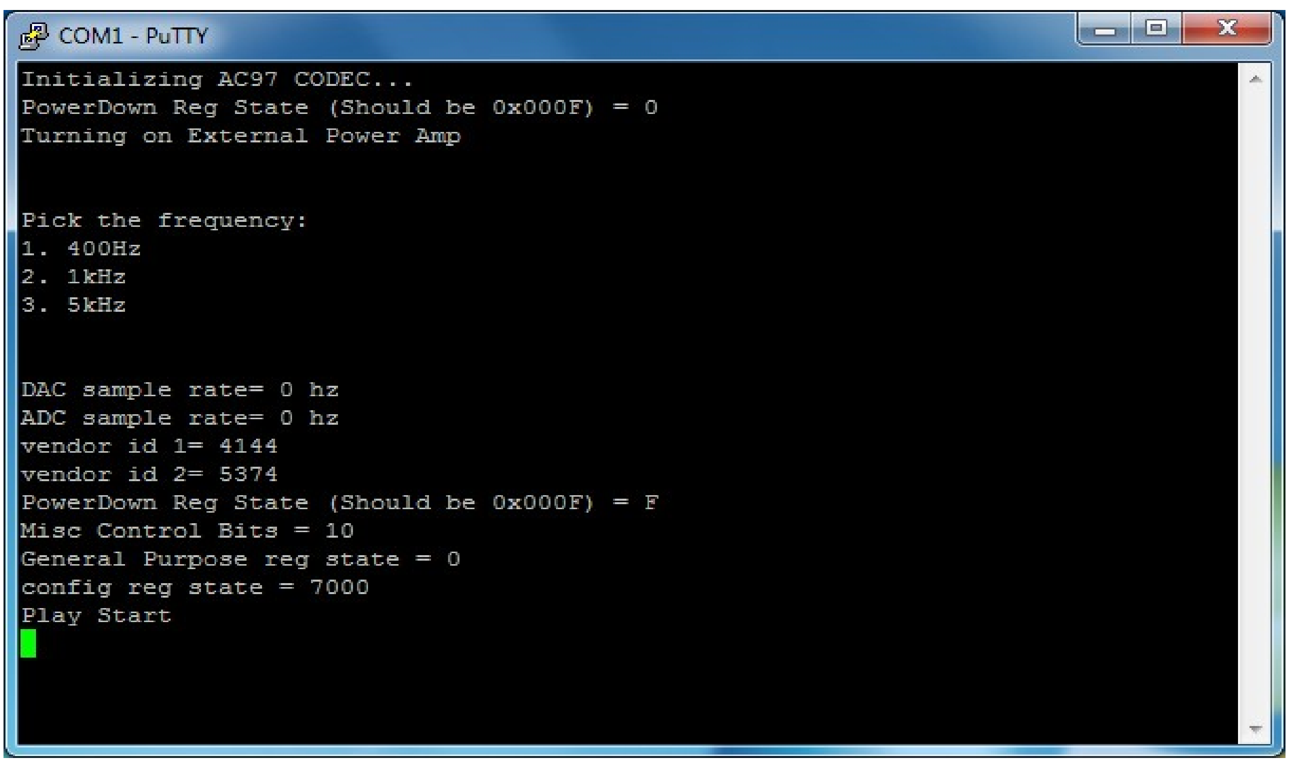
Then in SDK click Xilinx Tools → Program FPGA.



Make sure the bistream and BMM files point to the files you exported from XPS, then set your play_sine.elf file to be initialized onto the block RAM then click Program:



The FPGA is now programmed. Look to the terminal window for the directions and choose your sine wave by typing on your PC keyboard:



Some instructions are also displayed on the LCD:

SDK will create a download.bit file located in : C:\AC97sine\sdk\ac97sine. I suggest you rename this to something like playsine.bit

Programming your FPGA with the Compact Flash

Summary

You can save your programs on the compact flash so that when you turn on the ML506 it will automatically run your program without having to open up iMPACT or Xilinx SDK to program it. To do this we need the bitstream for your project and then we convert it to a .ace file. Then we save this file into one of the configuration folders located in the file system of the compact flash. We then set the configuration mode on SW3 to correspond to the folder number we saved our .ace file into and turn on the board.

We will use the bitsream created from the play sine wave tutorial. It is located in(on my computer):

C:\AC97sine\sdk\ac97sine

SDK names the file download.bit but I renamed it to playsine.bit

Creating an .ace file

This tutorial comes directly from this website:

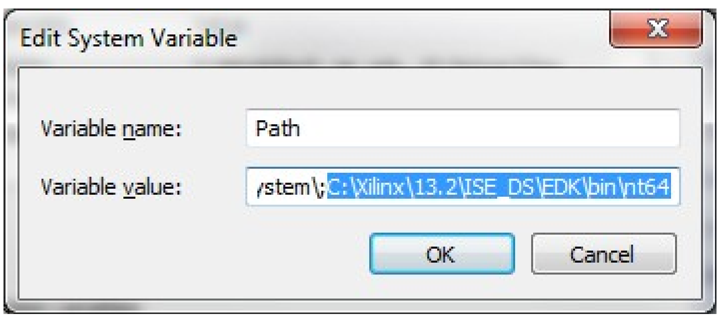
http://www.fpgadeveloper.com/2009/10/convert-bit-files-to-system-ace-files.html

I will summarize it here for convenience:

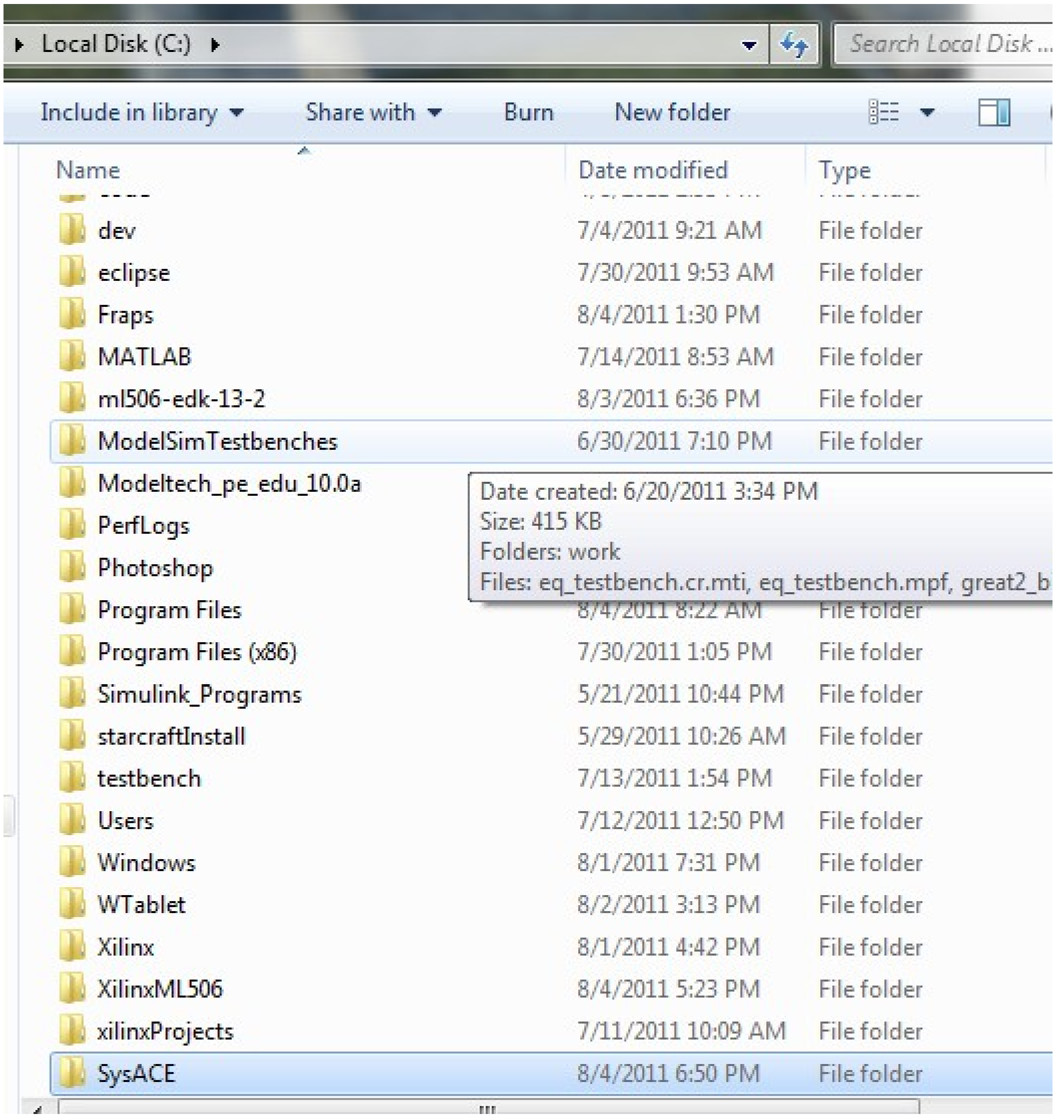
First add the Xilinx command: xmd

to your system path (google 'how to add to path windows 7' if you don't know how):

To get to this window right-click Computer, click Advanced system settings Environment Variables.., Edit the variable called 'Path' under System variables:



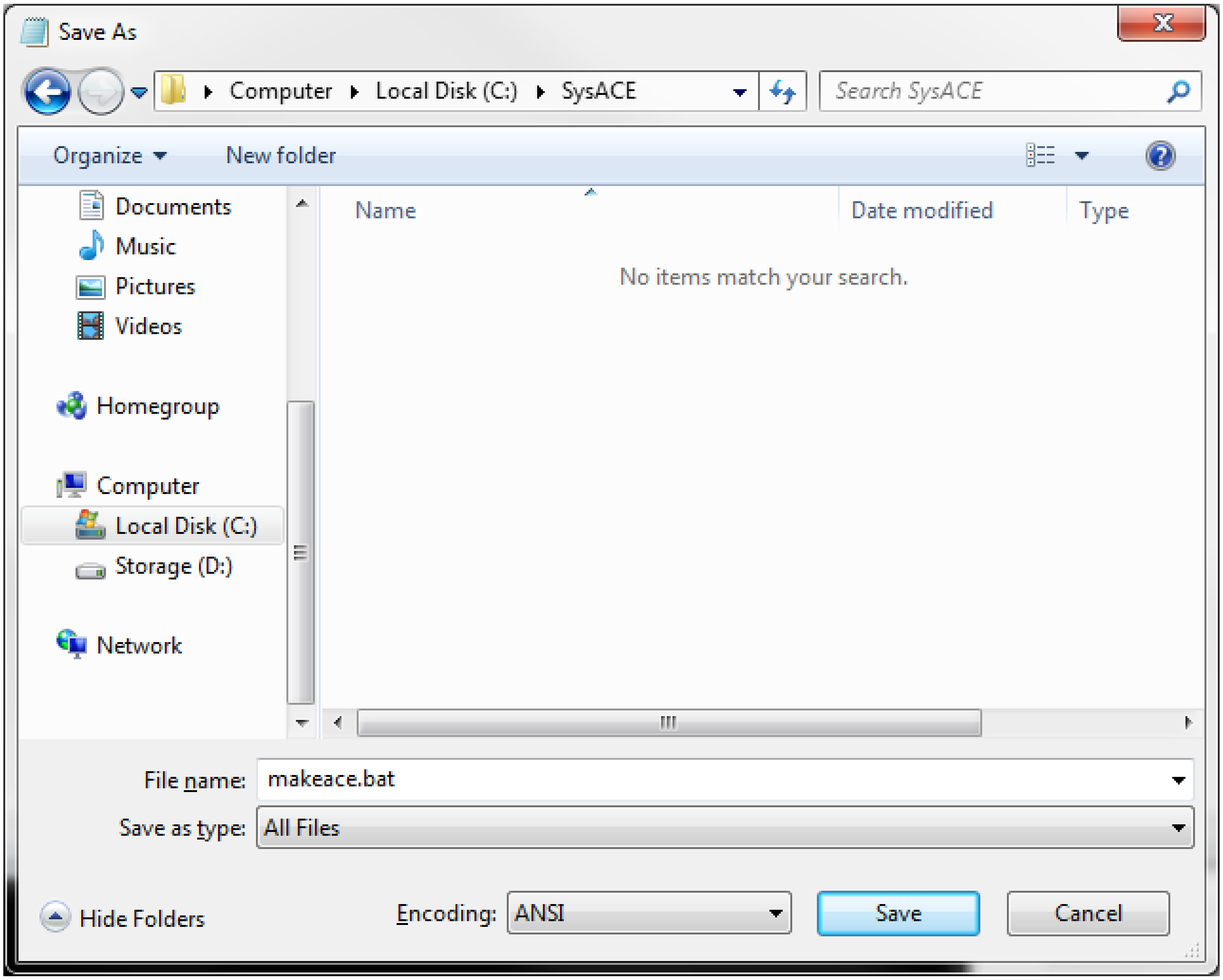
Create a folder on your local drive called: SysACE



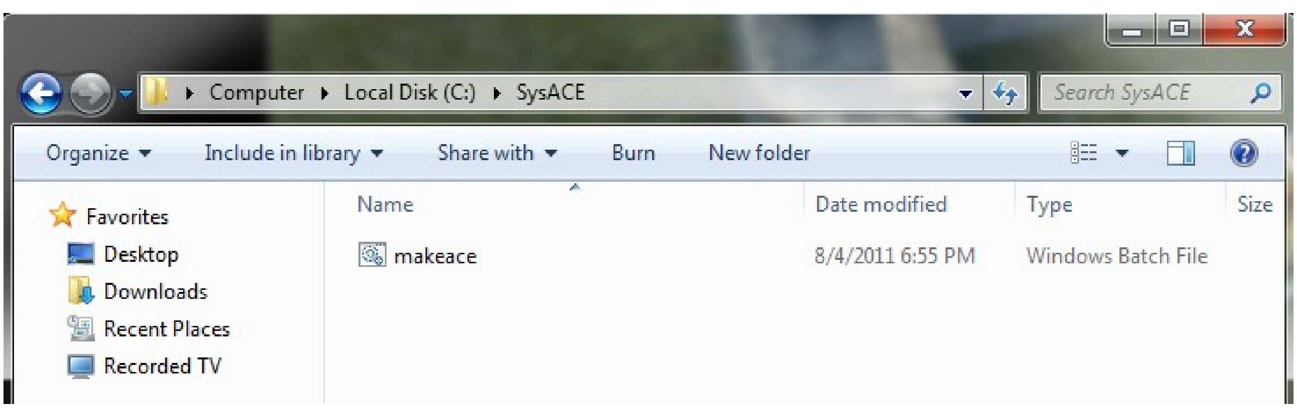
Open up a text editor and copy and paste this code:

```
@echo off
if "%1" == "" goto error
xmd -tcl ./genace.tcl -jprog -hw %1.bit -board ml505 -ace my_%1.ace
goto end
:error
echo Makeace - by FPGA Developer http://www.fpgadeveloper.com
echo.
echo Usage:   makeace bitfile (without .bit extension)
echo Example: makeace project
:end
echo.
```

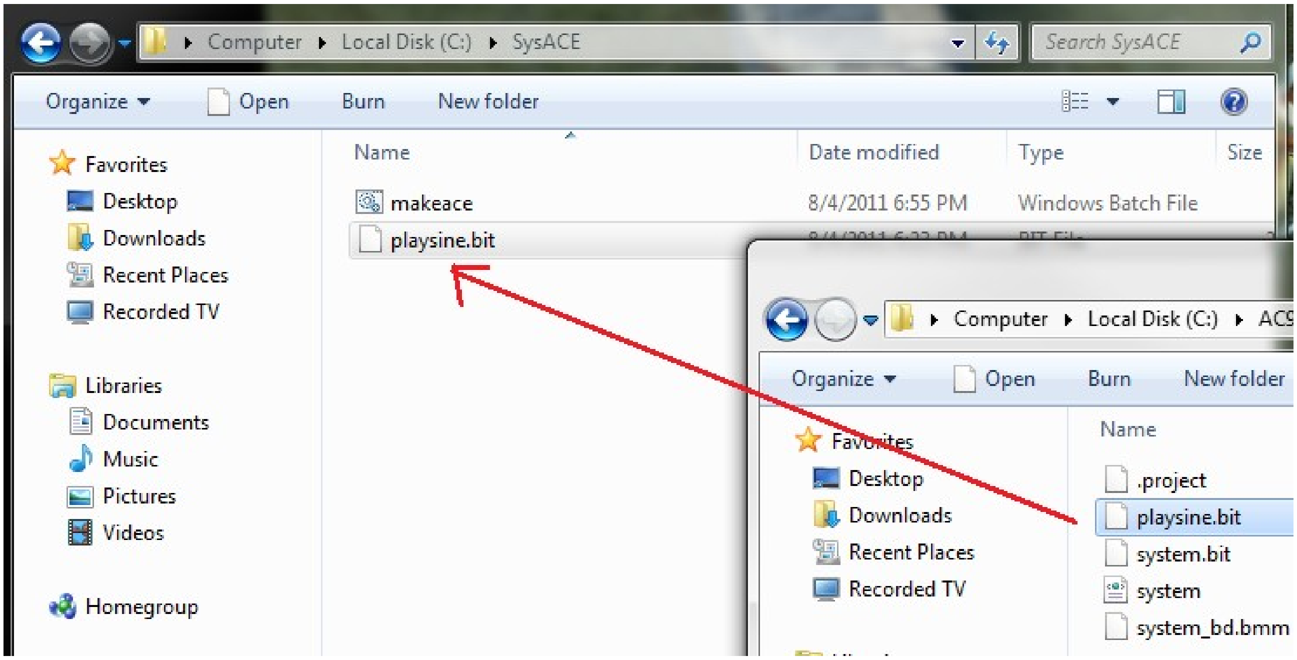
Then save the file as makeace.bat into the SysACE folder:



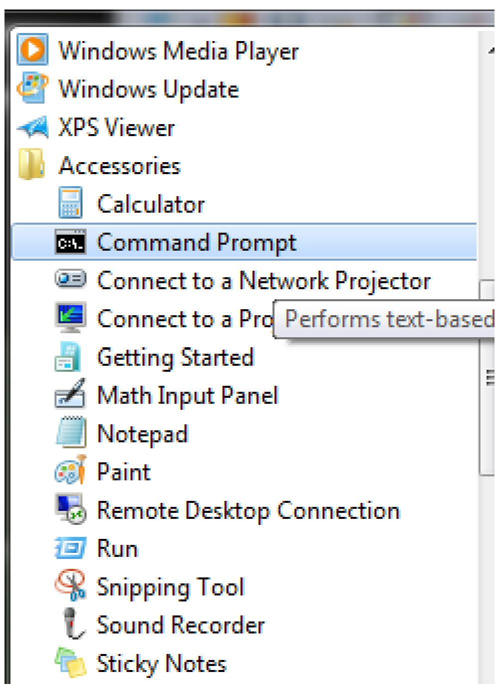
Now you should have this file in you SysACE folder:



Now we need the bitstream of our project we want to convert to an ACE file. Lets use the playsine.bit from the previous tutorial (if you didn't rename it, it would be called download.bit). Copy and paste this file into the SysACE folder:



Open up a windows command prompt:



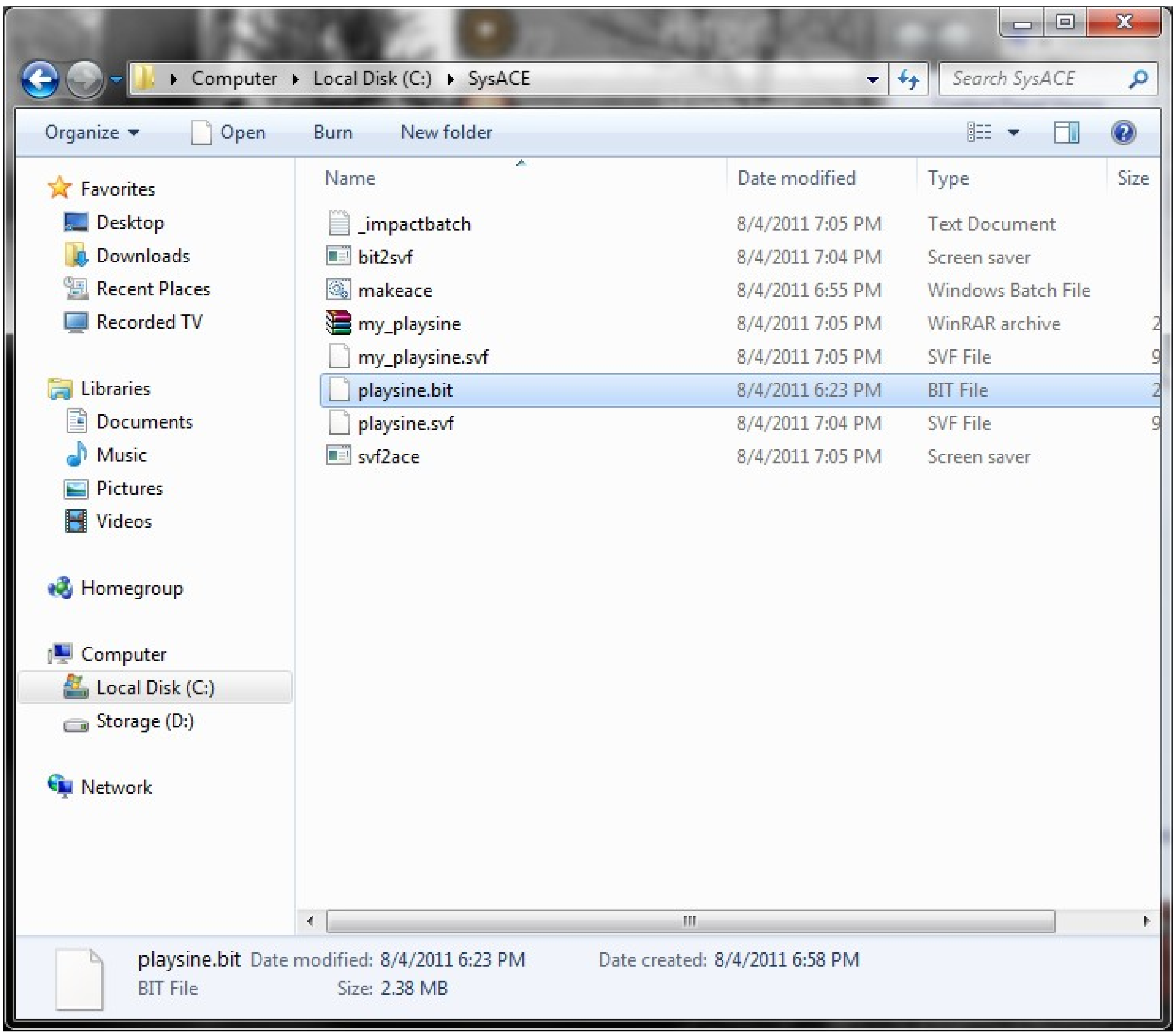
Type these commands to change directory to the SysACE folder:

cd \SysACE

Then to convert the file type:

makeace playsine

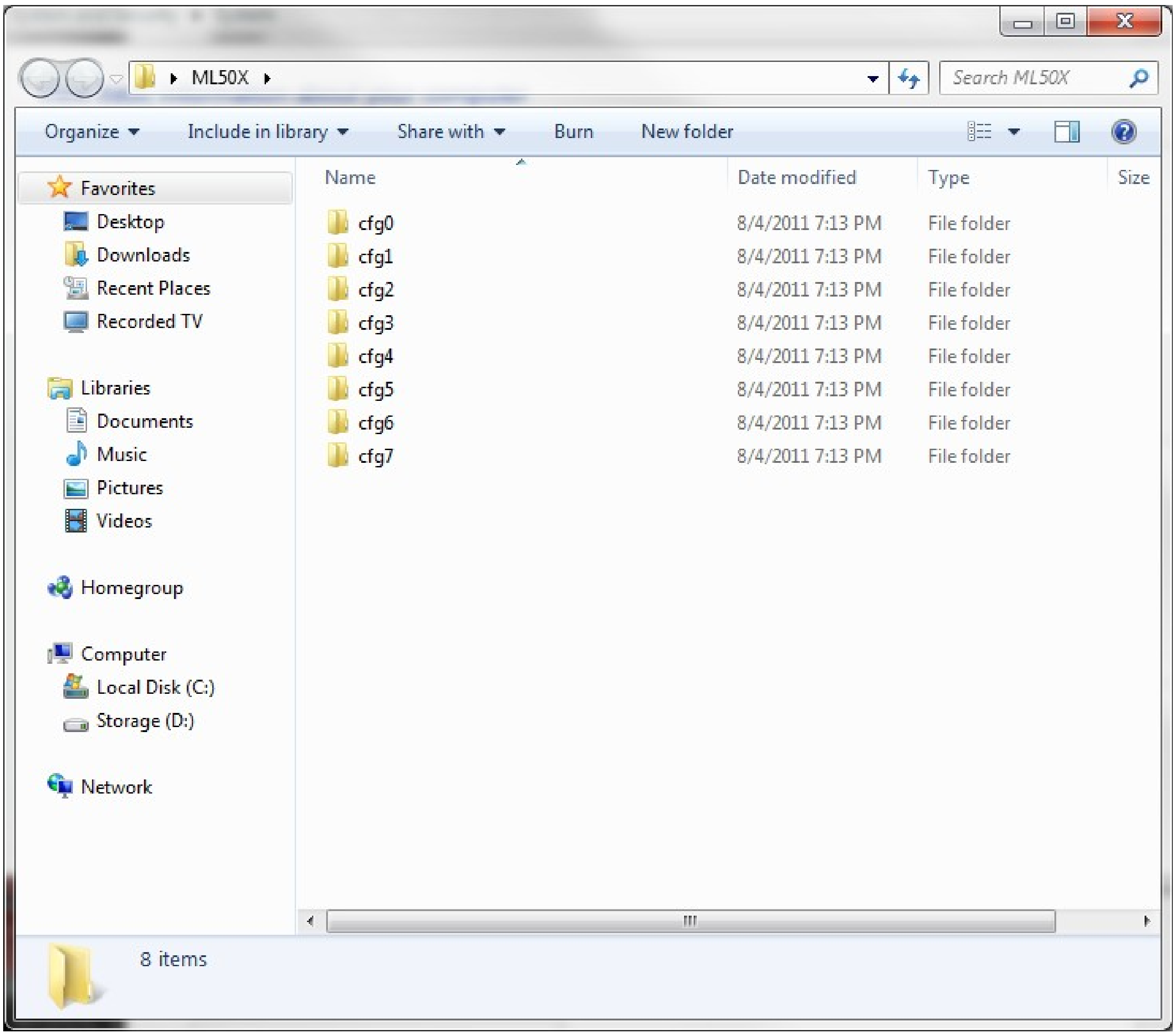
Once this finishes, several files will be created in the SysACE folder, we are interested in the my_playsine.ace file:



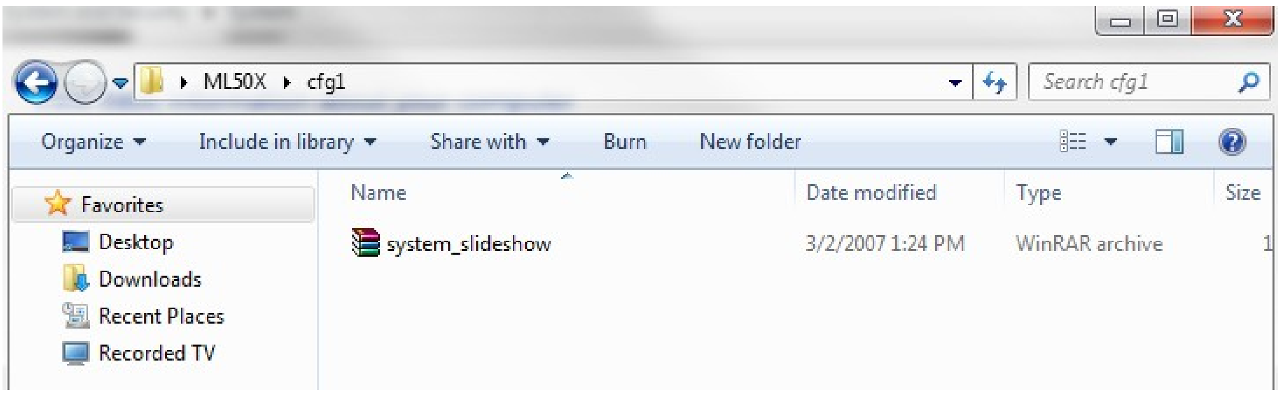
Setting up the Compact Flash

Make sure that your compact flash is properly formatted, see the beginning of this entire tutorial.

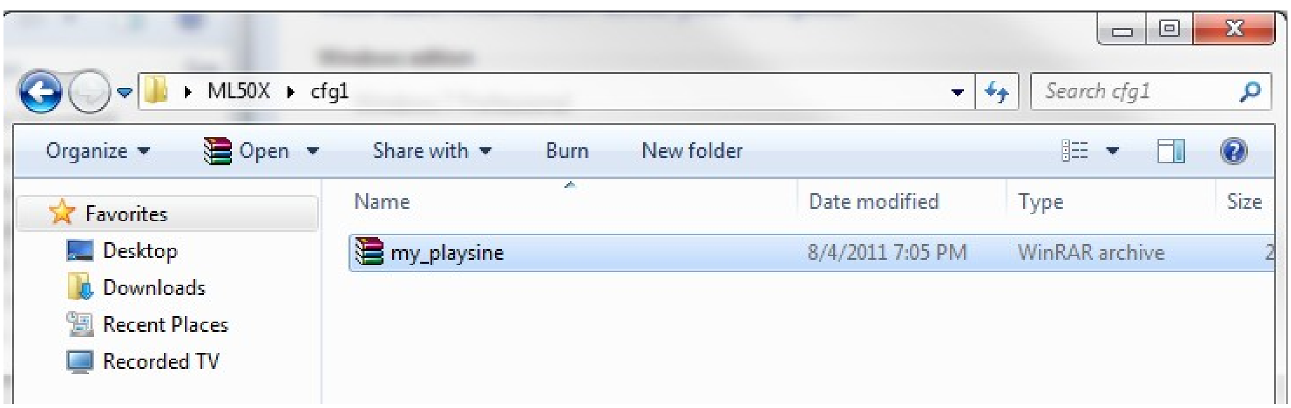
Turn of the ML506 and gently remove the compact flash. Insert the compact flash into your compact flash reader and open up the ML50X folder:



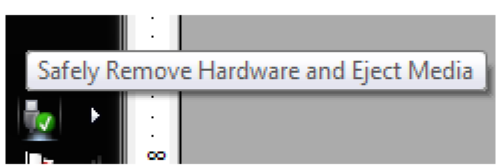
Pick one of these folder (I chose cfg1) and open it:



Inside you will notice some factory files, delete this file and replace it with the my_playsine.ace file we created (note: deleting this file will render the Xilinx Demo program inoperable):



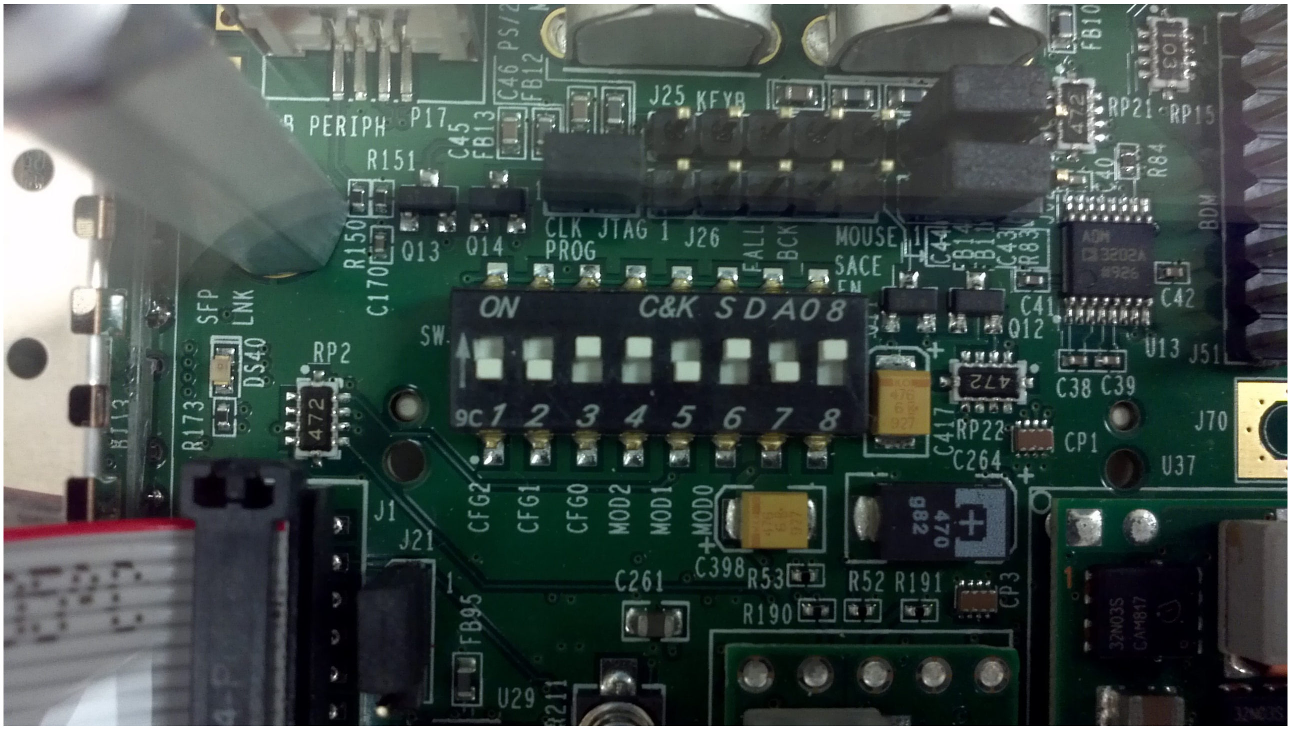
Safely eject, the compact flash:



Then insert the compact flash back into the ML506.

Since our file is in cfg1, we need to change the first three bits of SW3 to : 001 which is 1 in binary and leave the rest of the bits as: 10101.

Here is a picture:



Now turn on the board and your program should start automatically.

You can put more designs into the other folders, just change the first three bits of SW3 to correspond to the folder number, then turn on the board.

Conclusion

More Designs

By finishing this tutorial, you should be familiar with the many capabilities of the ML506 and the various Xilinx softwares such as ISE and EDK. There are plenty of tutorials online and from Xilinx. Here are some links if you want to learn more:

System Generator: http://www.xilinx.com/support/sw_manuals/sysgen_user.pdf

Xilinx EDK: http://www.xilinx.com/support/documentation/sw_manuals/xilinx13_1/edk_ctt.pdf

FPGA Tutorials: http://www.fpgadeveloper.com/