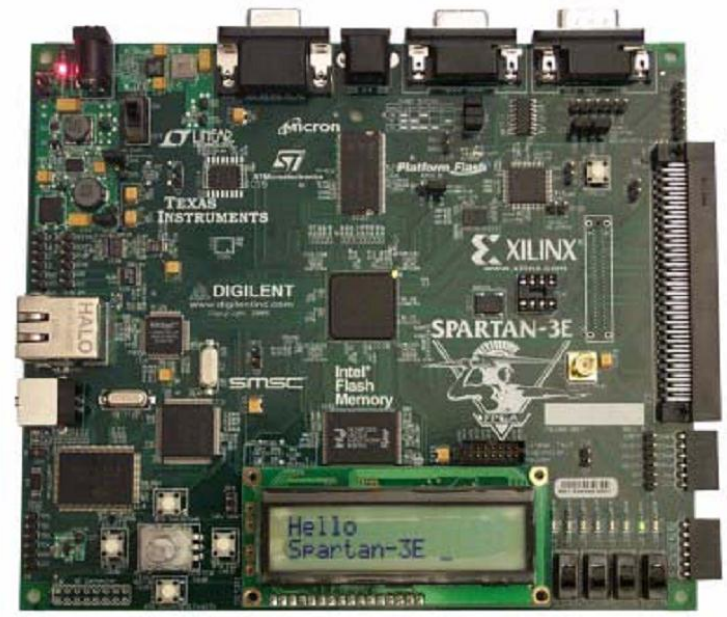
Beginner's Guide to Xilinx Spartan-3E FPGA Starter Kit Board

(Revision 2)

By: Nick Desaulniers and Cody Cziesler

RIT Department of Computer Engineering

5/5/11



Introduction

This guide is for students new to FPGAs who are using the Spartan-3E Starter Kit Board for a class such as Digital Systems Design (0306-561). This guide is a crash course in getting code onto the FPGA and mapping the various components on the board to your design.

Table of Contents

Section 1: Xilinx ISE Webpack, License, and Drivers <optional>

Installing the Xilinx ISE Webpack is optional. It is highly suggested so that way you can work on your FPGA at home. At your home computer, visit the Xilinx download website http://www.xilinx.com/support/download/index.htm or search for "Xilinx ISE Webpack" and install the ISE Design Suite for your platform. Create a Xilinx account, download the .tar file, unzip it, install the software, and follow the wizard to getting a license file (.lic). To install drivers, power up the board first, then connect the USB cable. Windows should automatically find and install drivers. If not, you can go into device manager to install the drivers manually.

Section 2: Hardware Description Language Programming and Simulation

It is implied that you know either VHDL or Verilog by this point if you are trying to figure out how to program a FPGA. Refer to your notes from Hardware Description Languages (0306-351) on how to write code. In this section, we provide you with sample code that way you can follow along. Take this code and save it in a file "bc8.v". It is written in Verilog HDL.

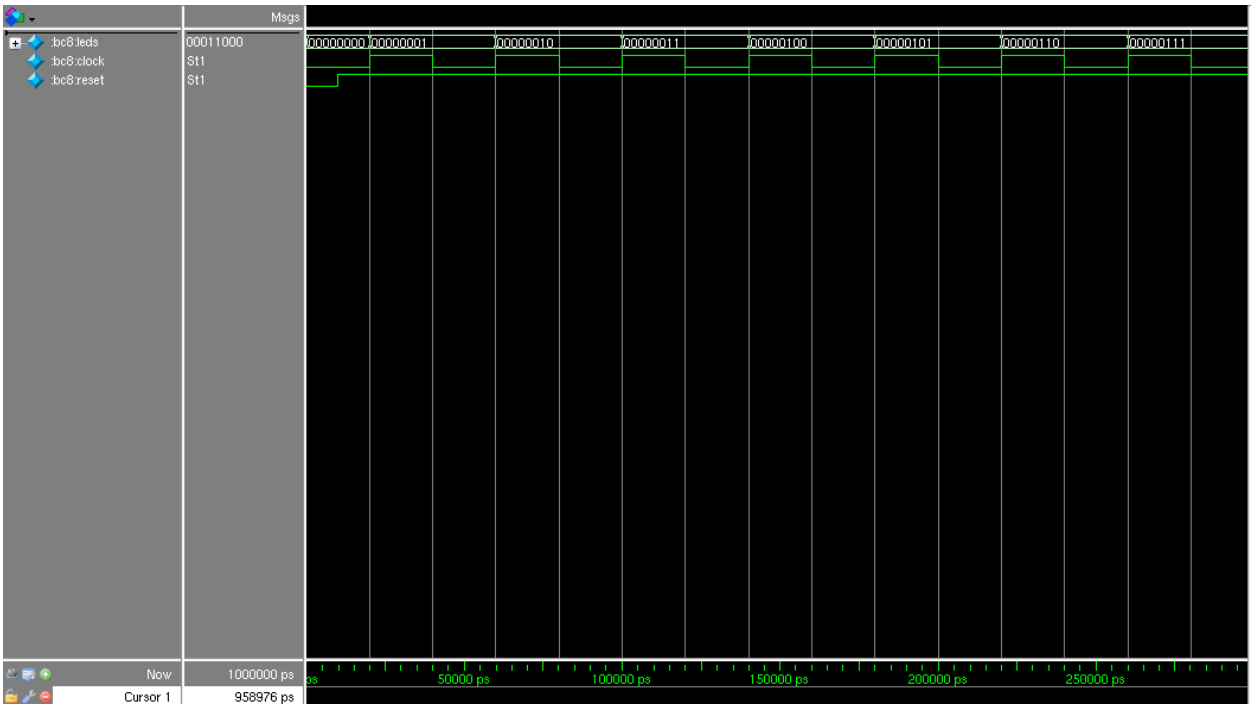
bc8.v

```
/*
--File: bc8.v
--Module: bc8
--Author: Nick Desaulniers, Cody Cziesler
--Created: 3/28/11
--Verilog HDL
--Description: 8 Bit Counter
*/
module bc8(
  output reg [7:0] leds,
  input wire clock,
  input wire reset
);
  reg [23:0] state;

  always @( posedge clock or negedge reset ) begin
    if( reset == 1'b0 ) begin
      state <= 24'b0;
      leds <= 8'b0;
    end else begin
      state <= state + 24'b1;
      if(state == 24'hFFFFFF) begin
            leds <= leds + 1'b1;
      end
    end
  end
endmodule
```

This example is an 8-bit binary counter. The input “reset” clears the output, and every clock cycle changes the state. 2^{24} or 16,777,216 states are used as a clock divider. Clock dividers are important because the onboard clock on the Spartan-3E Starter Kit Board operates at 50 MHz. The 50 MHz clock would change the LEDs so fast you would not be able to see it. Instead, we use 16,777,216 states and increase the count once every 16,777,216 states. This slows down the rate at which the LEDs change. Here is a simulation of the code without the clock dividers so that we can assure that the output “leds” are changing:

Waveform for Simulation (Made in ModelSim)



This shows our code counts up. The vector “leds” is tied to the eight LEDs on the Spartan-3E Starter Kit Board. This is done via a *.ucf, or constraint file, which is shown in the next section. Just for reference here is the force file used in simulation:

```
force reset 0 0, 1 10ns

force clock 0 0, 1 20ns -repeat 40ns
```

Simulation of code solves most problems faced later on in the ISE Design Suite; if your code is compliable and able to simulate correctly, then most errors are avoided in the Synthesis and Implementation steps.

Section 3: Constraints

Constraints are used to tie physical components such as switches, LEDs, knobs, external pins, and other various components on the evaluation board to pins on the FPGA specified in your HDL code. The Spartan-3E Starter Kit Board User Guide, Appendix B has an example User Constraint File (.ucf) with every declaration for every component on the board. You can find the guide here: http://www.digilentinc.com/Data/Products/S3EBOARD/S3EStarter_ug230.pdf or by searching "Spartan-3E User Guide." Here is the User Constraint file we used in our example. Please copy the following into a file and save it as "counter.ucf".

counter.ucf

```
# ==== Clock Source ====
NET "clock" LOC = "C9" | IOSTANDARD = LVCMOS33;
NET "clock" PERIOD = 5ns HIGH 40%;

# ==== Discrete LEDs (LED) ====
NET "leds<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "leds<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "leds<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "leds<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "leds<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "leds<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "leds<6>" LOC = "E9"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
NET "leds<7>" LOC = "F9"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;

# ==== Slide Switches (SW) ====
NET "reset" LOC = "L13" | IOSTANDARD = LVTTL | PULLUP ; # SW1
```

All of these lines of code were copied directly from the Spartan 3E Starter Kit Board User Guide. Code in blue is one of our input or output signals specified in our code. The locations ("LOC") are printed on the PCB of the FPGA board next to each component. The first line ties our input clock signal to the onboard clock. The second line changes the clock period and thus clock frequency. The clock was can be edited to have a period between 5ns and 20ns. The middle block of code shows how each signal of the leds vector is tied to an LED on the board. The LEDs are active high, but you already knew that since you read the User Guide, right? The User Guide has a lot of information to help you understand how the various components work. The last line of the UCF ties our reset signal to SW1 (Switch 1 on the board).

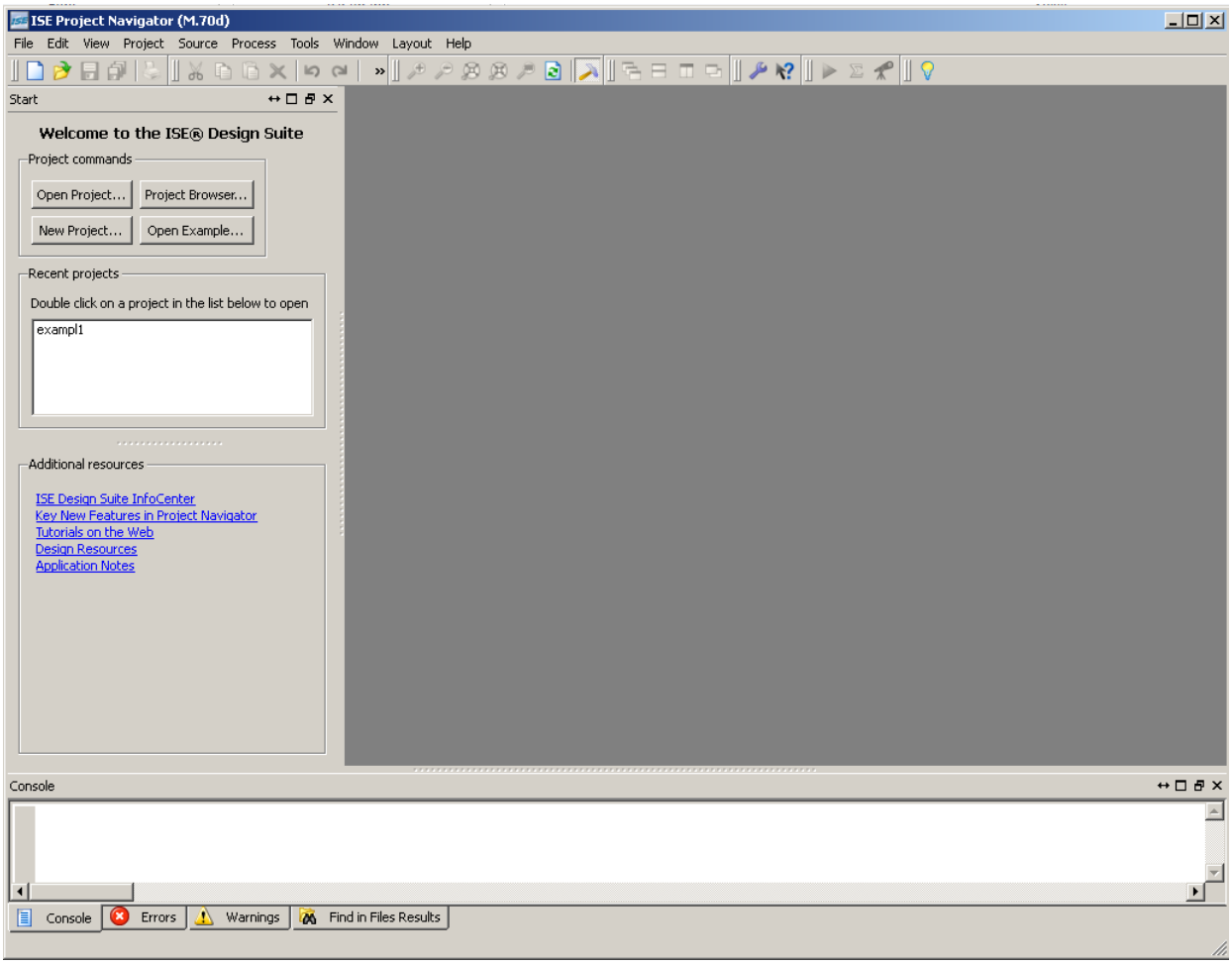
With your design (.vhd or .v) and constraint (.ucf) you are now ready to begin synthesis.

Section 4: Synthesis, Implementation, and Generating Bit File

On a lab computer, open the Integrated Design Environment (IDE):

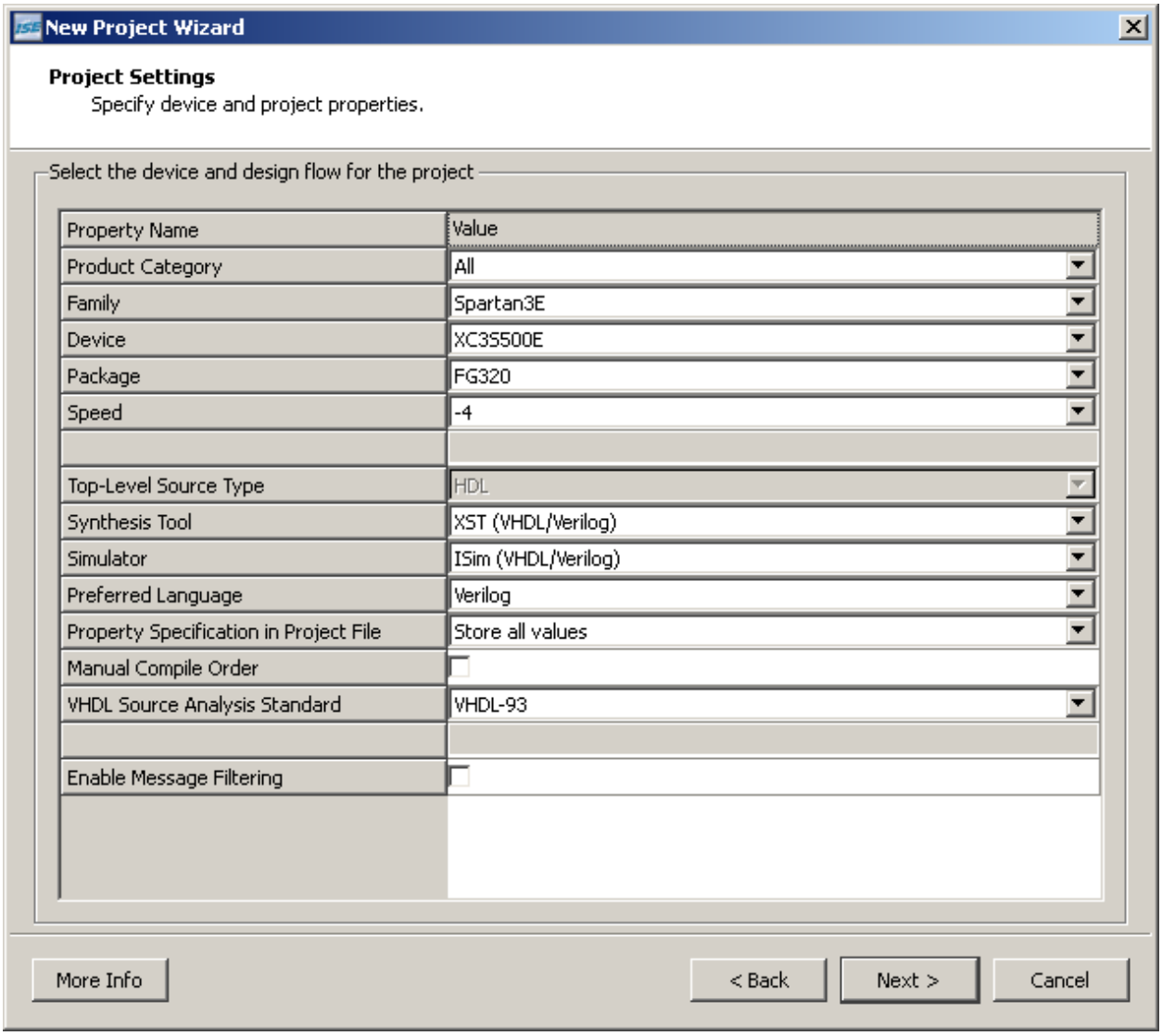
Start → All Programs → Xilinx ISE Design Suite 12.3 → ISE Design Tools → Project Navigator

ISE Project Navigator Startup Window



Click “New Project…” and give it a title (A folder with the title is created by default. This location will contain your bit file at the end of this section). Then, click “Next”, and use the following Project Settings:

New Project Wizard in ISE Project Navigator



Click “Next”, and then “Finish”.

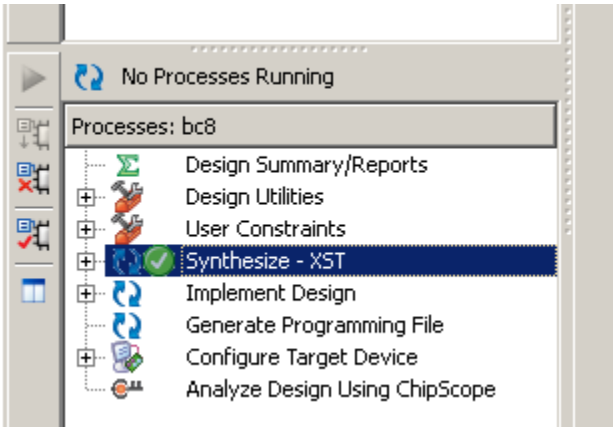
Now it is time to add your design file(s) and constraint file. On the left, click the icon with a green plus and single document “Add Source”.

Adding Source

Select both "bc8.v" and "counter.ucf", and then click "OK". If you have multiple entities or modules, you will need to specify which module is the top module. This can be done by right clicking on the entity/module on the left "Hierarchy" window and clicking on "Set as Top Module". You can also view your code by double clicking on it.

The first step in porting your code is to synthesize it. Synthesizing your code generates the necessary hardware to implement the desired behavior. To begin synthesis, double click on the left where it says "Synthesize – XST."

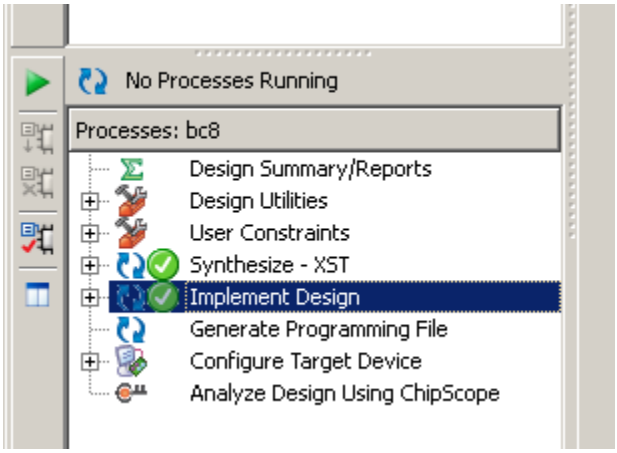
Synthesis Complete



As long as your code is synthesizable, you should not have a problem here. If you do, verify your code can be synthesized in the Spartan-3E FPGA.

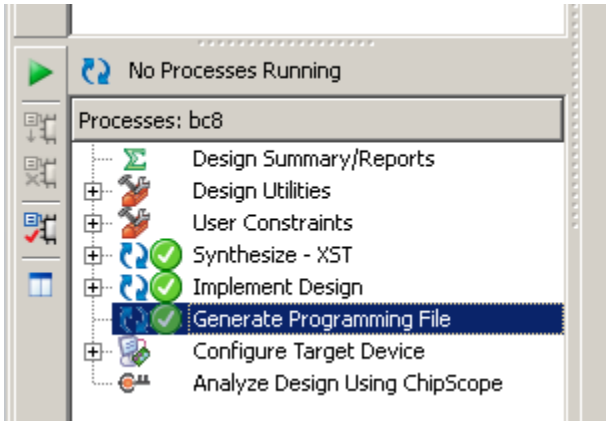
Implementation takes your design and translates, maps, and routes your design to your FPGA chip. Run implementation by double clicking on "Implement Design" on the left.

Implementation Complete



The final step in this stage is generating the *.bit file. The bit file is created from the source code (.vhd or .v) and the constraint file (.ucf). The bit file (.bit) is what is put onto the FPGA chip to program it. To create the bit file, double click on “Generate Programming File” on the left.

Generation Complete



Now, in the same directory as your project, you should have a file with the name of your top module with the extension “.bit”. If you are following along with our example code then you should have a file “bc8.bit”.

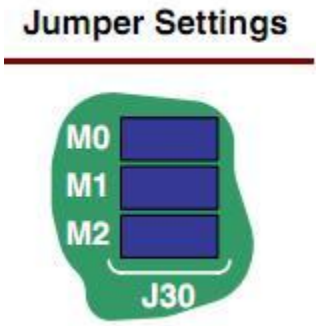
The last step is to load the bit file onto your FPGA. Do not close the ISE Project Navigator just yet.

Section 5: Jumper Configuration, Generating PROM File, and Loading Bit File

To load the bit file, start by connecting the power and turning on your board. Then, connect it to the PC via a USB cable. A red LED near the power switch on the board will turn on when the board is powered on, and a green LED by the USB shows a proper connection.

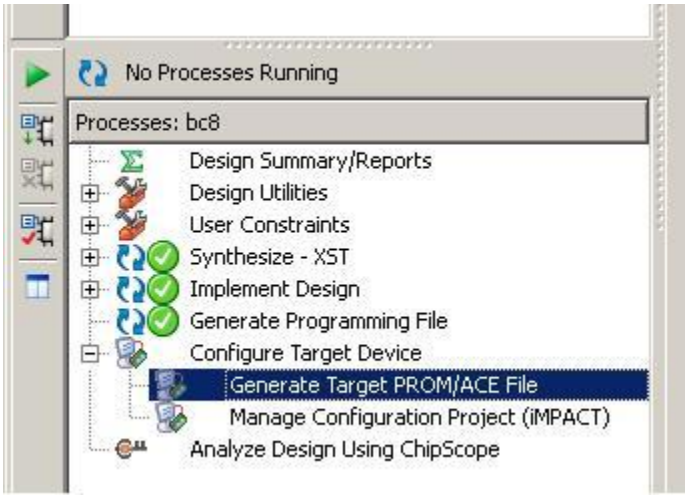
Jumpers on the board (J30) need to be set to configure the FPGA to load the configuration from Platform Flash PROM. This grouping, J30, is located near the middle Serial Port. Configure the jumpers as shown below. All jumpers should be connected as shown below.

Jumper Settings for J30 Jumpers to Load Configuration from Platform Flash PROM



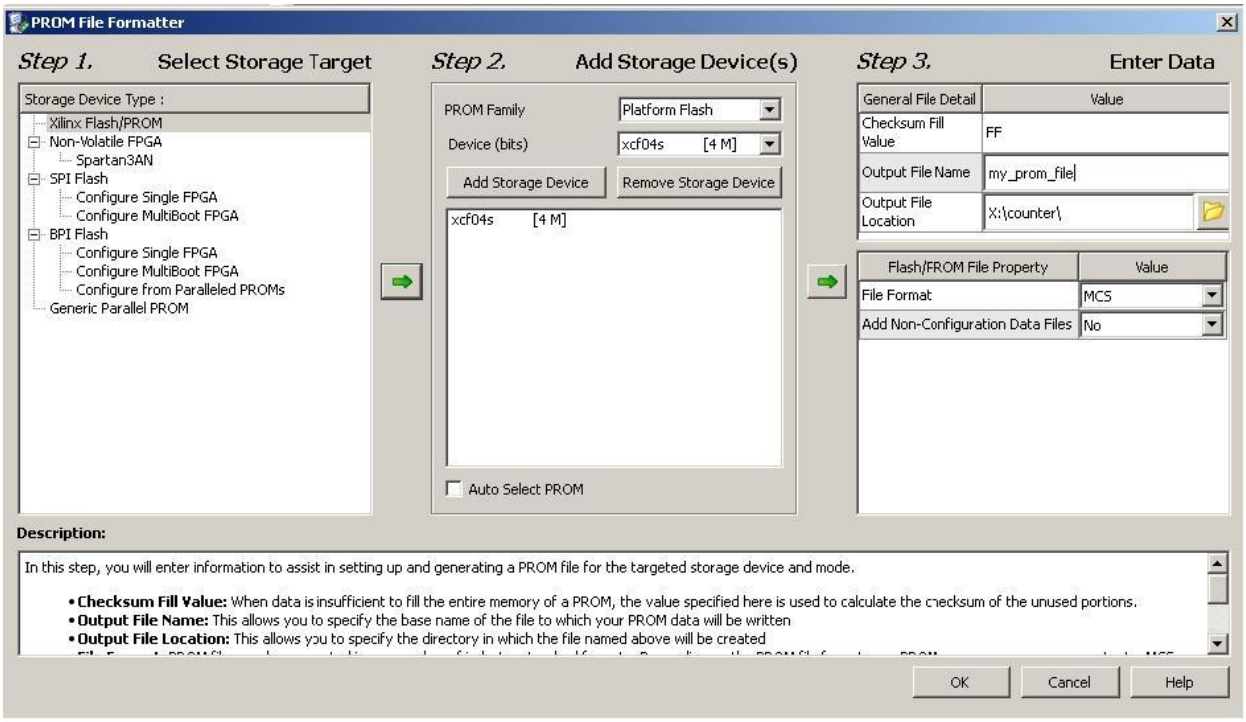
To open the software needed, expand “Configure Target Device” in the ISE Project Navigator and double click “Generate Target PROM/ACE File”.

Selecting Generate Target PROM/ACE File



Click “OK” on the warning. This will open up ISE iMPACT. On the left window, double click “Create PROM File (PROM File Formatter).” In *Step 1: Select Storage Device*, select Storage Device Type: “Xilinx Flash/PROM,” then click the green arrow. In *Step 2: Add Storage Device(s)*, select Device (bits): “xcf04s [4M]” from the dropdown list, click the “Add Storage Device” button, and then click the green arrow. In *Step 3: Enter Data*, title your PROM file in the Output File Name field, as you will need it later.

PROM File Formatter



Click “OK” in the “PROM File Formatter” box, “OK” in the “Add Device” box, select your bit file (“bc8.bit”), “Open”, “No” in the “Add Device” box, and finally “OK” in the “Add Device” box. In the lower left window, “iMPACT Processes,” double click “Generate File…” Generate Succeeded should appear.

Generate Succeeded

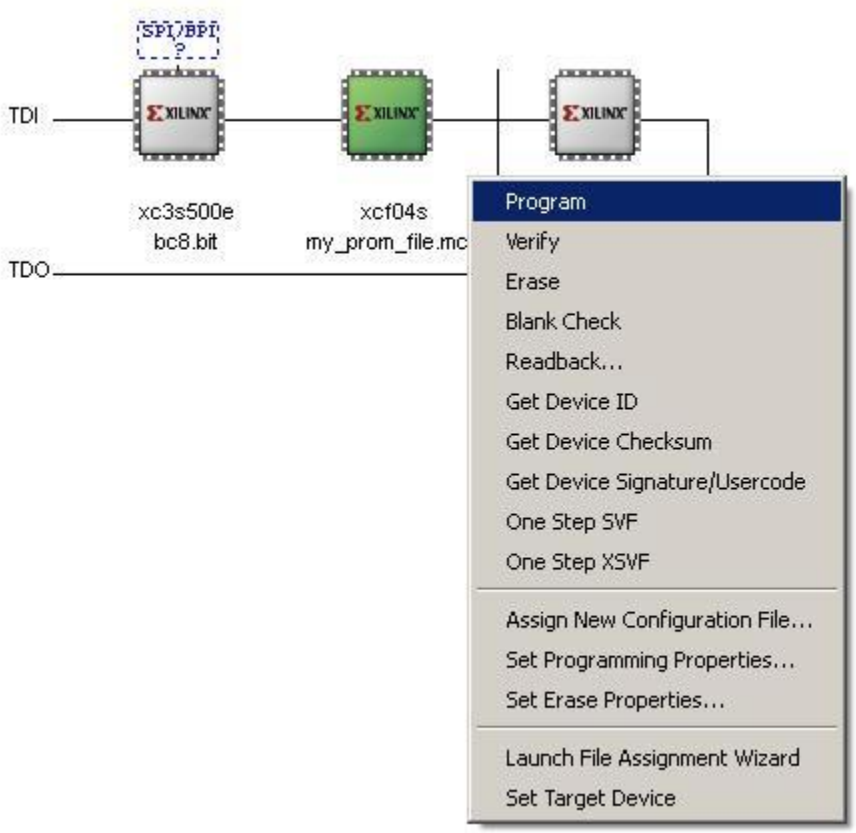
In the upper left window, “iMPACT Flows,” double click “Boundary Scan” then right click in the middle window near the “Right click to Add Device or Initialize JTAG chain” and click “Initialize Chain.” This should run correctly if your board is connected.

Initialize Chain

Click “Yes” on the “Auto Assign Configuration Files Query Dialog” box. You should now be able to browse to your bit file. Do so and click “Open”, and click “No” in the “Attach SPI or BPI PROM” box. You should now be able to browse to your PROM file (my_prom_file.mcs). Do so and click “Open”, then “Bypass” in the next box. In the “Device Programming Properties – Device 1 Programming Properties” box, in the left Category window, expand Boundary Scan and click “Device 2 (PROM xcf04s),” then check the “Load FPGA” checkbox, and finally click “OK.”

The FPGA chip “xc3s500e” should have your bit file underneath (bc8.bit in our example), the Platform Flash PROM chip “xcf04s” should have your PROM file underneath (my_prom_file.mcs in our example), and the last chip should be bypassed since we are not using it. Finally, right click on the “xcf04s” chip and click “Program”.

Programming the FPGA with the specified bit file



You should now see the phrase “Program Succeeded”:

Program Succeeded Notification

Also, an amber LED on the FPGA board should light up to show that the chip was successfully programmed. It is near the jumpers by a VGA port. Now your board is programmed! Check it out by throwing the reset switch HIGH (towards the LEDS). As long as you keep your jumpers configured for Platform Flash PROM (marked “M.S.” on the board), you should also be able to shut the off the FPGA

and disconnect power, then reconnect power and power on the board. The Amber LED will light up showing the FPGA has been reconfigured, and you should be good to go.

LED Binary Counter

Section 6: Helpful Tips

- Common courtesy is to erase the flash when you are done with the FPGA board. In iMPACT instead of clicking Program, click Erase.
- Make sure you use a constraint (*.ucf) file and check it twice. If there is a mistake in the ucf file that causes a VCC power pin to be shorted to a GND pin, there is a good possibility of frying the FPGA.
- If you think you followed the directions to a tee and the FPGA does not perform as expected, try rerunning the Synthesis process. Sometimes, the Xilinx flow will not run correctly.