EEDG/CE 6370
Design and Analysis of Reconfigurable Systems
Homework 2
Prototyping on the Terasic DE1-SoC board

1. Laboratory Objectives

- Synthesize designs targeting a particular prototyping board.
- Program the FPGA on a prototyping board and have a fully functional design.
- Understand the use constraint files.
- Learn how to interface with the prototyping board through switches and buttons.

2. Summary

The following design will output the 4 MSB of a 32-bit up and down counter to 4 LEDs of the DE1-SoC prototyping board. The design will count UP or DOWN based on the position of a switch.

3. Pre-lab

- Review the lecture slides that covers the basic structure of an FPGA and DE1-SoC board
- Collect the Terasic DE1-SoC board.

4. Tool Requirements

- Quartus Prime Lite
- Questa
- DE1-SoC board

5. Terasic DE1-SoC Board

The Terasic DE1-SoC prototyping board contains a single Cyclone V FPGA connected to a series of peripherals and interfaces that allows it to communicate with other devices. Figure 1and 2 show an overview of the board

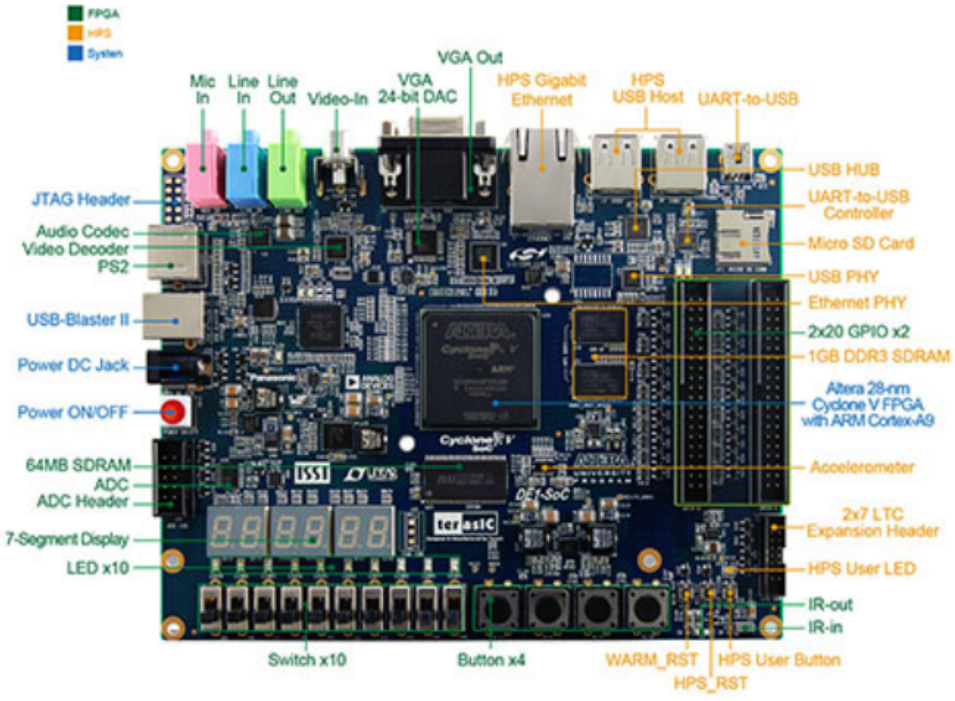


Figure 1 Terasic DE1-SoC Evaluation board (Front)

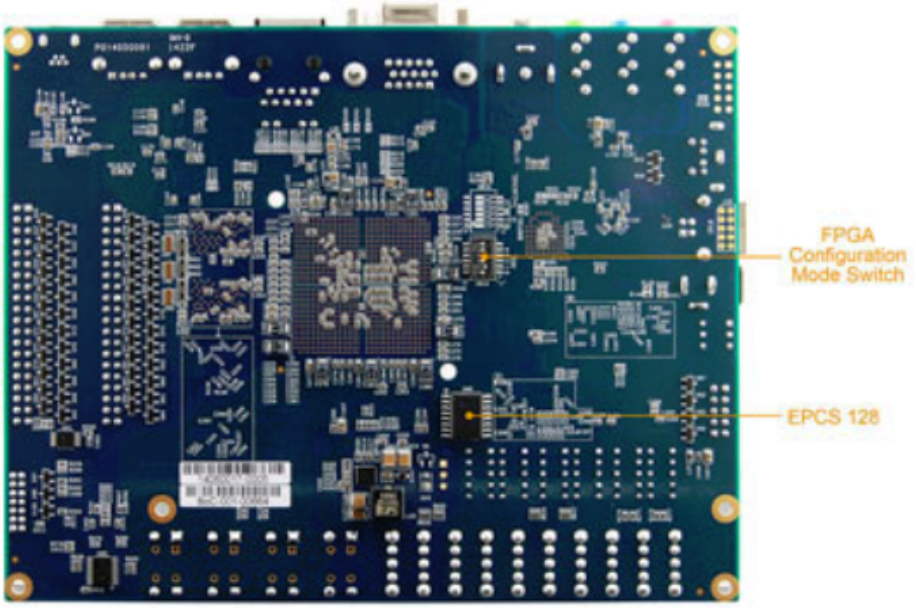


Figure 2 Terasic DE1-SoC Evaluation board

The DE1-SoC board contains the following:
FPGA:

- Cyclone V SoC 5CSEMA5F31C6 Device
- Dual-core ARM Cortex-A9 (HPS)
- 85K Programmable Logic Elements
- 4,450 Kbits embedded memory
- 6 Fractional PLLs
- 2 Hard Memory Controller

Switches:

- 4 User Keys (FPGA x4)
- 10 User switches (FPGA x10)
- 11 User LEDs (FPGA x10 ; HPS x 1)
- 2 HPS Reset Buttons (HPS_RST_n and HPS_WARM_RST_n)
- Six 7-segment displays
- Memories:
- 64MB (32Mx16) SDRAM on FPGA
- 1GB (2x256Mx16) DDR3 SDRAM on HPS
- Micro SD Card Socket on HPS

Communications:

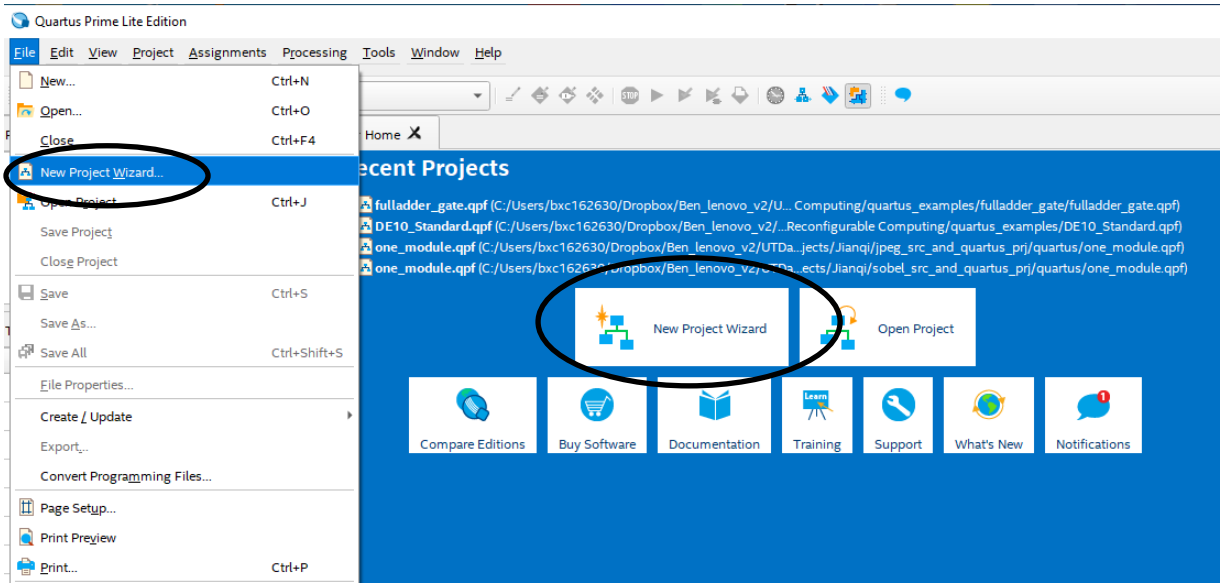
- Two Port USB 2.0 Host (ULPI interface with USB type A connector)
- USB to UART (micro USB type B connector)
- 10/100/1000 Ethernet
- PS/2 mouse/keyboard
- IR Emitter/Receiver

More information can be found at the manufacturers board web page *de1-soc.terasic.com*

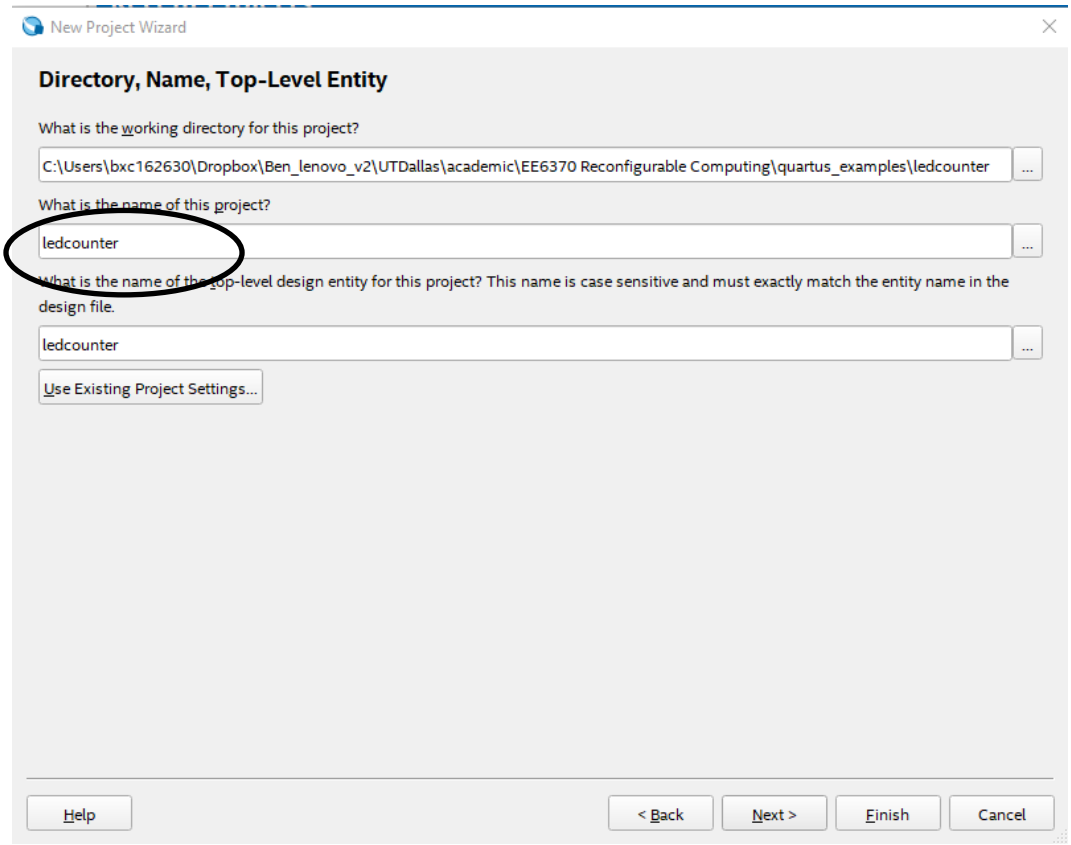
- Open Quartus Prime click in icon→ Run as) 

In case of Windows run the tools always as administrator (right

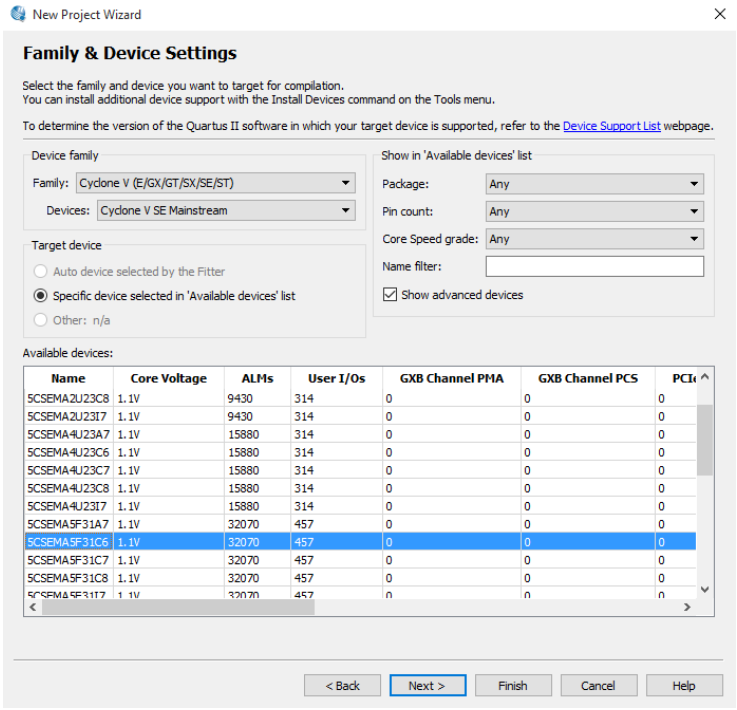
- Create a new project.
  File ➔ New Project Wizard



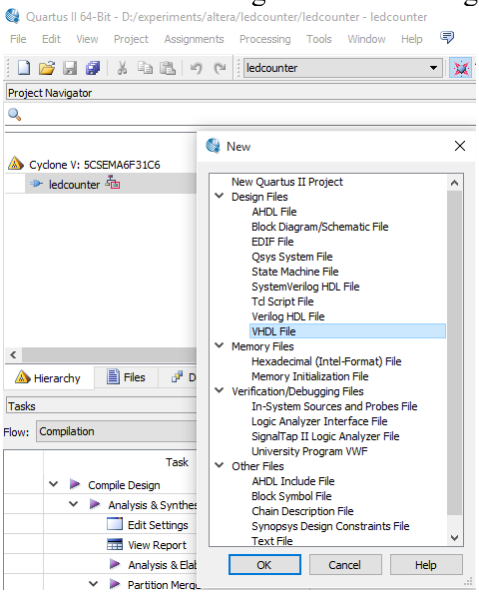
Name the project “ledcounter”



- Click next → empty project →
- Click next (do not add any files)
- Select Cyclone V 5CSEMA5F31C6 device
  (**Note:** This is very important: This FPGA has to match the FPGA in the DE1-SoC board)



- Click next and Finish.
- New → Design Files → Verilog or VHDL

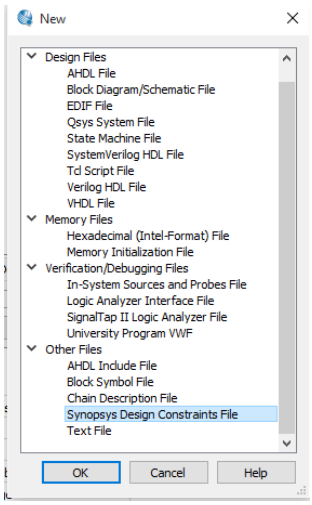


- Call the file ledcounter.vhd or ledcounter.v

- Edit source code of the 32-bit counter given below and/or re-write it in Verilog if you are more comfortable using Verilog instead of VHDL
  The counter should only output the 4 most significant bits of the 32-bit counter.
- Simulate the design using Questa to make sure that the program works as specified. Review instructions from HW #1 for this.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL; -- NEEDED to USE  +/- for STD_LOGIC data types

entity ledcounter is
 port(  clk:  in std_logic;
    reset:  in std_logic;
    direction:  in std_logic;
    count:  out std_logic_vector(3 downto 0)
  );
end ledcounter;

architecture Behavioral of ledcounter is
  signal pre_count: std_logic_vector(31 downto 0);

begin
     process(clk, reset)
     begin
      if (reset = '0') then
        pre_count <= (others => '0');
      elsif (clk='1' and clk'event) then
        if (direction = '1') then
            pre_count <= pre_count + '1';
      else
            pre_count <= pre_count – '1';
        end if;
      end if;
     end process;

--assign the 4 most significant bits to the output

end Behavioral;
```

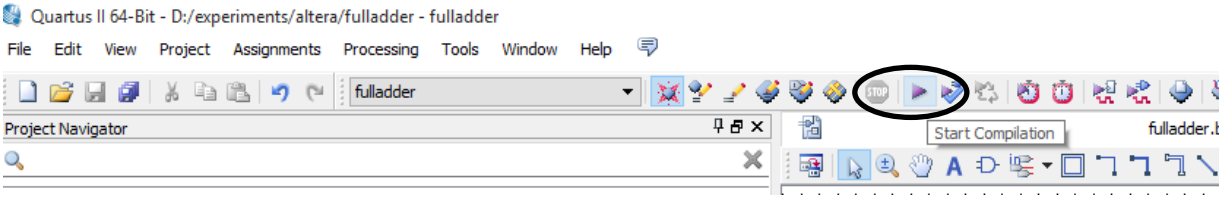
- Create a Synopsys Design Constraints file (.sdc) in which to declare that the design has a clock and specify its name. Other constraints can also be specified here.



- Add the following to the .sdc file where the name of the clock has to be the same as in your Verilog/VHDL file:

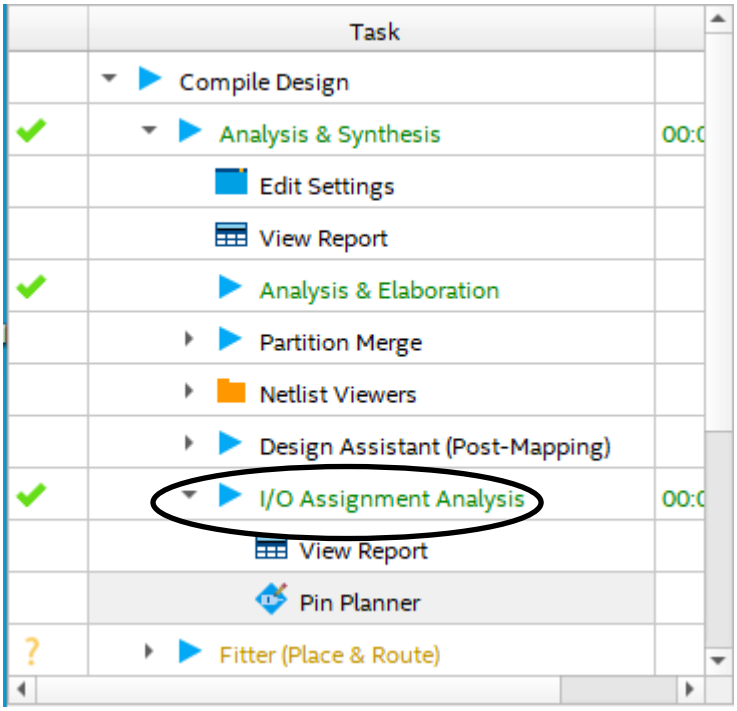
  ```
  create_clock -name "clk" -period 20.000ns [get_ports {clk}]
  derive_pll_clocks
  derive_clock_uncertainty
  ```

- The period of the clock is 20 ns because the DE1-SoC board has a 50MHz clock (1/50MHz=20ns)
- Compile the design to make sure that there are no errors.



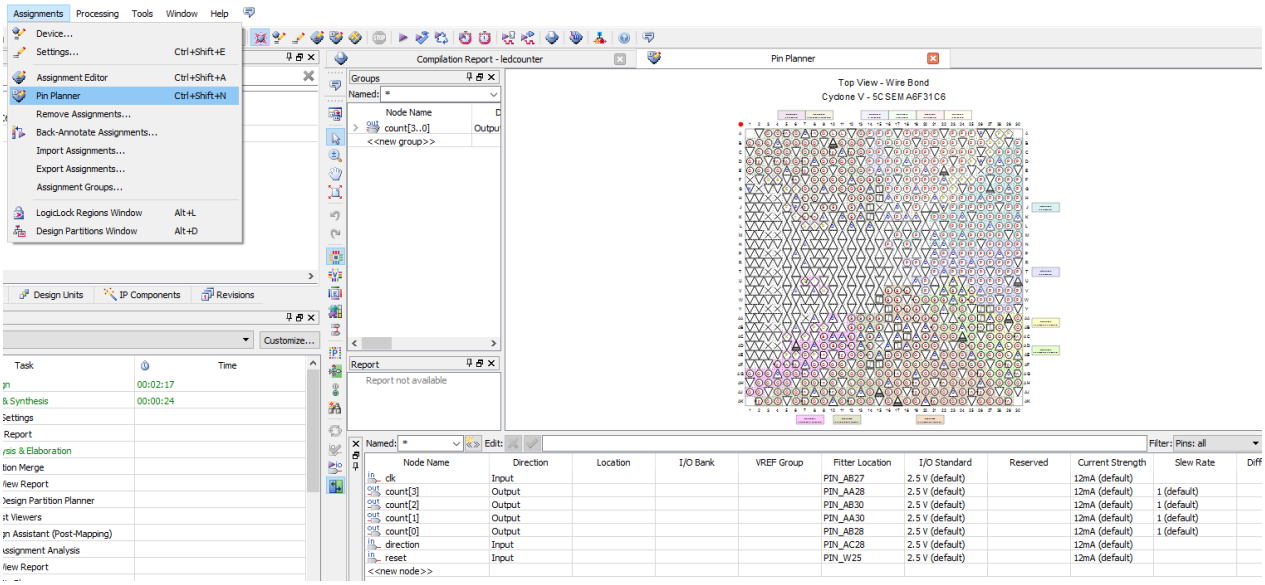
- Verify that the synthesis processes have finished successfully.

Note: You need to synthesize at least until the I/O assignment analysis state in Quartus to allow the pin planner to edit the pint assignment



Module IOs to FPGA pin assignment

- Create constraint file to assign IOs to fixed FPGA pins. Click on Assignments→Pin Planner
- This should open a new window with the entity ports displayed and a schematic of the FPGA pins.



- Based on the following DE1-SoC manufacturer's datasheet, assign the module's IOS to the following FPGA pints in the location column.

Table 3-6 Pin Assignment of Slide Switches

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
SW[0]	PIN_AB12	Slide Switch[0]	3.3V
SW[1]	PIN_AC12	Slide Switch[1]	3.3V
SW[2]	PIN_AF9	Slide Switch[2]	3.3V
SW[3]	PIN_AF10	Slide Switch[3]	3.3V
SW[4]	PIN_AD11	Slide Switch[4]	3.3V
SW[5]	PIN_AD12	Slide Switch[5]	3.3V
SW[6]	PIN_AE11	Slide Switch[6]	3.3V
SW[7]	PIN_AC9	Slide Switch[7]	3.3V
SW[8]	PIN_AD10	Slide Switch[8]	3.3V
SW[9]	PIN_AE12	Slide Switch[9]	3.3V

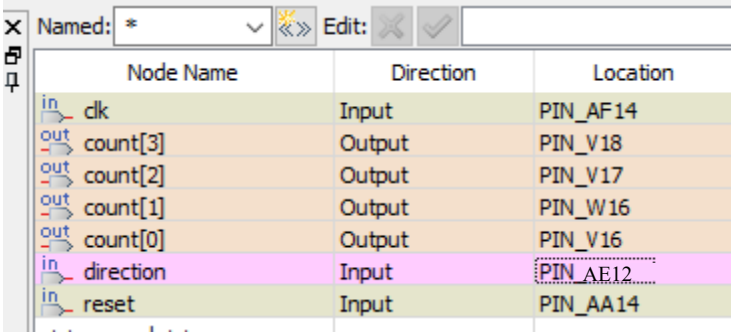
Table 3-7 Pin Assignment of Push-buttons

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
KEY[0]	PIN_AA14	Push-button[0]	3.3V
KEY[1]	PIN_AA15	Push-button[1]	3.3V
KEY[2]	PIN_W15	Push-button[2]	3.3V
KEY[3]	PIN_Y16	Push-button[3]	3.3V

Table 3-8 Pin Assignment of LEDs

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
LEDR[0]	PIN_V16	LED [0]	3.3V
LEDR[1]	PIN_W16	LED [1]	3.3V
LEDR[2]	PIN_V17	LED [2]	3.3V
LEDR[3]	PIN_V18	LED [3]	3.3V
LEDR[4]	PIN_W17	LED [4]	3.3V
LEDR[5]	PIN_W19	LED [5]	3.3V
LEDR[6]	PIN_Y19	LED [6]	3.3V
LEDR[7]	PIN_W20	LED [7]	3.3V
LEDR[8]	PIN_W21	LED [8]	3.3V
LEDR[9]	PIN_Y21	LED [9]	3.3V

Assign the following pins to the different IOs

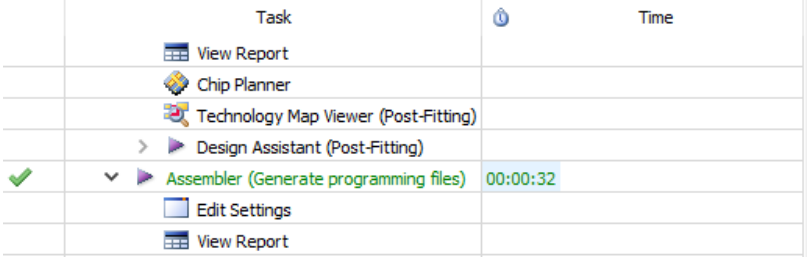


- These assignments will be saved in ledcounter.qsf project file as follows:

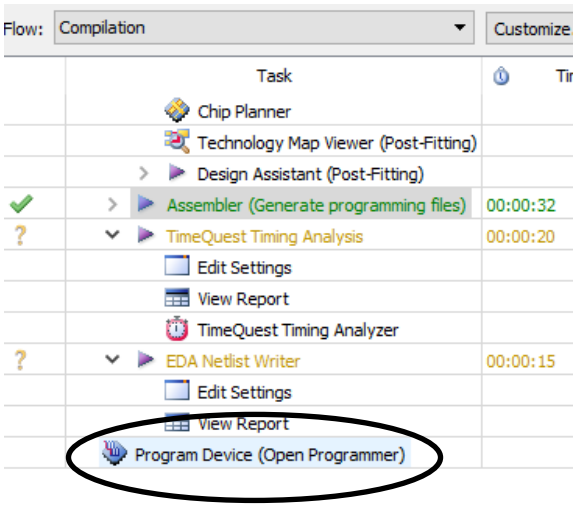
```
set_location_assignment PIN_AF14 -to clk
set_location_assignment PIN_V18 -to count[3]
set_location_assignment PIN_V17 -to count[2]
set_location_assignment PIN_W16 -to count[1]
set_location_assignment PIN_V16 -to count[0]
set_location_assignment PIN_AE12 -to direction
set_location_assignment PIN_AA14 -to reset
```

Close the pin planner and check that the assignment has been recorded into the ledcounter.qsf file.

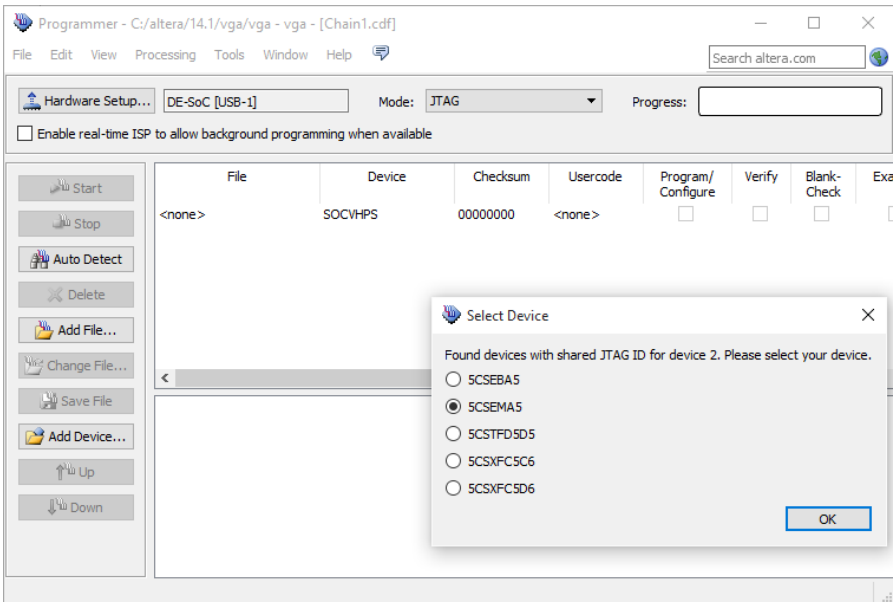
- Resynthesize the entire project and generate a programming file (.sof file in output_files folder)



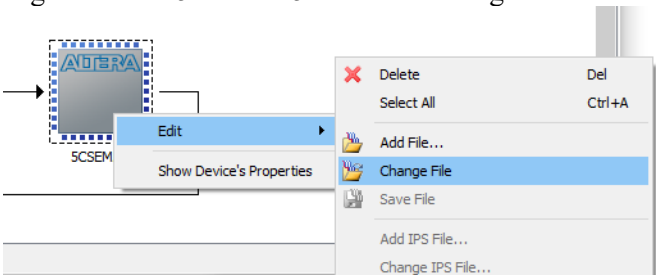
- Connect the FPGA board to the power, the USB Blaster cable (big USB connector next to the power supply connector)
- Power the FPGA on
- Open the programmer



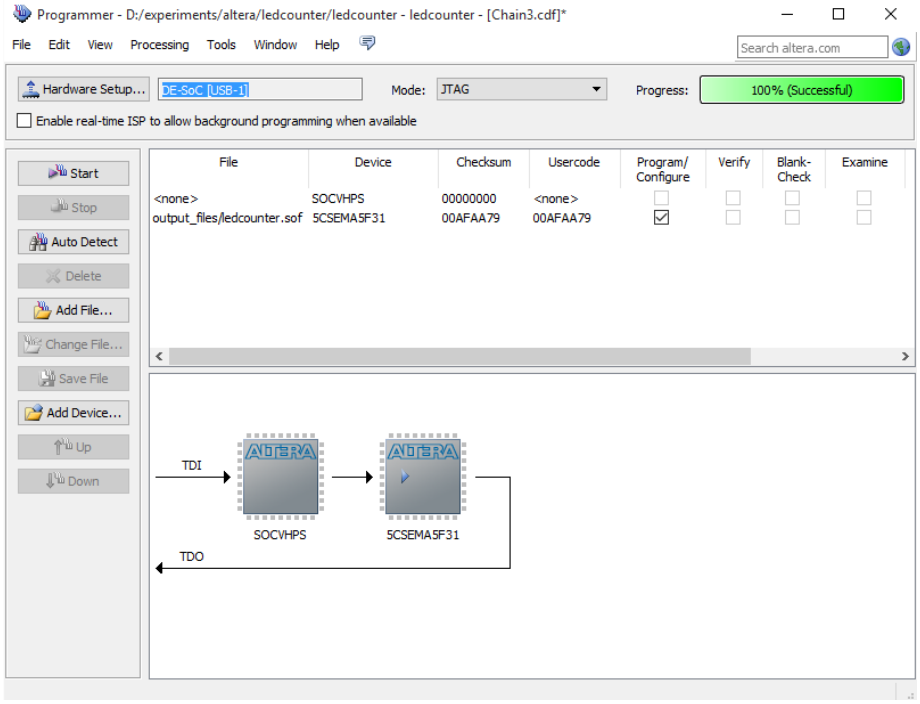
- Click on Hardware Setup and select DE-SoC [USB-1] FPGA
- Click on Auto Detect
- Select 5CSEMA5



Right click on 5CSEMA5 device → change file → select /output_files/ledcounter.sof



- Click on Auto-detect if the start button is not enabled → Start



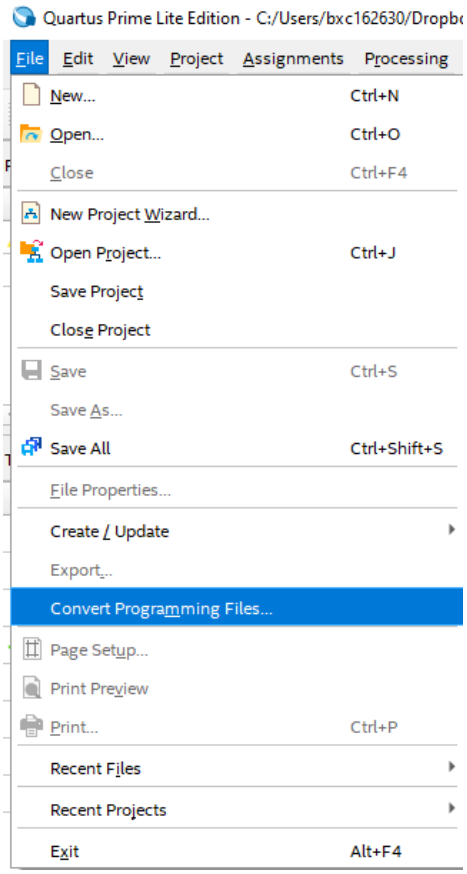
- The FPGA should have been successfully programmed.
- Test that four of the LEDs count up and downwards.

6. Storing the FPGA configuration (.sof) file into DE1-SoC Flash Memory

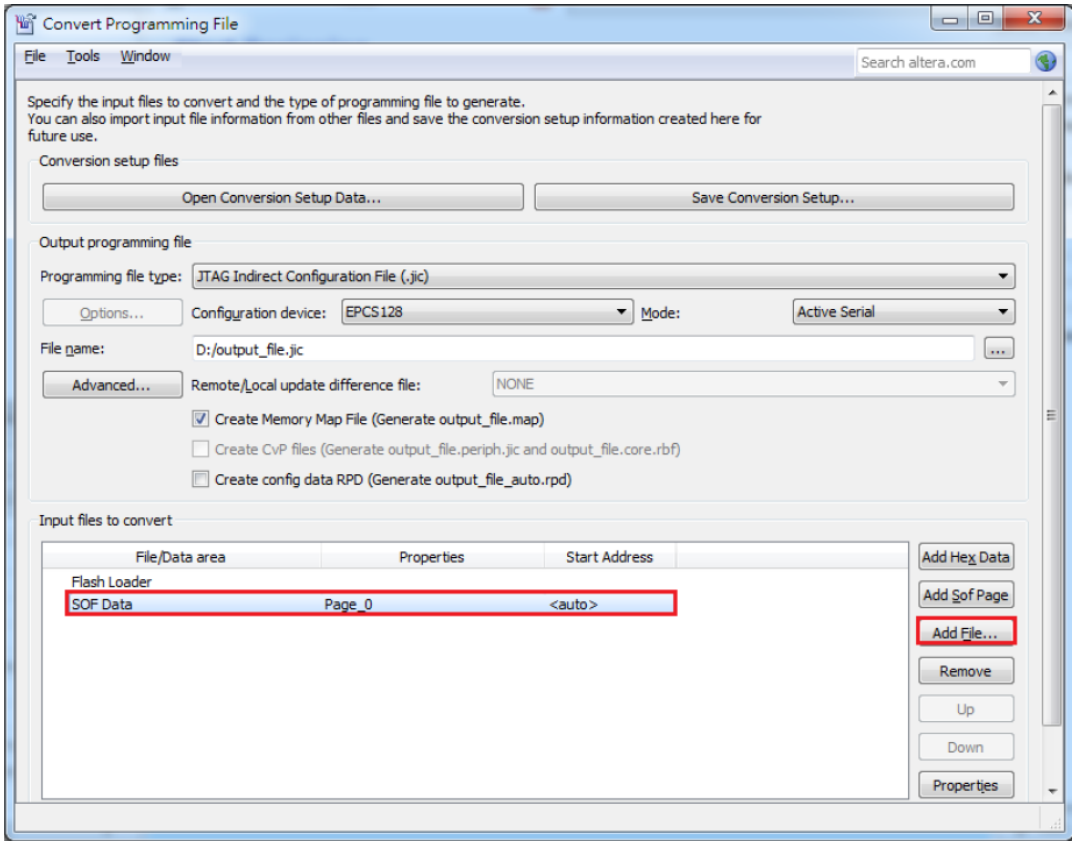
When programing the FPGA with the previously generated .sof file the bitstream will be downloaded to the DE-SoC SRAM memory. This implies that once the board is turned off, it will lose the bitstream.

The DE1-SoC board comes with Flash memory where the .sof file can be stored indefinitely. For this you need to convert the .sof file into a .jif file as follows:

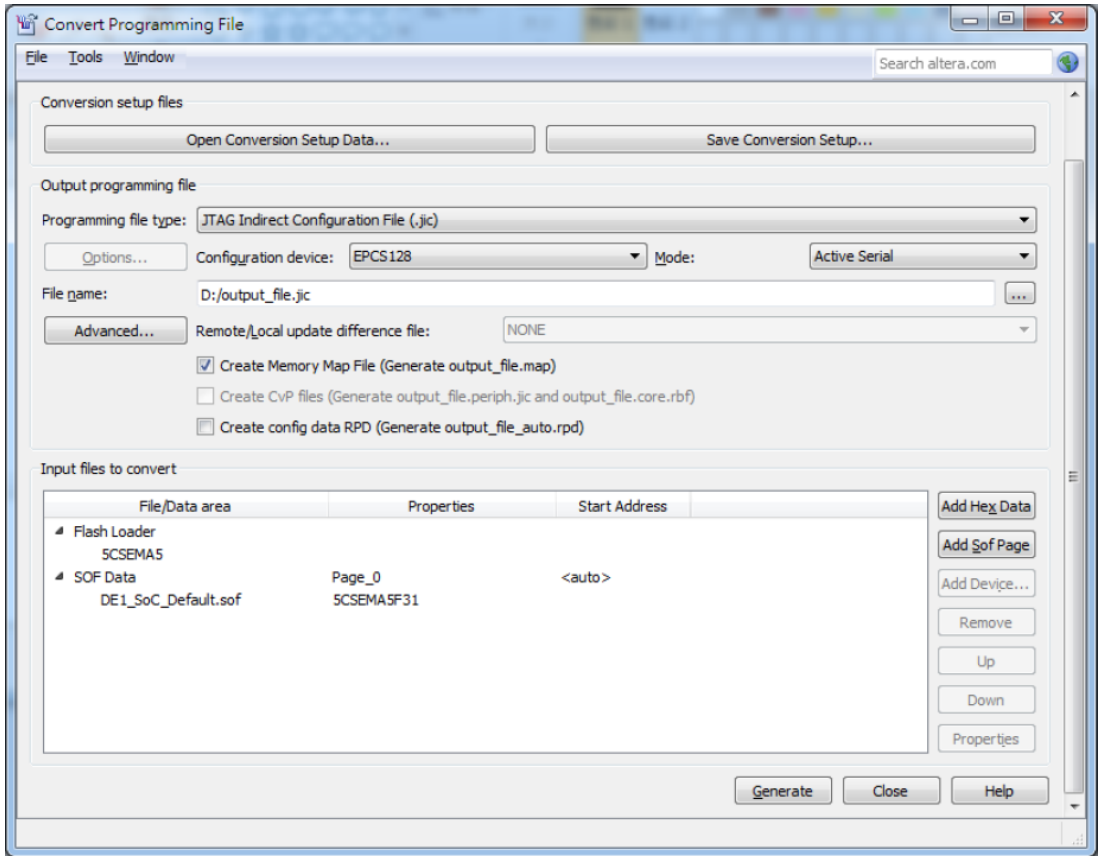
- Choose Convert Programming Files from the File menu in Quartus Prime as shown in the figure



- Select **JTAG Indirect Configuration File (.jic)** from the **Programming file type** field in the dialog of Convert Programming Files.
- Choose **EPCS128** from the **Configuration device** field.
- Choose **Active Serial** from the **Mode** filed.
- Browse to the target directory from the **File name** field and specify the name of output file.
- Click on the **SOF data** in the section of **Input files to convert**, as shown below



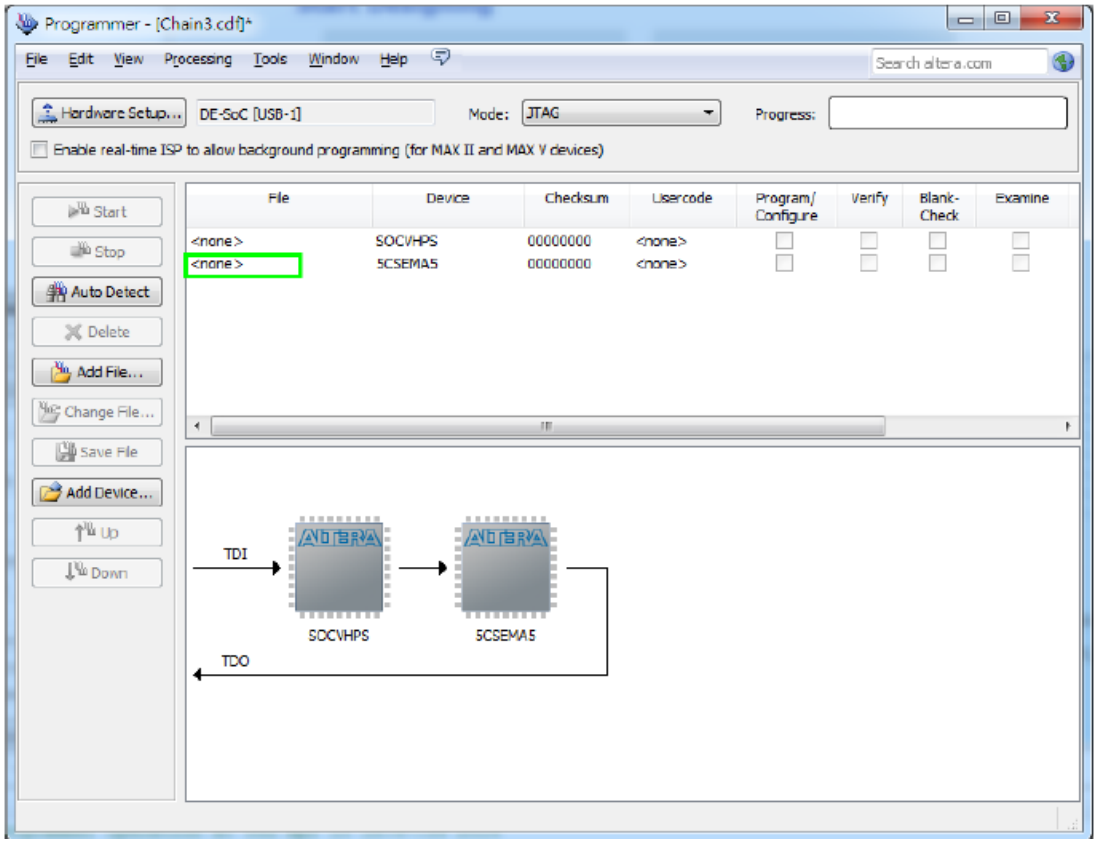
- Click **Add File**.
- Select the .sof to be converted to a .jic file from the Open File dialog.
- Click **Open**.
- Click on the **Flash Loader** and click **Add Device**
- Click **OK** and the **Select Devices** page will appear
- Select the targeted FPGA to be programed into the EPCS



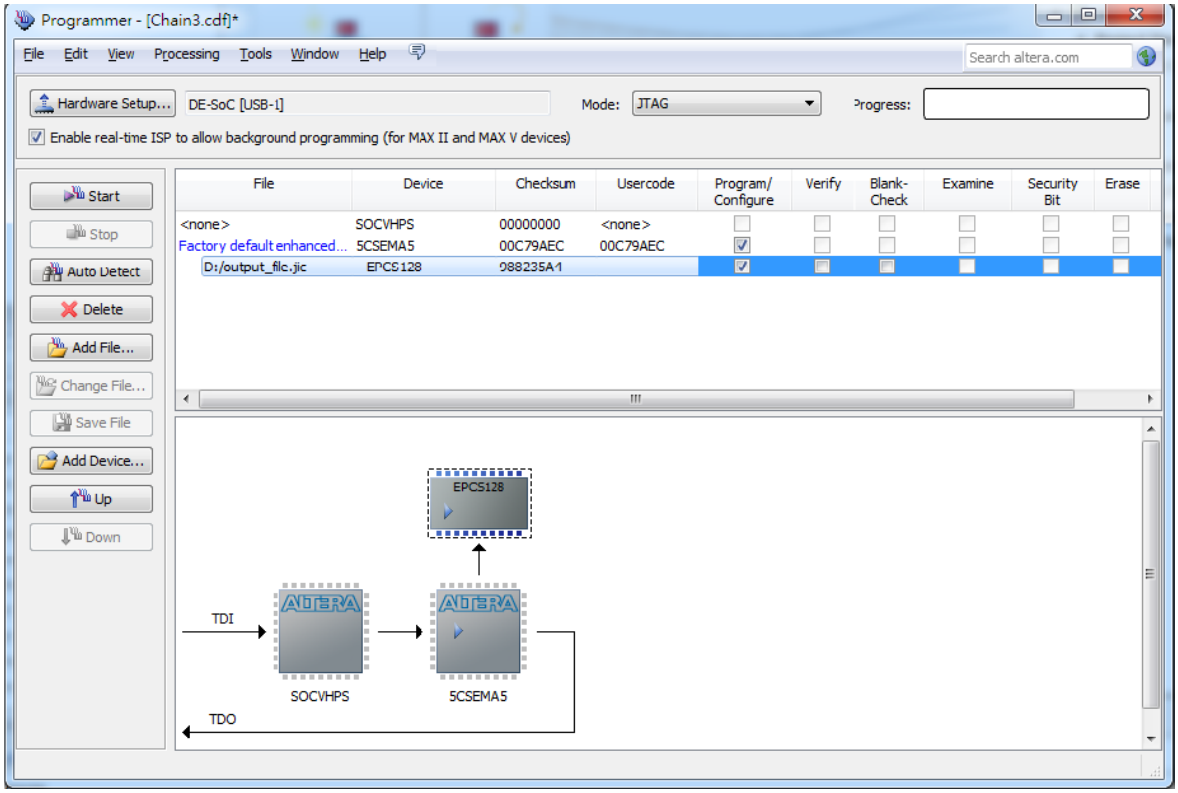
- Click **OK** and the **Convert Programming Files** page will appear
- Click **Generate**.

To program the Flash memory follow these steps:

- Set MSEL[4..0] = “10010”
- Choose **Programmer** from the Tools menu and the **Chain.cdf** window will appear
- Click **Auto Detect** and then select the correct device. Both FPGA device and HPS should be detected
- Double click the green rectangle region shown in the figure shown and the **Select New**



- **Programming File** page will appear. Select the .jic file to be programmed.
- 5. Program the EPCS device by clicking the corresponding **Program/Configure** box. A factory default SFL image will be loaded
- Click **Start** to program the EPCS device.



UT Dallas Honor Code

UT Dallas values academic integrity. Therefore, all students must understand the meaning and consequences of cheating, plagiarism and other academic offences under the Code of Student Conduct and Disciplinary Procedures.

Group assignments must be completed solely by the members of the group. Cross-group work is not allowed. Moreover, similar assignments have been offered before at UT Dallas and other universities. Any use of information from previous assignments is prohibited. The tutorials are to be taken individually. Failure to respect this rule constitutes dishonesty and is a direct violation of the University Honor Code.

[END]