Synthesis and FPGA programming walk-through with Altera Quartus II

The FPGA used in the walkthrough is Altera Cyclone III on the Altera DE0 Development and Education Board (Figure 1).

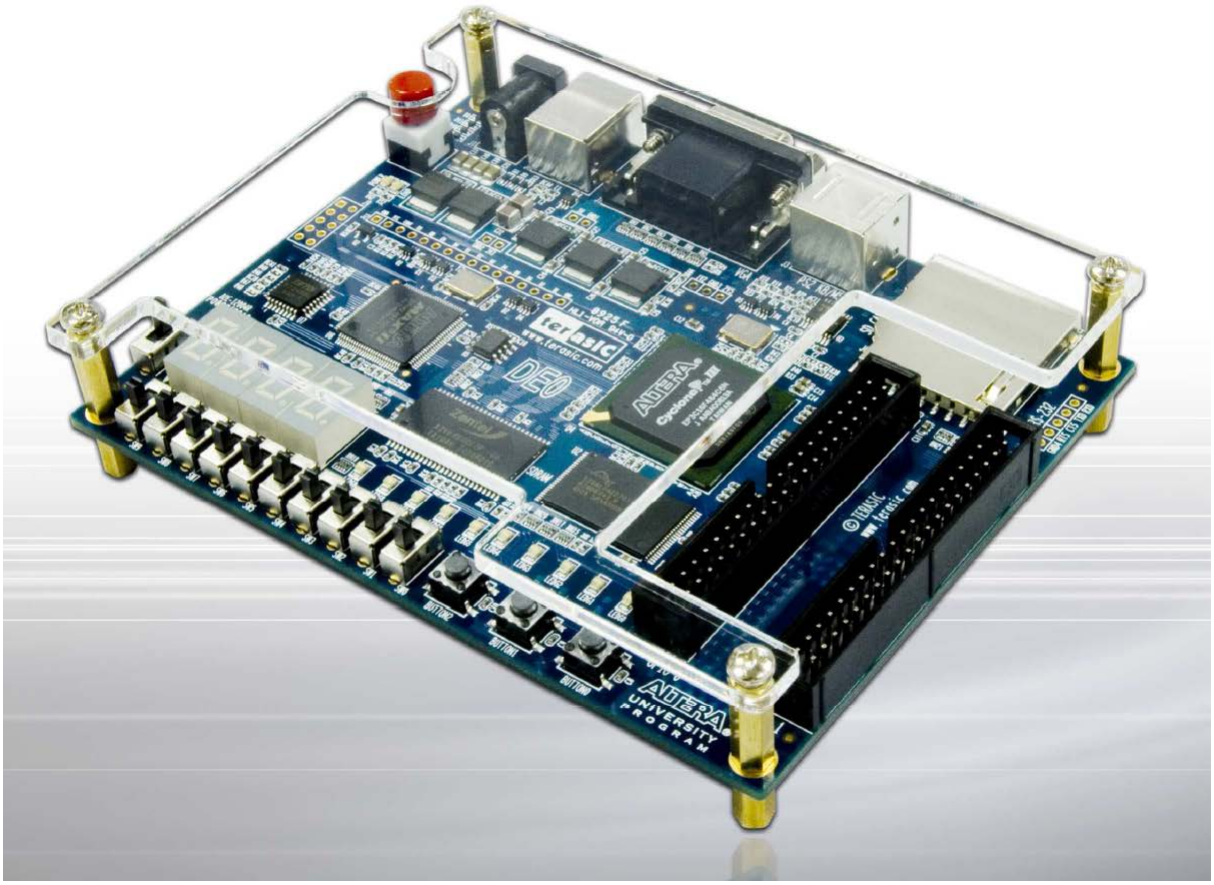

Figure 1. Altera DE0 Development and Education Board with Cyclone III EP3C16 FPGA .

Don’t forget that the FPGA is clocked at 50MHz!

1. RTL Sequential Logic Synthesis

Start Altera Quartus II available from the CAD menu. The general operation of this tool is similar to that of Synplify Pro which you have used in previous laboratory experiments. Altera Quartus II is a dedicated application specifically for FPGAs and Programmable Logic Devices manufactured by Altera. It is a comprehensive and user-friendly tool which performs RTL synthesis from high-level descriptions in a number of Hardware Description Languages, including SystemVerilog, and which connects directly to the development board via a USB link for programming.

We will illustrate how to use Quartus using the picoMIPS program counter (PC_. The project will consist of three SystemVerilog files: pc.sv, which describes the program counter, counter.sv which is a counter that we will use to slow down the 50MHz clock provided on the Altera DE0 board and pcenc.sv which is an encapsulating module that will comprise the counter, program counter and will bind the PC ports to the slow clock and the DE0 switches and LEDs. The slow clock, of the order of about 1Hz, allows to observe the changes on the PC output. The code for all the three modules is shown below and can also be downloaded from the laboratory notes pages on the elec6016 notes site.

```
//-----------------------------------------------------
// File Name : pc.sv
// Function : simplified picoMIPS Program Counter
// functions: increment and absolute branches only
// Author: tjk
// Last rev. 28 Feb 14
//-----------------------------------------------------
module pc #(parameter Psize = 3) // up to 8 instructions
(input logic clk, reset, PCincr,PCabsbranch,
 input logic [Psize-1:0] Branchaddr,
 output logic [Psize-1 : 0]PCout
);

//------------- code starts here---------
always_ff @ ( posedge clk or posedge reset) // async reset
  if (reset) // sync reset
     PCout <= {Psize{1'b0}};
  else if (PCincr) // increment
     PCout <= PCout + 1'b1;
  else if (PCabsbranch) // absolute branch
     PCout <= Branchaddr;

endmodule // module pc



// ------------------------------------------------------------
// counter for slow clock
module counter #(parameter n = 24) //clock divides by 2^n,
                                   //  adjust n if necessary
  (input logic fastclk, output logic clk);

logic [n-1:0] count;

always_ff @(posedge fastclk)
    count <= count + 1'b1;

assign clk = count[n-1]; // slow clock

endmodule
```

```
//-----------------------------------
// File Name   : pcenc.sv
// Function    : picoMIPS Program Counter encapsulating module
//               for elec6016 fpga lab
// Author      : tjk
// Last revised: 28Feb14
//-----------------------------------
module pcenc (
  input logic fastclk,  // 50MHz clock
  input logic [2:0] push_button, // active low push buttons
  input logic[9:0] SW, //  switches
  output logic[9:0] LED // LEDs
);

parameter Psize = 4; // 4-bit program counter –
                     // up to 16 instructions

// declarations of local signals that connect the modules
logic PCincr,PCabsbranch;
logic [Psize-1:0] Branchaddr;
logic [Psize-1 : 0]PCout;
// slow clock from counter for the CPU
logic clk;
// active high reset
logic reset;

//------------- code starts here ---------
// module instantiations
counter #(.n(24)) c1 (
          .fastclk(fastclk),.clk(clk)); // counter for slow clk

pc  #(.Psize(Psize)) progCounter (.clk(clk), // slow clk
        .reset(reset), // active high reset
        .PCincr(PCincr),
        .PCabsbranch(PCabsbranch),
        .Branchaddr(Branchaddr),
        .PCout(PCout) );

// invert active low reset (push butto 2)
assign reset = ~push_button[2];

// connect program counter control signals and branch address to
switches
assign PCincr = SW[9]; // if SW9==1, PC increments
assign PCabsbranch = ~SW[9]; // if SW9 == 0, PC loads branch address
assign Branchaddr = SW[Psize-1:0]; // Switches to select
                                   // the branch address

// connect PCout to LEDs
assign LED[Psize-1:0] = PCout;

endmodule
```

Save the project files in a directory you have a write access to, e.g. on your home drive. When you start Quartus, the startup screen shown in Fig. 2 appears.

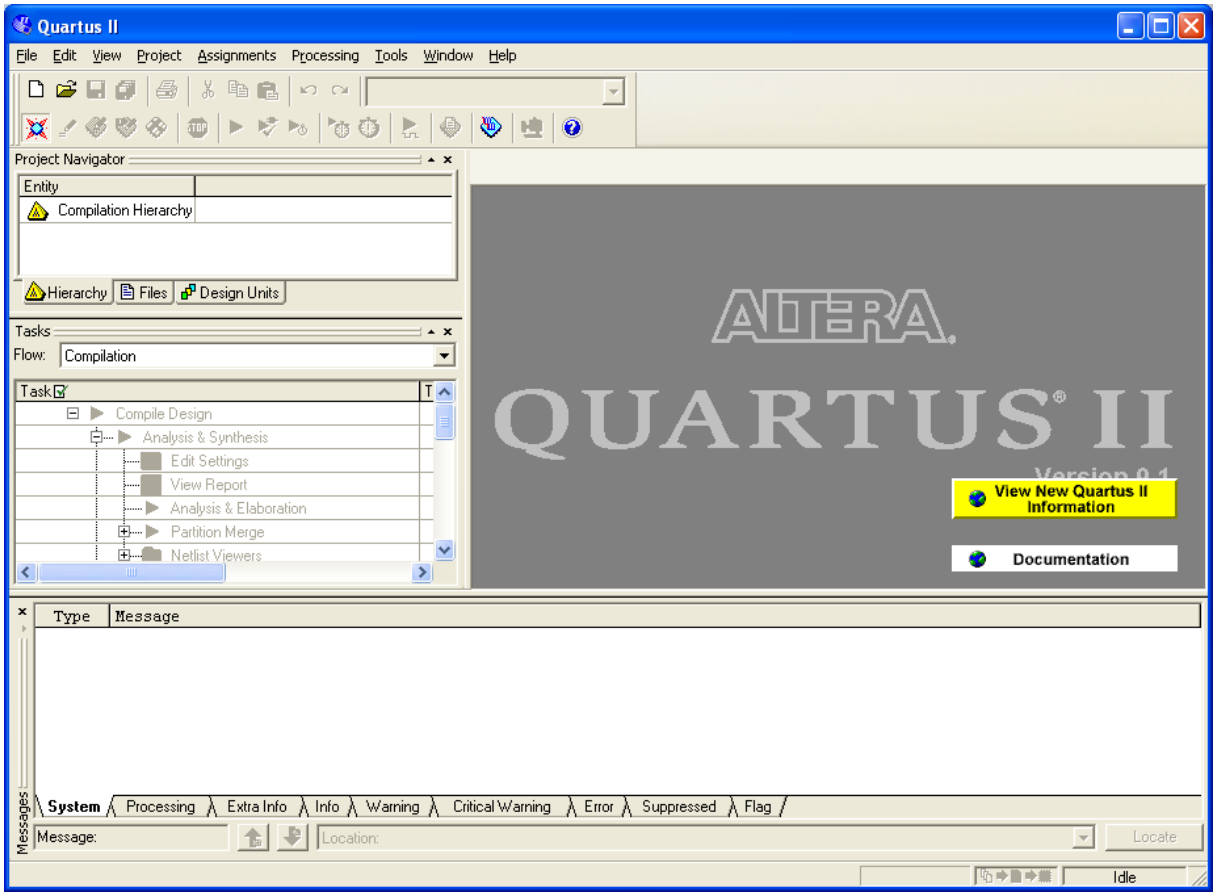

Figure 2. Altera Quartus II startup screen.

Click on File->New Project Wizard. Click Next. Select a working directory for the project (make sure you have a write permission for your working directory), type a project name and specify the name of the top-level design module, ie. **pcenc** ((Figure 3).

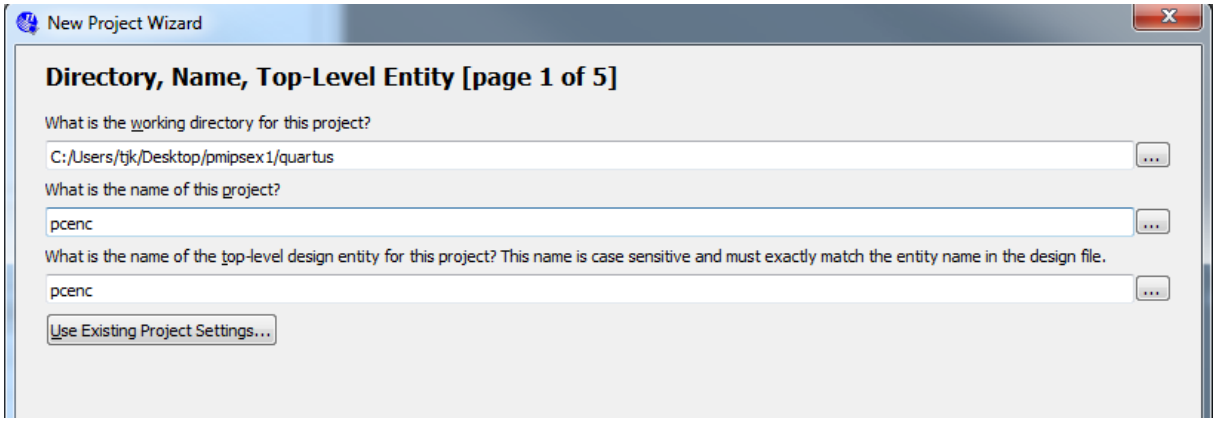

Figure 3. Quartus New Project Wizard.

Click Next, navigate to the directory with your lift controller SystemVerilog module (lift_controller.sv) and add it to the project. Click Next to bring up the Device Settings Window (Figure 4).

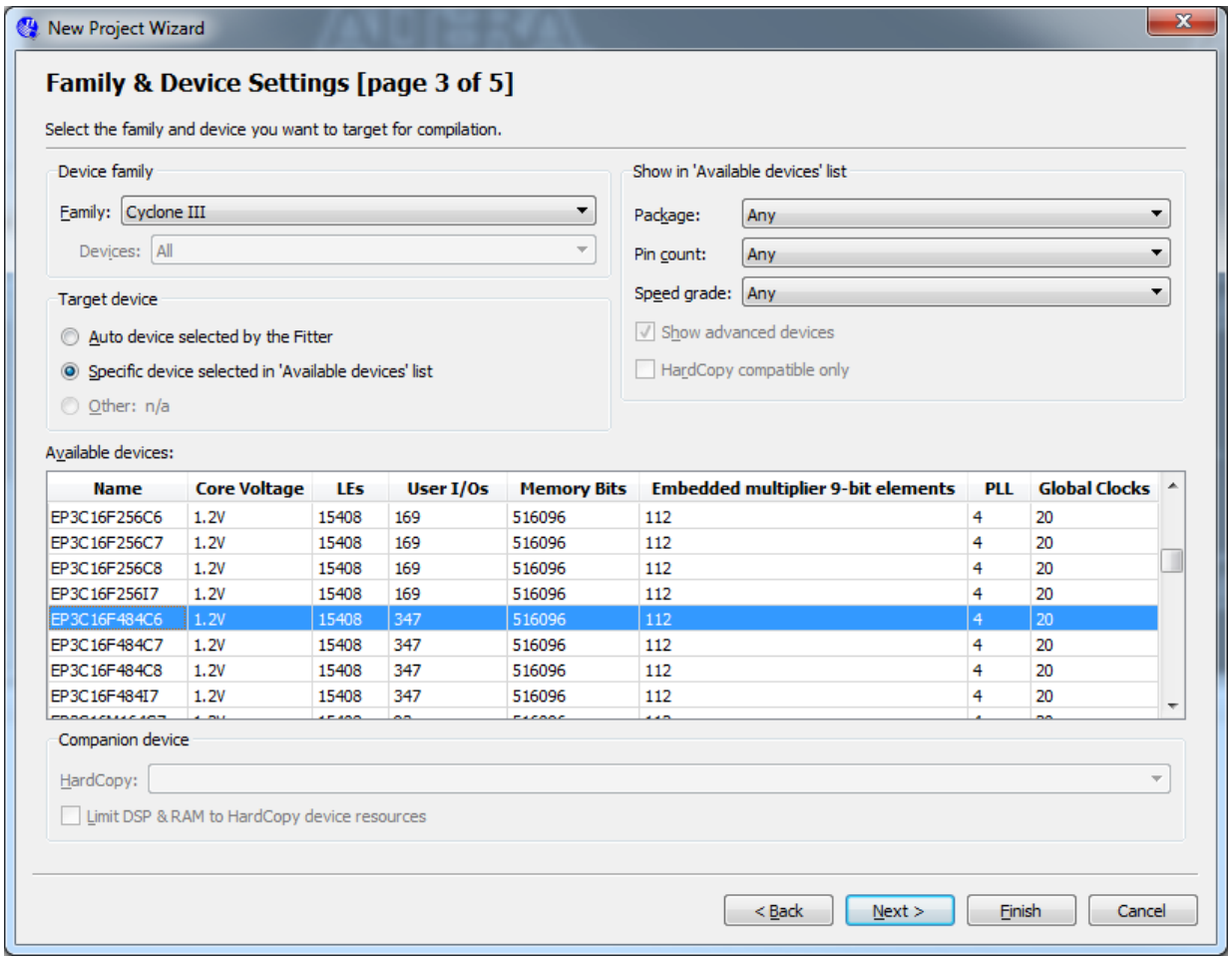

Figure 4. Device Settings window.

Select Cyclone III as the device family ant the device EP3C16F484C6 Click Next->Next->Finish. The main Quartus window now shows the project details in the Project Navigator. Now click the Analysis & Synthesis button (Figure 5).

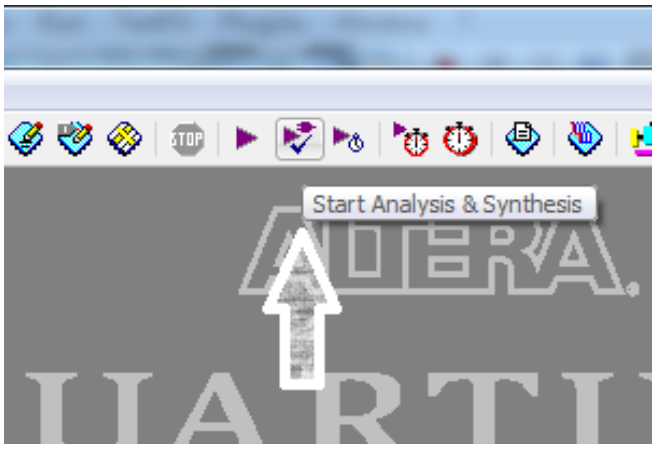

Figure 5. Start Analysis and Synthesis button.

This is a fast run as it does not involve full placement and routing for the FPGA and is intended to reveal any syntax errors or other problems with the SystemVerilog code. If successful, the analysis and synthesis process creates a number of files, including the pin allocation file pcenc.qsf. You will need to edit this file to provide the pin assignments for the

switches, a push button and LEDs according the Altera DE0 board documentation. The required pin definitions are provided in Figure 6 and also in the file pc-pins.qsf downloadable from the elec6016 notes web site.

```
set_location_assignment PIN_G21 -to fastclk
set_location_assignment PIN_F1 -to push_button[2]
set_location_assignment PIN_G3 -to push_button[1]
set_location_assignment PIN_H2 -to push_button[0]
set_location_assignment PIN_D2 -to SW[9]
set_location_assignment PIN_E4 -to SW[8]
set_location_assignment PIN_E3 -to SW[7]
set_location_assignment PIN_H7 -to SW[6]
set_location_assignment PIN_J7 -to SW[5]
set_location_assignment PIN_G5 -to SW[4]
set_location_assignment PIN_G4 -to SW[3]
set_location_assignment PIN_H6 -to SW[2]
set_location_assignment PIN_H5 -to SW[1]
set_location_assignment PIN_J6 -to SW[0]
set_location_assignment PIN_B1 -to LED[9]
set_location_assignment PIN_B2 -to LED[8]
set_location_assignment PIN_C2 -to LED[7]
set_location_assignment PIN_C1 -to LED[6]
set_location_assignment PIN_E1 -to LED[5]
set_location_assignment PIN_F2 -to LED[4]
set_location_assignment PIN_H1 -to LED[3]
set_location_assignment PIN_J3 -to LED[2]
set_location_assignment PIN_J2 -to LED[1]
set_location_assignment PIN_J1 -to LED[0]
```

Figure 6. File pc-pins.qsf.

You can either copy the text in Figure 6 and paste it directly into the file pcenc.qsf which has just been created by Quartus, or download the file pc-pins.qsf from the notes web site, save it in your Quartus project directory and type the following line at the end of the Quartus pcenc.qsf file:

```
Source pc-pins.qsf
```

Now run the full synthesis as well as placement and routing by clicking the Start Compilation button in Quartus (Figure 7):

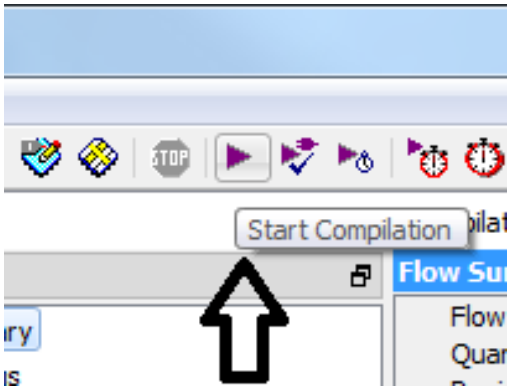

Figure 7. Start Compilation button.

1.1. FPGA programming

On successful compilation, plug in the Altera DE0 board to a USB socket on your machine and click the Program Device (Open Programmer) task in the Compilation pane (Figure 8).

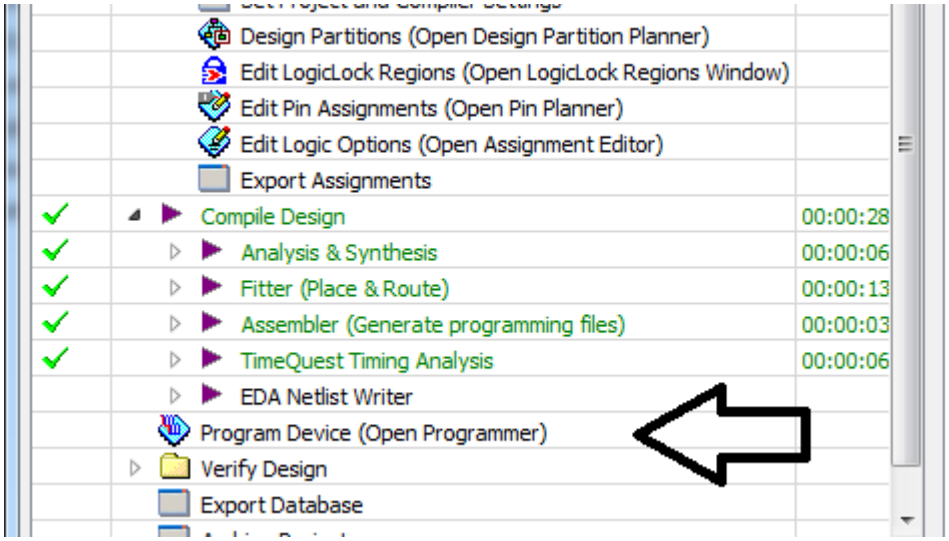

Figure 8. Program Device task.

The programmer window pops up as shown in Figure 9. Make sure that the FPGA placement and route file lift_controller.sof exists and click Start to program the FPGA.

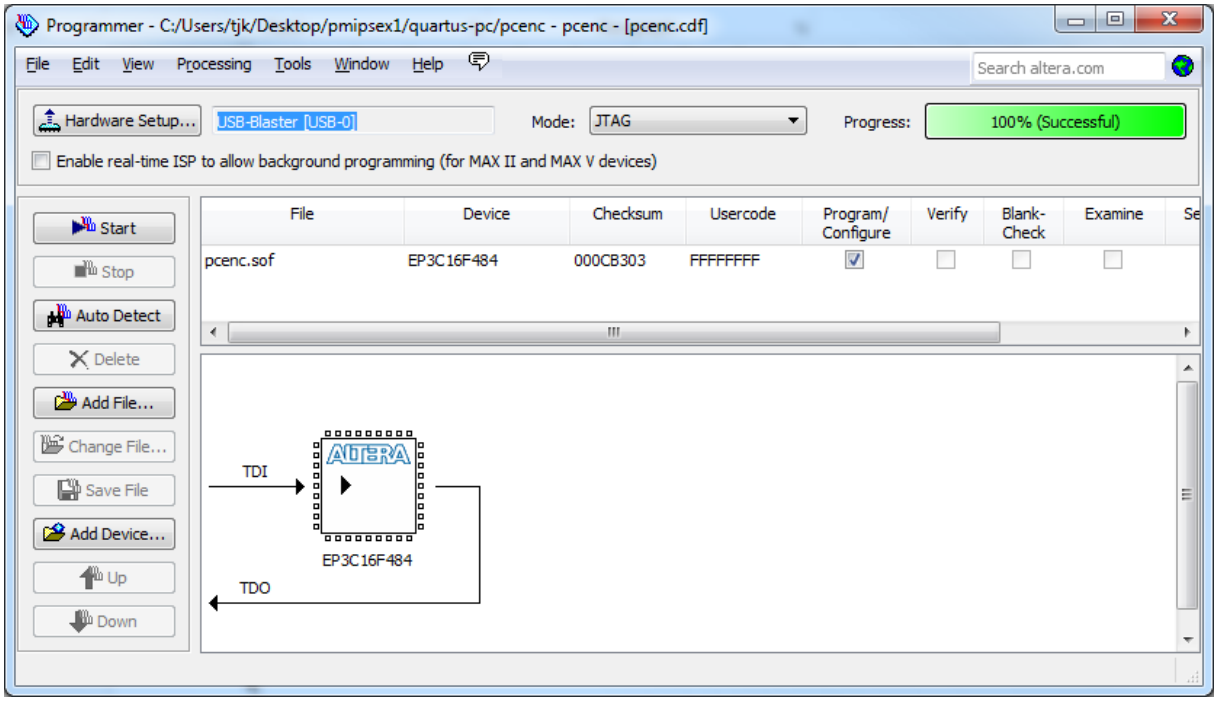

Figure 9. Quartus programmer.

Play with program counter by selecting the switches and pressing the push button 2 to reset as appropriate (see Figure 10). Observe the program counter output on the LEDs to make sure that the PC behaviour matches that in the SystemVerilog description operation of the controller.

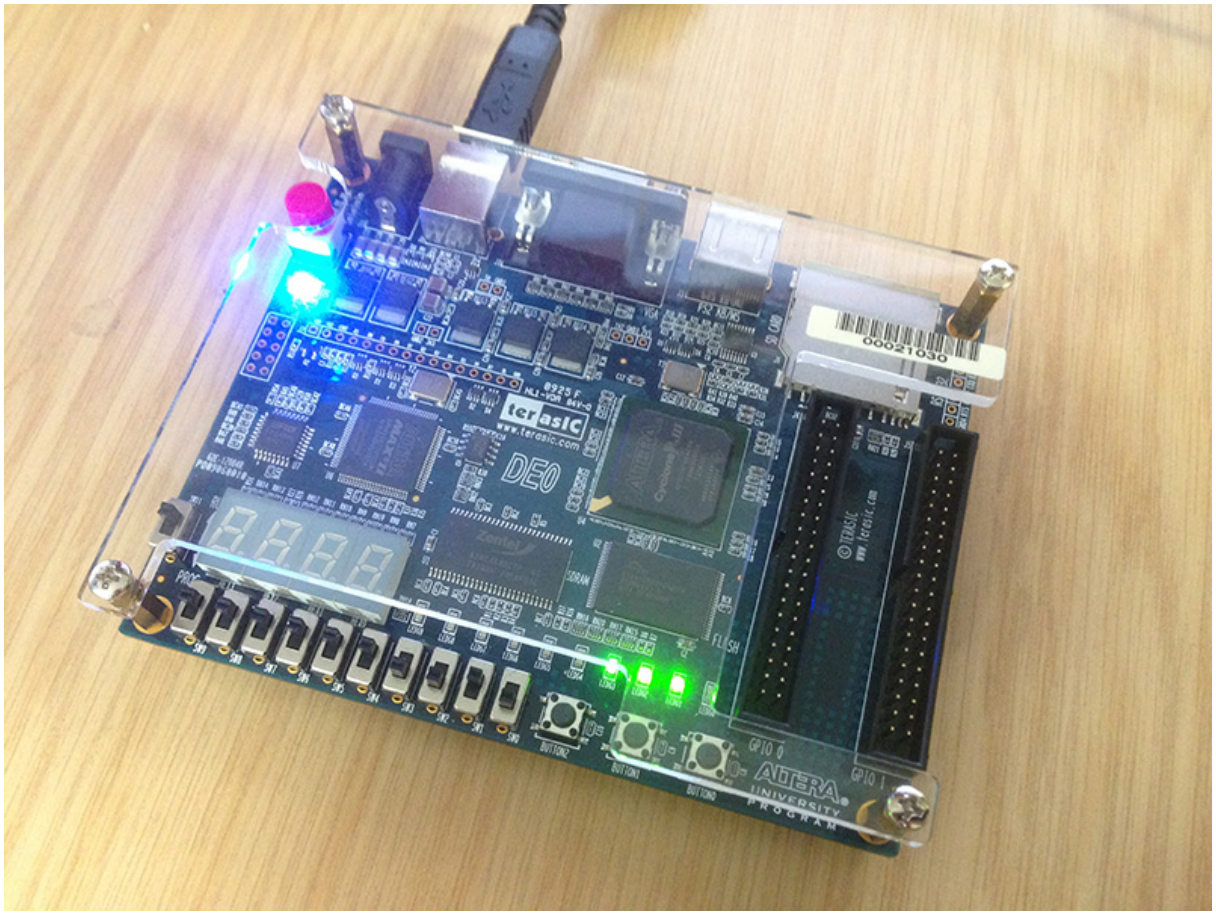

Figure 10. Altera DE0 programmed with the picoMIPS program counter.

2. ALU synthesis example

If you have time, follow the steps above to synthesise the picoMIPS ALU. As the ALU has in total 19 input lines (two 8-bit operands and a 3-bit function select input) , there are not enough input pins on the DE0 to drive them. Test the ALU but hardwiring some of the input pins to fixed value, for example, as shown in the encapsulating module code below.

```
//-----------------------------------------------------
// File Name   : aluenc.sv
// Function    : alu encapsulating module for testing on Altera DE0
// Author:  tjk
// Last rev. 28 Feb'14
//-----------------------------------------------------

`include "alucodes.sv"
// inputs and outputs of the encapsulating module
module aluenc (input logic [7:0] SW, output logic[9:0] LED);
  parameter n =8; // data bus width
  logic [n-1:0] a, b; // ALU input operands
  logic [`Fsize-1:0] func; // ALU function code
  logic N,Z,C; // ALU flags
  logic [n-1:0] result; // ALU result

//------------- code starts here ---------
// input a is a fixed number, say 2:
assign a = 8'b0000_0010;
```

```
// input b is driven by the switches
assign b = SW;
// select alu function as RADD
assign func = `RADD;
// the ALU output and flags Z,C drive the LEDs
assign LED[7:0] = result;
assign LED[8] = Z;
assign LED[9] = C;

//
alu #(.n(n)) alu1
(.a(a),.b(b),.func(func),.N(N),.C(C),.Z(Z),.result(result));  //
create alu

endmodule //end of module aluenc
```

Create a Quartus project for the ALU by following the steps above. The source files needed in the synthesis are: alu.sv, alucodes.sv, and aluenc.sv. They can be downloaded from the elec6016 notes site.

tjk 3/4/2014