MachXO2 Pico Evaluation Kit - synthesis and programming walk-through

1. Introduction

This document describes how to synthesise SystemVerilog descriptions of a simple combinational logic system and a simple state machine to MachXO Mini Development Kit. The software tools you will need are:

- **Modelsim** for simulation of your SystemVerilog description,
- **Lattice Diamond** for synthesis into an RTL level structure, placement and routing of the RTL level design into a MachXO2 CPLD and for programming the MachXO2 chip.

2. MachXO2 Pico Evaluation Kit

The MachXO2 Pico Evaluation Board (part number LCMXO2- 1200ZE-P1-EVN), shown in Figure1, features a MachXO2 LCMXO2-1200ZE CPLD device in a 132-ball csBGA package, a 4 Mbit SPI Flash memory, current sensor circuits using Delta-Sigma ADC, an LCD driven with PWM analogue output circuitry, a 32-pin expansion header, 4 capacitive touch sense buttons, a standard USB cable for device programming and an RS-232/USB and JTAG/USB interface. The board is powered from USB or a 2032 coin battery.

Figure 1. Lattice MachXO Mini Development Kit.

In your MachXO2 Pico Evaluation Kit projects you will need the information about the MachXO2 device pin allocation provided in the document Lattice.Schematic.pdf. In your

projects you will be using an LED board with 8 LEDs connected to the expansion header as shown in Figure 2.

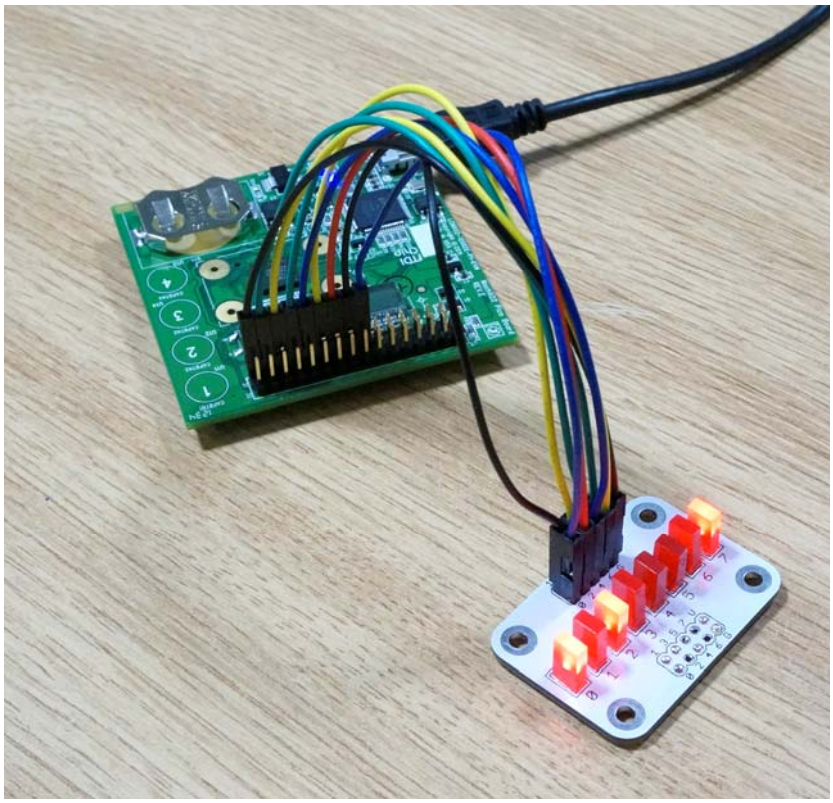


Figure 2. MachXO2 Pico Kit with LED board.

The table below shows how the LED diodes in Figure 2 are connected to the Pico Evaluation Board header pins and how the header pins are mapped to MachXO2 I/O pins:

Table 1. MachXO2 Pico kit header pinout.

LED Board	Pico Kit header pin	MachXO2 CPLD pin name
G (ground)	31	GND
LED 0	15	M7 / PB11A
LED 1	17	N8 / PB11B
LED 2	19	B1 / PL2A
LED 3	21	B2 / PL2B
LED 4	23	C1 / PL2C
LED 5	25	C3 / PL2D
LED 6	27	C2 / PL3A
LED 7	29	D1 / PL3B

The push button on the Pico Evaluation Board is connected to the MachXO2 pin N3 / PB6A.

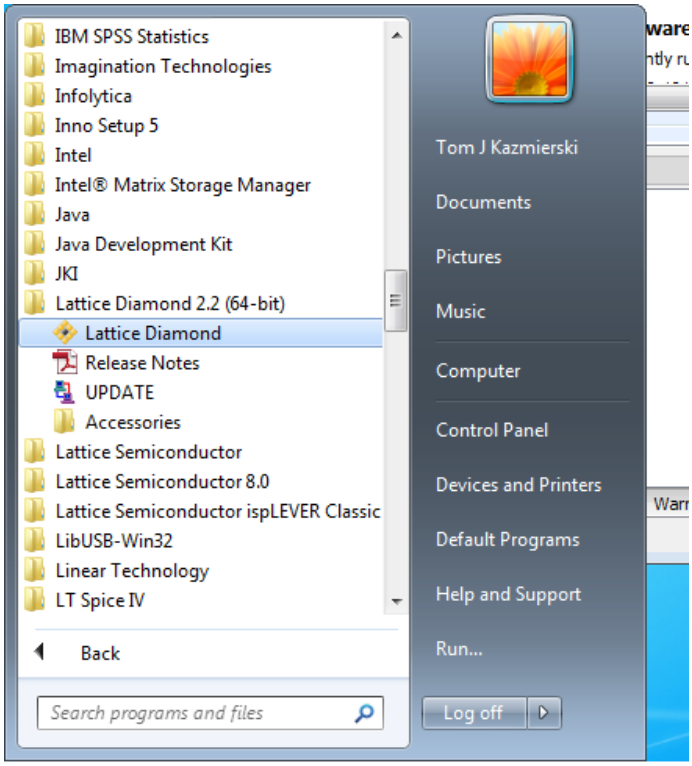
3. Create a Lattice Diamond project

To practice the CPLD design workflow, implement the following simple SystemVerilog module which drives the LEDs with a counter causing the LEDs to blink.

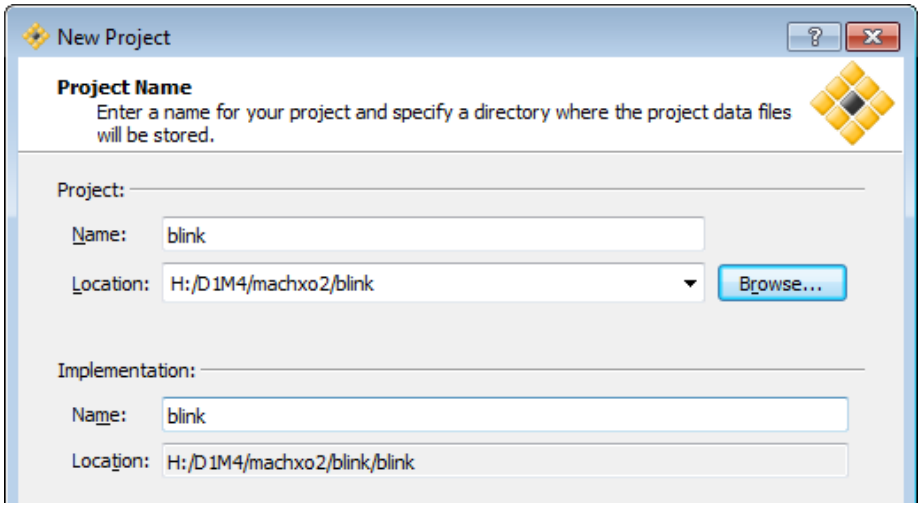
```
////// LED Blinker in SystemVerilog Using On-Chip Oscillator
////////////////////////////
//*********************************************************
// Version 1.0 21 Oct 2013 Tom Kazmierski
// Based on Lattice example blinking_led.v

////  The default on-chip oscillator frequency is 2.08MHz.
//// Supported frequencies (in MHz) include:
////  2.08        4.16         8.31          15.65
////  2.15        4.29         8.58          16.63
////  2.22        4.43         8.87          17.73
////  2.29        4.59         9.17          19.00
////  2.38        4.75         9.50          20.46
////  2.46        4.93         9.85          22.17
////  2.56        5.12        10.23         24.18
////  2.66        5.32        10.64         26.60
////  2.77        5.54        11.08         29.56
////  2.89        5.78        11.57         33.25
////  3.02        6.05        12.09         38.00
////  3.17        6.33        12.67         44.33
////  3.33        6.65        13.30         53.20
////  3.50        7.00        14.00         66.50
////  3.69        7.39        14.78         88.67
////  3.91        7.82        15.65       133.00


module blink( output logic [7:0] LED);
//// ---------------- internal constants --------------
      parameter          N= 28;      // sets counter size
////---------------- internal variables ---------------
      logic [N-1 : 0]   count; // counter
      logic           osc_clk; // clock generated by internal osc
//// ----------------------------------------------------

//// Internal Oscillator, 3.33 MHz is selected
      defparam OSCH_inst.NOM_FREQ = "3.33";
      OSCH OSCH_inst
            (
            .STDBY(1'b0), // 0=Enabled, 1=Disabled
            .OSC(osc_clk),
            .SEDSTDBY()
            );

//// counter
      always_ff @ (posedge osc_clk)
            count <= count + 1;

//// connect the top 8 bits of the counter to the leds
   assign LED = count[N-1:N-8];

endmodule
```

Firstly, download the above code from the ELEC2202 notes website and save it a project directory on your H: drive. The file name is blink.sv.

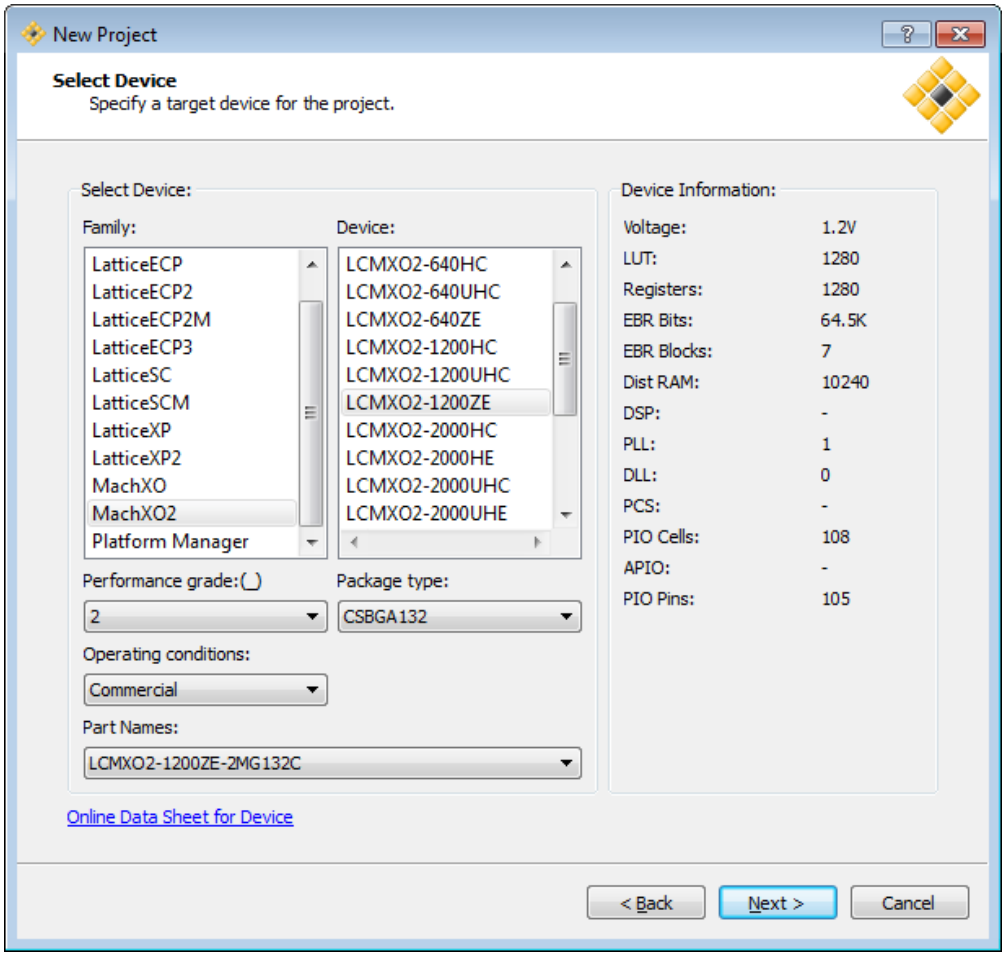
Find Lattice Diamond in the Start Menu of your lab PC and launch it.



From the Diamond main menu select File -> New -> Project. In the New Project window give your project a name and browse to its location as shown below.

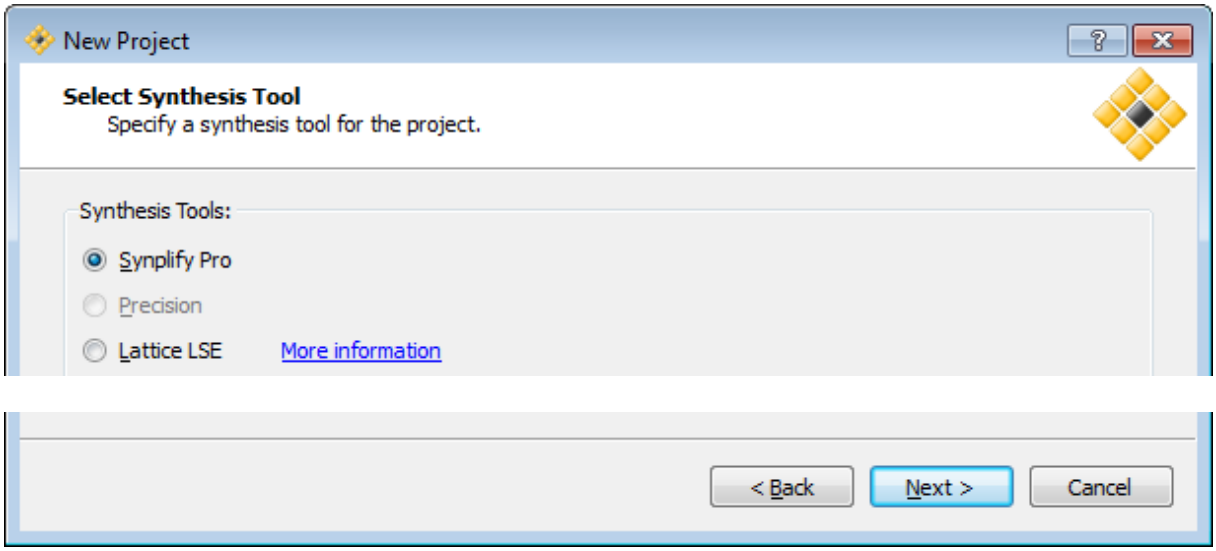


Click Next. The Add Source window will show but do not add a source file at this stage, just click Next to proceed to the Select Device window. See the picture below.



Select the Device as follows: Family: MachXO2, Device: LCMXO2-1200ZE, Part Name: LCMXO2-1200ZE-2MG132C. Click Next.

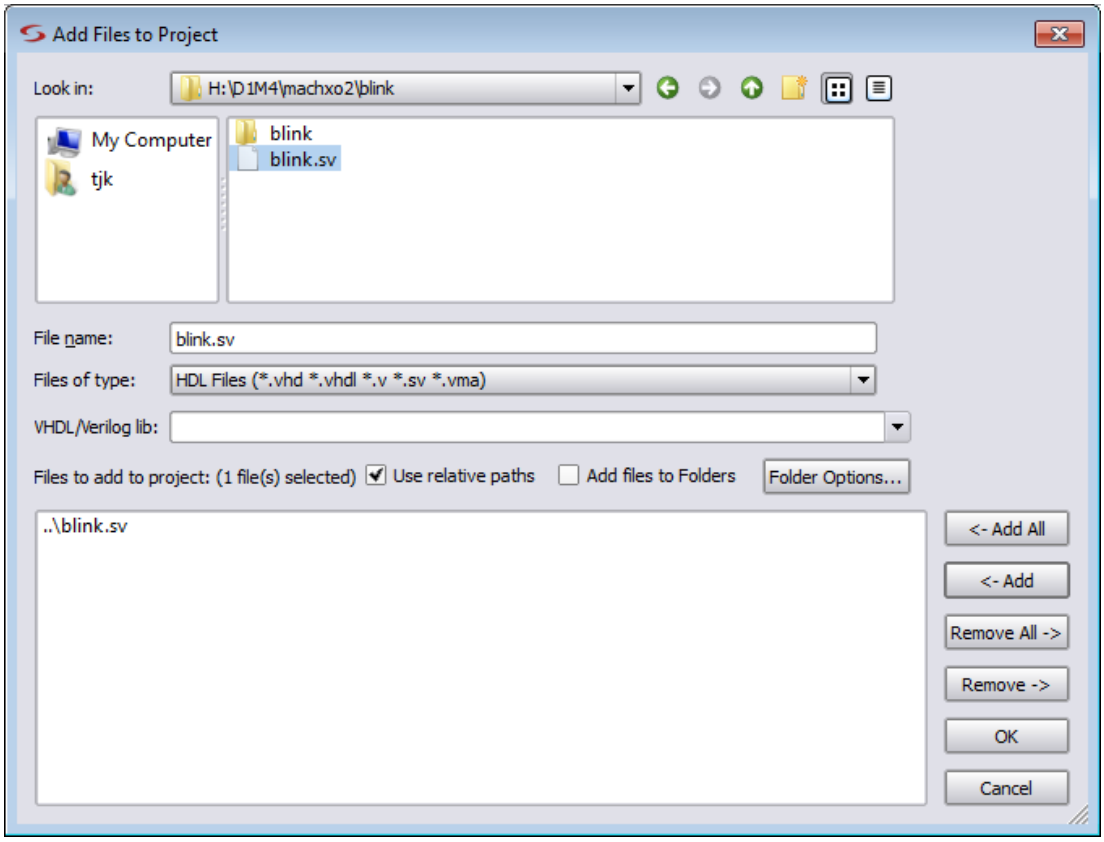
Select Synplify Pro as the synthesis tool. Click Next and then Finish to close the New Project window.



4. Synthesis with SynplifyPro

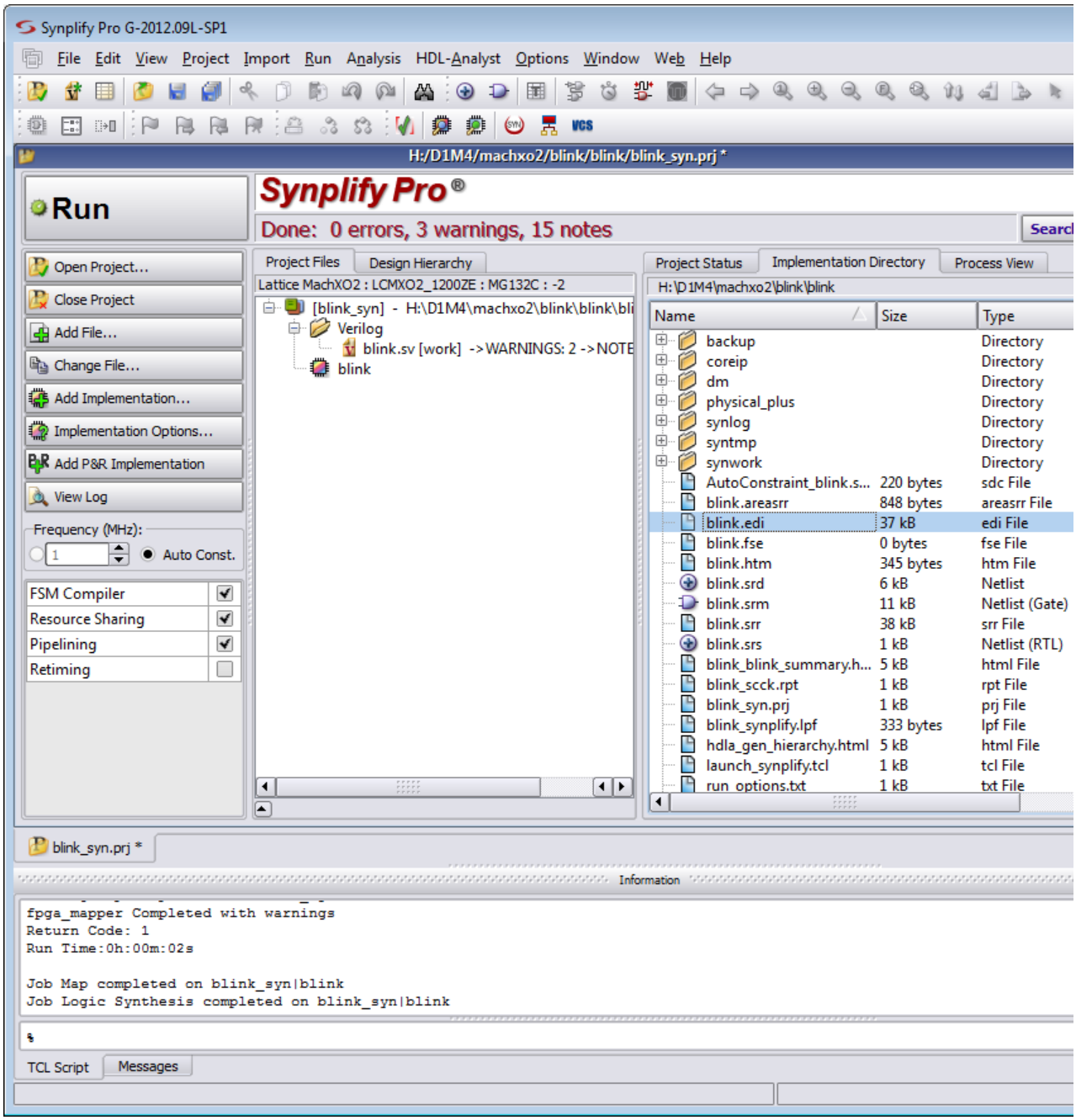
From the Diamond main menu select Tools -> “Synplify Pro for Lattice” to launch Synplify Pro. Go to the Synplify Pro main window. Note that Synplify will already have the correct device selected for you: LCMXO2-1200ZE-2MG132C.

In Synplify click the Add File button. The “Add Files to Project” window pops up as shown below. Navigate to your SystemVerilog source file blink.sv and add it to the Synplify Project.

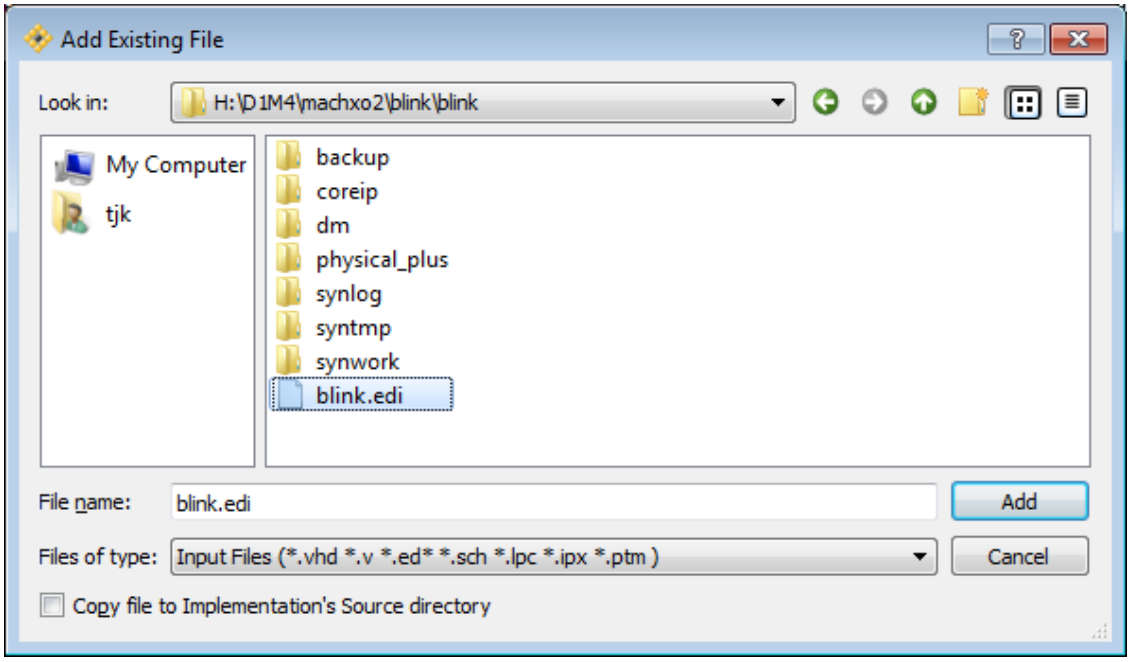


Click OK to close the “Add Files to Project” window

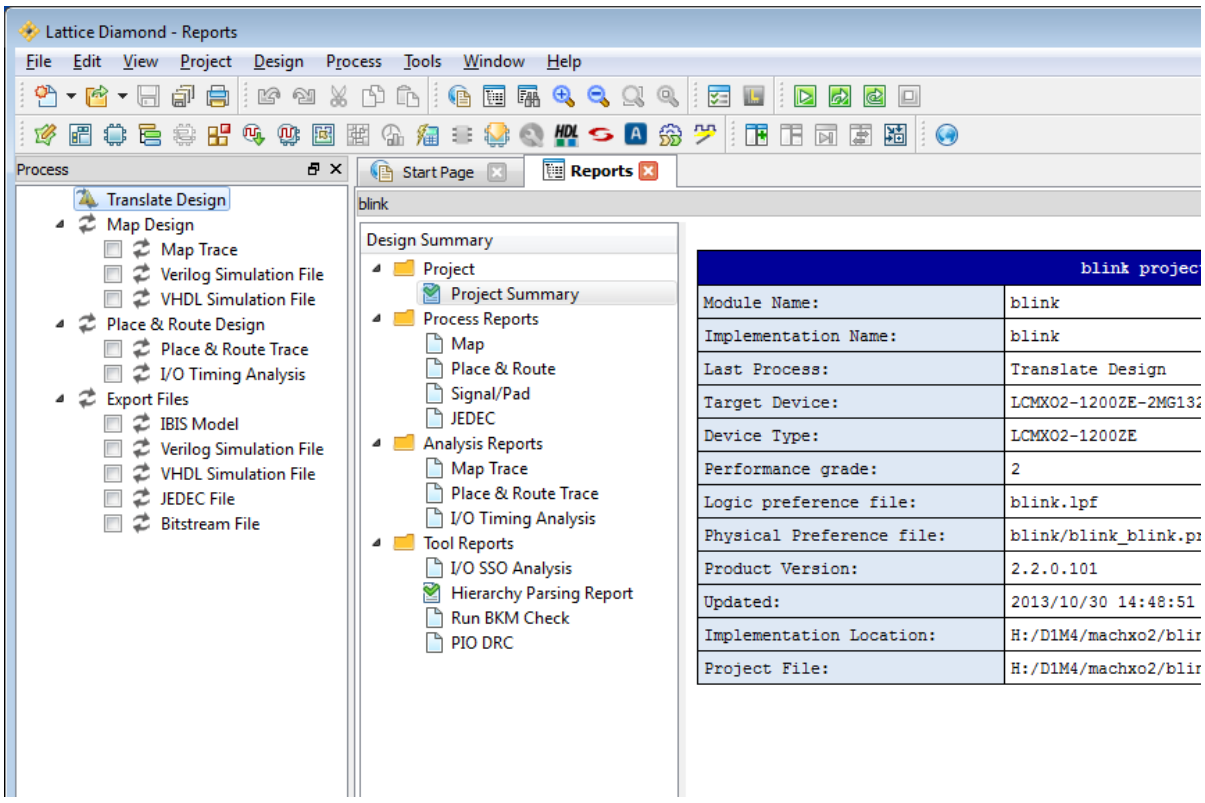
In the main Synplify window click the large Run button to carry out the synthesis. When the synthesis run is complete, make a note of the location of the EDIF file generated by Synplify. It will be named blink.edi or blink.edn as shown in the picture below.



The EDIF file will be used by the Diamond place and route algorithm. Return to the Diamond main window and select from the Diamond main menu File -> Add -> Existing File. Navigate to the EDIF file generated by Synplify and click the Add button as show in the picture below.



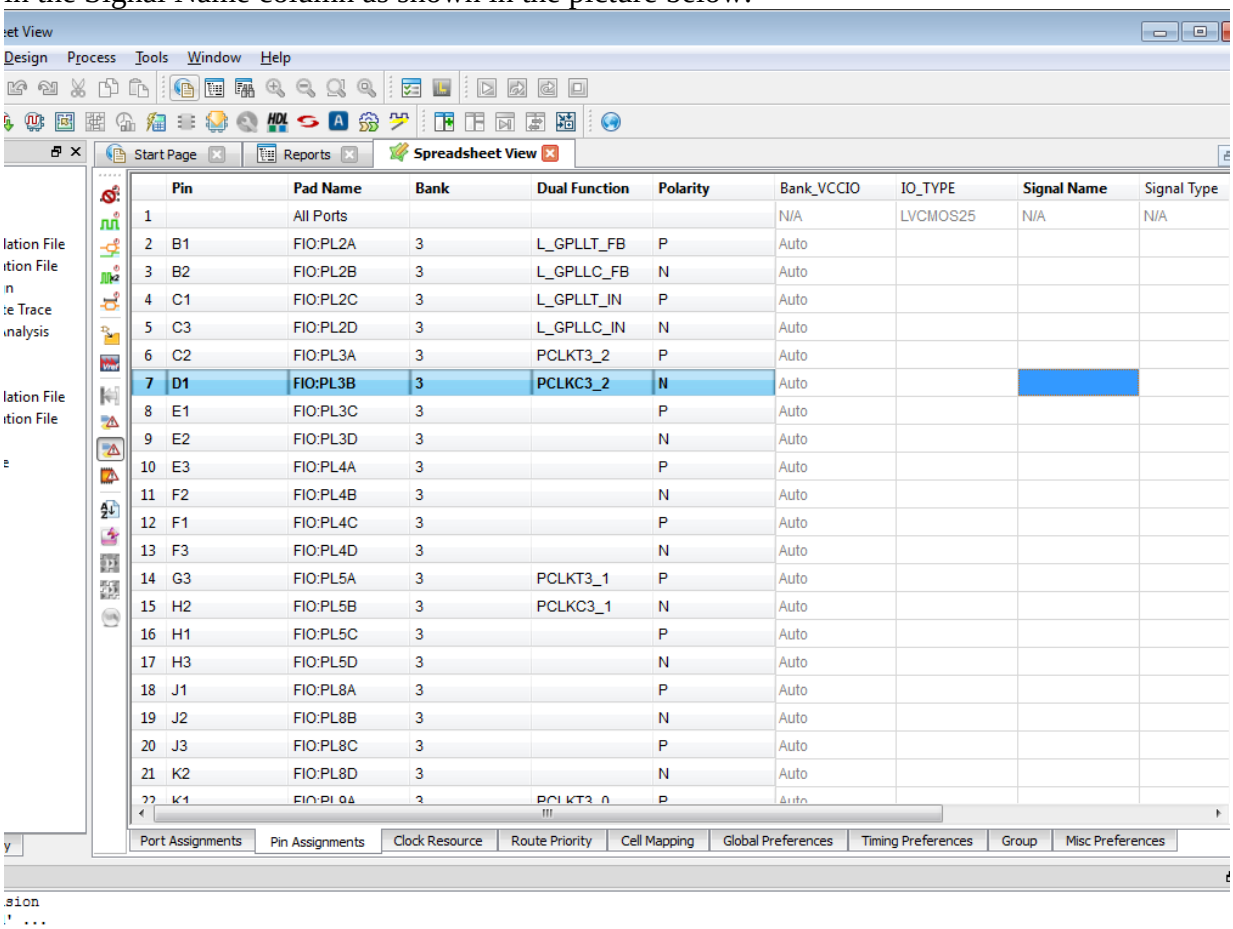
Having added the EDIF file to your Diamond project, double click on “Translate Design” in the Process pane of the main Diamond window as shown below.



5. I/O pin assignments

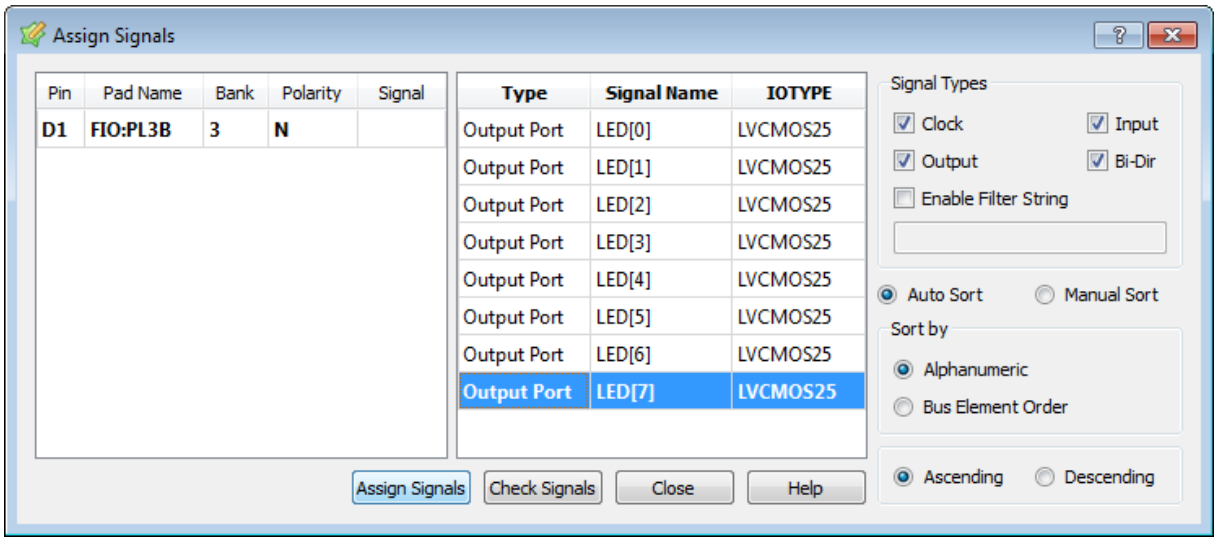
Then select from the Diamond main menu Tools->Spreadsheet View and select the Pin Assignments tab. In this step the I/O signals of your design (in the blink program it is the vector: output logic [7:0] LED) will be connected to MachXO2 I/O pins. To connect a signal

to a pin, find the row of the pin you want to connect, e.g. PL3B, and double click on the cell in the Signal Name column as shown in the picture below.

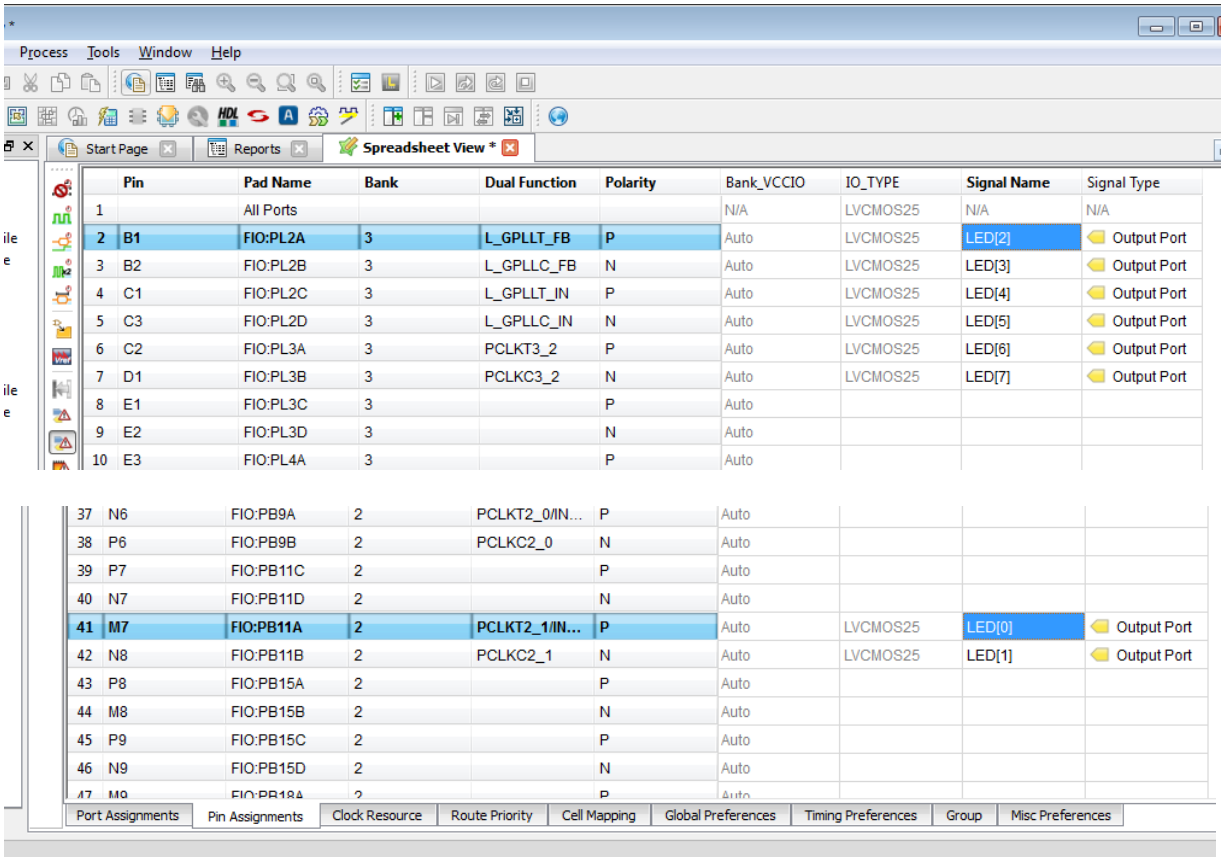


	Pin	Pad Name	Bank	Dual Function	Polarity	Bank_VCCIO	IO_TYPE	Signal Name	Signal Type
1		All Ports				N/A	LVCMOS25	N/A	N/A
2	B1	FIO:PL2A	3	L_GPLLT_FB	P	Auto			
3	B2	FIO:PL2B	3	L_GPLLC_FB	N	Auto			
4	C1	FIO:PL2C	3	L_GPLLT_IN	P	Auto			
5	C3	FIO:PL2D	3	L_GPLLC_IN	N	Auto			
6	C2	FIO:PL3A	3	PCLKT3_2	P	Auto			
7	D1	FIO:PL3B	3	PCLKC3_2	N	Auto			
8	E1	FIO:PL3C	3		P	Auto			
9	E2	FIO:PL3D	3		N	Auto			
10	E3	FIO:PL4A	3		P	Auto			
11	F2	FIO:PL4B	3		N	Auto			
12	F1	FIO:PL4C	3		P	Auto			
13	F3	FIO:PL4D	3		N	Auto			
14	G3	FIO:PL5A	3	PCLKT3_1	P	Auto			
15	H2	FIO:PL5B	3	PCLKC3_1	N	Auto			
16	H1	FIO:PL5C	3		P	Auto			
17	H3	FIO:PL5D	3		N	Auto			
18	J1	FIO:PL8A	3		P	Auto			
19	J2	FIO:PL8B	3		N	Auto			
20	J3	FIO:PL8C	3		P	Auto			
21	K2	FIO:PL8D	3		N	Auto			

The Assign Signals window will pop up (see below). Select the signal you want to assign, e.g. LED[7], and click the Assign Signals button.

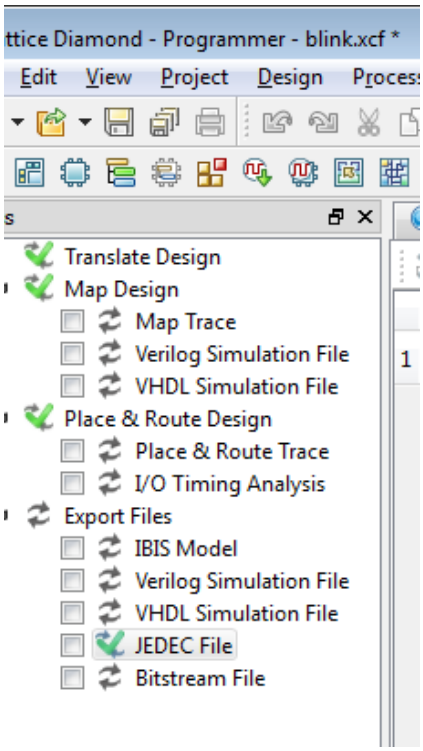


Repeat this process until all the I/O signals of your design are allocated to pins. Use the pin assignments shown in Table 1 above.



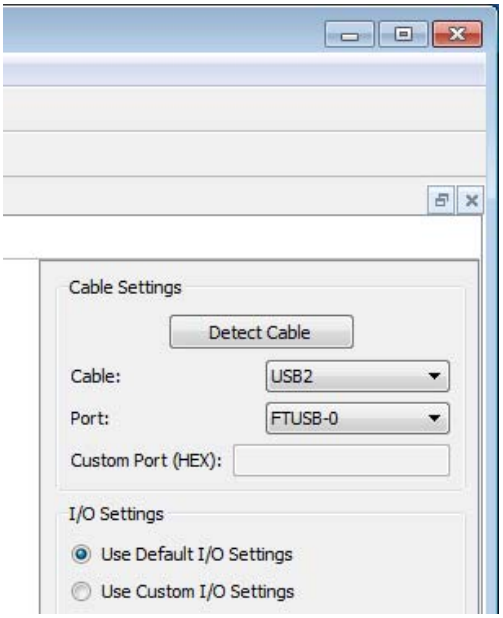
6. Creating a JEDEC fuse map and programming the CPLD

Now create a JEDEC fuse map. To do this, double click on “JEDEC File” in the process pane.

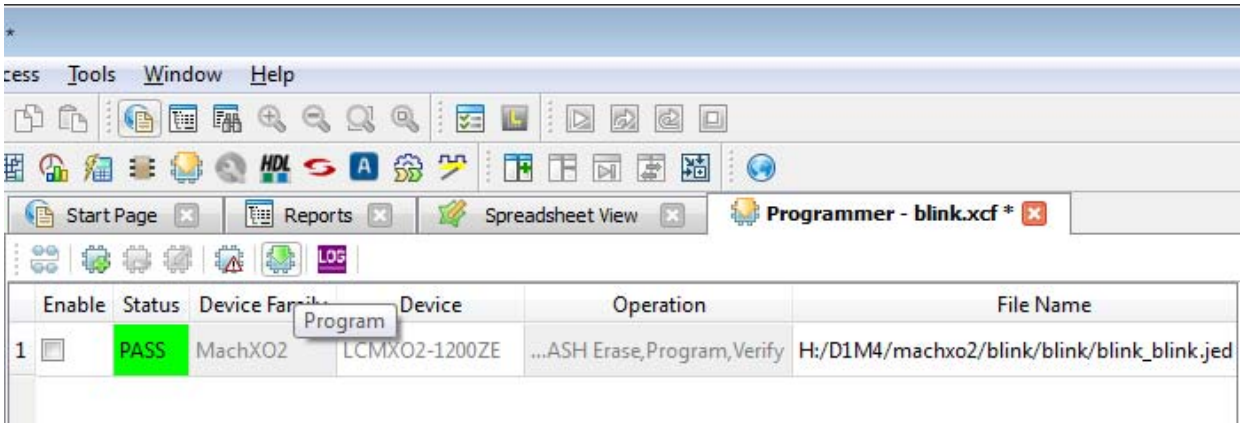


Connect your MachXO2 Pico Kit to a USB port of your lab PC. From the main Diamond manu select Tools->Programmer to launch the Programmer. Check that the Cable and Port

options are as shown in the figure below. Click the “Detect Cable” button and observe the messages in the Message pane at the bottom of the main window. If no cable is detected, select a different Cable and Port setting from the Cable and Port drop-down menus.



Select the Programmer tab and click the Program icon to run the Programmer as shown in the picture below. When the device is programmed successfully, a PASS status is displayed highlighted in green.



7. Notes on implementing the D1 assignment

Now proceed with a more complex, sequential logic example, such as the multiplier you develop in the design exercise D1. Use the MachXO2 internal oscillator as in the Blink example and a counter to slow down the clock to about 1Hz. Any bouncing of the push button switch should settle within a few milliseconds, i.e. much less than the clock period. Also, the slow clock will allow you to observe the outputs of your design in real time. A counter example is provided on the ELEC2202 notes website. It assumes that the selected frequency of the internal MachXO2 clock is 3.3MHz. Note that a 20-bit counter does not stretch the available CPLD resources much as this particular MachXO2 device (LCMXO2-1200ZE) features 1280 flip-flops.

The code of the main multiplier module below also illustrates how to connect output signals to the active-low LEDs on the Mini Development Kit. This code is also available for

download on the ELEC2202 notes site, Note that in this implementation the multiplier Q and multiplicand M are hardwired in the code, i.e. have fixed logic values. This eliminates the need to use switches as external inputs to the CPLD.

```
/* D1 encapsulating module
   author: tjk
   last revision: 17 Oct' 13
*/

module multiplier(input logic start, output logic[7:0] AQ);

logic osc_clk, clock, C, ready, reset, shift, add;
logic [3:0] M,Qin, Sum, count;

//// Internal Oscillator 3.33MHz
      defparam OSCH_inst.NOM_FREQ = "3.33";
      OSCH OSCH_inst (
            .STDBY(1'b0),  // 0=Enabled, 1=Disabled
            .OSC(osc_clk),
            .SEDSTDBY()
      );

assign M = 4'b0101; // multiplicand - hardwired
assign Qin = 4'b0111; // multiplier - hardwired


counter c(.*); // produces slow clock
adder A(.A(AQ[7:4]), .M(M), .C(C), .Sum(Sum));
register R(.*);
sequencer S(.start(start), .clock(clock), .reset(reset), .Q0(AQ[0]),
 .add(add), .shift(shift), .ready(ready));

endmodule
```

Figure 12. Main multiplier module for MachXO2 implementation.

```
/* counter.sv - produces a slow clock from MachXO2 internal clock
  (e.g 3.33MHz)
  author: tjk
  last revision: 17 Oct' 13
*/

module counter #(parameter N = 20)
(input logic osc_clk, output logic clock);
logic [N-1:0] count; declare an n-bit counter
// N=20: 2^20 * 1/3.33MHz = ~0.3 seconds
// N=18: 2^18 * 1/3.33MHz = ~1.2 seconds

always_ff @(posedge fastclk) begin
 count <= count + 1;
end

// copy top bit to output - slow clock
assign clock = count[N-1];

endmodule
```

Figure 13. Counter used to produce a slow clock .