Department of Electrical Engineering

ASIC/FPGA Chip Design

Laboratory Manual

ASIC/FPGA Chip Design Laboratory Manual

Dr. Mahdi Shabany

Spring 2025

Contents

Department of Electrical Engineering

ASIC/FPGA Chip Design

Lab 2 - FSM & Modular Integration

Lab 2 - FSM & Modular integration

1.1 Objective

In this lab, you will refine your expertise in Mealy and Moore finite state machines (FSM) while mastering the integration of multiple modules within a comprehensive design schematic. You will also gain hands-on experience with Altera DE2, to familiarize yourself with working on various chips and hardware platforms.

1.2 Introduction

You got familiar with the zynq 7010 in the previous lab. The Zynq-7010 is a member of the Xilinx Zynq-7000 SoC (System-on-Chip). It combines a dual-core processor (PS) with a programmable FPGA fabric (PL) in a single chip. It is designed to work seamlessly with Xilinx Vivado. The Zynq-7010 is an industrial grade chip, designed primarily for professional applications in industrial environments. As a result, it is not equipped with an abundance of LEDs or buttons. Consequently, we utilized the additional pins to interface with an extension board, thereby enhancing its peripheral capabilities.

For educational purposes, there are specialized development boards available, such as the Altera DE2. It is an educational development board designed by Terasic, featuring an Altera Cyclone II FPGA chip. It is widely used for learning digital design, hardware development, and prototyping. The board is supported by Intel's Quartus, a robust FPGA design software, which provides an intuitive environment for developing and testing digital systems. The DE2 board provides a rich set of peripherals and interfaces, making it ideal for both academic purposes and entry-level FPGA applications. It provides such following peripherals initially mounted on the board:

- 18 switches(ACTIVE HIGH) and 4 push buttons(ACTIVE LOW)
- 27 LEDs (18 Red, 9 Green) (ACTIVE HIGH)
- 8 7-segments (ACTIVE LOW)
- LCD module
- mouse/keyboard connector
- microphone/speaker aux jacks
- VGA and Serial port (RS-232)

The DE2 boasts a significantly higher number of logic elements compared to the Zynq 7010 FPGA fabric, offering an increase of nearly 2.5 times. However, these advanced capabilities come at a premium, with the DE2 priced approximately five times higher than the Zynq 7010. It is also noteworthy that the comparatively lower cost of the Zynq 7010 is, in part, a result of its high-volume production, as it enjoys widespread adoption across various industries.

You can take a closer look at the altera DE2-70 in the following figure:

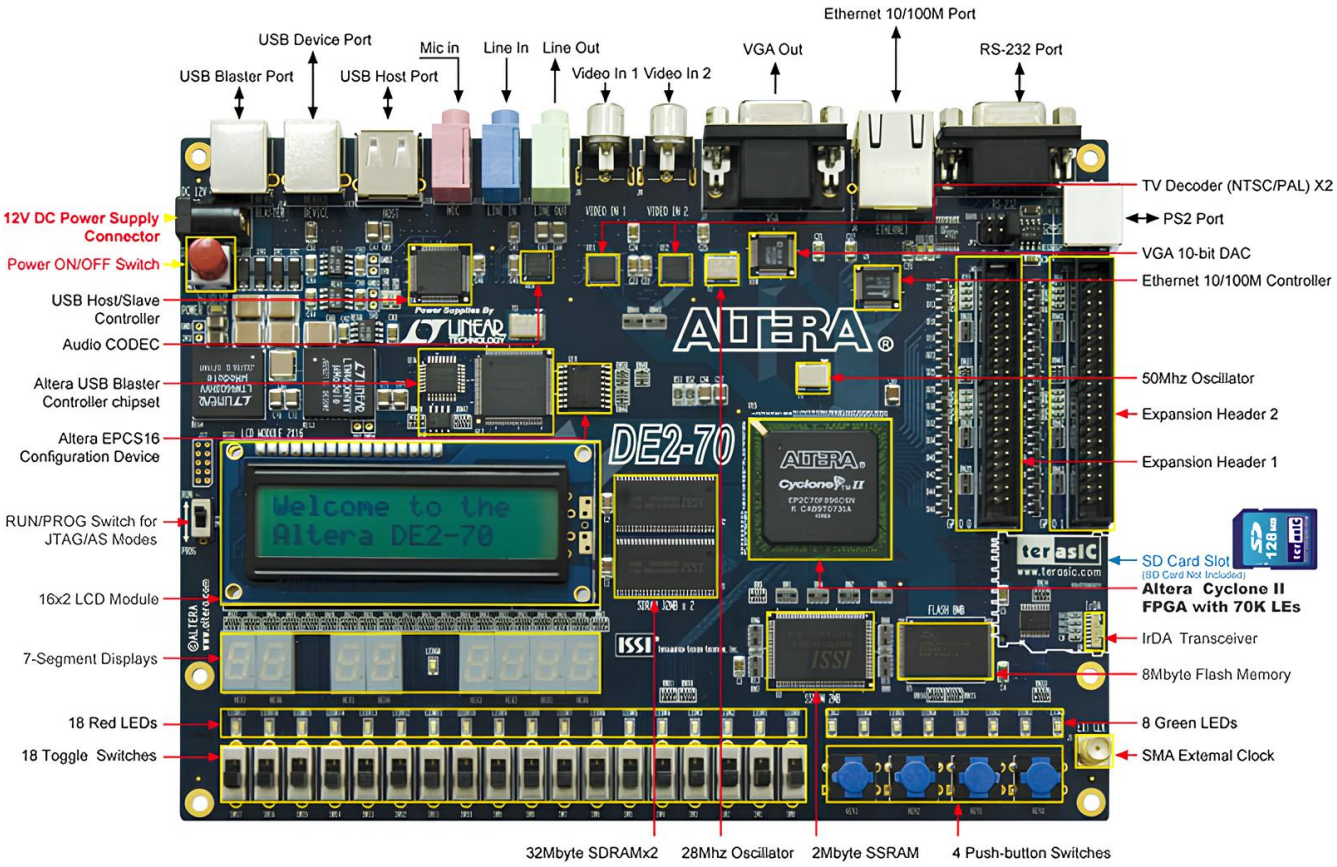


Figure 1.1: Altera DE2-70 developement board

Although it appears that the DE2 surpasses the 7010 in nearly every aspect, albeit at a higher price point, it is important to note that the DE2 lacks multiplication and DSP units that provide the Zynq with a significant advantage in applications like signal processing .

Ultimately, selecting the right FPGA for a project is a careful consideration of the specific requirements and goals of the intended application. The unique demands of the task at hand should guide the decision-making process, ensuring the chosen FPGA aligns seamlessly with the project's objectives.

In alignment with the objectives outlined, we will be engaging with a new FPGA chipset. The Quartus II (Web edition) serves our needs for the Altera DE2 boards. Make sure to have it downloaded and installed.

It is highly recommended to review Appendix A, as it provides comprehensive guidance on properly installing Quartus and adhering to the FPGA design flow.

1.3 Instructions

You have been hired as FPGA engineers in a company developing gaming consoles. Your job is to design and implement FPGA-based solutions for the critical components of the console. Throughout these instructions, you will tackle various tasks on both zynq 7010 and altera DE2

1.3.1 Controller receiver

You are provided with a sender module that operates as follows:
Each button press generates a unique 4-bit symbol representing that specific button. These bits are transmitted sequentially as a bitstream. The signals use a Non-Return-to-Zero (NRZ) encoding, meaning the output remains at 1 when the controller is idle, with no buttons pressed. When a button is pressed, the transmission begins with a leading 0 to indicate the start of data transfer. The subsequent bits are then sent one by one. Upon completion of the transmission of all four bits, the signal automatically returns to its idle state (logic 1), indicating that the system is once again waiting for input. This guarantees that each button press produces a uniquely distinguishable and well-structured transmission sequence, effectively preventing any overlap or interference between symbols. You can see the corresponding sequence for each button illustrated in figure 1.2.

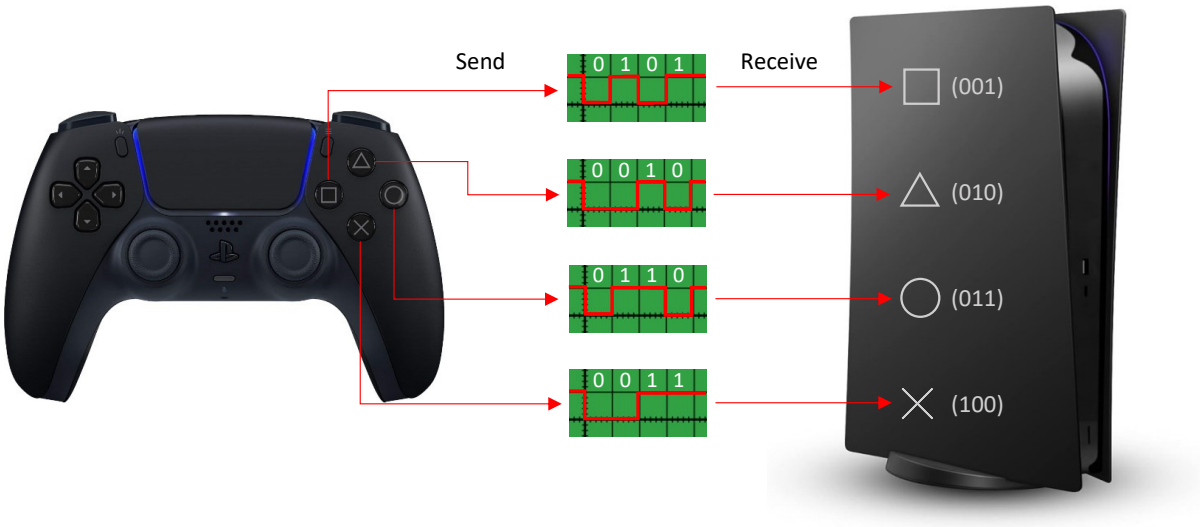


Figure 1.2: Controller sender and receiver protocol

Here is a comprehensive explanation of the sender module (download link):

```
module sender (
    input wire clk,
    input wire [3:0] button,
    output reg Tx
    );

    // BUTTONS //
    localparam SQUARE   = 4'b1000;      // button[3] --> Square
    localparam TRIANGLE = 4'b0100;      // button[2] --> Triangle
    localparam CIRCLE   = 4'b0010;      // button[1] --> Circle
    localparam CROSS    = 4'b0001;      // button[0] --> Cross

    // SYMBOLS //
    localparam SQUARE_SYMBOL   = 4'b0101;
    localparam TRIANGLE_SYMBOL = 4'b0010;
    localparam CIRCLE_SYMBOL   = 4'b0110;
    localparam CROSS_SYMBOL    = 4'b0011;


    // The rest of the code ...


endmodule
```

Snippet 1.1: Sender module

The module accepts 4 button inputs, corresponding to the 4 designated controller buttons mapped from "Square" to "Cross," as specified in snippet 1.0. Upon activation of a button, its respective encoded sequence is transmitted via the "Tx" output. Notably, each sequence is transmitted only once per button press, ensuring a singular, non-repetitive signal. Following transmission, the module seamlessly transitions into an idle state, maintaining this condition until the previously pressed button is released and a new input is detected.

Since you have access to the sender module, you are assigned to design the receiver module on the console side. The receiver should interpret the received 4-bit symbol and determine which button was pressed with a 3-bit output. ("000" shows that no button has been pressed yet)

Rest assured, simultaneous button presses are not a concern, as we guarantee such a scenario will not occur. Moreover, even if they were to happen, it would be inconsequential on the receiver side, as multiple 4-bit symbols would still be received as a serialized bitstream.

To achieve this, you'll have to create a sequence detector finite state machine (FSM). Complete the Moore machine provided in figure 1.3 and write the receiver module. (The module holds the previously detected button on the output, until a new pattern is received)

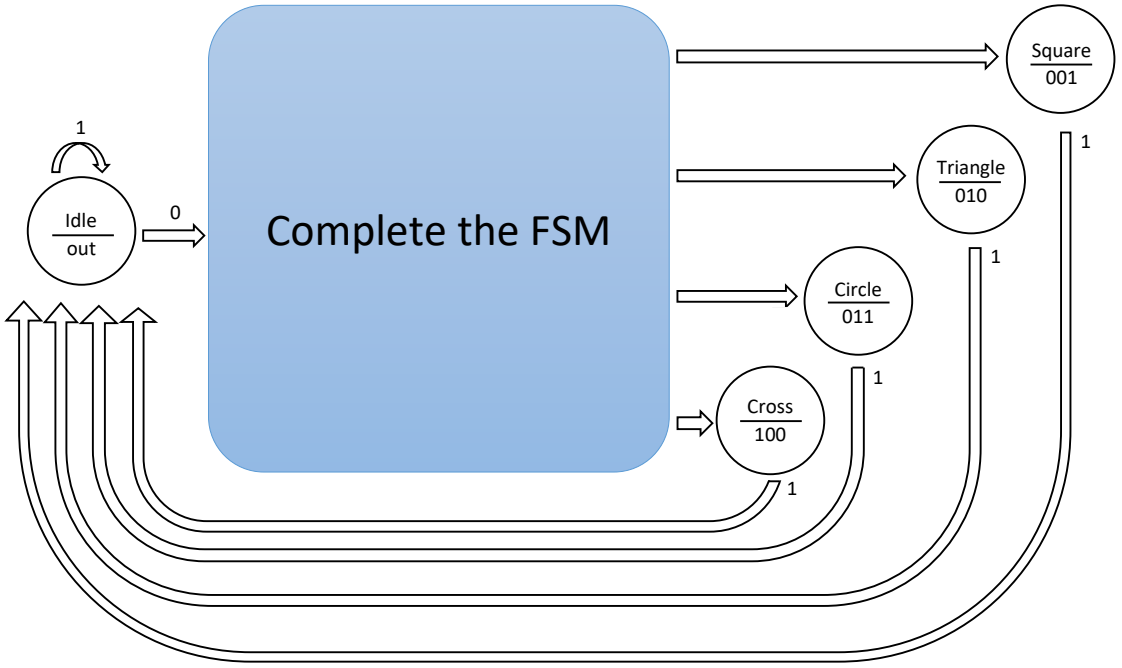


Figure 1.3: Incompleted receiver FSM

To ensure your module functions correctly, take an instance of the given sender and your receiver module. Then connect them as illustrated in figure 1.4. The input gets connected to 4 buttons and the 50 Mhz clock pin. The output will then be shown on 3 LEDs.

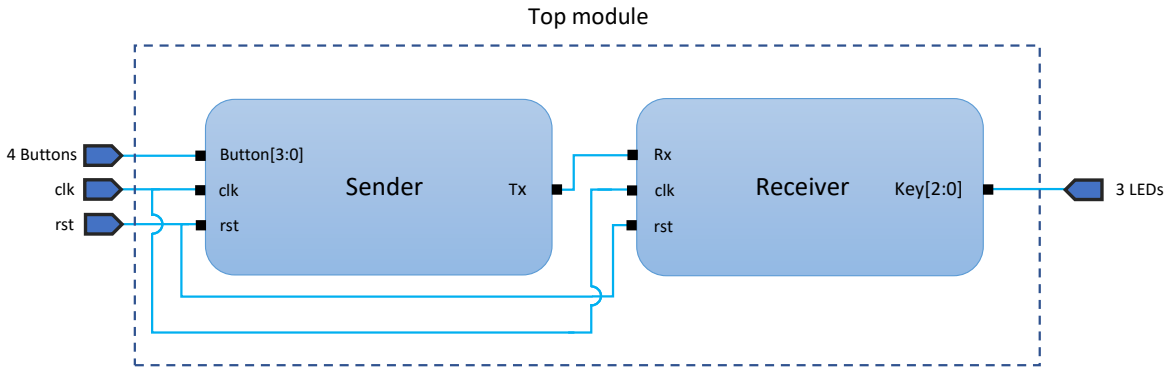


Figure 1.4: Block diagram of the sender/receiver interconnections

1.3.2 BCD counter

The console is equipped with parental control features, including a 3-digit timer to limit playtime. Your task is to design a module that displays the timer value on the 7-seg displays (HEX2-HEX0) from 0 to 999. A circuit diagram illustrating the system is shown in figure 1.5.

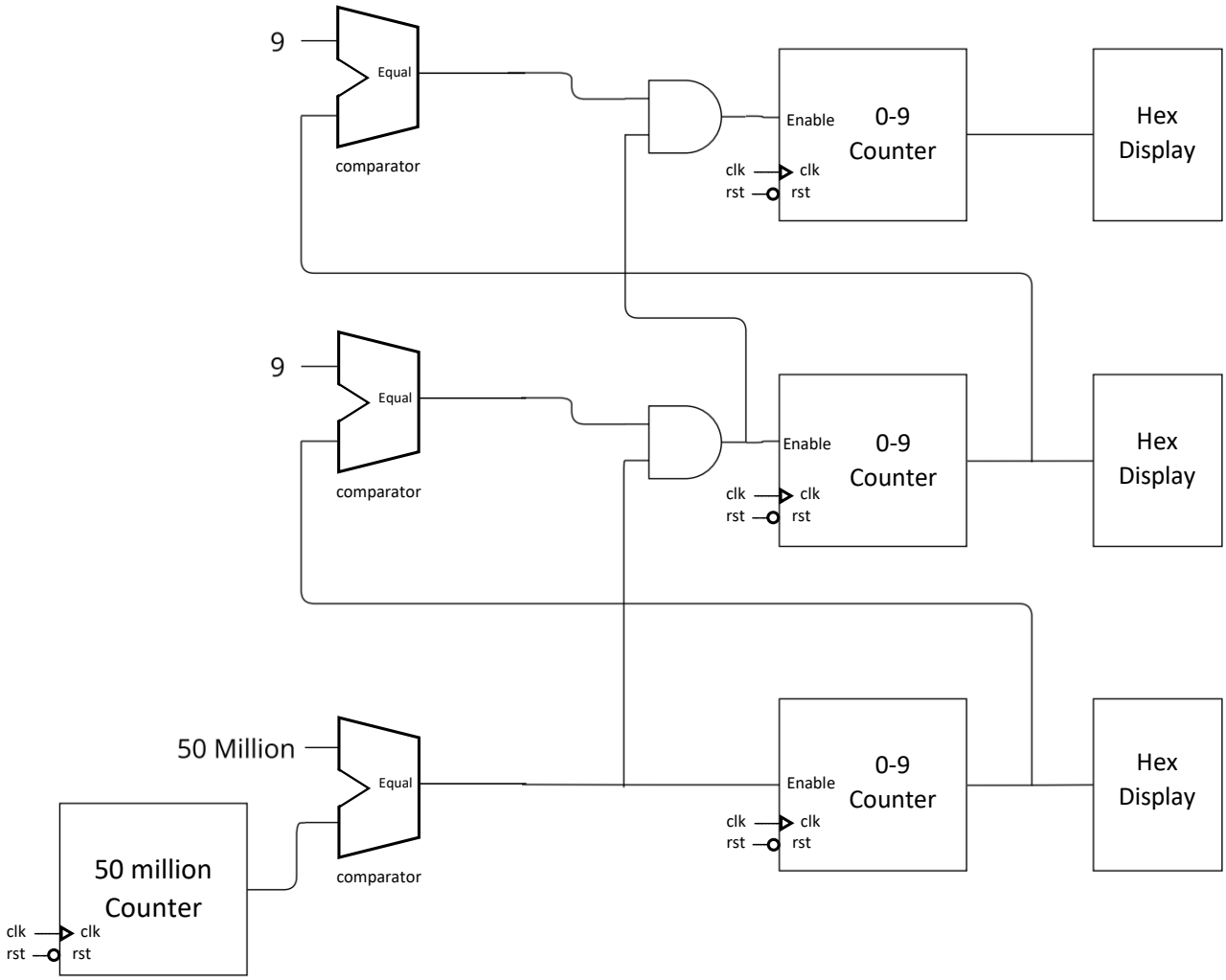


Figure 1.5: Block diagram of the 3-digit 7-segment counter

You need to implement 3 modules:

- **50 Million Counter**:
  It counts up to 50 Million when connected to a 50 Mhz clock, so it takes precisely 1 second.It also includes a reset feature that allows the counter to start from zero.
- **0-9 Counter**:
  It counts from 0 to 9 if the enable input is set to HIGH. It also has a reset to count from zero.
- **Hex Display**:
  It takes a 4-bit input as a hex number and gives out a 7-bit output indicating what segments of 7-segment should be turned on to light up the given number on the 7-segment display.

After implementing these modules, you can instantiate and connect them as depicted in figure 1.5

⚠Before compiling your code in Quartus II, you must explicitly set the Synthesis tool to use the state assignment specified in your Verilog code. Otherwise, Quartus II will assign state codes automatically, ignoring your definitions. To do this, navigate to Assignments → Settings → Analysis and Synthesis → More Settings, and set State Machine Processing to "User-Encoded".

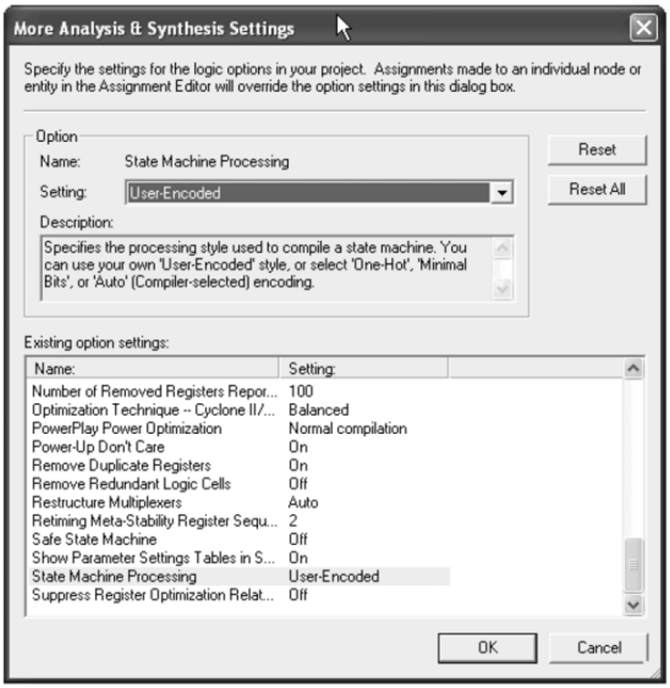


Figure 1.6: Specifying the state assignment method in Quartus

To test out the true functionality of your module:

1. Assign the clock to the 50 MHz clock pin of the DE2.
2. Assign the rst pin to one of the push buttons.
3. Assign the 3 7-bit outputs of hex displays to HEX2-HEX0.

The pin assignments of the altera DE2-70 is provided in the following link.

To examine the circuit produced by Quartus II open the RTL Viewer tool. The RTL viewer can be launched from the Quartus menu (Tools → Netlist viewers → RTL viewer). Double-click on the box shown in the circuit that represents the FSM, and determine whether the state diagram that it shows properly corresponds to yours.
To see the state codes used for your FSM, open the Compilation Report, select the Analysis and Synthesis section of the report, and click on State Machines. You must also record the size of your circuit and the number of flip-flops (also called registers) used. The size is given as the number of Logic Elements (LEs) in the Compilation Report, and number of flip-flops is given as the total number of dedicated registers. You can refer back to figure 1.7 for a sample compilation report.

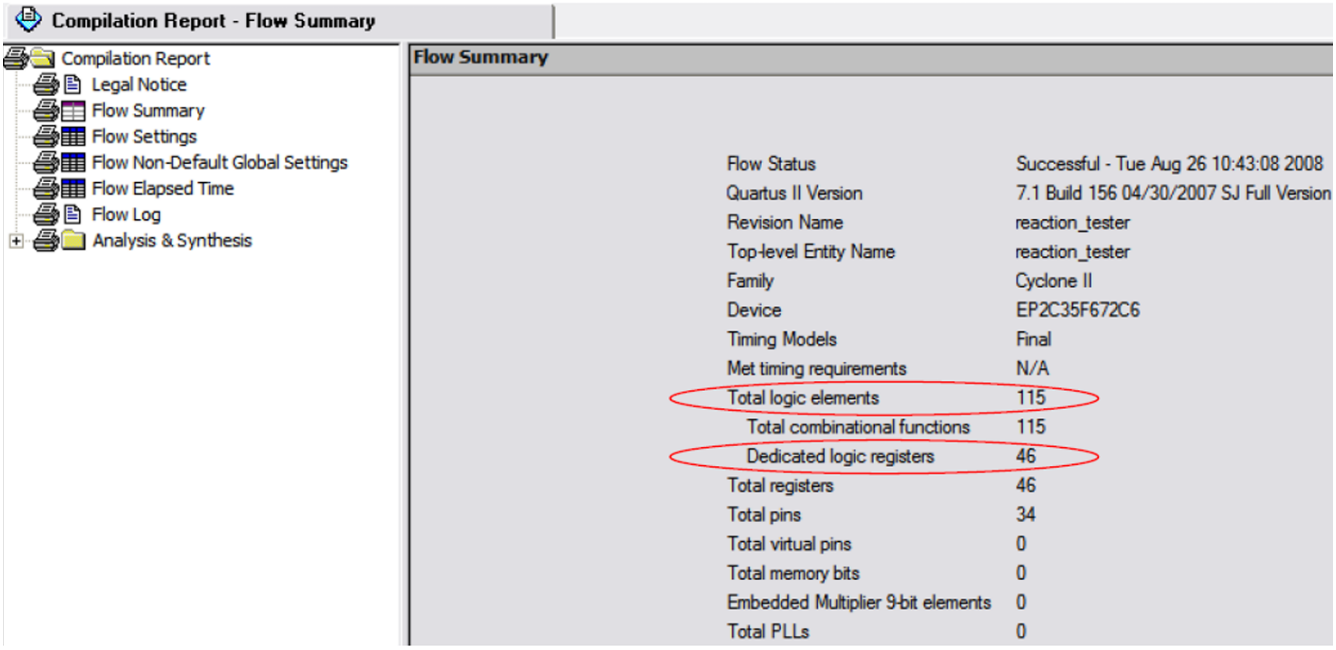


Figure 1.7: Specifying the state assignment method in Quartus

1.3.3 Secret sequence detector

As a kid, you were always frustrated by the playtime limitations. Since you were involved in both tasks, you have the opportunity to secretly embed a hidden pattern that allows the timer to be manually reset. If the user presses the specific sequence of buttons in the designated pattern provided below, they will be able to restart the timer.

Figure 1.8: Secret pattern to restart the timer

Complete the Mealy machine provided in figure 1.9 and write the secret FSM module.

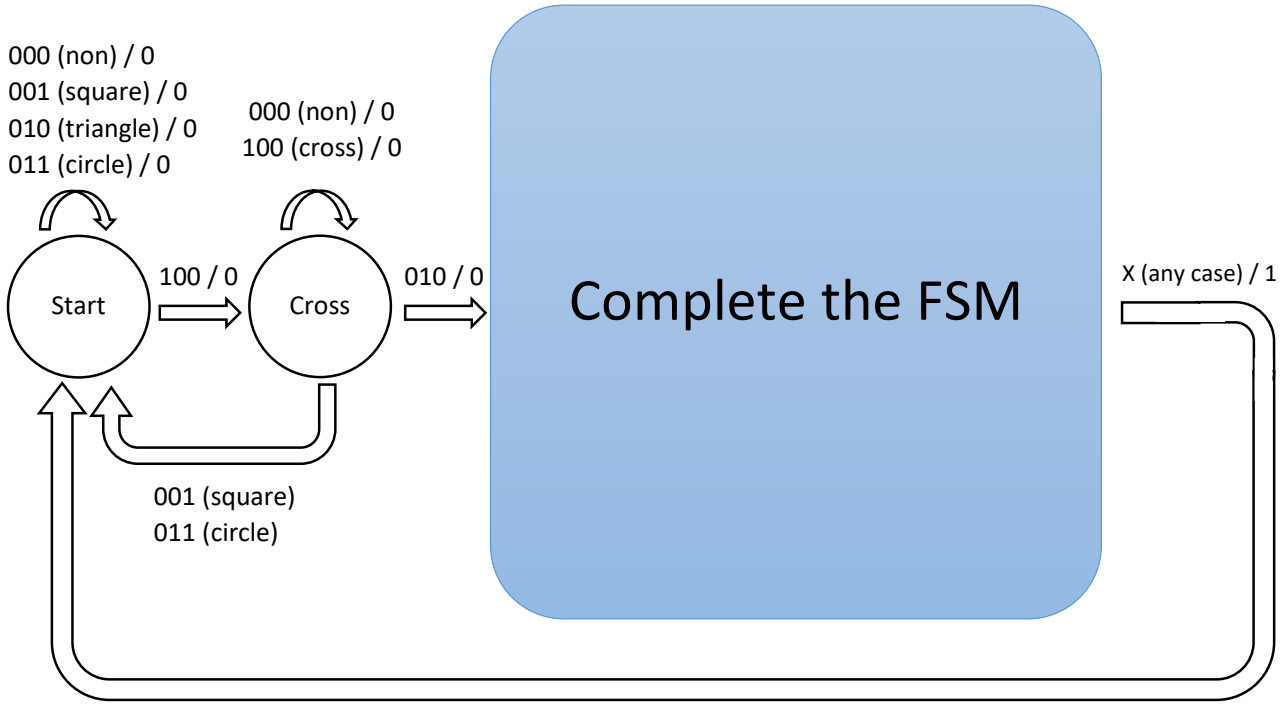


Figure 1.9: Incomplete FSM of the secret pattern

Now to see if the secret pattern works, you will have to integrate all the modules you used in the previous parts and connect them like the block diagram provided in figure 1.10. Press the buttons in the specified order and observe whether the timer is reset.

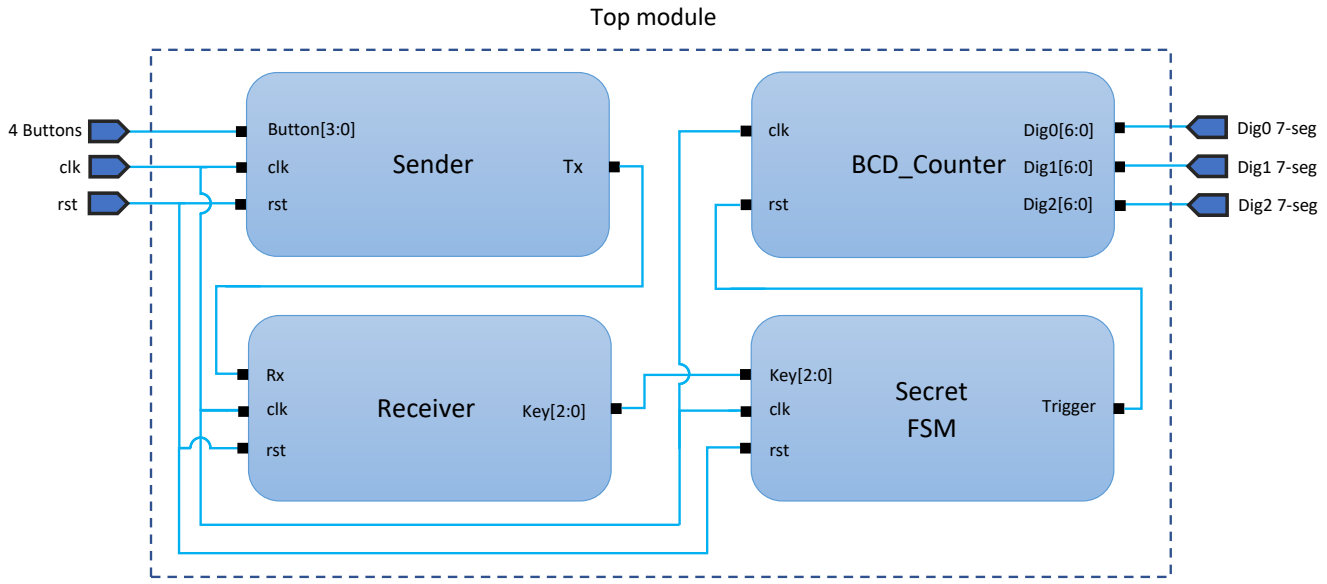


Figure 1.10: Final Block Diagram Incorporating the New Reset Feature Utilizing the Secret Sequence

1.3.4 Bonus

The FSM provided in figure 1.9 doesn't function properly in some cases. If the secret pattern contained two same successive buttons like in figure 1.11, it won't be able to tell if a button is pressed for a second consecutive time. Although it may still function, entering either "Cross, Cross, Square, Triangle" or "Cross, Square, Triangle" would both restart the timer, which is undesired.

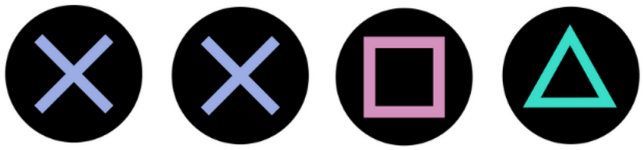


Figure 1.11: The issued secret pattern

Feel free to adjust the FSM or any of modules you designed to address this issue. Then write another secret FSM for figure 1.9 and see if "Cross, Cross, Square, Triangle" resets the timer & "Cross, Square, Triangle" is ignored!

Appendix - Quartus II

In this appendix, you will be guided through the essential steps of using Quartus II, including how to set up a new project, compile your code, and run it on an FPGA device. We will use Altera DE2-35 and DE2-70 in some Labs, which require you to work with Quartus II since they both include the Cyclone II chip. You can download Quartus II (Web edition) via the Intel website.

⚠ The final release of Quartus II dates back to 2013, and as such, it has not been optimized for compatibility with Windows 11. If you are currently using Windows 11, it is strongly recommended to install a virtualized instance of Windows 10 via VMware or VirtualBox, where Quartus II can be seamlessly installed and operated.

A.1 VM resource allocation

When installing the virtual machine, we should provide it with a sufficient amount of resources so Quartus can run smoothly. It is recommended to allocate half of your CPU cores, maximum amount of RAM that does not cause swapping (around half), and 50GB-60GB of storage.

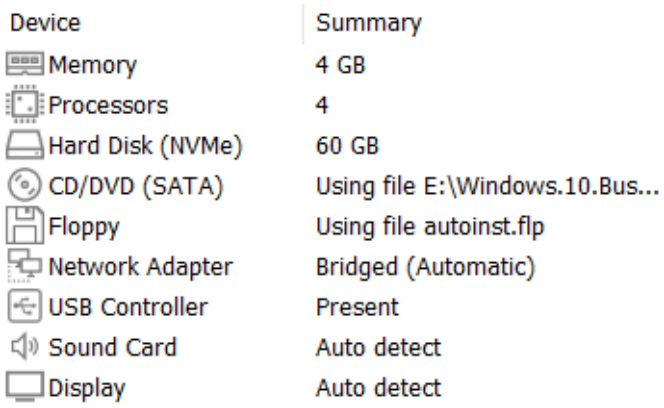


Figure A.1: An instance of resource allocation

A.2 Quartus installation

The only part you need to pay close attention to is the "Select Components" tab. By default, the chosen options are as shown in figure A.2. You don't need to change anything. Just make sure that the "Cyclone II/III/IV" device is checked. The rest of the installation is straightforward.

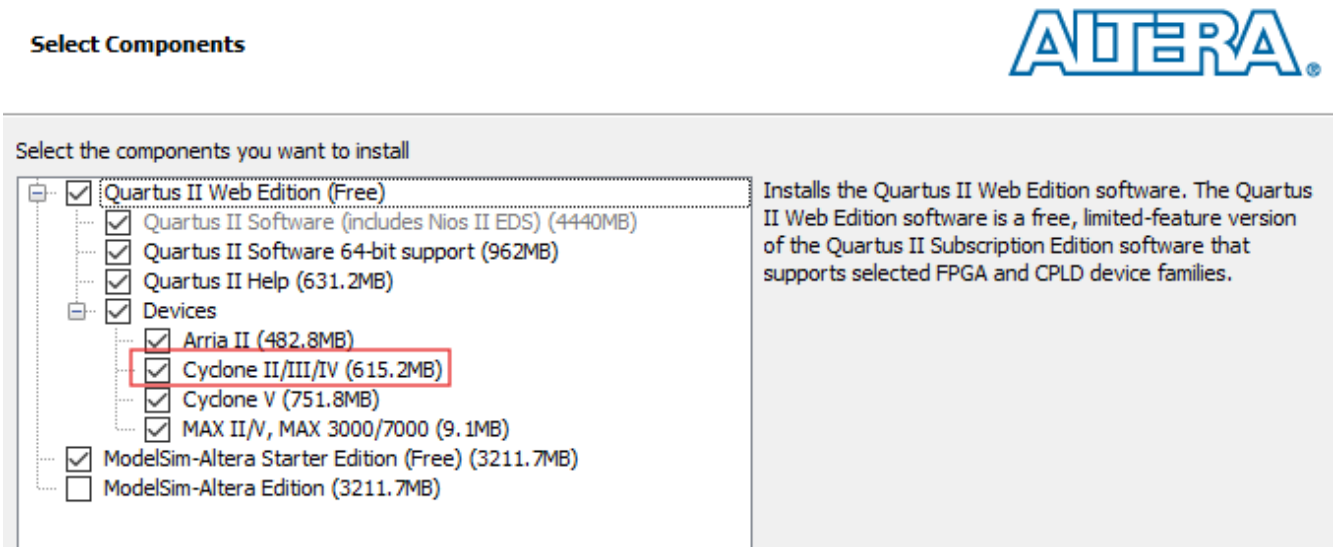


Figure A.2: An instance of resource allocation

A.3 Making a new project

Open Quartus and click on "File → New → New Quartus II Project". In the 1st tab, enter the address where you want to save the project, project name, and the top-level design entity name.

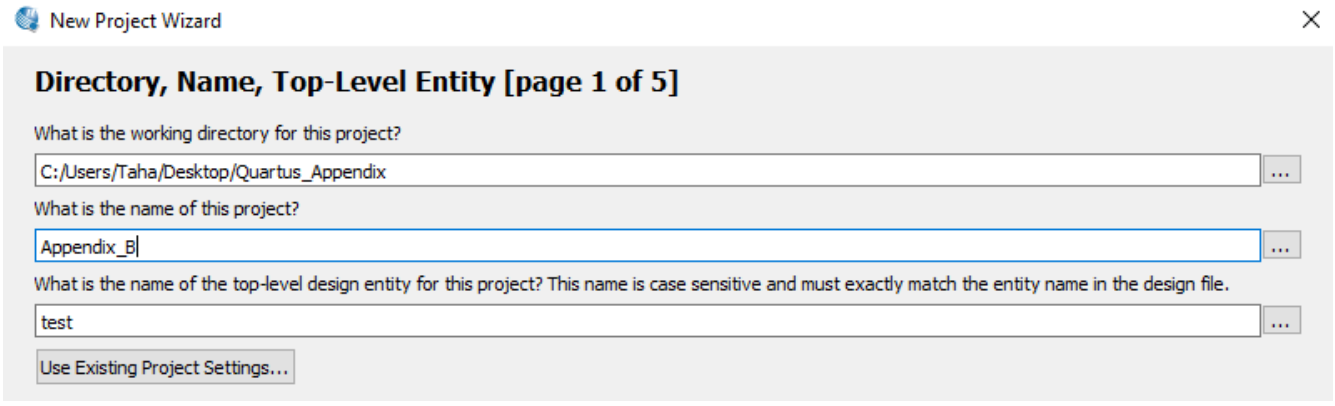


Figure A.3: "Select Components" tab in the installation process

⚠ Do not save the project in default directories such as "Desktop", "Downloads" or "Documents", as this may lead to an error illustrated in figure A.4, making the project read-only and preventing further edits. Instead, create a dedicated folder within your preferred directory and save the project there to ensure proper functionality and accessibility. An example of proper addressing is mentioned in figure A.3

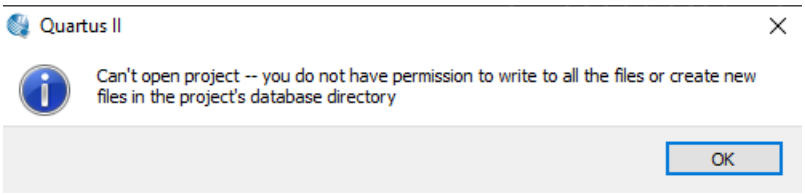


Figure A.4: Read-only project error

In the 2nd tab, you will have to add your pre-written modules. If you have not prepared any yet, you can skip this page.

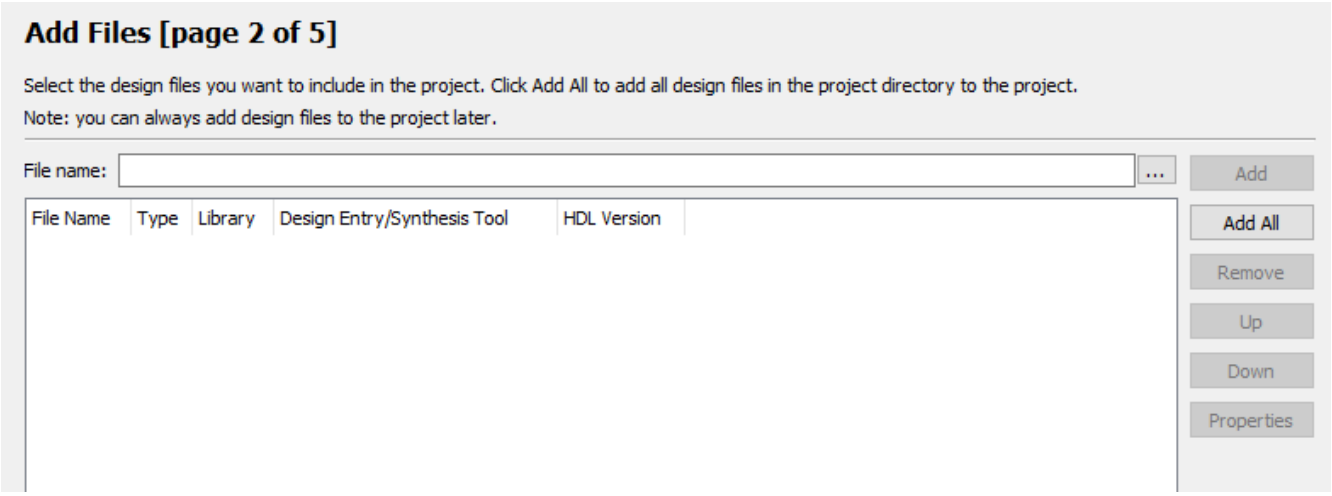


Figure A.5: "Add files" page

Moving on to the 3rd page, you will need to select the chip with which you intend to work. There are two types of DE2 boards available in the lab, so make sure to choose the chip corresponding to the board that has been assigned to you.

The DE2-35 and DE2-70 boards available in the lab are equipped with chips bearing the respective serial numbers "EP2C35F672C6" and "EP2C70F896C6." The meaning of each letter and number in these serial numbers is detailed in figure A.6.

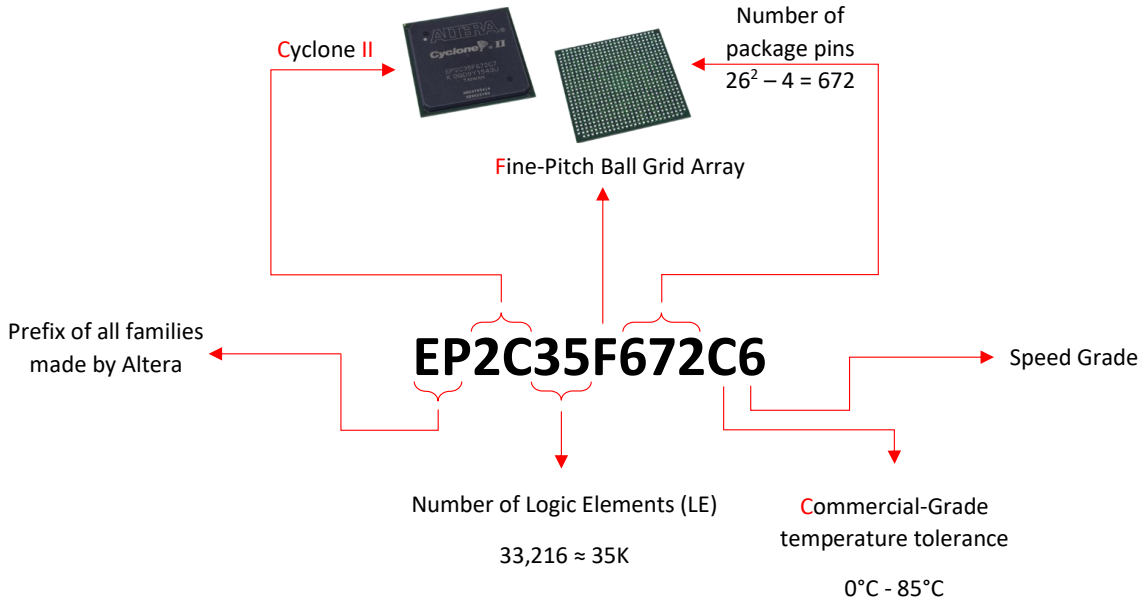


Figure A.6: Chip serial number analysis

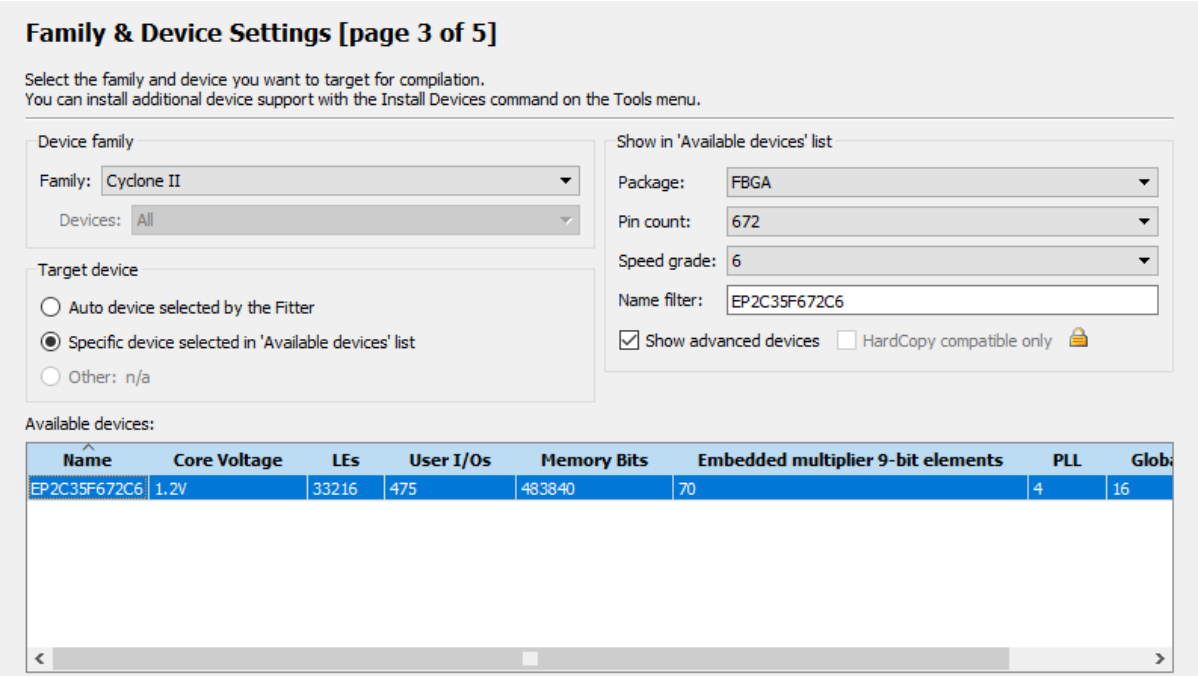


Figure A.7: Selecting the chip serial number

You can either set the family, package, pin count & speed grade to find your desired chip or alternatively, enter the full name in the "Name filter" box as shown in figure A.7

In the 4th page, you are given options to choose your desired tool for synthesis, simulation, verification , etc. As we are not concerned with using any particular tool, we leave it as is, so they will be set to the default ones.

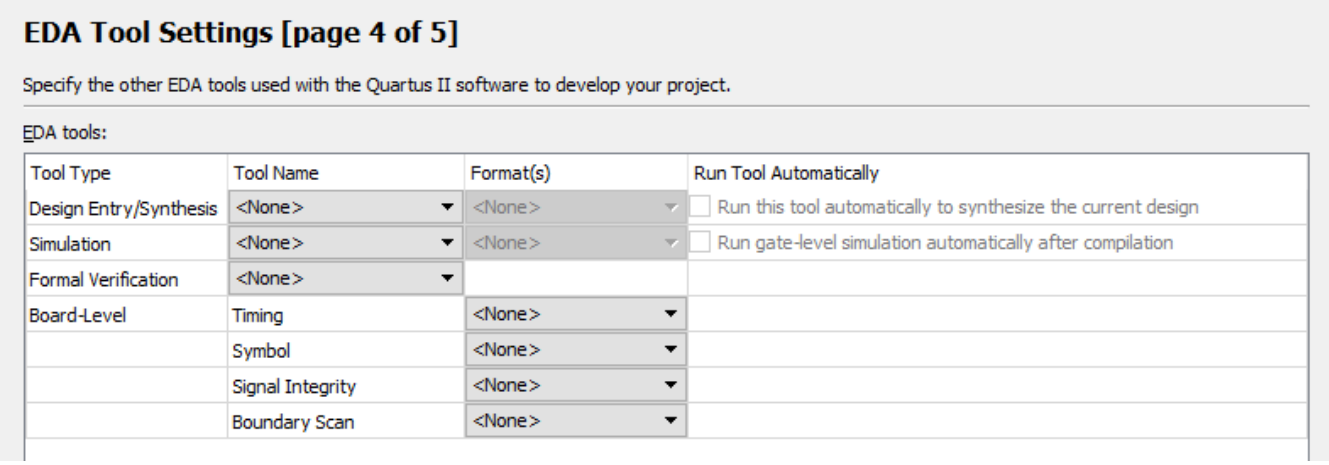


Figure A.8: "EDA tool settings" page

In the last page, you can see a brief summary of the settings you made.

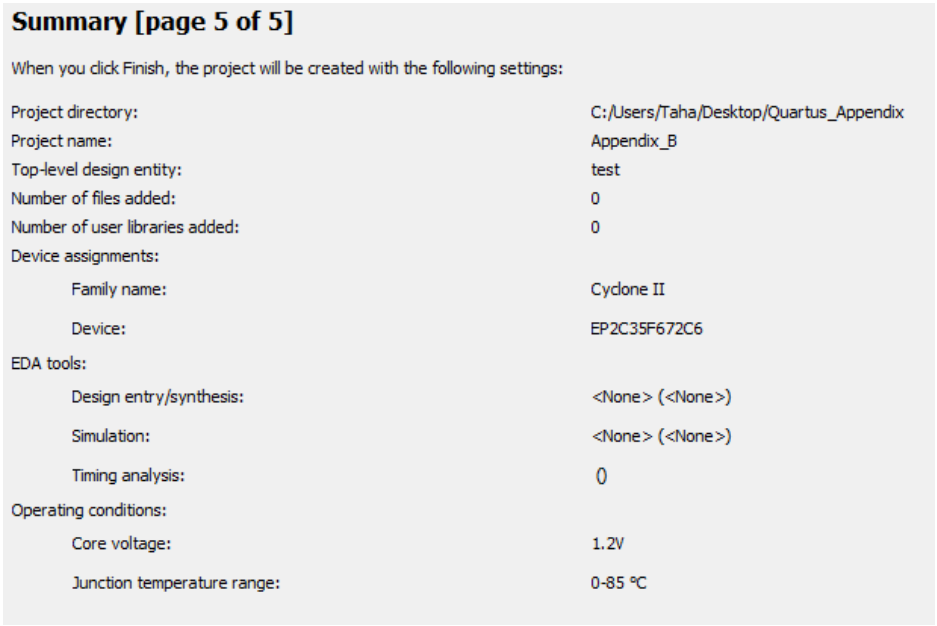


Figure A.9: "Summary" page

A.4 FPGA design flow

By pressing finish, the project will be generated. To make a Verilog module, you should head to "File → New → Design Files → Verilog HDL File". A new window will pop up. To go through the process, we will enter with the following code and save it in the same directory as the project.

```
module test(
    input wire [17:0] SW,
    output wire [17:0] LEDR
    );

    assign LEDR = SW;
endmodule
```

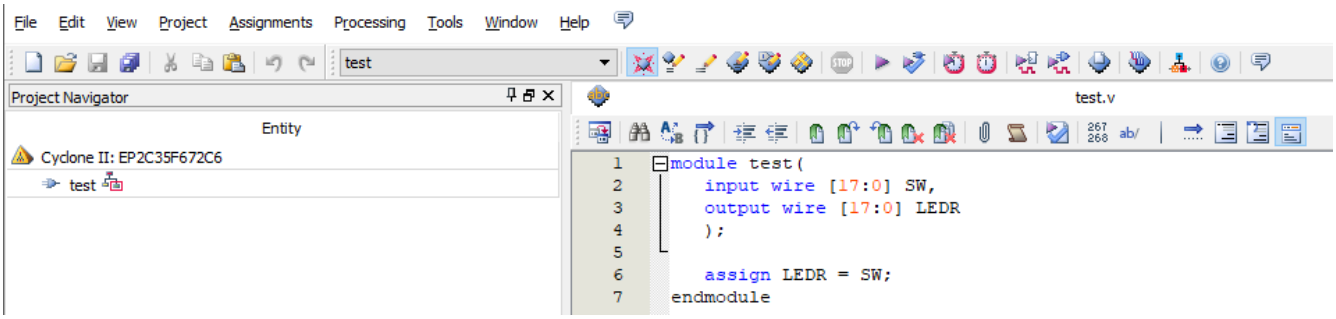


Figure A.10: "Quartus project snapshot

Set the flow to "Compilation". Now to make sure there is no syntax issue in the saved module, click on "Analysis & Sythesis" and wait for it to be done.

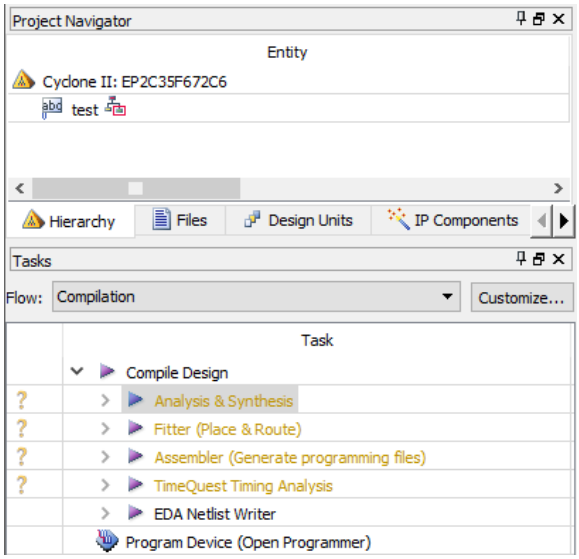


Figure A.11: FPGA design flow

Now before continuing the rest of the flow, we need to assign our input/outputs to the pins of the board using one of two options mentioned below.

1. Manual Pin Assignment

Go to "Assignments → Pin Planner". The window illustrated in figure A.12 will pop up.

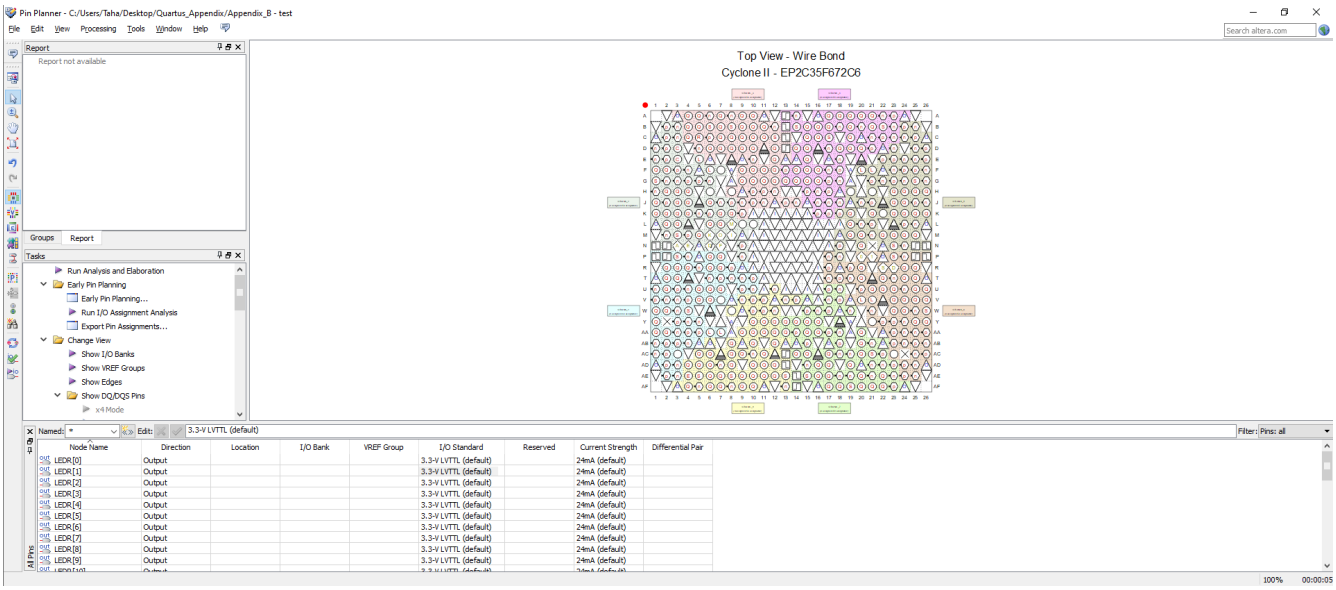
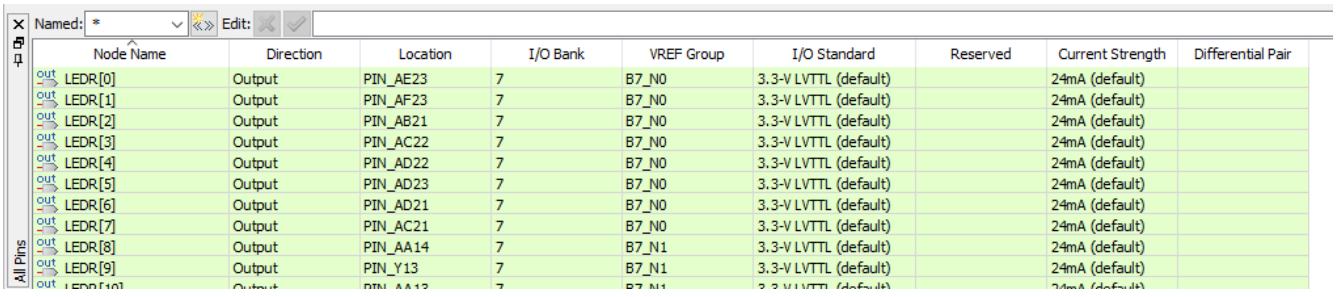


Figure A.12: Pin Planner Window

Now using the DE2-35 manual or DE2-70 manual find your desired pins and assign them to the corresponding inputs/outputs like figure A.13. (You do not need to type the "PIN_" part. It will be written automatically)

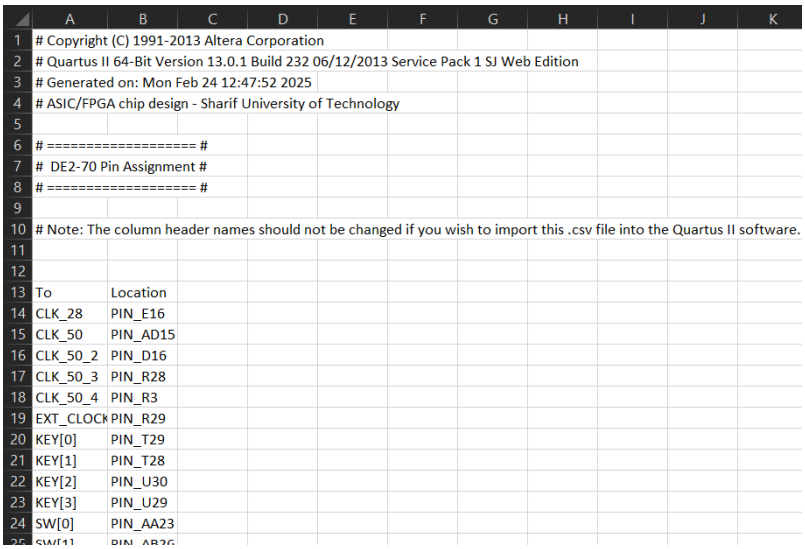


Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Differential Pair
LEDR[0]	Output	PIN_AE23	7	B7_N0	3.3-V LVTTL (default)		24mA (default)	
LEDR[1]	Output	PIN_AF23	7	B7_N0	3.3-V LVTTL (default)		24mA (default)	
LEDR[2]	Output	PIN_AB21	7	B7_N0	3.3-V LVTTL (default)		24mA (default)	
LEDR[3]	Output	PIN_AC22	7	B7_N0	3.3-V LVTTL (default)		24mA (default)	
LEDR[4]	Output	PIN_AD22	7	B7_N0	3.3-V LVTTL (default)		24mA (default)	
LEDR[5]	Output	PIN_AD23	7	B7_N0	3.3-V LVTTL (default)		24mA (default)	
LEDR[6]	Output	PIN_AD21	7	B7_N0	3.3-V LVTTL (default)		24mA (default)	
LEDR[7]	Output	PIN_AC21	7	B7_N0	3.3-V LVTTL (default)		24mA (default)	
LEDR[8]	Output	PIN_AA14	7	B7_N1	3.3-V LVTTL (default)		24mA (default)	
LEDR[9]	Output	PIN_Y13	7	B7_N1	3.3-V LVTTL (default)		24mA (default)	

Figure A.13: Manually assigned pins

2. Loading the csv file of pin assignments

Using the provided DE2-35 & DE-70 pin assignment csv files, you can set them automatically. The content of the file is shown in figure A.14.



	A	B	C	D	E	F	G	H	I	J	K
1	# Copyright (C) 1991-2013 Altera Corporation										
2	# Quartus II 64-Bit Version 13.0.1 Build 232 06/12/2013 Service Pack 1 SJ Web Edition										
3	# Generated on: Mon Feb 24 12:47:52 2025										
4	# ASIC/FPGA chip design - Sharif University of Technology										
5											
6	# ==================== #										
7	# DE2-70 Pin Assignment #										
8	# ==================== #										
9											
10	# Note: The column header names should not be changed if you wish to import this .csv file into the Quartus II software.										
11											
12											
13	To	Location									
14	CLK_28	PIN_E16									
15	CLK_50	PIN_AD15									
16	CLK_50_2	PIN_D16									
17	CLK_50_3	PIN_R28									
18	CLK_50_4	PIN_R3									
19	EXT_CLOCK	PIN_R29									
20	KEY[0]	PIN_T29									
21	KEY[1]	PIN_T28									
22	KEY[2]	PIN_U30									
23	KEY[3]	PIN_U29									
24	SW[0]	PIN_AA23									

Figure A.14: Pin assignment csv file

Note : To be able to have automatic assignment, you have to either use the names provided in the csv file, or change them to match your input/output names.

⚠ The CSV file generated by Quartus does not strictly adhere to standard CSV formatting conventions. Consequently, editing and saving it via Excel may introduce additional commas, rendering the file unreadable by Quartus. To avoid this issue, it is recommended to use a text editor to modify the required changes. Alternatively, if you prefer using Excel, ensure that after saving the file, you review it in a text editor and manually remove the extraneous commas, as highlighted in figure A.15.

```
# Copyright (C) 1991-2013 Altera Corporation,
# Quartus II 64-Bit Version 13.0.1 Build 232 06/12/2013 Service Pack 1 SJ Web Edition,
# Generated on: Mon Feb 24 12:47:52 2025,
# ASIC/FPGA chip design - Sharif University of Technology,
,
# ==================== #,
# DE2-70 Pin Assignment #,
# ==================== #,
,
# Note: The column header names should not be changed if you wish to import this .csv file into the Quartus II software.,
,
,
To,Location
CLK_28,PIN_E16
```

Figure A.15: Redundant commas added by Excel

Now to apply it, go to "Assignments → Import Assignments...", enter the address of the csv file and press ok. Now to ensure the pins are assigned, you can either check the pin planner or go to "Assignments → Assignment editor". As you can see, Quartus set the pins automatically.

	Status	From	To	Assignment Name	Value
1	Ok		SW[0]	Location	PIN_N25
2	Ok		SW[1]	Location	PIN_N26
3	Ok		SW[2]	Location	PIN_P25
4	Ok		SW[3]	Location	PIN_AE14
5	Ok		SW[4]	Location	PIN_AF14
6	Ok		SW[5]	Location	PIN_AD13
7	Ok		SW[6]	Location	PIN_AC13
8	Ok		SW[7]	Location	PIN_C13
9	Ok		SW[8]	Location	PIN_B13
10	Ok		SW[9]	Location	PIN_A13
11	Ok		SW[10]	Location	PIN_N1
12	Ok		SW[11]	Location	PIN_P1
13	Ok		SW[12]	Location	PIN_P2
14	Ok		SW[13]	Location	PIN_T7
15	Ok		SW[14]	Location	PIN_U3
16	Ok		SW[15]	Location	PIN_U4
17	Ok		SW[16]	Location	PIN_V1
18	Ok		SW[17]	Location	PIN_V2
19	Ok		LEDR[0]	Location	PIN_AE23
20	Ok		LEDR[1]	Location	PIN_AF23

Figure A.16: Auto pin assignment result

Go back to the sidebar/flow tab, and run each step one by one until they all get marked as verified.

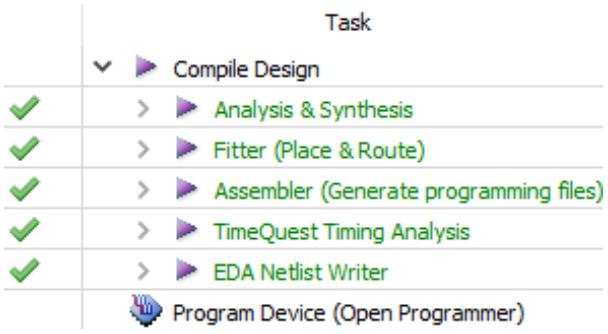


Figure A.17: Completed Design flow

A.5 Installing the USB-Blaster driver

Before we program the chip, we need to install the USB-Blaster driver. Connect the board to your Laptop/PC via the "Blaster" port using a Type-B USB cable. Then head to the "Device manager". If the driver has not been installed before, it will be shown under the "Other devices" section.

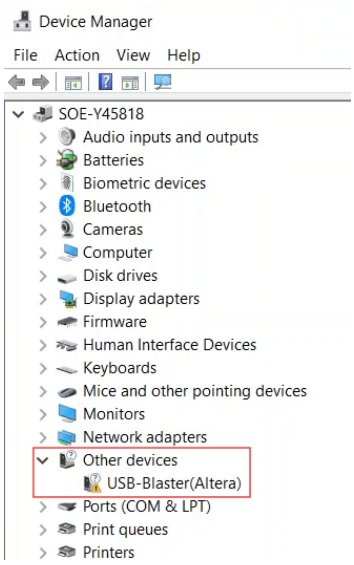


Figure A.18: Device manager window

Right click on it and select "Update Driver". Then select "Browse my computer for driver software".

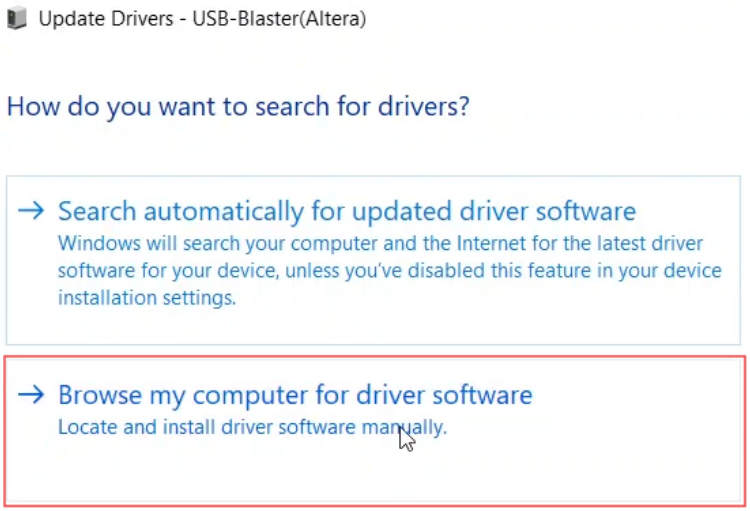


Figure A.19: Update Drivers window

Then browse into this address : "<Quartus file location>\<Version number>\quartus\drivers". Check the "Include subfolder" option and click on next.

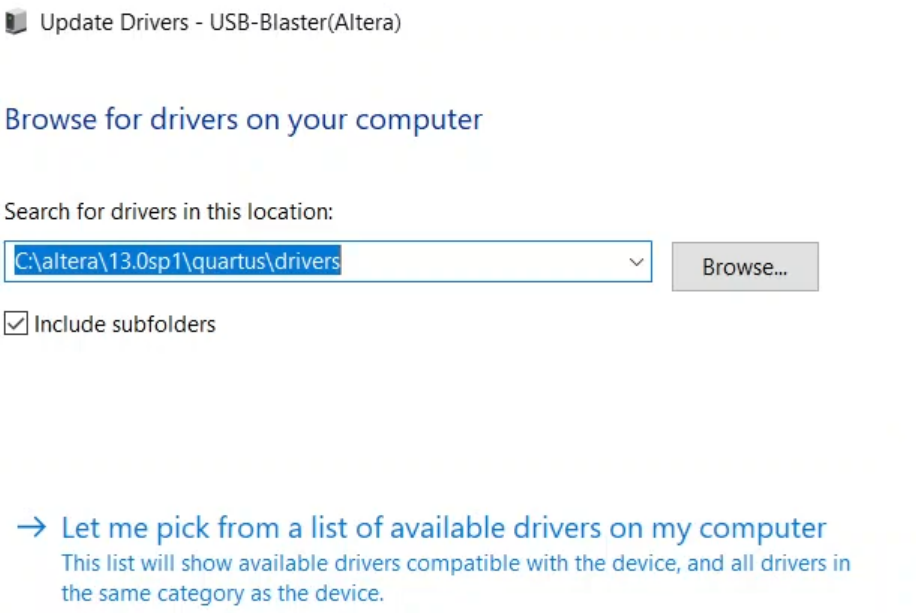


Figure A.20: The driver location

It will search for a few seconds and then a "Windows Security" window pops up, showing that the driver is detected. Click "Install" and wait for it to do its job.

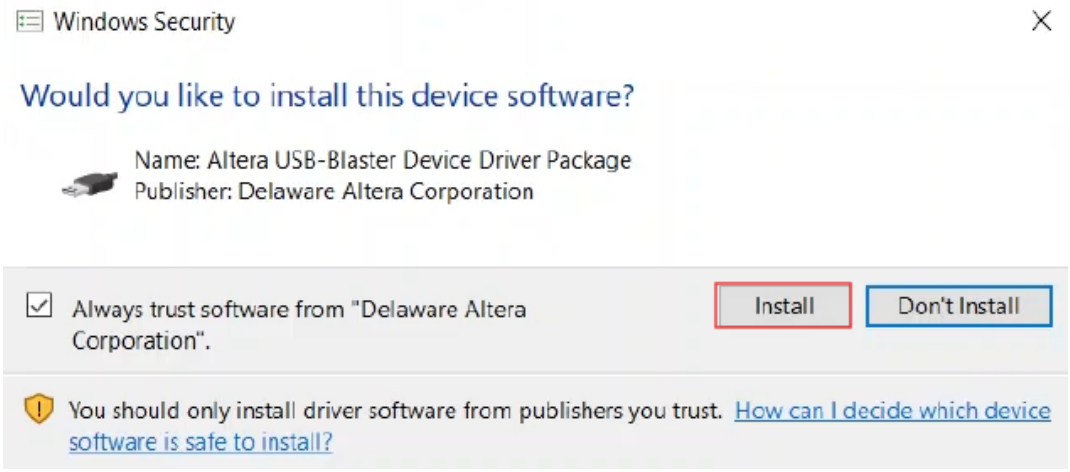


Figure A.21: Detected USB-Blaster driver notification

The notification shown in figure A.22 indicates that the driver is successfully installed.

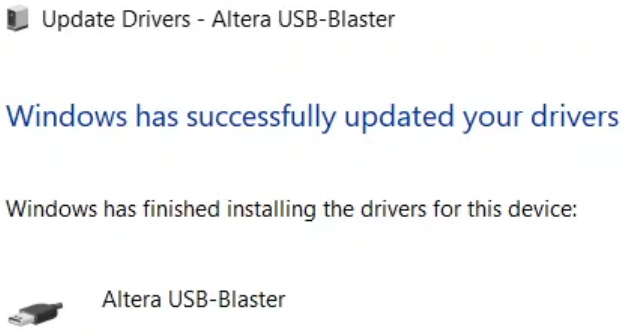


Figure A.22: Successful driver installation

Note that installing the driver is a one-time procedure, and you will not be required to repeat this process for subsequent programming sessions with the board.

To make sure of the proper driver installation, open the device manager again. The "Altera USB-Blaster" is recognized under the name of USB controllers and no longer carries the alert sign.

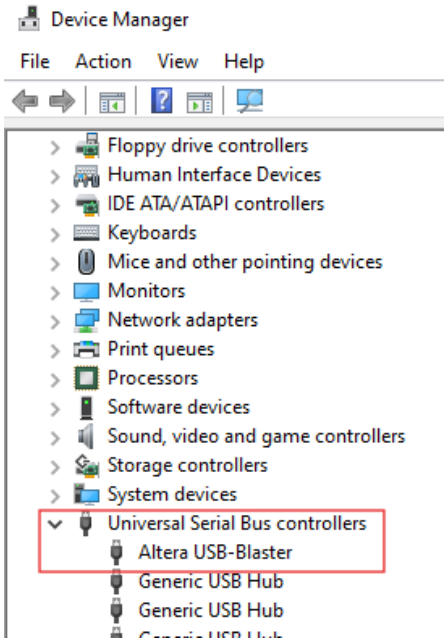


Figure A.23: Detected USB driver in device manager

A.6 Programming the chip

Go back to the flow side bar(figure A.17) and click on "Program Device (Open Programmer)". Once the programmer is launched, click on "Hardware Setup..."

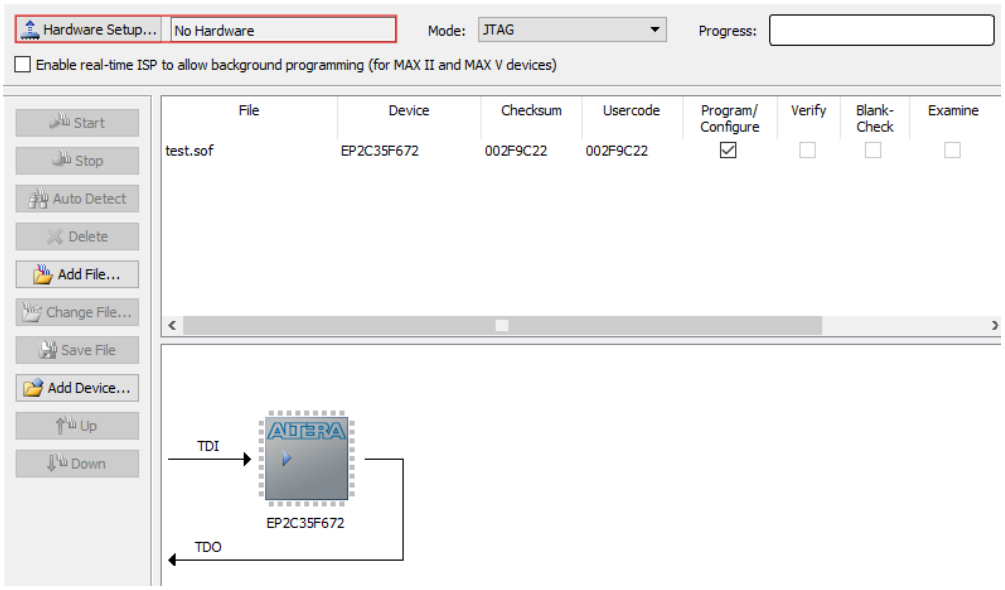


Figure A.24: Programmer window

Select the USB-Blaster as the selected hardware and close the window.

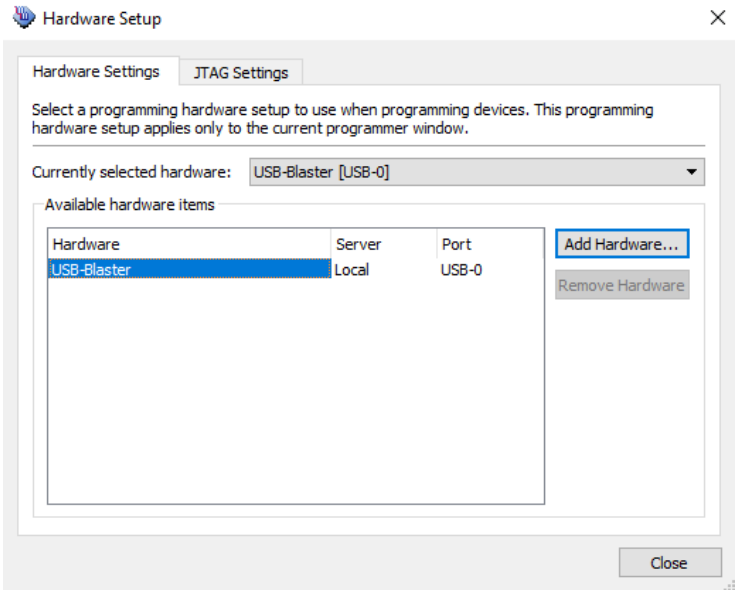


Figure A.25: Hardware setup window

Make sure the switch on the left side of the board is set on "RUN". Then hit start and wait for the bitstream to be fully uploaded.

Figure A.26: Uploading the bitstream

Here is the result of the board. As you can see, only ON switches have the corresponding LEDs lit.

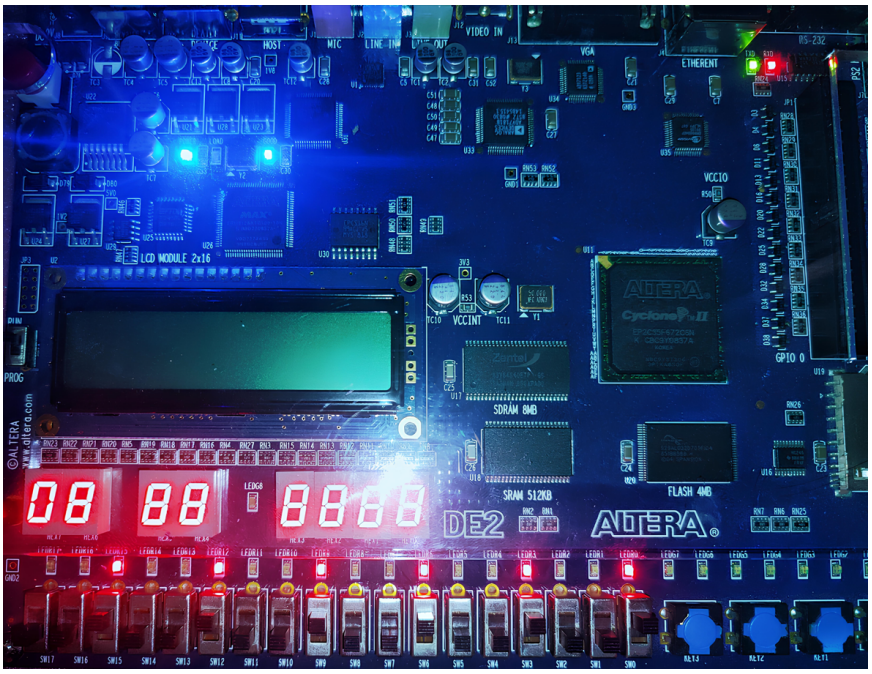


Figure A.27: DE2 board output

Department of Electrical Engineering