CHAPTER 3

METHODOLOGY AND DESIGN

3.1 Introduction

In order to create an implemental source code to the FPGA board, a good work plan is required. The summary of the design flow of this study is summarized in Figure 3.1.

The first thing need to be carried out is to plan the function of the system. This is including the interfaces of the system and how users are going to use them. This will later be the manual for end users. It is important that this part covers all the possibilities that may occur during system's operation.

Upon completing the function of the system, a thorough design of the source code can be done. It is written using Verilog in Altera Quatus® II environment. The flow of the program is using "bottom-top" method.. Having this method will create a systematic structure of the code, making redesigning for error correction easier.

Compilation and simulation is the following method. It is done using Simulator tool in Altera Quartus® II. Compilation is where the source code is verified to legally meet the Verilog-2001 rules, so as the simulation. Any violation of the rules will cause error in compiling. Important to have clean compiled source code due to proceed to simulation of the code. For error in simulation, the source code had to be modified until the desired waveform in the simulation is as wished.

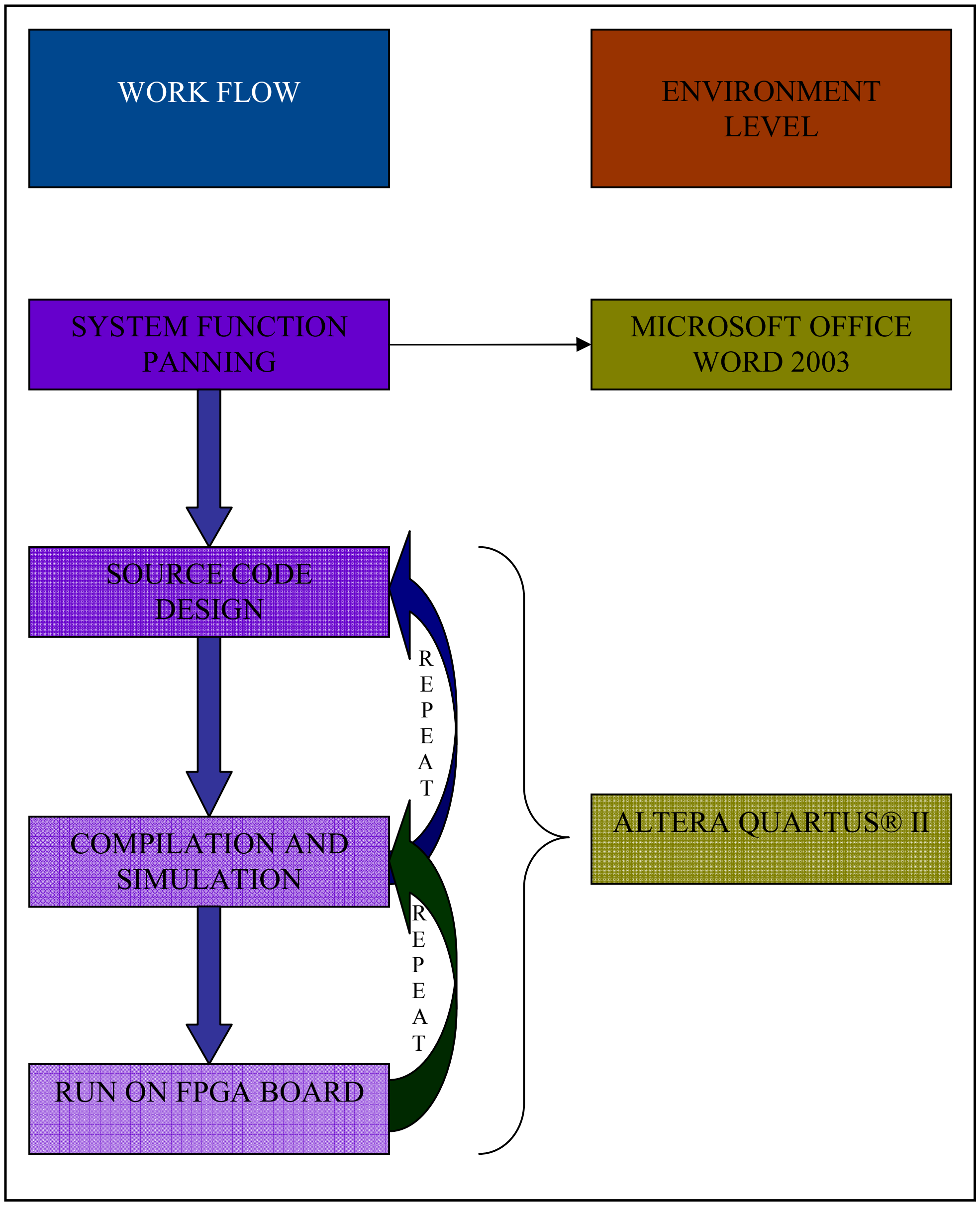


Figure 3.1 Flowchart showing work plan on the study

Following after compilation is testing the source code onto the FPGA board. This is done by downloading the source code into the FPGA board and running the downloaded source code as an application.

3.2 System Design

A user friendly but high level of security system is going to be designed. The input method is using pushbuttons. There are four pushbuttons available for use on the Altera FPGA board. Pressing the pushbuttons gives High and Low signal to the system that creates passkey. The source code will be pre-programmed with the passkey on which user must enter these passkey to gain access to the car. This passkey however, cannot be changed by the user directly from the system, to avoid hackers or thief from modifying the internal system's code.

The system will initially check the key's existence. If there is any, the system will disable itself automatically. Thus, if there is no key, the system will check the status of the pushbuttons. It waits until the correct passkey inserted by user to open the car's door. Else, the system will remain locking the car.

Upon gaining access, after the correct passkey has been inserted, the system will disable or disarm the internal car alarm. This is because latest car have its own internal alarm that will trigger itself if the car is opened without key. The system will disable the function for the while, and enable it again when the door is closed and locked.

Some users may encounter the problem of having to turn on the car's internal light manually after opening the door, and they have to search for the light's switch. This system will however, automatically open the light after the door is opened. User will not have to face the hardship to search the light manually, and this feature will greatly help them at nights and in the darks.

As the system is implemented using FPGA board, the conditions of the system and the progress of the system will be reported using the seven segment display. This is useful for the later programmer to monitor the status of the system. They can also be used to detect the error if there is any, so it will be easier to fix the code. This feature is however not for end user.

3.2.1 The Altera Quartus® II Software

To design a source code, a compiler and simulator is required. Computer Aided Design (CAD) made it easy to implement a desired logic circuit by using a programmable logic device, such as a field-programmable-logic-array (FPGA). Altera Quartus® II is used for this study due to its simplicity and free license. The compiler uses the Verilog-2001 rules during compilation. Other than Verilog, this software supports VHDL too.

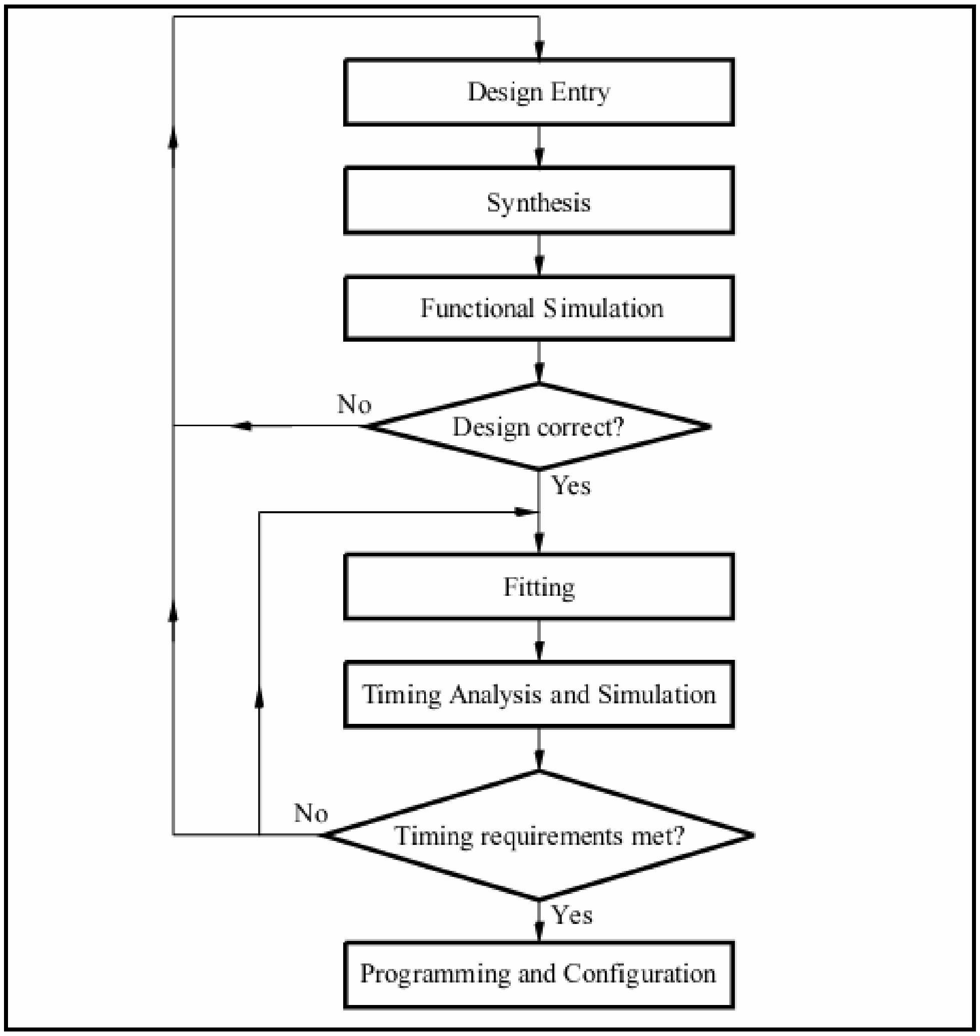


Figure 3.2 CAD Design Flow

3.2.2 Source Code

The first line to write is the top-level name of the module. The one used here is named version_11. Thus, the name of the file must be version_11 too. The ports of the module are listed here. They are not in accordance of input and output but to make it easier to read by programmer, it must be in sequence. Below is the top-level declaration of the system.

```
module version_11
(man_lock,sw,clk,display1,display2,lock,key,disarm_alarm,light);
```

The `man_lock` port is the manual lock representation of the door's manual locking knob which is normally located beside the window pane. When the door is locked, this knob is closed, which gives a Low signal to the system.

The `sw` port is the input pushbuttons for the system. It is four bits, and provides High signal to the system when idle. The switches are active Low. The `clk` port is the heart of the system which is connected to the 25.175MHz crystal on the FPGA board. It provides clocking for the system which trigger states and data movement enable in it.

As for programmer, the ports are `display1` & `display2` provided. They are connected to the seven segment display on the FPGA board. Low signal is required to turn the seven segments on. Two seven segments are required, thus, two ports were declared.

The `lock` port is used to represent the lock status. When the system is locked, the LED will light, indicating the lock status. The same goes to the other ports, `disarm_alarm` & `light`. When the alarm is disarmed, the LED will light, so as the lighting system.

Later, the input and output ports were declared. They are as follow.

```
input clk,key,man_lock;
input [3:0]sw;
```

```
output [7:0]display1,display2;
output lock,disarm_alarm,light;
```

The square brackets are indicating the length of the port. Since there are fout pushbuttons, it will be declared as four bits, ranging from zero to three. Note that in digital world, zero is the starting of every counting number.

The output ports must be declared as registers. The register size must be equal to the port's size at the output ports declaration. The declarations are as follow.

```
reg lock,disarm_alarm,light;
reg [3:0]state;
reg [2:0]swcount;
reg [7:0]display1,display2;
```

In Verilog, some constant can be named and declared as parameter. They are used to make the programmer use the constants easier by having to only call the name, instead of its value. Following are declaration of parameter.

```
parameter zero=8'b00000011,
             one=8'b10011111,
             two=8'b00100101,
             three=8'b00001101,
             four=8'b10011001,
             five=8'b01001001,
             six=8'b01000001,
             seven=8'b00011111,
             eight=8'b00000001,
             nine=8'b00001001;
```

For the parameter shown above, signal zero (Low) is used to light up the seven segments. Following is the diagram of the seven segment used and each segment's labeling. The labeling *a* is at the Most Significant Bit (MSB) of the binary assignments.

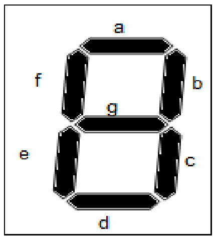


Figure 3.3 Seven Segments Labeling

Every state is numbered. They are as in the follow table. Whenever the states changed, it is visible to the programmer. The numbers are assigned randomly.

Table 3: Display Assignments for Seven Segments

State	First Digit Display	Second Digit Display
Check Key	Six	Six
Wait Button A	One	One
Wait Button B	Two	One
Wait Button C	Two	Three
Wait Unlock	Three	Two
Wait Lock	Three	Three

All the state changes in the source code is based on the clock's edge change. When the clock changes from Low to High, the states will trigger themselves according to the input conditions. The code is as follow.

```
always @ (posedge clk)
```

As for the states, they are changing according to the previous and current conditions. A *case* statement is used to reach the mean. The state machine diagram is as in Figure 3.3. The buttons are checked in sequence if there is no key checked in the earlier state. If the keys are not in sequence, the system will go back to the earlier state and wait for the correct input to be inserted.

The pre-programmed passkey here is A, B and then C. The button D is programmed for cancel, in case of user wants to cancel the input. It is also used to check the initial status of the system too examine the functionality of the system.

For every true condition met, the system will proceed to open the door. The system will give an enable signal to the door unlocking system to unlock the door using the manual lock knob internally. While the internal door system tries to unlock the door, the keyless entry system will wait until the manual lock knob gives the unlock signal, and will automatically open the car's door.

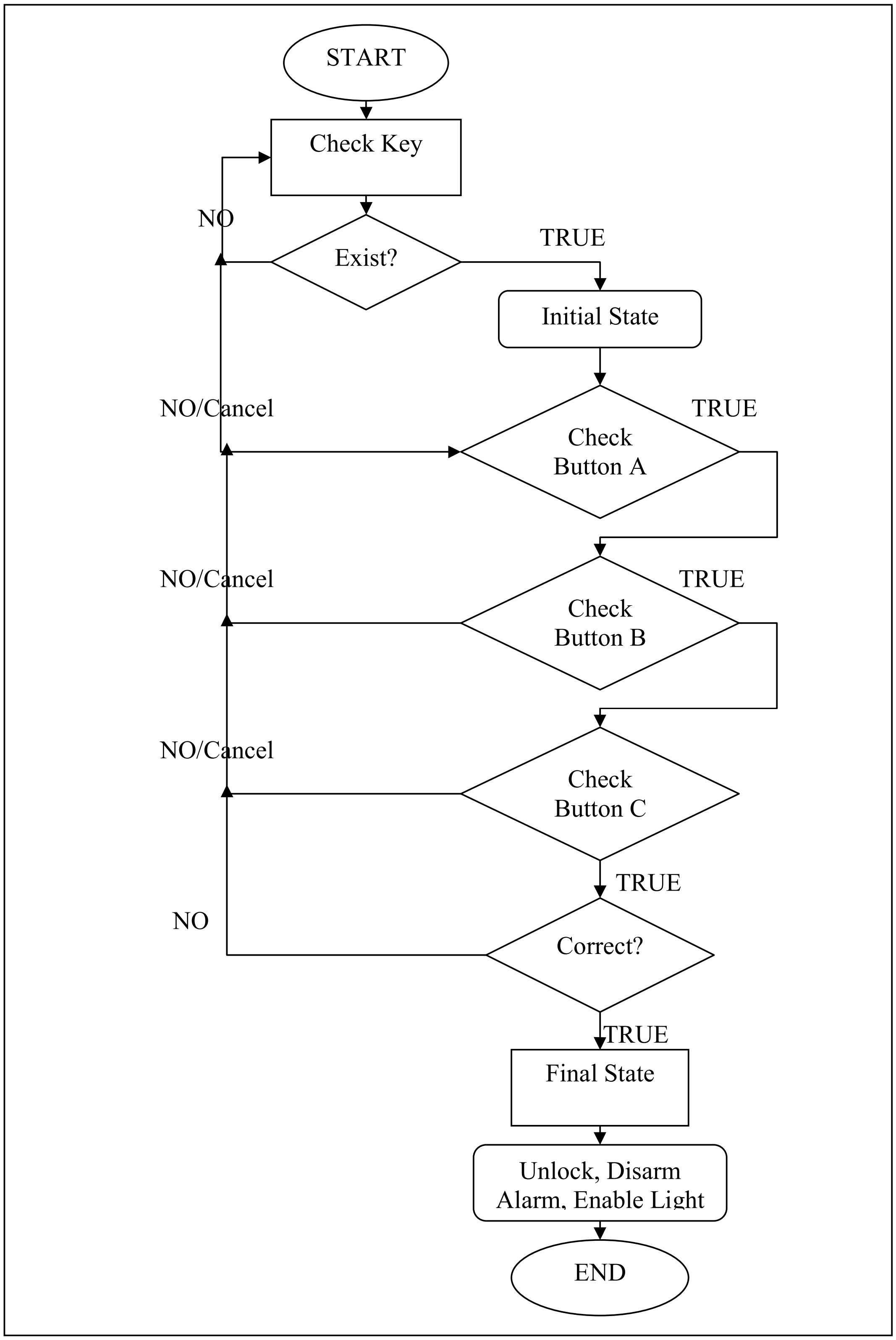
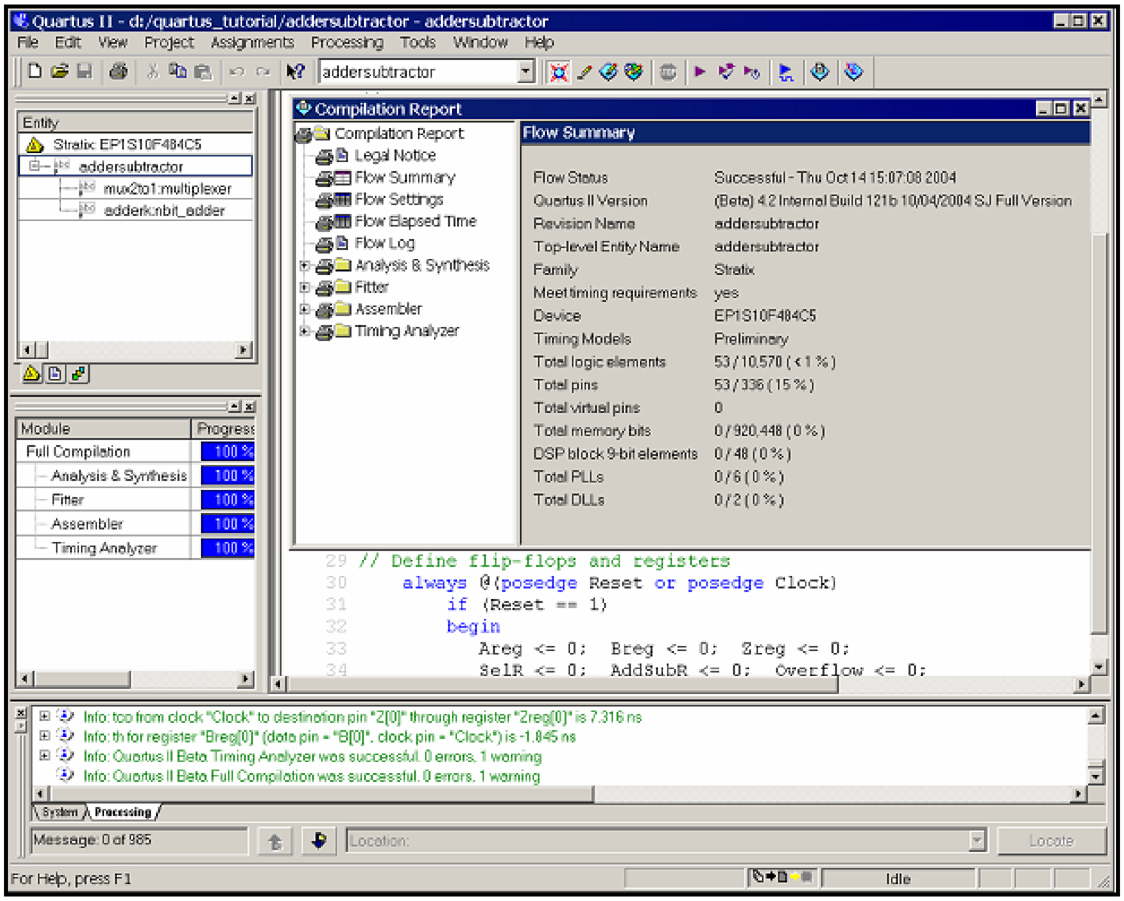


Figure 3.4 State Diagram for version_11.v

3.2.3 Compiling

The Verilog code is processed by several Quartus II tools that analyze the code and generate an implementation of it for the target chip. The tools are controlled by the application program called the *Compiler*.

It can be found by selecting **Processing>Start Compilation**. As the compilation moves through the various stages, its progress is reported in the window on the left side using meter bars. Successful (or unsuccessful) compilation is indicated in a pop-up box. Acknowledge it by clicking **OK**. In the message window, at the bottom of the Figure 3.5, various messages are displayed. In case of errors, there will be appropriate messages given written in red font. They must be cleared by correcting the corresponding errors until there are no more errors.



Module	Progress
Full Compilation	100 %
Analysis & Synthesis	100 %
Fitter	100 %
Assembler	100 %
Timing Analyzer	100 %

Flow Summary

Flow Status	Successful - Thu Oct 14 15:07:08 2004
Quartus II Version	(Beta) 4.2 Internal Build 121b 10/04/2004 SJ Full Version
Revision Name	addersubtractor
Top-level Entity Name	addersubtractor
Family	Stratix
Meet timing requirements	yes
Device	EP1S10F484C5
Timing Models	Preliminary
Total logic elements	53 / 10,570 (< 1 %)
Total pins	53 / 336 (15 %)
Total virtual pins	0
Total memory bits	0 / 920,448 (0 %)
DSP block 9-bit elements	0 / 48 (0 %)
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 2 (0 %)

```
// Define flip-flops and registers
   always @(posedge Reset or posedge Clock)
      if (Reset == 1)
      begin
         Areg <= 0;  Breg <= 0;  Zreg <= 0;
         SelR <= 0;  AddSubR <= 0;  Overflow <= 0;
```

```
Info: tco from clock "Clock" to destination pin "Z[0]" through register "Zreg[0]" is 7.316 ns
Info: th for register "Breg[0]" (data pin = "B[0]", clock pin = "Clock") is -1.845 ns
Info: Quartus II Beta Timing Analyzer was successful. 0 errors, 1 warning
Info: Quartus II Beta Full Compilation was successful. 0 errors, 1 warning
```

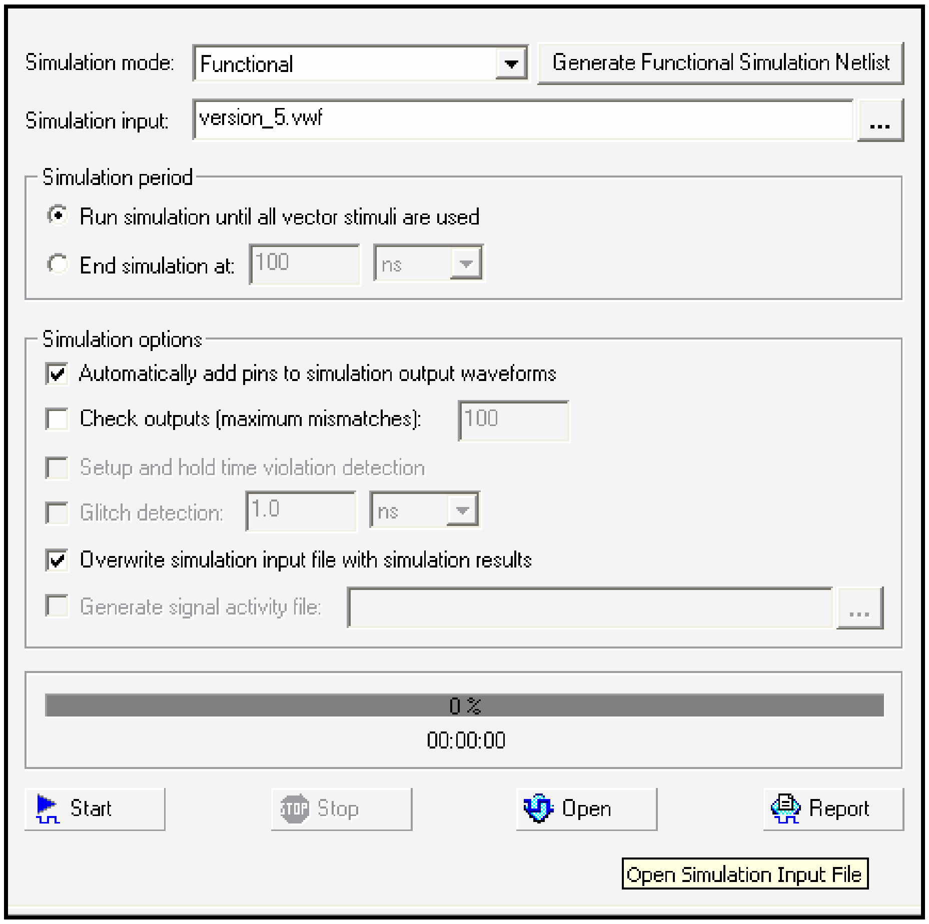
Figure 3.5 Window after a successful compilation

3.2.4 RTL Viewer

Quartus II software includes a tool that cam display a schematic diagram of the designed circuit. The display is at Register Transfer Level of detail, and the tool is called the *RTL Viewer*. Click **Tools>RTL Viewer**. To save the diagram as a picture, go to **File>Export**, and rename the file and select the destination directory. The RTL view of the source code is at Appendix B.

3.2.5 Simulator Tool

Simulator tool is to simulate the source code without downloading the code into the FPGA board. It can be found by selecting **Tools>Simulator Tools**. Choose Functional for simulation mode. Click Open, and a new window as depicted in Figure 3.6 will appear. On Figure 3.6, tick the check box of *Overwrite simulation input file with simulation results.* Simulation timing is left as default and so the other values.


Simulation mode: Functional

Simulation input: version_5.vwf

Simulation period

- [✓] Run simulation until all vector stimuli are used
- [ ] End simulation at: 100 ns

Simulation options

- [✓] Automatically add pins to simulation output waveforms
- [ ] Check outputs (maximum mismatches): 100
- [ ] Setup and hold time violation detection
- [ ] Glitch detection: 1.0 ns
- [✓] Overwrite simulation input file with simulation results
- [ ] Generate signal activity file:

0 %
00:00:00

Generate Functional Simulation Netlist
Start
Stop
Open
Report
Open Simulation Input File

Figure 3.6 Simulation start window

Upon entering Figure 3.7, the left pane on the window is empty. Right click on the empty space to enter the *Node or bus*. Select the input pins and the output pins to list them out on the left pane. Select **OK** after selecting all the desired pins. Save the waveform file by selecting File>Save and name it the same as the module's top-level name. The extension is *.vwf*.

To start simulating, go back to Simulator Tool window. Click on **Generate Functional Simulation Netlist**. Ensure that this box is pressed every time a change is made at the waveform file. Press **Ctrl+I** to start simulation. Upon running simulation, the progress bar on Simulator Tool window will be triggered showing the process done. Click **OK** on the pop-up box that appears after simulation is done.

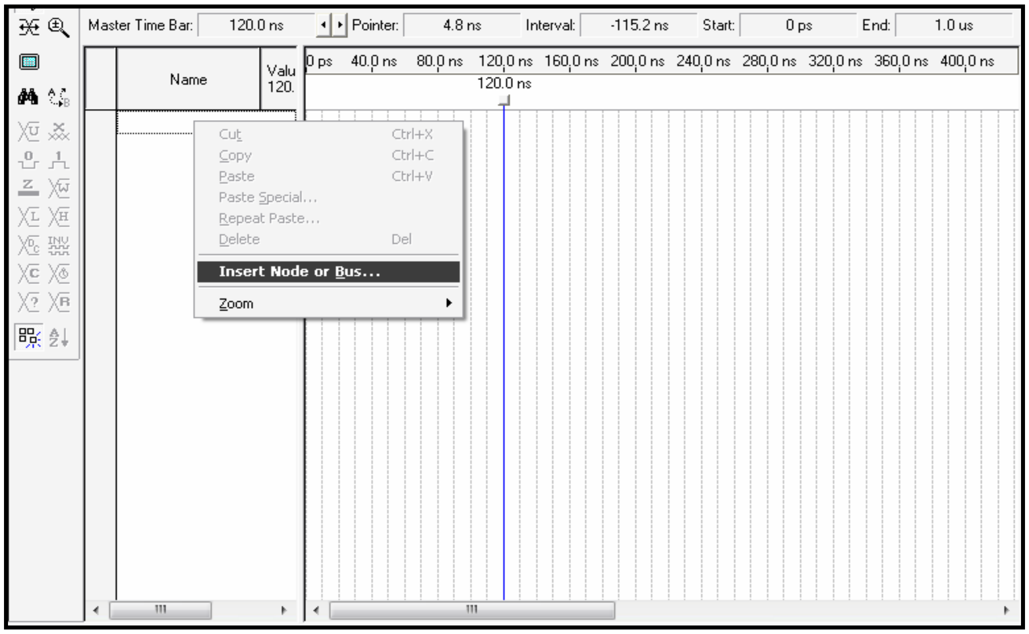


Figure 3.7 Entering pins for the waveform window

After simulation is done, the window as Figure 3.7 will be filled with waveform, in case of no errors. It is shown in Figure 3.8. Note that the inputs were assigned with cases that might occur to the system.

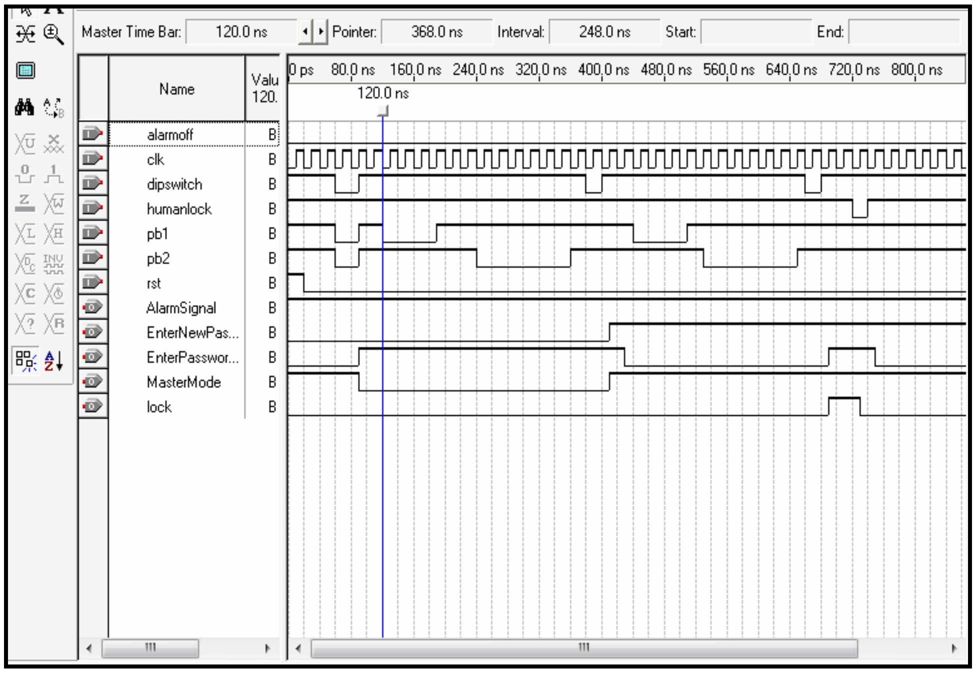
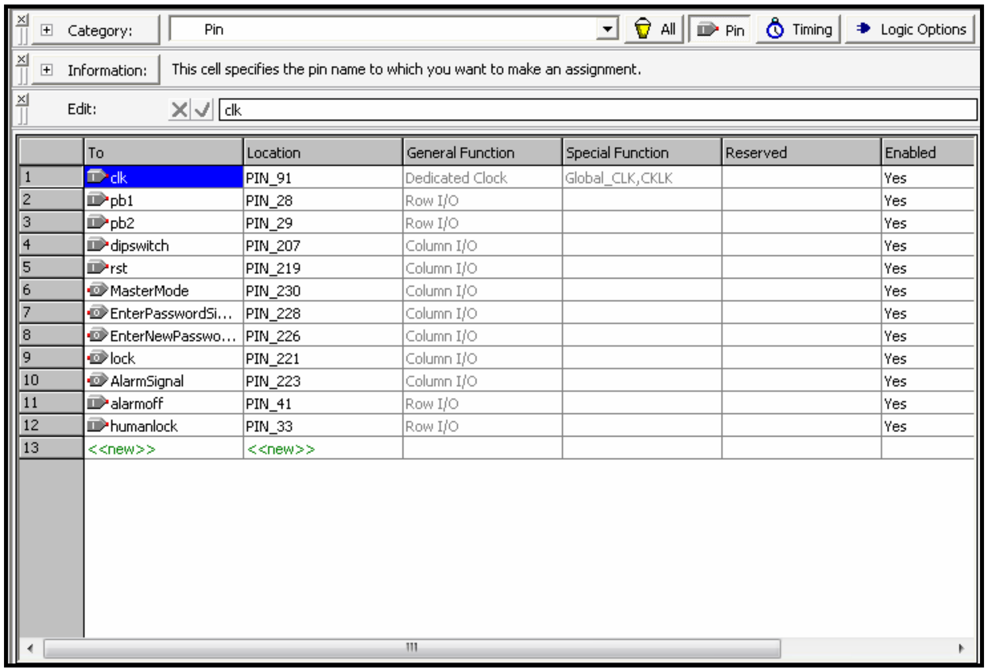


Figure 3.8 Successful simulation waveform windows



This cell specifies the pin name to which you want to make an assignment.

clk

	To	Location	General Function	Special Function	Reserved	Enabled
1	clk	PIN_91	Dedicated Clock	Global_CLK,CKLK		Yes
2	pb1	PIN_28	Row I/O			Yes
3	pb2	PIN_29	Row I/O			Yes
4	dipswitch	PIN_207	Column I/O			Yes
5	rst	PIN_219	Column I/O			Yes
6	MasterMode	PIN_230	Column I/O			Yes
7	EnterPasswordSi...	PIN_228	Column I/O			Yes
8	EnterNewPasswo...	PIN_226	Column I/O			Yes
9	lock	PIN_221	Column I/O			Yes
10	AlarmSignal	PIN_223	Column I/O			Yes
11	alarmoff	PIN_41	Row I/O			Yes
12	humanlock	PIN_33	Row I/O			Yes
13	<<new>>	<<new>>				

Figure 3.9 Pin Assignments window

As the waveform is analyzed and is determined to be correct, the next step is to download the source code into the chip. This is done by accessing **Assignments>Pins**. A new window will appear as depicted in Figure 3.9.

3.3 Using the FPGA Board

The board used for the study is from University Program UP2 Education Kit. The board comes with two chips:

i. An EPF10K70 device in a 240-pin power quad flat pack (RQFP) package
ii. An EPM7128S device in an 84-pin plastic J-lead chip carrier (PLCC) package.

The first chip is selected for this study for it has 3744 logic elements and nine embedded logic array, compared to the other one which has only 2,500 gates. It is due to safety margin in case the source code implies a large number of logic elements. Thus, insufficient number of logic elements will not be encountered. The board used is in Figure 3.10. Note that the rightmost chip is the FLEX10K while the other one is the EPM7128S chip.

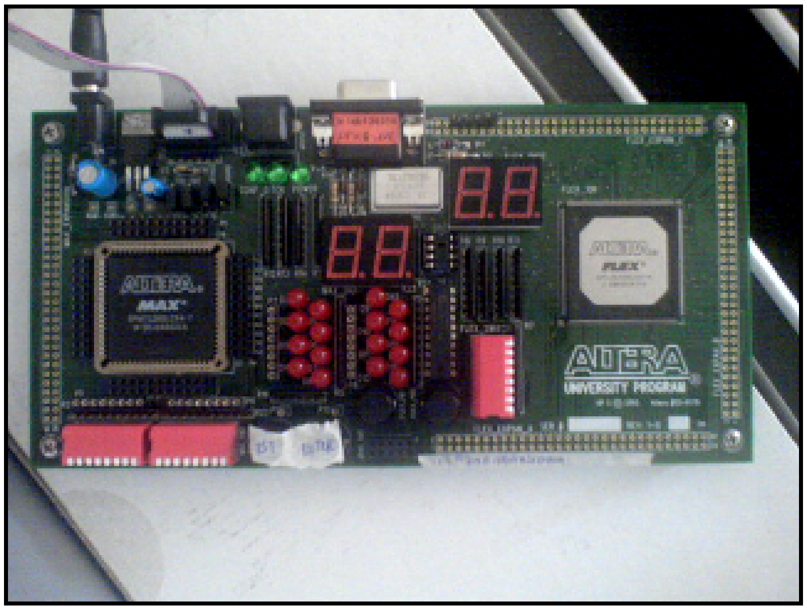


Figure 3.10 Altera FPGA Board.

To interface the board with computer, ByteBlasterII cable is used. It is a cable with LPT1 port at an end of the cable and the other end is JTAG port. LPT1 port is the old printer port and usually is colored in pink. To use the cable, one must install the driver of the cable into the computer since it is not installed upon installing Quartus II software.

The driver is located in the Quartus II installed directory, within the folder **win2000**, under the file name of *win2000.inf*. Install the driver manually, through **Computer Management**. Restart the computer after installation.

3.3.1 Configuring the FPGA board

There are four jumpers on the FPGA board. They are TDI, TDO, DEVICE and BOARD. To use the EPF10K70 chip, set the jumpers as Figure 3.11.

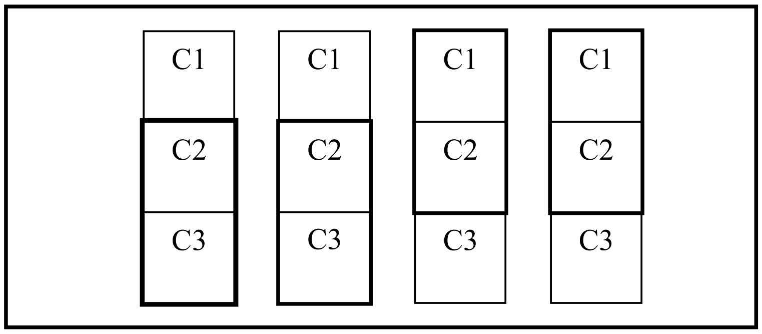


Figure 3.11 Jumper Settings for Configuring FLEX10K70

3.3.2 Setting up Programming Hardware in Quartus II Software

Choose **Programmer** from the Tools menu. The Programmer window will open. Click the **Hardware Setup...** button to open the Hardware Setup window. Note that:

i. The selected programming hardware is identified as *Currently Selected Hardware.*

ii. Programming hardware that is already set up appears in the *Available hardware* items window.

Click the **Add Hardware** button to open the *Add Hardware* window if the programming hardware you would like to use is not listed in the *Available hardware* items window.

a. Select the appropriate programming cable or programming hardware from the *Hardware Type* list.
b. Select the appropriate port and baud rate if necessary.
c. Click **OK**.

Select the programming hardware you would like to use by choosing it in the Available hardware items list. Click **Close**. Now, the program is ready to download the source code into the board.

3.3.4 Downloading Source Code

Still in the Programming window, make sure the file *.sof* existed in the list. Connect the ByteBlasterII cable to the computer printer port and the other to the JTAG port on then board. The connection is as in Figure 3.12. Tick on the **Program/Configure** checkbox.

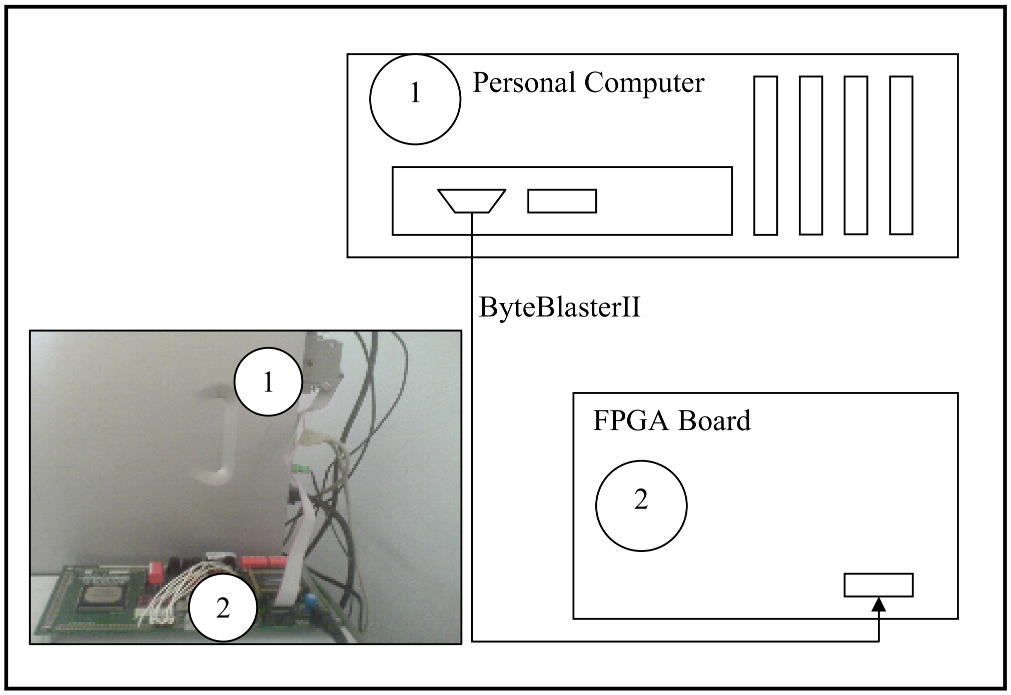


Figure 3.12 Connected ByteBlasterII cable

As the power is turned on, click on Start to start downloading the source code. During the progress, the progress bar inside the Programming window will show the process done. At the same time, on the board, the **TCK** LED will light up to show that the chip is being programmed. After the programming is done, the LED will turn off while the **CONF_D** LED will light up. This is to show that the chip has been successfully configured.

3.4 Hardware Simulation Setup

Hardware simulation is to simulate the source code through hardware mean, the FPGA board. Following is the table for the FPGA board's connection pins.

Table 3.1 Pin Assignments for the FPGA board

Pin Names	**FLEX Pin**	**Function**	**Board Pin**
clk	PIN_91	Global Clock	N/A
Sw[0]	PIN_29	Push Button D	N/A
Sw[1]	PIN_28	Push Button C	N/A
Sw[2]	PIN_48	Push Button B	Hole 48 (A)
Sw[3]	PIN_45	Push Button A	Hole 45 (A)
Man_lock	PIN_34	Manual Lock Knob	FLEX Switch 8
Key	PIN_33	Key	FLEX Switch 7
lock	PIN_162	Lock Trigger	Hole 55 (B)
Disarm_alarm	PIN_159	Disarm Alarm	Hole 53 (B)
Light	PIN_157	Light Trigger	Hole 51 (B)

The pins are connected to their respective signals. The input pins were connected to the switches while the outputs were shown using LED signals. Note that the on-board LEDs are Active Low, thus supplying Low signals to them will make them light up. The following table shows their connected ends.

Table 3.2 Pins Connection on the FPGA board

Function	**Connection**
Dedicated Clock	Internal Clock
Push Button A	MAX_PB1
Push Button B	MAX_PB2
Push Button C	FLEX_PB1
Push Button D	FLEX_PB2
Light	LED
Disarm ALarm	LED
Lock Trigger	LED

Shown in Figure 3.13 is the final hardware wiring for the keyless entry system. The expansion holes on the FPGA board is connected to the wires using female jumper headers, and so at the veroboard. They are not soldered to the FPGA board anyhow, but the legs are bent to keep a firm hold to the board.

The LEDs on the verobod are connected so they are active Low. They anode part of the LEDs were connected to 5V supply. Their cathodes were in series with 1kΩ resistor. The resistor's ends were connected to the FPGA board.

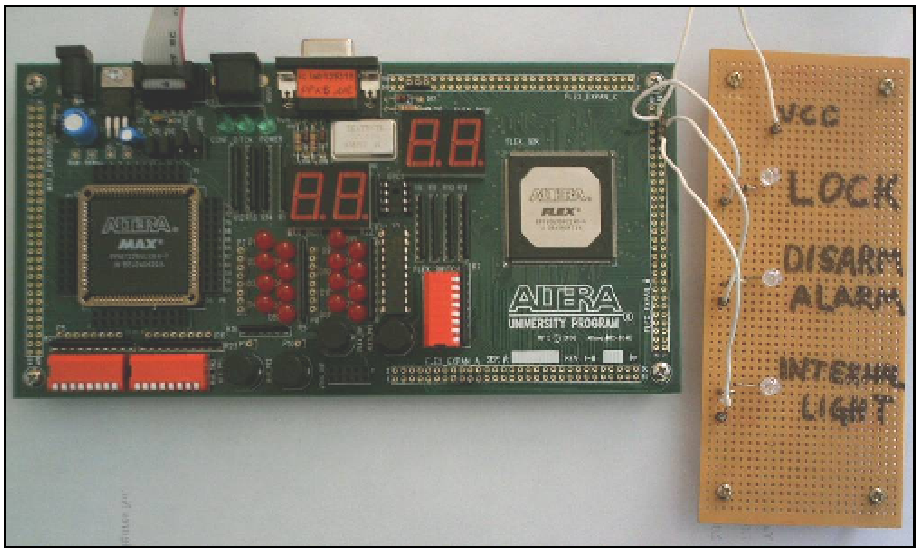


Figure 3.13 Hardware Wiring

3.5 Summary

This important section provides the in-depth details of the auto keyless entry system design from the source code to the hardware simulation. The application of how the software is used explained here. The software explained was Altera Quartus II 5.0 including the sub functions, such as Simulator Tool, Programming and RTL Viewer. Graphical display of the setting was shown, as well as the setup for programming. Finally, pin configurations for the system are also explained in detail.