DEVELOPMENT OF AN FPGA BASED LOW POWER MESSAGE DISPLAYING SYSTEM USING SCANNING TECHNIQUE

by

S.M. Tofayel Ahmad

MASTER OF ENGINEERING IN INFORMATION AND COMMUNICATION TECHNOLOGY

Institute of Information and Communication Technology

BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY

October, 2010

The project titled "Development of an FPGA based low power message displaying system using scanning technique" submitted by S.M. Tofayel Ahmad, Roll No. M04053125, Session April, 2005 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Master of Engineering (ICT) held on 9th October, 2010.

BOARD OF EXAMINERS

1.	Dr. Md. Liakot Ali Associate Professor Institute of Information and Communication Technology BUET, Dhaka-1000.	Chairman
2.	Dr. S. M. Lutful Kabir Professor and Director Institute of Information and Communication Technology BUET, Dhaka-1000.	Member
3.	Dr. Md. Abul Kashem Mia Professor and Associate Director Institute of Information and Communication Technology BUET, Dhaka-1000.	Member

Candidate's Declaration

It is hereby declared that this report or any part of it has not been submitted elsewhere for the award of any degree or diploma.

_____

S.M. Tofayel Ahmad

Dedicated

to

My Parents

Table of Contents

List of Figures

Title **Page No.**

List of Tables

LIST OF ABBREVIATIONS, SYMBOLS AND TECHNICAL TERMS

ALUT	Adaptive Look-UP Table
ASIC	Application Specific Integrated Circuit
CLB	Configurable Logic Block
CPLD	Computer Programmable Logic Device
e.g.	For example
EDA	Electronic Design Automation
FD, FDR, FDE	Various flip-flop primitives in Xilinx FPGAs
FIR	Finite Impulse Response
FIPS	Federal Information Processing Standards
FIPS PUB	Federal Information Processing Standards Publication
FPGA	Field Programmable Gate Array
GF	Galois Field
HDL	Hardware Description Language
LUT	Look-UP Table
µC	Microcontroller
MAC	Multiply and Accumulate
NIST	National Institute of Standards and Technology
NoC	Network on Chip
NRE	Non Recurring Engineering
OCN	On Chip Network
PCB	Printed Circuit Board
RTL	Register Transfer Level
SRL16	A 16-bit Shift Register in Xilinx FPGAs
TFT	Thin-Film Transistor
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration
Xor	Exclusive-OR

Acknowledgement

First of all I would like to thank the almighty Allah for giving me the opportunity to conduct this project. I would like to express my sincere thanks to my research project supervisor, Dr. Md. Liakot Ali, for giving me the opportunity to conduct this project. Without his ever helping personalities, this project would not have got the success.

I would like to convey my thanks to Professor Dr. S. M. Lutful Kabir, Director, IICT, BUET and Professor Dr. Abul Kashem Mia, Associate Director, IICT, BUET. Their motivation and inspiration gave me the courage to do this work.

I gratefully acknowledge the restless support and advice of my fellow classmate and friend Mohammad Mohidur Rahman Khan and Md. Bozlul Karim during the design and implementation phase of this project. My special thanks to all the teachers, students and staffs of IICT, BUET.

I would like to thank all of my friends and family members for their continuous support and inspiration throughout the whole period of this undertaking.

Abstract

Power efficient solution is an essential criteria for any portable electronic system. Electronic display is now the most conventional means of presenting information all over the world. Due to its attractive brightness and simple operability, Seven Segment Display is one of the most popular medium of presenting different types of information to the mass people. The display elements in an embedded system usually consume the major portion of the total power required to run the whole system. When large amount of display elements are used, power dissipation issue becomes more acute. A microcontroller based system using scanning technique for low power message display was developed by other researcher. The system was burdened with many limitations such as physical security, no. of pins, portability and slow processing speed. To overcome the limitations an FPGA based embedded system implementing scanning technique is proposed in this project. The FPGA based intelligent controller scans all the display elements continuously at a certain speed to ensure only one display unit is ‘ON’ and others are ‘OFF' at a given time but human eye cannot detect it due to speedy scanning of the controller. Here an FPGA based embedded system for ‘Muslim Calendar’ containing date, time and prayer times for five salat has been developed using 30 seven segment display units. Experimental result shows that dynamic scanning makes the current consumption 88% less and power requirement 82% less than that of the static display. It also shows that the proposed FPGA based system consumes 55% less current and 22% less power than the previous microcontroller based system. The same scanning technique can also be applied to drive the display system in other applications.

Chapter 1
Introduction

1.1 Introduction

An electronic display is a device which is used for presentation of text and images for visual reception, without producing a permanent record. The use of electronic displays for presentation of graphs, symbols, alphanumeric, and still or video pictures has doubled every several years, in parallel with the rapid expansion of microelectronics [1-3]. Electronic displays have largely replaced traditional mechanical devices, counters, galvanometers, and to a degree, hardcopy (paper) means for presenting information. This change is due to the increased use of computers, microprocessors, very large-scale integration (VLSI) electronics, and digital mass memories. Electronic display-we see it everywhere, in computers, watches, DVD players and many other electronic devices to display numeric and alphabetic characters [4-8]. It is also used for showing messages in digital calendars, billboards, shopping malls, airports, stadiums and many other places. It is a commonly used and efficient way of displaying information. Although electronic display is widely used, power consumption is still an important issue [9-12].

Portable embedded system is a vision of this day. It is usually a battery operated electronic products. Low power dissipation is a desirable and/or even essential in these equipments to have reasonable battery life and weight of the system. In an electronic system having display elements, the major part of the total power is consumed by the display elements. If a large amount of display units are used in a system then the power requirement to run the system will also increase proportionately. In that case extra cooling arrangements may be required to keep the system workable. Researchers have proposed plasma screen or organic light-emitting devices (OLED) to reduce power consumption for display element [13-14]. A microcontroller (μC) based system using scanning technique is proposed for low power message display by previous researcher [15]. In that project a Muslim Calender for displaying date, time and five prayer time

for salat was developed using seven segment display. A microcontroller based system is usually burdened with the problem of limited number of pins, limited physical security and slow processing speed. Seven segment display with integrated controller and other circuitry are proposed to reduce the burden from main controller [16-17]. But in this case cost will be increased and power will be consumed by display control circuitry. Now the Field Programmable Gate Array (FPGA) technology outperforms microcontroller technology and offers improved performance, reduced power consumption, reduced system size, more physical security and shorter time to market. Besides this FPGA has parallel processing feature to enhance the speed performance. So it is widely used in many applications. In this research project, FPGA has been used replacing the microcontroller technology to enhance the performance of the system as shown in the literature [15].

1.2 Objectives with Specific Aims and Possible Outcome

The major objectives of the project are as follows:

i) To study different types of display devices,
ii) To design the display controller using Verilog HDL,
iii) To simulate the design using EDA tools,
iv) To implement the system using FPGA and display units.
v) To conduct test for measurements of current and power and then compare with the previous microcontroller based system.

Here as a test case a Muslim Calendar will be developed where the following information will be displayed using seven segment display:

a. Date
b. Time and
c. Prayer times of five Salah for any given date.

1.3 Organization of the Project

Chapter 1 of this report starts with impact of power consumed by display units in an electronic system followed by objectives and organization of this report.

In Chapter 2 of this report, issues from all aspects in developing the proposed system have been reviewed.

Methodology to develop the proposed system has been discussed elaborately in chapter 3.
In chapter 4 results and discussion on the proposed research have been discussed.

In the final chapter (Chapter 5) conclusion and recommendation for future works have been stated.

The project report ends with an appendix that contains the program code of the system.

Chapter 2

FPGA and Display Systems

2.1 Display System

A display device is a device for presentation of information for visual reception, acquired, stored, or transmitted in various forms. When the input information is supplied as an electrical signal, the display is called "electronic display". An electronic component is used to convert electrical signals into visual imagery in real time suitable for direct interpretation by a human operator. It serves as the visual interface between human and machine. The visual imagery is processed, composed, and optimized for easy interpretation and minimum reading error.

2.2 Basic Types of Electronic Display

There are two basic types of displays. They are:

- Analog Electronic Display
- Digital Electronic Display

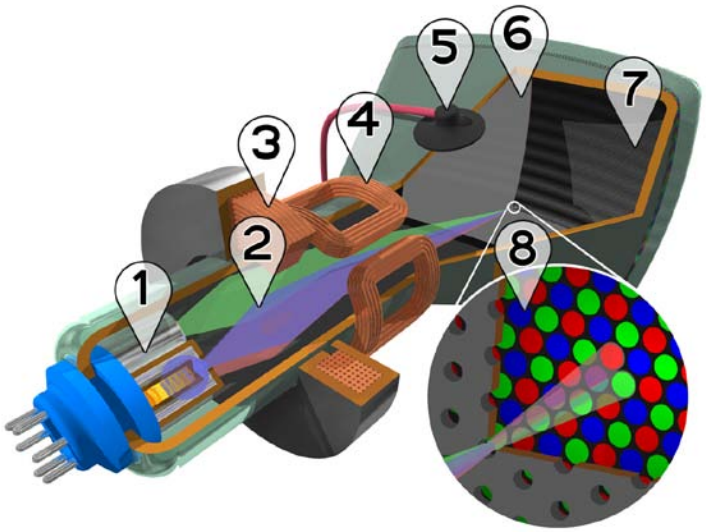
The following subsections briefly describe different types of display systems that are commonly used.

2.2.1 Commonly Used Analog Electronic Display

The cathode ray tube (CRT) display is the most commonly used analog electronic display. The cathode ray tube (CRT) as shown in Figure 2.1 is a vacuum tube containing an electron gun (a source of electrons) and a fluorescent screen, with internal or external means to accelerate and deflect the electron beam, used to form images in the form of light emitted from the fluorescent screen. The image may represent electrical waveforms (oscilloscope), pictures (television, computer monitor), radar targets and others.

The single electron beam can be processed in such a way as to display moving pictures in natural colors.

It consists of the following units.



1. Electron guns
2. Electron beams
3. Focusing coils
4. Deflection coils
5. Anode connection
6. Mask for separating beams for red, green, and blue part of displayed image
7. Phosphor layer with red, green, and blue zones
8. Close-up of the phosphor-coated inner side of the screen

Figure 2.1 Cathode Ray Tube (CRT) Display

2.2.2 Commonly Used Digital Electronic Display

There are many examples of digital electronic displays. Among them light emitting diode (LED), liquid crystal display (LCD) and seven segment display are most common.

a) Light Emitting Diode (LED)

A light-emitting diode is a semiconductor diode that emits incoherent narrow-spectrum light when electrically biased in the forward direction of the p-n junction, as in the common LED circuit. This effect is a form of electroluminescence.

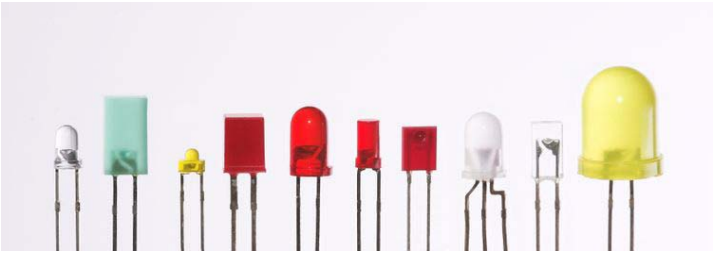


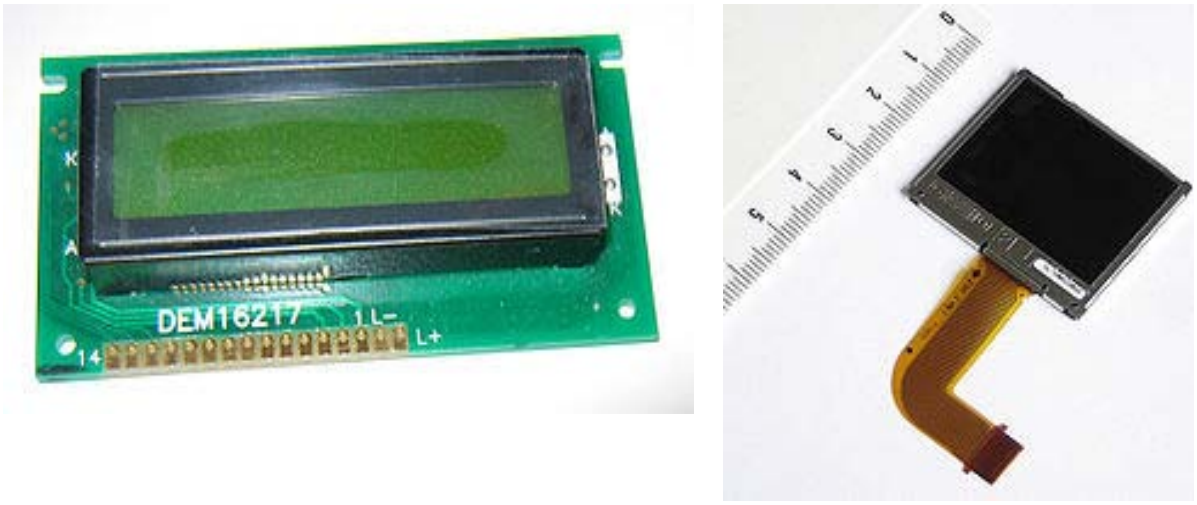
Figure 2.2 Different types of Light Emitting Diode (LED)

Figure 2.2 shows different types of LED. Generally an LED has two legs. One is cathode another is anode. An LED is usually a small area light source, often with optics added to the chip to shape its radiation pattern. LEDs are often used as small indicator lights in electronic devices and increasingly in higher power applications such as flashlights and area lighting. The color of the emitted light depends on the composition and condition of the semiconducting material used, and can be infrared, visible, or ultraviolet. LEDs can also be used as a regular household light source.

b) Liquid Crystal Display (LCD)

A liquid crystal display (LCD) is a thin, flat electronic visual display that uses the light modulating properties of liquid crystals (LCs). LCs do not emit light directly. A liquid crystal display (LCD) is a thin, flat display device made up of any number of color or monochrome pixels arrayed in front of a light source or reflector. A low-power flat-panel display used in many laptop computers, calculators and digital watches, made up of a liquid crystal that is sandwiched between layers of glass or plastic and becomes opaque when electric current passes through it. The contrast between the opaque and transparent areas forms visible characters.

LCD is a display technology that uses rod-shaped molecules (liquid crystals) that flow like liquid and bend light. When not energized, the crystals direct light through two polarizing filters, allowing a natural background color to show. When energized, they redirect the light to be absorbed in one of the polarizers, causing the dark appearance of crossed polarizers to show. The more the molecules are twisted, the better the contrast and viewing angle. They are used in a wide range of applications including: computer monitors, television, instrument panels, aircraft cockpit displays, signage, etc. They are common in consumer devices such as video players, gaming devices, clocks, watches, calculators, and telephones. They are usually more compact, lightweight, portable, less expensive, more reliable, and easier on the eyes. They are available in a wider range of screen sizes than CRT and plasma displays, and since they do not use phosphors, they cannot suffer image burn-in.



(a) (b)

Figure: 2.3 LCD Display

Figure 2.3(a) is a general purpose alphanumeric LCD, with two lines of 16 characters and Figure 2.3(b) is a Casio 1.8 in colour TFT liquid crystal display which equips the Sony Cyber-shot DSC-P93A digital compact cameras.

c) Seven Segment Display

A seven-segment display, or seven-segment indicator, is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot-matrix displays. Seven-segment displays are widely used in digital clocks, electronic meters, and other electronic devices for displaying numerical information. Each digit is formed by selective illumination of up to seven separately addressable bars. A seven segment display, as its name indicates, is composed of seven elements. Individually ‘ON’ or ‘OFF’, they can be combined to produce simplified representations of the English-Arabic numerals. Often the seven segments are arranged in an oblique, or italic, arrangement, which aids readability. Figure 2.4 shows a typical seven segment LED display component with decimal point. It has eight pins. The 3rd and 8th pin are reserve for common cathode /anode.

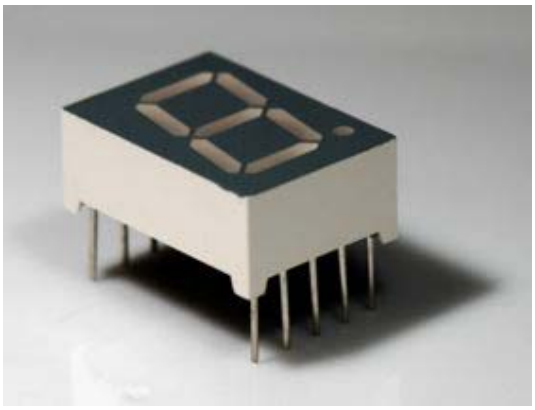


Figure 2.4 Seven Segment Display IC

The seven segments are arranged as a rectangle of two vertical segments on each side with one horizontal segment on the top and bottom. Additionally, the seventh segment bisects the rectangle horizontally. There are also fourteen-segment displays and sixteen-segment displays (for full alphanumeric); however, these have mostly been replaced by dot-matrix displays. The seven segment display layout is shown in following Figure 2.5.

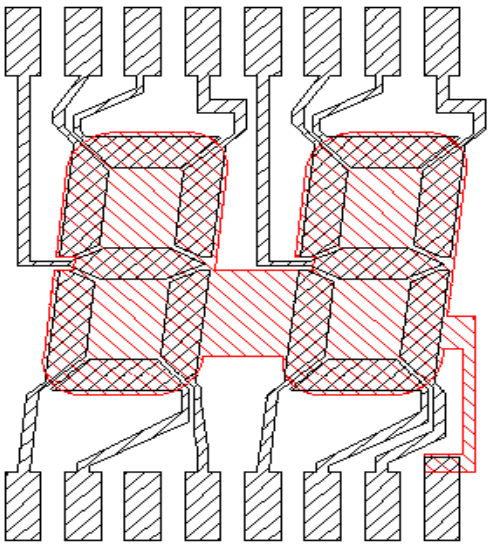


Figure 2.5 Seven Segment Display Layout

The individual segments of a seven-segment display referred to by the letters A to G is shown by Figure 2.6.

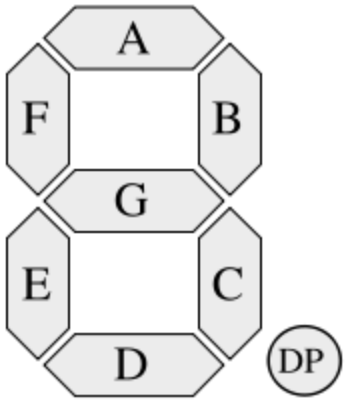


Figure 2.6 Segments of a Seven Segment Display

There is an optional DP decimal point (an "eighth segment") is used for the display of non-integer numbers.

Seven-segment displays are made with seven LEDs. They are aligned in a pattern that represents the number ‘8’ (Figure 2.6). It can display 0-9 by turning different LEDs ‘ON or ‘OFF’ (Figure 2.7).

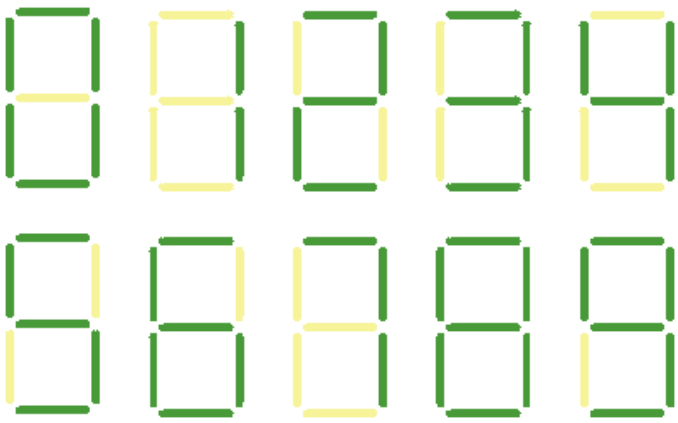


Figure 2.7 Displaying 0 to 9

2.3 Field-Programmable Gate Array (FPGA)

A Field-Programmable Gate Array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, partial re-configuration of the portion of the design and the low non-recurring engineering costs relative to an ASIC design (not withstanding the generally higher unit cost) offer advantages for many applications.

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"- somewhat like a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory. The Altera FPGA’s Chip is shown in Figure 2.7. It is FLEX10K device chip and built in UP2 board.

Figure 2.8 Altera Flex FPGA Chip

In addition to digital functions, some FPGAs have analog features. The most common analog feature is programmable slew rate and drive strength on each output pin, allowing the

engineer to set slow rates on lightly loaded pins that would otherwise ring unacceptably, and to set stronger, faster rates on heavily loaded pins on high-speed channels that would otherwise run too slow.

2.3.1 Major Manufacturers of FPGA

Xilinx and Altera are the current FPGA market leaders and long-time industry rivals. Together, they control over 80 percent of the market, with Xilinx alone representing over 50 percent. Both Xilinx and Altera provide free Windows and Linux design software. Other competitors include Lattice Semiconductor (SRAM based with integrated configuration Flash, instant-on, low power, live reconfiguration), Actel (antifuse, flash-based, mixed-signal), SiliconBlue Technologies (low power), Achronix (RAM based, 1.5 GHz fabric speed), and QuickLogic (handheld focused CSSP, no general purpose FPGAs). In March 2010, two FPGA companies that had previously worked secretly, announced their new FPGA technology: Tabula and Tier Logic; Tabula uses time-multiplexing and Tier Logic 3D-FPGA technology.

2.3.2 Architecture of FPGA

The most common FPGA architecture consists of an array of logic blocks (called Configurable Logic Block, CLB, or Logic Array Block, LAB, depending on vendor), I/O pads, and routing channels. Generally, all the routing channels have the same width (number of wires). Multiple I/O pads may fit into the height of one row or the width of one column in the array.

An application circuit must be mapped into an FPGA with adequate resources. While the number of CLBs/LABs and I/Os required is easily determined from the design, the number of routing tracks needed may vary considerably even among designs with the same amount of logic. For example, a crossbar switch requires much more routing than a systolic array with the same gate count. Since unused routing tracks increase the cost (and decrease the performance) of the part without providing any benefit, FPGA manufacturers try to provide just enough tracks so that most designs that will fit in terms of LUTs and IOs can be routed.

This is determined by estimates such as those derived from Rent's rule or by experiments with existing designs.

Figure 2.9 shows a logic block (CLB or LAB) consists of a few logical cells (called ALM, LE, Slice etc). A typical cell consists of a 4-input Lookup table (LUT), a Full adder (FA) and a D-type flip-flop, as shown in Figure 2.9. The LUT are in this figure split into two 3-input LUTs. In normal mode those are combined into a 4-input LUT through the left mux. In arithmetic mode, their outputs are fed to the FA. The selection of mode are programmed into the middle mux. The output can be either synchronous or asynchronous, depending on the programming of the mux to the right, in the figure example. In practice, entire or parts of the FA are put as functions into the LUTs in order to save space.

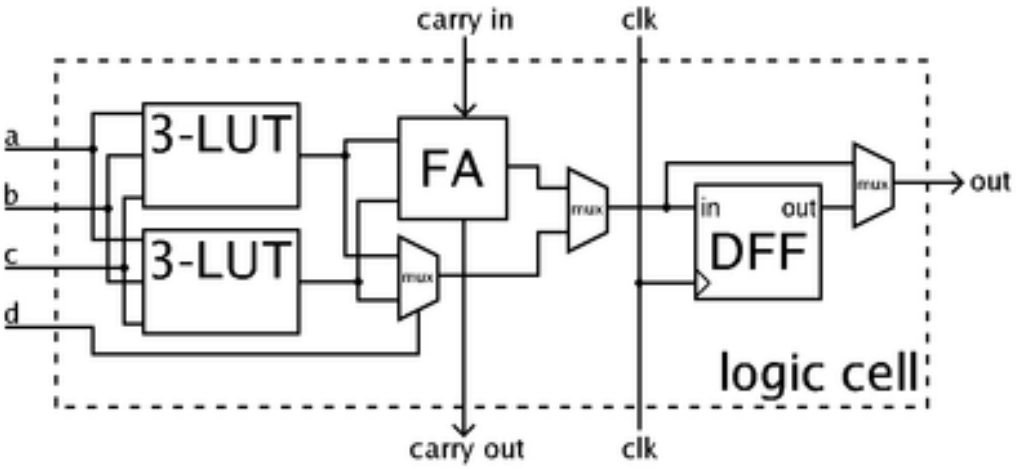


Figure 2.9 Simplified Example Illustration of a Logic Cell

ALMs and Slices usually contains 2 or 4 structures similar to the example figure, with some shared signals. CLBs/LABs typically contains a few ALMs/LEs/Slices. In recent years, manufacturers have started moving to 6-input LUTs in their high performance parts, claiming increased performance. Since clock signals (and often other high-fanout signals) are normally routed via special-purpose dedicated routing networks in commercial FPGAs, they and other signals are separately managed. For this example architecture, the locations of the FPGA logic block pins are shown in Figure 2.10. Each input is accessible from one side of the logic block, while the output pin can connect to routing wires in both the channel to the right and the channel below the logic block.

Each logic block output pin can connect to any of the wiring segments in the channels adjacent to it.

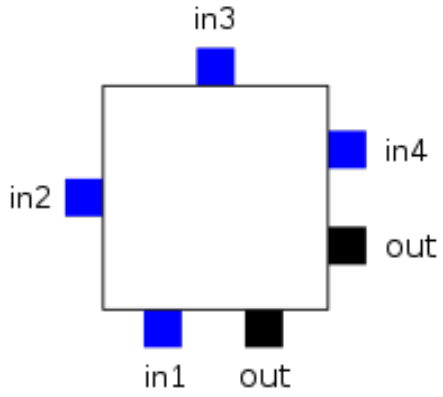


Figure 2.10 Logic Block Pin Locations

Similarly, an I/O pad can connect to any one of the wiring segments in the channel adjacent to it. For example, an I/O pad at the top of the chip can connect to any of the W wires (where W is the channel width) in the horizontal channel immediately below it.

Generally, the FPGA routing is unsegmented. That is, each wiring segment spans only one logic block before it terminates in a switch box. By turning on some of the programmable switches within a switch box, longer paths can be constructed. For higher speed interconnect, some FPGA architectures use longer routing lines that span multiple logic blocks.

Whenever a vertical and a horizontal channel intersect, there is a switch box. Switch box is shown in Figure 2.11. In this architecture, when a wire enters a switch box, there are three programmable switches that allow it to connect to three other wires in adjacent channel segments. The pattern, or topology, of switches used in this architecture is the planar or domain-based switch box topology. In this switch box topology, a wire in track number one connects only to wires in track number one in adjacent channel segments, wires in track number 2 connect only to other wires in track number 2 and so on. Figure 2.11 illustrates the switch box topology.

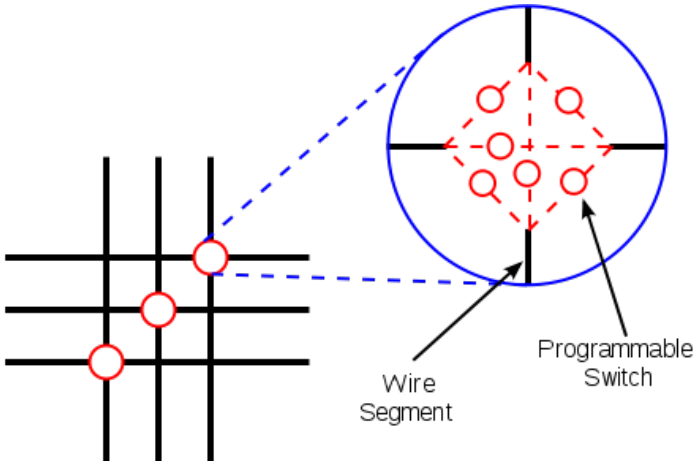


Figure 2.11 Switch box topology

Modern FPGA families expand upon the above capabilities to include higher level functionality fixed into the silicon. Having these common functions embedded into the silicon reduces the area required and gives those functions increased speed compared to building them from primitives. Examples of these include multipliers, generic DSP blocks, embedded processors, high speed IO logic and embedded memories.

FPGAs are also widely used for systems validation including pre-silicon validation, post-silicon validation, and firmware development. This allows chip companies to validate their design before the chip is produced in the factory, reducing the time-to-market.

2.3.3 Common Features of FPGA

- Large number of gates available.
- FPGAs typically have tens of thousands to several million logic gates which are greater than CPLD and PAL.
- Some provisions for logic more flexible than sum-of-product expressions, including complicated feedback paths between macro cells, and specialized logic for implementing various commonly-used functions, such as integer arithmetic.
- FPGA products also offer models with embedded configuration memory.
- FPGAs are internally based on Look-up tables (LUTs).

2.3.4 Advantages of FPGAs

1. FPGA is a customized IC. It can implement most digital logic.
2. A micro-controller won't have enough processing power in most tele-com applications, especially the data path. In those applications, one needs direct hardwired logic to process the packets. The only choices are FPGA or ASIC
3. FPGA is excellent to implement the glue logic of the system of different chips. It really glues all of them together.
4. FPGAs can be very powerfull, much more powerfull than any microcontroller.
5. The real advantage of FPGA's are the ease of prototyping, time to market and no NRE and Low volume to use.
6. It is reconfigurable and strongly flexible. The limitation of microcontroller is not existed.
7. It is possible to customize the process fully without wasting the resources as what we do in controllers, also that FPGA has ability to carry out very big and complex process.
8. It is possible to make a special purpose processor using FPGA. Also it can operate at very high clock speeds than to controllers.
9. It is not necessary to think about layout design, it is enough to repair the code only.
10. FPGA can be field programmed. But if the power is turn off, the logic will be lost. If we need change some logic in our product we can easily change it.
11. The project on high speed communication protocol can't be implemented only using microcontroller. In such case, FPGA and ASIC are needed.
12. FPGA is parallel processing. So it is widely used in high-speed and real-time processing field. That means speed can be very fast, and multiple control loops can run on a single FPGA device at different rates.
13. FPGA design tools are increasingly available, allowing embedded control system designers to more quickly create and adapt FPGA hardware.
14. A field programmable gate array (FPGA) contains a matrix of reconfigurable gate array logic circuitry that, when configured, is connected in a way that creates a hardware implementation of a software application. Increasingly sophisticated tools

are enabling embedded control system designers to more quickly create and more easily adapt FPGA-based applications.

15. Unlike processors, FPGAs use dedicated hardware for processing logic and do not have an operating system.
16. Because the processing paths are parallel, different operations do not have to compete for the same processing resources.
17. The reconfigurability of FPGAs can provide designers with almost limitless flexibility.
18. In manufacturing and automation contexts, FPGAs are well-suited for use in robotics and machine tool applications, as well as for fan, pump, compressor and conveyor control.
19. FPGA devices are very attractive for realizing modern, complex digital controller designs.

2.3.5 Applications of FPGA

Applications of FPGAs include digital signal processing, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection and a growing range of other areas.

FPGAs originally began as competitors to CPLDs and competed in a similar space, that of glue logic for PCBs. As their size, capabilities, and speed increased, they began to take over larger and larger functions to the state where some are now marketed as full systems on chips (SoC). Particularly with the introduction of dedicated multipliers into FPGA architectures in the late 1990s, applications, which had traditionally been the sole reserve of DSPs, began to incorporate FPGAs instead.

FPGAs are increasingly used in conventional high performance computing applications where computational kernels such as FFT or Convolution are performed on the FPGA instead of a microprocessor.

The inherent parallelism of the logic resources on an FPGA allows for considerable computational throughput even at a low MHz clock rates. The flexibility of the FPGA allows

for even higher performance by trading off precision and range in the number format for an increased number of parallel arithmetic units. This has driven a new type of processing called reconfigurable computing, where time intensive tasks are offloaded from software to FPGAs.

The adoption of FPGAs in high performance computing is currently limited by the complexity of FPGA design compared to conventional software and the turn-around times of current design tools.

Traditionally, FPGAs have been reserved for specific vertical applications where the volume of production is small. For these low-volume applications, the premium that companies pay in hardware costs per unit for a programmable chip is more affordable than the development resources spent on creating an ASIC for a low-volume application. Today, cost and performance dynamics have broadened the range of many applications, like:

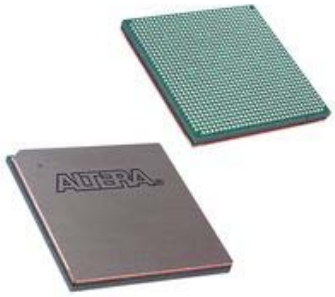
1) Reprogrammable HARDWARE
2) Supports parallel processing (so finds application in VLSI signal processing)
3) Optimization
4) Laboratory prototyping

2.3.6 FPGA vs. Microcontroller

FPGA is mainly for programmable logic but microcontroller is mainly for hardcore processing. Microcontroller is running sequentially regardless of how fast the controller is. In digital signal processor, the hardcore would enhance the harvard architecture by increasing pipelining to certain level of parallel instruction processing. Instead, FPGA is totally based on programmable hardware. The parallel processing in FPGA does not depend on the pipelining, but it is hardware based parallel architecture. For general application, microcontroller is good enough for system implementation. However, in some critical arithmetic processing such as (Digital Signal Processing) DSP would need real time processing that is time critical. In this case, FPGA would be the best solution. FPGAs especially find applications in any area or algorithm that can make use of the massive parallelism offered by their architecture. One such area is code breaking, in particular brute-force attack, of cryptographic algorithms. FPGA is

faster than microcontroller. A comparison study between FPGA and microcontroller is given below:

Table 2.1 Comparison between FPGA and microcontroller

	FPGA	**Microcontroller**
		
Advantages	• fills the gap between hardware and software • much higher performance than software • higher level of flexibility than hardware	• software is very flexible to change
Disadvantages	• need expert programmer to configure the device	• performance can suffer if clock is not fast • fixed instruction set by hardware

2.3.7 FPGA Design and Programming

To define the behavior of the FPGA, the user provides a hardware description language (HDL) or a schematic design. The HDL form is more suited to work with large structures because it's possible to just specify them numerically rather than having to draw every piece by hand. However, schematic entry can allow for easier visualisation of a design.

Then, using an electronic design automation tool, a technology-mapped netlist is generated. The netlist can then be fitted to the actual FPGA architecture using a process called place-and-route, usually performed by the FPGA company's proprietary place-and-route software. The user will validate the map, place and route results via timing analysis, simulation, and other verification methodologies. Once the design and validation process is complete, the binary

file generated (also using the FPGA company's proprietary software) is used to (re)configure the FPGA.

Going from schematic/HDL source files to actual configuration: The source files are fed to a software suite from the FPGA/CPLD vendor that through different steps will produce a file. This file is then transferred to the FPGA/CPLD via a serial interface (JTAG) or to an external memory device like an EEPROM.

The most common HDLs are VHDL and Verilog, although in an attempt to reduce the complexity of designing in HDLs, which have been compared to the equivalent of assembly languages, there are moves to raise the abstraction level through the introduction of alternative languages.

To simplify the design of complex systems in FPGAs, there exist libraries of predefined complex functions and circuits that have been tested and optimized to speed up the design process. These predefined circuits are commonly called IP cores, and are available from FPGA vendors and third-party IP suppliers (rarely free, and typically released under proprietary licenses). Other predefined circuits are available from developer communities such as OpenCores (typically released under free and open source licenses such as the GPL, BSD or similar license), and other sources.

In a typical design flow, an FPGA application developer will simulate the design at multiple stages throughout the design process. Initially the RTL description in VHDL or Verilog is simulated by creating test benches to simulate the system and observe results. Then, after the synthesis engine has mapped the design to a netlist, the netlist is translated to a gate level description where simulation is repeated to confirm the synthesis proceeded without errors. Finally the design is laid out in the FPGA at which point propagation delays can be added and the simulation run again with these values back-annotated onto the netlist.

2.4 Decoder

Decoder is a multiple-input, multiple-output logic circuit that converts coded inputs into decoded outputs, where the input and output codes are different. e.g. n-to-2^n, BCD decoders. Enable inputs must be on for the decoder to function, otherwise its outputs assume a single "disabled" output code word. Decoding is necessary in applications such as data multiplexing, seven segment display and memory address decoding. The simplest decoder circuit would be an AND gate because the output of an AND gate is "High" (1) only when all its inputs are "High". Such output is called as "active High output". If instead of AND gate, the NAND gate is connected the output will be "Low" (0) only when all its inputs are "High". Such output is called as "active low output". Slightly more complex decoder would be the n-to-2^n type binary decoders. These type of decoders are combinational circuits that convert binary information from 'n' coded inputs to a maximum of 2^n unique outputs. We say a maximum of 2^n outputs because in case the 'n' bit coded information has unused bit combinations, the decoder may have less than 2^n outputs. We can have 2-to-4 decoder, 3-to-8 decoder or 4-to-16 decoder. We can form a 3-to-8 decoder from two 2-to-4 decoders (with enable signals). The simple decoder IC 4511 is shown by Figure 2.12. There are 16 pins in IC-4511. Pin 7, 1, 2 and 6 are four inputs A, B, C and D respectively. Pin 9 to 15 are output pins. Another pins are indicated in the diagram.

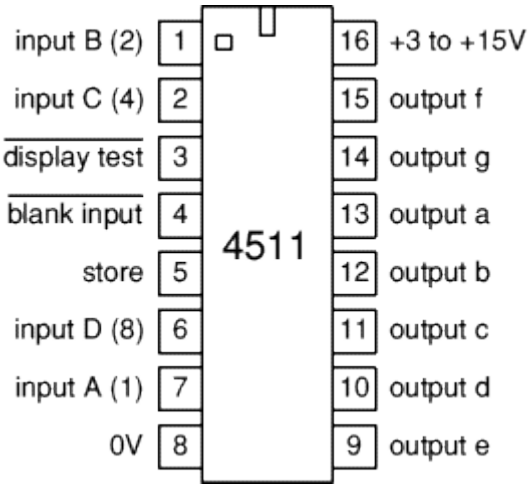


Figure 2.12 Decoder IC 4511

2.5 ULN (Unique Line Number)

The eight NPN Darlington connected transistors in this family of arrays are ideally suited for interfacing between low logic level digital circuitry (such as TTL, CMOS or PMOS/NMOS) and the higher current/voltage requirements of lamps, relays, printer hammers or other similar loads for a broad range of computer, industrial, and consumer applications. All devices feature open–collector outputs and free wheeling clamp diodes for transient suppression. The ULN2803 is designed to be compatible with standard TTL families while the ULN2804 is optimized for 6 to 15 volt high level CMOS or PMOS. ULN2803 IC is shown in Figure 2.13. There are 18 pins in ULN2803. Among the pins 8 pins work as input and 8 pins work as output. Rest two pins are reserve for GND and Vcc.

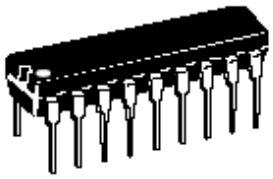


Figure 2.13 ULN IC

The detail pin connection of ULN is shown in Figure 2.14.

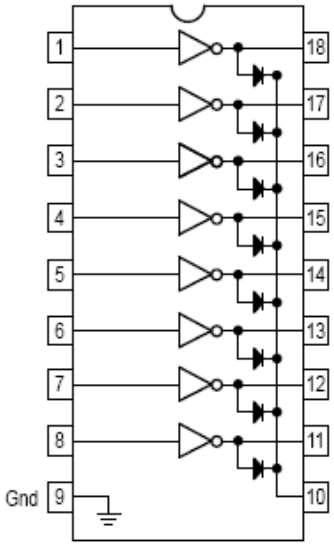


Figure 2.14 Pin Connection of ULN

The ULN2801A-ULN2805A each contain eight Darlington transistors with common emitters and integral suppression diodes for inductive loads. Each darlington features a peak load current rating of 600mA (500mA continuous) and can withstand at least 50V in the off state. Outputs may be paralleled for higher current capability. Five versions are available to simplify interfacing to standard logic families : the ULN2801A is designed for general purpose applications with a current limit resistor ; the ULN2802A has a 10.5kW input resistor and zener for 14-25V PMOS ; the ULN2803A has a 2.7kW input resistor for 5V TTL and CMOS ; the ULN2804A has a 10.5kW input resistor for 6-15V CMOS and the ULN2805A is designed to sink a minimum of 350mA for standard and Schottky TTL where higher output current is required. All types are supplied in a 18-lead plastic DIP with a copper lead from and feature the convenient input opposite- output pin out to simplify board layout.

The block diagram of ULN 2803 is shown in Figure 2.15. There are 8 inputs and 8 outputs in an ULN 2803. Each output is the inverter of corresponding input. Pin 9 is grounded and Pin 10 is reserve for common (+)ve.

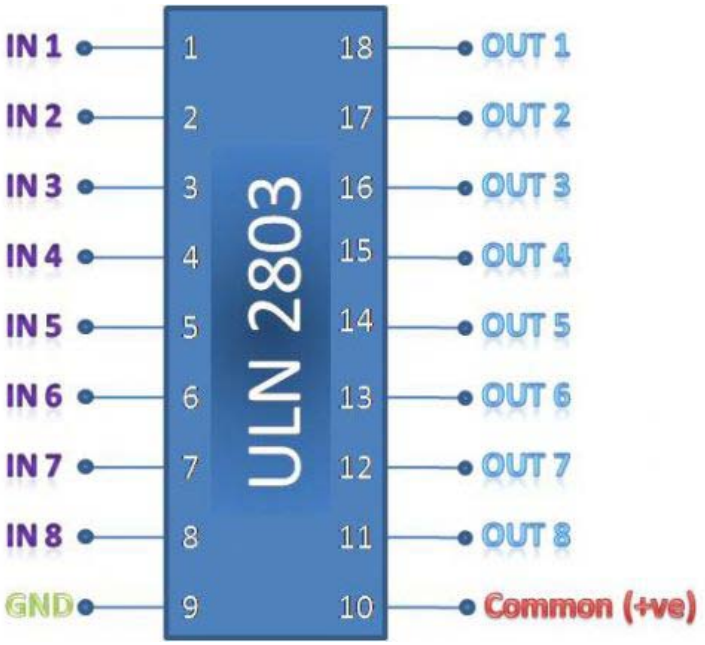


Figure 2.15: ULN 2803 Block diagram

2.5.1 Common Features of ULN

1. Eight Darlingtons with Common Emitters
2. Output Current To 500 mA
3. Output Voltage To 50 V .
4. Integral suppression Diodes.
5. Versions for all popular logic families
6. Output can be paralleled.
7. Inputs pinned opposite outputs to simplify board layout

Chapter 3
FPGA Implementation of the Embedded System

3.1 Introduction

In electronic circuits the issue of managing heat and power dissipation has become increasingly significant. The display units are the major power consuming units of an electronic circuit. Power consumption increases when the size or the number of display units increases. Sometimes it requires extra cooling arrangements for large display units. Besides one has to ensure that the power supply unit he/she uses for the circuit is good enough to supply enough current to illuminate the display units. In this chapter the detail FPGA implementation technique is discussed for the proposed system.

3.2 Scanning Technique

This project concentrates on the power dissipation issues for display devices. A scanning technique has been introduced in the proposed system to minimize the power consumption for display elements. The objective is to connect all the display elements in parallel with each other and turning only one unit ‘ON’ at a time for a very short period and move to the next unit. If each unit can be illuminated periodically in a very short span of time the human eye will see the entire message as if all the units are turned ‘ON’ whereas only one unit is ‘’ON at any given time.

Here seven segment displays are used as display units and as a test case a Muslim Calendar (displays Date, Time and Prayer times of Five Salah) is developed using 30 units of seven segment display. ULN is used to select the corresponding display unit. ULN is a very fast device to turn “ON” and turn “OFF” of display units. As the algorithm for illuminating display units follow the scanning technique where only one unit of seven segment display is turned ‘’ON at any given time. So, theoretically the total power consumption for illuminating display units is reduced by 30 times. Hence 30 units of seven segment displays can be illuminated by only the power required by one unit.

3.3 Block Diagram of the Proposed System

The basic units of the proposed system can be divided into the following 6 Parts.

1. FPGA
2. Display elements
3. Display Decoder
4. ULN
5. Power Unit and
6. Switches

Block diagram of the proposed system is shown in Figure 3.1.

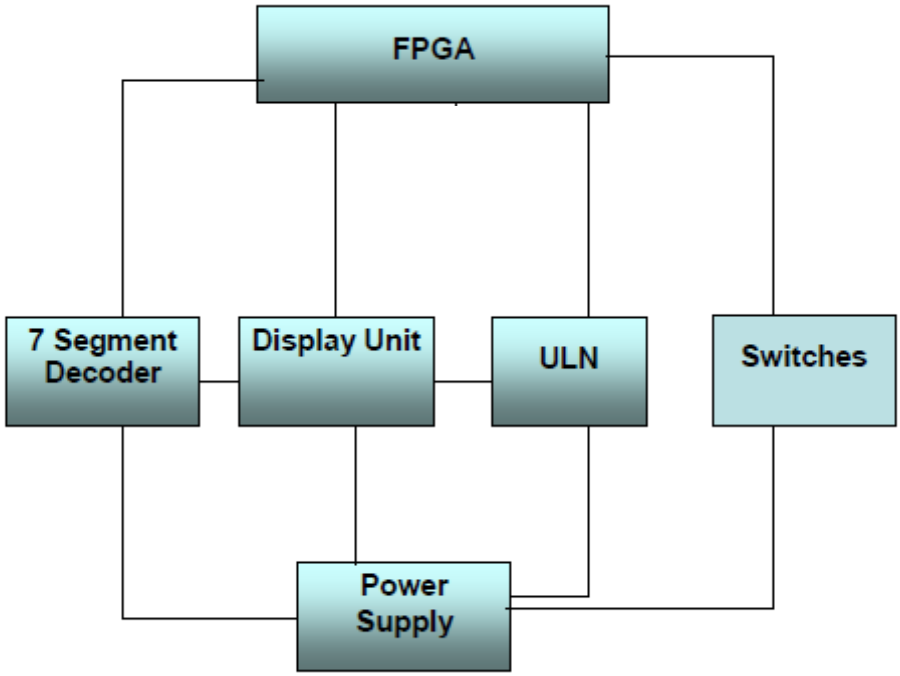


Figure 3.1 Block Diagram of the Proposed System

Here the FPGA based controller controls the input of the ULN. The output of the ULN are connected to the common cathodes of 30 seven segment displays. The FPGA initializes all the output pins of the ULN (sends logic 1 thirty times in the input of the ULN), making all the seven segment displays to be turned ‘OFF’. At this point the FPGA sends logic 0 once and logic1 29 times so that each display receives the signal 0 sequentially while other displays

receive signal 1. In this process, only one display unit is turned ‘ON’ at a time and shifted sequentially.

The output pins of the Seven Segment Decoder are connected in parallel to all seven segment displays. The FPGA sends appropriate input data for the decoder when corresponding seven segment display has logic 0 at its common cathode.

The prayer times change with the change of day and month field of the date.

3.4 Circuit Board Design

Figure 3.2 shows the developed circuit board of the proposed project work. The main part of the project is the FLEX 10K FPGA. All other components are connected to it and controlled by it. The four input pins (D0-D3) of the seven segment decoder CD4511BE) are connected to the hole number of 15-18 of FLEX Expansion slot A of the FPGA. These pins are used to provide the appropriate data to be displayed by a particular seven segment display. Holes no. 1 to 14 and 57 to 60 of FLEX Expansion slot A of the FPGA are used in system purpose. The seven output pins (a-g) of the display decoder are connected to all seven segment displays’ input pins (a-g) in parallel along with seven 100 • registers.

Four ULN (ULN2803) are used in this project. The input pins (IN1 to IN 8) of the of ULN 1, ULN 2, ULN3 and ULN4 are connected to the hole number (19 to 26), (27 to 34), (35-38 & 47-48) and (41-46 & 49-50) of FLEX Expansion slot A of the FPGA respectively. The each output pins (OUTT 1 to OUT 8) of all four ULN are connected to the common cathode pin of each seven segment display. It means, one output pin of ULN is connected to the common cathode of a single display unit.

A 25.175MHz crystal oscillator is connected to the pin 91 of FLEX 10K FPGA.
The detail Pin configuration is shown by the following Tables 3.1 to 3.7.

Table 3.1 Pin Configuration for Clock

Pin Number	Purpose
Pin 91	Used for Clock input (25.175MHz)

Table 3.2 Pin Configuration for Decoder

Decoder Input	**Pin Number**	**Hole No in Flex Expansion Slot A**
A	45	15
B	46	16
C	48	17
D	49	18

Table 3.3 Pin Configuration for ULN 1

ULN Input Pin	**Pin Number**	**Hole No in Flex Expansion Slot A**	**Purpose**
1	55	23	Prayer time for Fazr
2	56	24	
3	61	25	
4	62	26	
5	50	19	Prayer time for Zuhr
6	51	20	
7	53	21	
8	54	22	
9	Ground	×	

Table 3.4 Pin Configuration for ULN 2

ULN Input Pin	**Pin Number**	**Hole No in Flex Expansion Slot A**	**Purpose**
1	63	27	Prayer time for Asr
2	64	28	
3	65	29	
4	66	30	
5	67	31	Prayer time for Maghrib
6	68	32	
7	70	33	
8	71	34	
9	Ground	×	

Table 3.5 Pin Configuration for ULN 3

ULN Input Pin	**Pin Number**	**Hole No in Flex Expansion Slot A**	**Purpose**
1	72	35	Prayer time for Esha
2	73	36	
3	74	37	
4	75	38	
5	87	48	For Displaying Date
6	86	47	
7	Not Used	Not Used	×
8	Not Used	Not Used	×
9	Ground	×	

Table 3.6 Pin Configuration for ULN 4

ULN Input Pin	**Pin Number**	**Hole No in Flex Expansion Slot A**	**Purpose**
1	84	46	For Displaying Hour
2	83	45	
3	82	44	For Displaying Minute
4	81	43	
5	80	42	For Displaying Second
6	79	41	
7	94	50	For Displaying Month
8	88	49	
9	Ground	×	×

Table 3.7 FLEX_SW1 Switches Pin Assignments

Switch	FLEX 10K Pin	Purpose
FLEX_SWITCH-1	41	For Adjusting Day
FLEX_SWITCH-2	40	For Adjusting Month
FLEX_SWITCH-3	39	For Adjusting Minute
FLEX_SWITCH-4	38	For Adjusting Hour
FLEX_SWITCH-5	36	Not Used
FLEX_SWITCH-6	35	Not Used
FLEX_SWITCH-7	34	Not Used
FLEX_SWITCH-8	33	For Check Counter

The developed circuit board using Vero board and other related equipments is shown in Figure 3.2. The circuit board is interfaces with the FLEX10K FPGA.

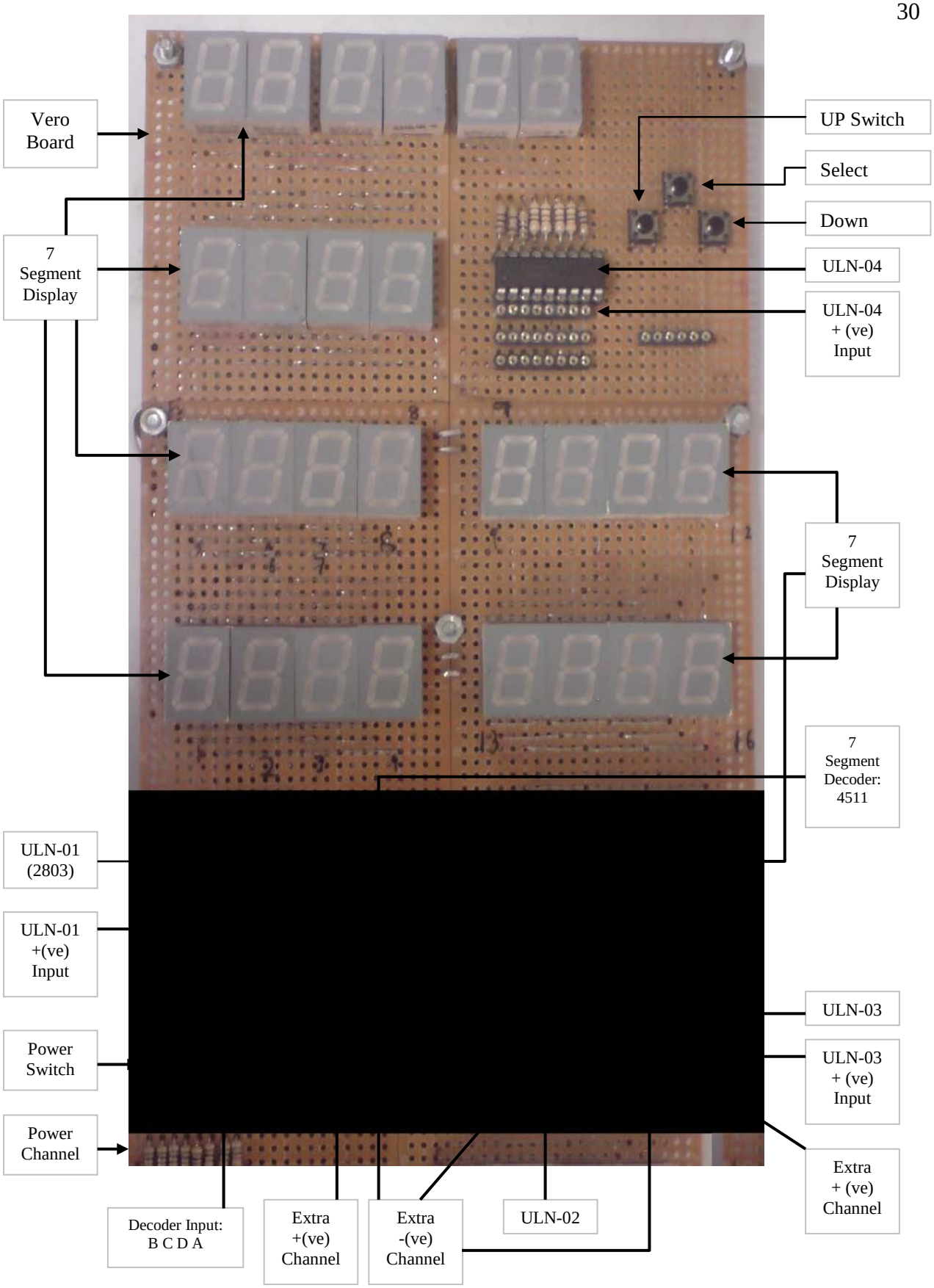


Figure 3.2 Circuit Board Description

The list of components used to develop the circuit board is given below.

Item	No. of Pcs
1. Vero Boards	:03
2. ULN 2803	:04
3. 7 Segment Decoder 4511	: 01
4. Power Switch	:01
5. Power switch IC-7805	:01
6. 7 Segment Display Unit	:30
7. Registers (100Ω, 10kΩ)	:100 (app)
8. Rail	:10
9. Different connecting wires	
10. Circuit legs	:08

3.5 Operation of the Proposed System

In the proposed system a controller scans all the display units continuously to ensure that only one unit is on at any given time. In this way power consumption will be much lower than that of existing static display system. Here a ‘Muslim Calendar’ containing date, time and prayer times for five salah has been developed as a test case. Seven segment displays are used as display units.

Here the FPGA controls the input of the ULN and Decoder. The FPGA initializes all the output pins of the ULN at level 1 (sends logic 1 thirty times in the input of the ULN), making all the seven segment displays to be turned ‘OFF’. At this point the FPGA sends logic 0 once and logic1 29 times so that each display receives the signal 0 sequentially while other displays receive signal 1. In this process, only one display is turned at a time and shifted sequentially.

3.6 Devices Used for the Proposed System

Some chips/devices have been used for the proposed system. Brief description on those devices is given bellow.

3.6.1 Field-Programmable Gate Array (FPGA)

For the proposed system Altera University Program kit UP2 was used. There are two FPGA chip in an Altera University Program FPGA. One is MAX another is FLEX. Figure 3.3 shows the detail about both FPGAs. There are 3 expansion slots Named FLEX_EXPAN_A, FLEX_EXPAN_B, and FLEX_EXPAN_C in FLEX10K Device. There are also two push buttons LEX_PB1 & FLEX_PB2,FLEX_SW1 Switches, FLEX_DIGIT Display and some other components. AS I need 36 pins for my project I only used FLEX_EXPAN_A slot.

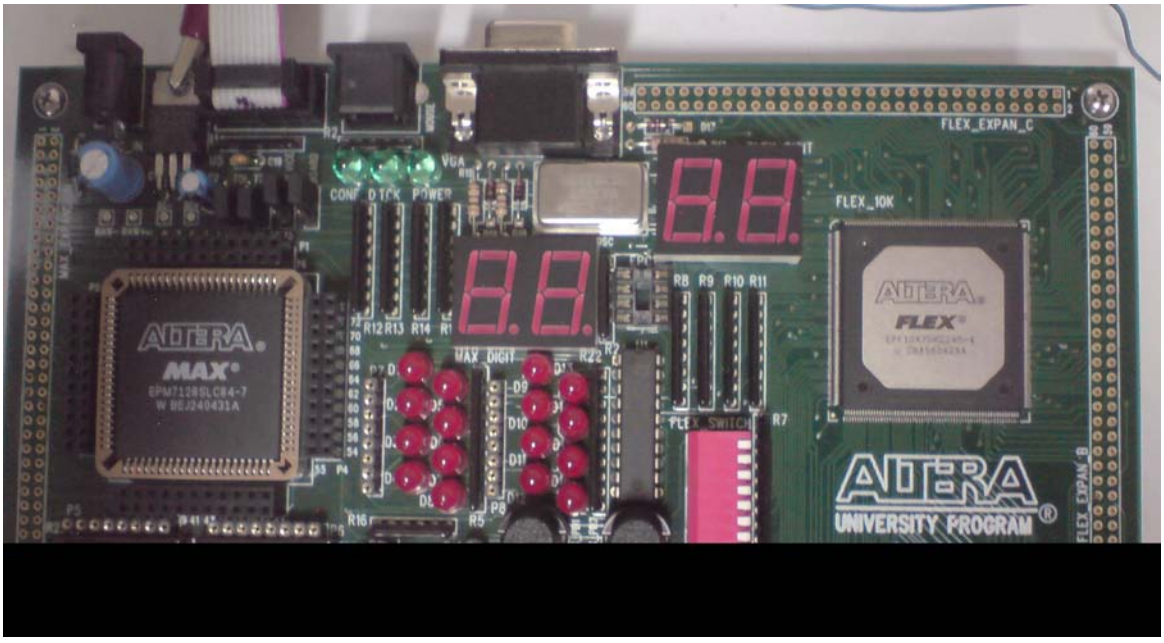


Figure 3.3 Altera FLEX10K FPGA

Although there are different types of FPGAs, the Altera Education board FPGA is widely used. UP2 is one of the FPGA platform for University program. The different components and devices of UP2 board which are used in the proposed system are discussed below.

(a) EPF10K70 Device

The EPF10K70 device is based on SRAM technology. It is available in a 240-pin RQFP package and has 3,744 logic elements (LEs) and nine embedded array blocks (EABs). Each LE consists of a four-input LUT, a programmable flipflop, and dedicated signal paths for carry-and-cascade functions. Each EAB provides 2,048 bits of memory which can be used to create RAM, ROM, or first-in first-out (FIFO) functions. EABs can also implement logic functions, such as multipliers, microcontrollers, state machines, and digital signal processing (DSP) functions. With 70,000 typical gates, the EPF10K70 device is ideal for intermediate to advanced digital design courses, including computer architecture, communications, and DSP applications.

Another chip of Altera US2 program FPGA is EPM7128S. There are only 128 logic cell in EPM7128S. The amount of logic cell of EPM7128S is not enough for my project work as I need about 2200 logic cell to run my program. So In my project work I used EPF10K70 Device because it has more logic cell then EPM7128S Device.

(b) Features of FLEX 10K Device

The UP2 Education Board provides the following resources for the FLEX 10K device. The pins from the FLEX 10K device are pre-assigned to switches and LEDs on the board.

- JTAG chain connection for the ByteBlaster II cable
- Socket for an EPC1 configuration device
- Two momentary push button switches
- One octal DIP switch
- Dual-digit seven-segment display
- On-board oscillator (25.175 MHz)
- VGA port
- Mouse port
- Three expansion ports, each with 42 I/O pins and seven global pins.

(c) FLEX_PB1 & FLEX_PB2 Push Buttons

FLEX_PB1 and FLEX_PB2 are two push buttons that provide active-low signals to two general-purpose I/O pins on the FLEX 10K device. FLEX_PB1 connects to pin 28, and FLEX_PB2 connects to pin 29. Each push button is pulled-up through a 10-KΩ resistor.

(d) FLEX_SW1 Switches

FLEX_SW1 contains eight switches that provide logic-level signals to eight general-purpose I/O pins on the FLEX 10K device. An input pin is set to logic 1 when the switch is open and set to logic 0 when the switch is closed. Table 3.8 lists the pin assignment for each switch.

Table 3.8 FLEX_SW1 Pin Assignments

Switch	FLEX 10K Pin
FLEX_SWITCH-1	41
FLEX_SWITCH-2	40
FLEX_SWITCH-3	39
FLEX_SWITCH-4	38
FLEX_SWITCH-5	36
FLEX_SWITCH-6	35
FLEX_SWITCH-7	34
FLEX_SWITCH-8	33

(e) FLEX_DIGIT Display

FLEX_DIGIT is a dual-digit, seven-segment display connected directly to the FLEX 10K device. Each LED segment on the display can be illuminated by driving the connected FLEX 10K device I/O pin with a logic 0.

(f) FLEX_EXPAN_A, FLEX_EXPAN_B & FLEX_EXPAN_C

FLEX_EXPAN_A, FLEX_EXPAN_B, and FLEX_EXPAN_C are dual rows of 0.1-inch spaced holes for accessing signal I/O pins and global signals on the FLEX 10K device, power, and ground. More detail is given to Appendix C.

Jumper Settings for Configuring Only the FLEX 10K Device is shown in Figure 3.4.

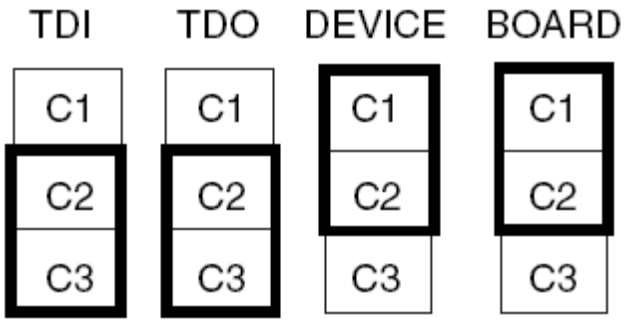


Figure 3.4 Jumper Settings for FLEX 10K Device

3.6.2 BCD to 7-Segment Latch / Decoder/Driver (CD4511BE)

The CD4511BE is a BCD to 7-segment latch/decoder/driver with four address inputs (D_A to D_D), an active LOW latch enable input ($\overline{EL}$), an active LOW ripple blanking input ($\overline{BL}$), an active LOW lamp test input ($\overline{LT}$), and seven active HIGH n-p-n bipolar transistor segment outputs (O_a to O_g). When ELis LOW, the state of the segment outputs (O_a to O_g) is determined by the data on D_A to D_D. When *EL* goes HIGH, the last data present on D_A to D_D are stored in the latches and the segment outputs remain stable. When LT is LOW, all the segment outputs are HIGH independent of all other input conditions. With LT HIGH, a LOW on BI forces all segment outputs LOW. The inputs LT and BL do not affect the latch circuit. Seven segment decoder IC HEF4511B is shown in Figure 3.5.

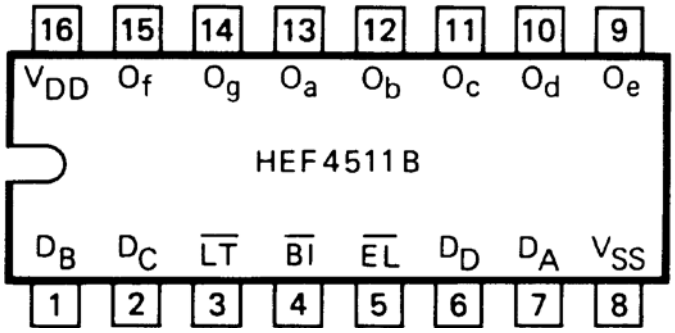


Figure 3.5 Pinning Diagram of Seven Segment Decoder

The functional table of seven segment decoder 4511 is given to Table 3.9.

Table 3.9 Function Table of 7-Segment Decoder

INPUTS							OUTPUTS							
$\overline{EL}$	$\overline{BI}$	$\overline{LT}$	D_D	D_C	D_B	D_A	O_a	O_b	O_c	O_d	O_e	O_f	O_g	DISPLAY
X	X	L	X	X	X	X	H	H	H	H	H	H	H	8
X	L	H	X	X	X	X	L	L	L	L	L	L	L	blank
L	H	H	L	L	L	L	H	H	H	H	H	H	L	0
L	H	H	L	L	L	H	L	H	H	L	L	L	L	1
L	H	H	L	L	H	L	H	H	L	H	H	L	H	2
L	H	H	L	L	H	H	H	H	H	H	L	L	H	3
L	H	H	L	H	L	L	L	H	H	L	L	H	H	4
L	H	H	L	H	L	H	H	L	H	H	L	H	H	5
L	H	H	L	H	H	L	L	L	H	H	H	H	H	6
L	H	H	L	H	H	H	H	H	H	L	L	L	L	7
L	H	H	H	L	L	L	H	H	H	H	H	H	H	8
L	H	H	H	L	L	H	H	H	H	L	L	H	H	9
L	H	H	H	L	H	L	L	L	L	L	L	L	L	blank
L	H	H	H	L	H	H	L	L	L	L	L	L	L	blank
L	H	H	H	H	L	L	L	L	L	L	L	L	L	blank
L	H	H	H	H	L	H	L	L	L	L	L	L	L	blank
L	H	H	H	H	H	L	L	L	L	L	L	L	L	blank
L	H	H	H	H	H	H	L	L	L	L	L	L	L	blank
H	H	H	X	X	X	X				*				*

3.6.3 ULN 2803

The ULN2803 is used for the proposed system. There are 8 input and 8 output pins in each ULN. Rest two pins are used for GND and Vcc. The Pin connections of ULN 2803 is shown in Figure 3.6.

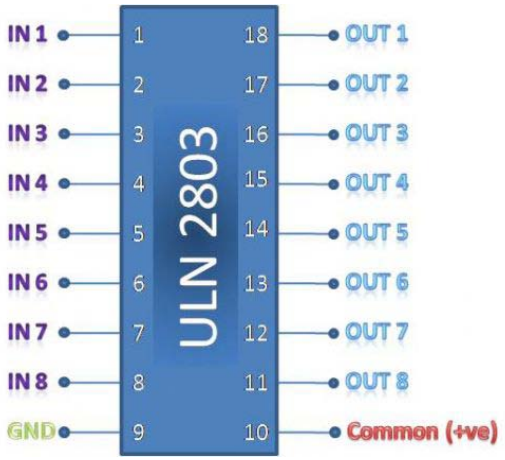


Figure 3.6 Pining Diagram of ULN 2803

3.7 Software Used for the Proposed System

Verilog HDL is one of the most popular high definition language to design digital circuit. In this project the proposed system is designed and simulated using Verilog HDL in Altera's Quartus II environment.

3.7.1 Verilog HDL (Hardware Definition Language)

In the earlier, the conventional approach such as hand-draw and schematic based design technique was the only choice to the designer to design a digital system. But now millions of transistors are being integrated on a single chip integrated circuit (IC) where the conventional design technique is insufficient to be used. It points towards having a new approach for designing today's complex digital system and that is hardware description language (HDL). HDL based design technique has been emerged as the most efficient solution. It offers the following advantages over conventional based design approaches.

- It is technology independent. If a particular IC fabrication process becomes outdated, it is possible to synthesize a new level design by only changing the technology file but using the same HDL code.
- HDL shortens the design cycle of a chip by efficiently describing and simulating the behavior of the chip. A complex circuit can be designed using a few lines of HDL code.
- It lowers the cost of design of an IC.
- It improves design quality of a chip. Area and timing of the chip can be optimized and analyzed in different stages of design.

There are different types of HDL available in the market. Some of these are vendor dependent where the HDL code is only useable under the software provided by the specific vendor. For example, Altera hardware description language (AHDL) from Altera company, Lola (Logic Language) from European Silicon Structure (ES2) company etc. However Verilog and VHDL (very high speed IC hardware description language) are the two vendor independent HDL which are now widely accepted industry standard electronic design automation (EDA) tool for designing digital system. Verilog HDL is introduced by Cadence Data Systems, Inc. and later its control is transferred to a consortium of companies and universities known as open Verilog

international (OVI) whereas VHDL is used primarily by defense contractors. Currently Verilog is widely used by IC designers. Verilog HDL is IEEE standard and easier than VHDL. It is less error prone. It has many pre-defined features very specific to IC design. For this reason Verilog is chosen to design and implement of the proposed system.

3.7.2 Development Tool Quartus II

The Proposed System is designed using Quartus II EDA tool (provided by Altera Company) which provides graphical user interface (GUI) to download the digital design of Proposed system into the FLEX 10K FPGA.

Quartus II software provides a simple, automated mechanism to allow designers to obtain the best performance for their designs. This software provides the way to design the solution through Verilog HDL and complile the design to ensure the workability and efficiency logically. The tool programmer allows using files generated by the compiler to program and/or configuring all devices supported by the Quartus II software. Programmer and supported programming hardware tool is used to easily program or configure a working device in minutes. After a successful compilation, download configuration data into a device through the, ByteBlaster or USB-Blaster communications cables, or through the Altera Programming Unit (APU).The program or configure devices can be in Passive Serial mode, Active Serial Programming mode, JTAG mode, or In-Socket Programming mode.

Programming an Altera Device

When the design is ready to program or configure a device, it needs to open the programmer and create a chain description file (.cdf) that stores device name, device order, programming and hardware setup information. CDFs can be used to program or configure one or more devices in a JTAG chain or a passive serial chain.

Compiling Mode

The Quartus II Compiler consists of a set of independent modules that check the design for errors, synthesize the logic, fit the design into an Altera device, and generate output files for simulation, timing analysis, software building, and device programming. The basic compiler

consists of the analysis & synthesis, fitter, assembler, and timing analyzer modules. Each of the compiler modules can be run individually or together from the Quartus II user interface. Alternatively, these modules can be run independently with the appropriate command line executable.

Compile the Design

The compiler automatically locates and uses all non-design files associated with the design, such as include files (.inc) containing AHDL function prototype statements; memory initialization files (.mif) or hexadecimal intel-format files (.hex) containing the initial content of memories; as well as Quartus II project Files (.qpf) and Quartus II settings files (.qsf) containing project and setting information. During compilation, the compiler generates information, warning, and error messages that appear automatically in the messages window.

Simulation Mode

Simulation allows testing a design thoroughly to ensure that it responds correctly in every possible situation before configuring a device. Depending on the type of information need, functional or timing simulation can be performed with the simulator. Functional simulation tests only the logical operation of a design by simulating the behavior of flattened netlist extracted from the design files, while timing simulation uses a fully compiled netlist containing timing information to test both the logical operation and the worst-case timing for the design in the target device. Before running a simulation, it is necessary to specify input vectors as the stimuli for the Quartus II Simulator. The simulator uses these input vectors to simulate the output signals that a programmed device would produce under the same conditions. The simulator supports input vector stimuli in the form of a vector waveform file (**.vwf**), vector table output file (**.tbl**), power input file (**.pwf**), or a Quartus II generated vector file (**.vec**) or simulator channel file (**.scf**).

Quartus II is a software tool produced by Altera for analysis and synthesis of HDL designs, which enables the developer to compile their designs, perform timing analysis, examine RTL diagrams, simulate a design's reaction to different stimuli, and configure the target device with the programmer.

Quartus II Web Edition is a free version of Quartus II that can be downloaded or delivered by mail for free. This edition provided compilation and programming for a limited number of Altera devices.

The low-cost Cyclone family of FPGAs is fully supported by this edition, as well as the MAX family of CPLDs, meaning small developers and educational institutions have no overheads from the cost of development software. License registration is required to use the Web Edition of Quartus II, which is free and can be renewed an unlimited number of times.

Design Flow for FPGA Implementation using Quartus II 7.0

Figure 3.7 shows flow diagram of a design to be realized into FPGA hardware. Once the sub-modules of a design are identified, each of the modules is designed, compiled and synthesized using FPGA vendor provided software. Then functional simulation is performed upon each module. The correct simulation results ensure the proper functionality of a design. Once the simulation results of all the sub-modules are as desired then they are integrated and simulated again. Then for hardware realization, suitable FPGA device is selected for the design, inputs and outputs are assigned to specific pins of the FPGA. It is again compiled and synthesized. After that timing simulation of the design is performed to ensure that the design functions in real time. Then the design is downloaded into the FPGA.The Design flow of FPGA implemetation are shown in Figure 3.7.

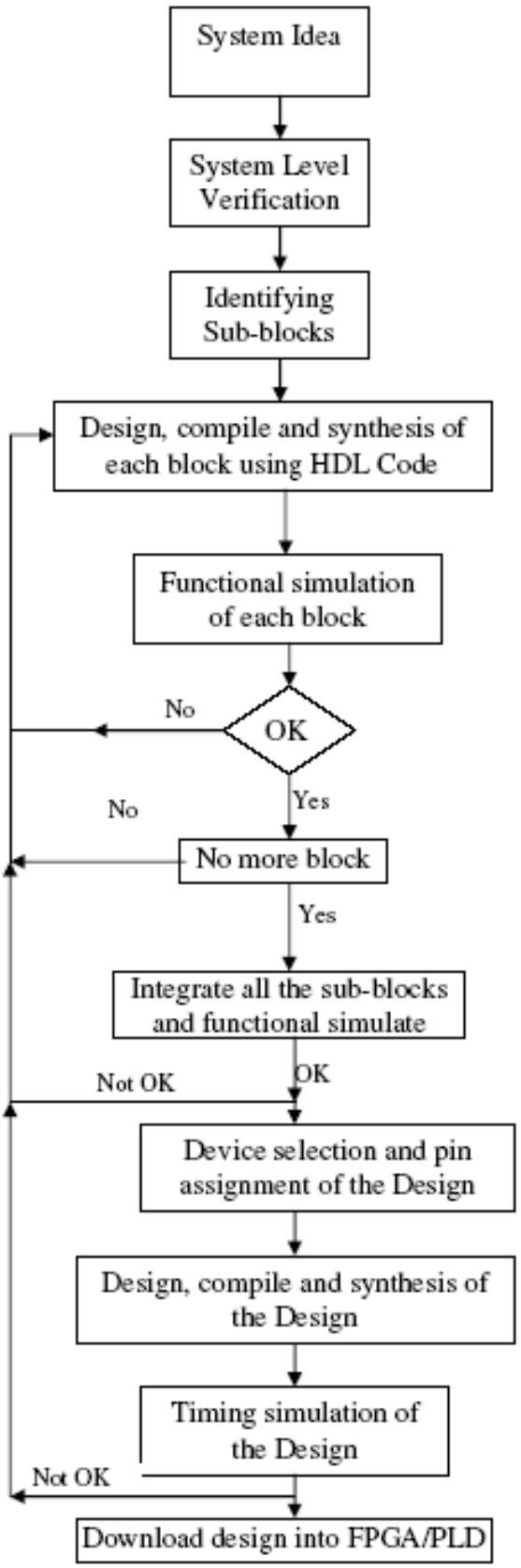


Figure 3.7 Digital System Design and Implementation using FPGA

So to implement the circuits that will be designed on the FPGA there are few key steps.

1. Create circuit using different components
2. Test the circuit manually
3. Start New Project using Quartus II
4. Create a new Verilog HDL file
5. Write the program using Verilog HDL.
6. Compile the code.
7. Correct syntax errors and other errors.
8. Create a Vector Web form file
9. Pin Assignment
10. Simulate the circuit to make sure that getting the expected behavior.
11. Download the program onto the FPGA
12. Test the circuit for operation

Quartus II helps to implement all of the above easily.

Chapter 4
Results and Discussions

4.1 Introduction

This chapter shows simulation and implementation results of the FPGA based controller system. Artview of the proposed system, current and power consumption of the test case design are given in this chapter. At the end of chapter there is a comparison study of present work with the previous research.

4.2 Simulation Result

(a)Decoder Data:

Figure 4.1 shows the decoder data simulation. There are four inputs A, B, C and D of seven segment decoder. Decimal output is shown in the simulation for prayer time, Date and time.

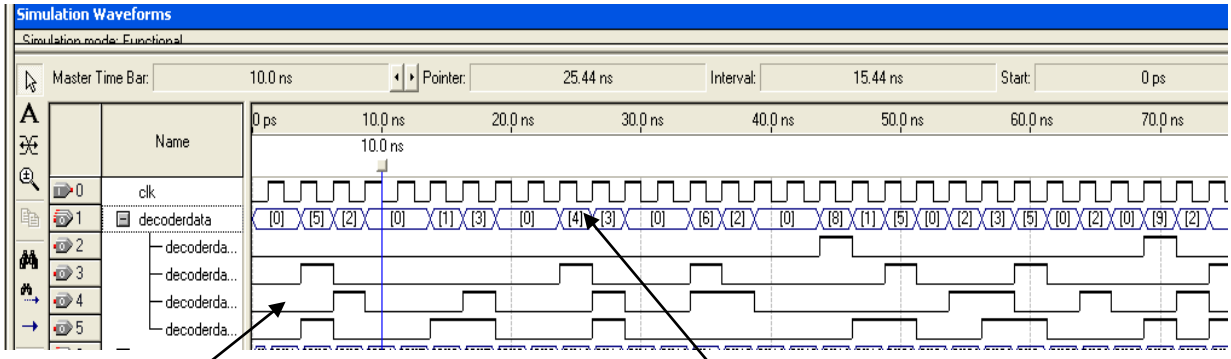


Figure 4.1 Decoder Data Simulation

The decoder data simulation is indicated with the arrow mark. The timing diagram and corresponding decimal values are shown in Figure 4.1.

(b) Counter and Decoder Data Simulation

Figure 4.2 shows the simulation of decoder data with counter. The counter is reset when its counter value becomes 30.

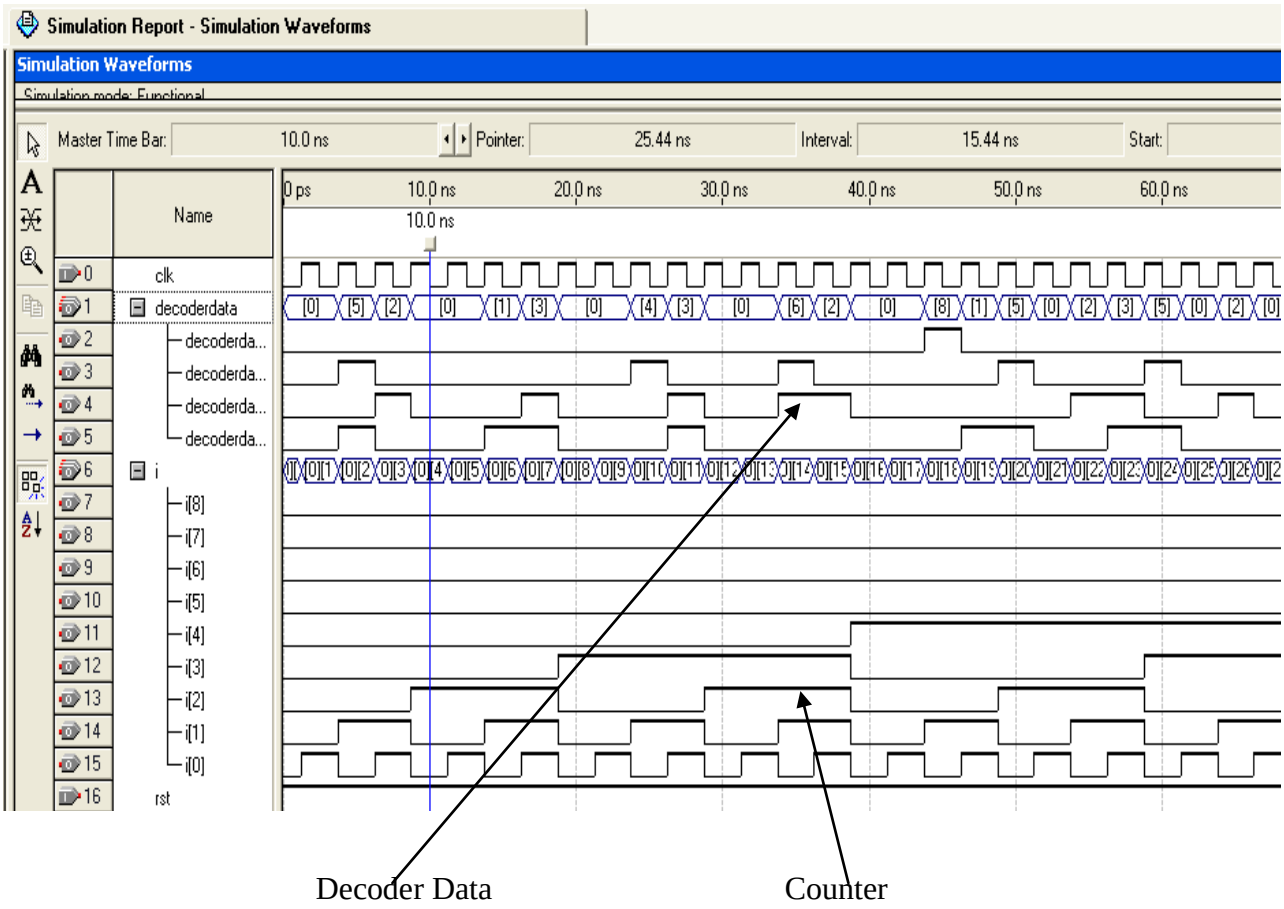


Figure 4.2 Counter and decoder data simulation

The simulation for decoder data and counter are indicated with the arrow mark. The timing diagram and corresponding decimal values are shown in Figure 4.2.

(c) ULN Data Simulation

Figure 4.3 shows ULN Data Simulation. When one pin ULN pass 1 then all other pin passes 0. In a particular instant of time only one ULN signal is activated.

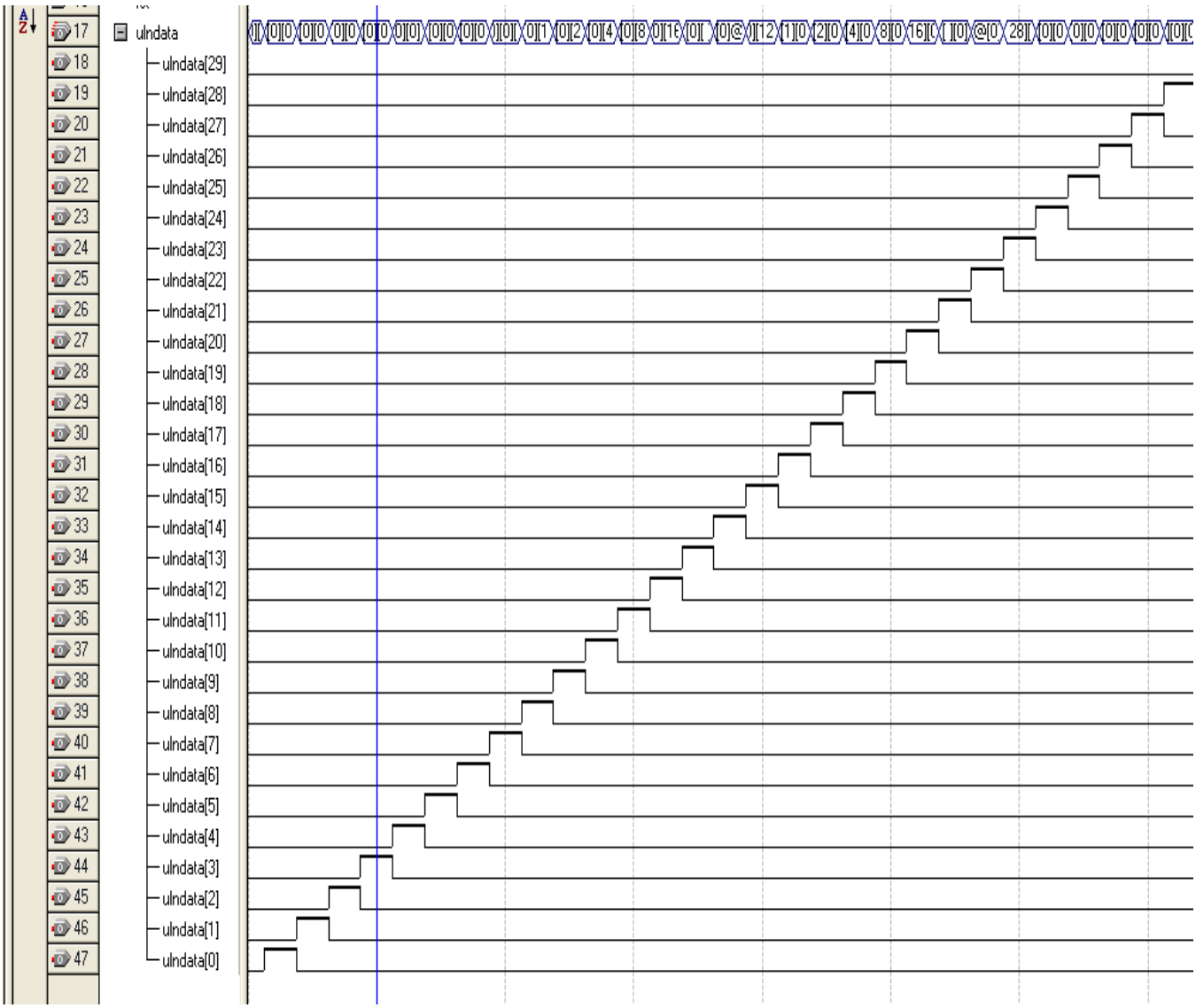


Figure 4.3 ULN Data Simulation

Figure 4.3 shows the simulation result for ULN output for corresponding 30 seven segment display units. In a particular instance of time only one ULN pin is HIGH and all other pins of ULN are in LOW state.

(d) Overall Simulation

Figure 4.4 shows the overall simulation of the proposed system. The system is simulated using the Quartus II 7.0 simulator. The overall simulation is done using to the EPF10K70RC240-4 FPGA.

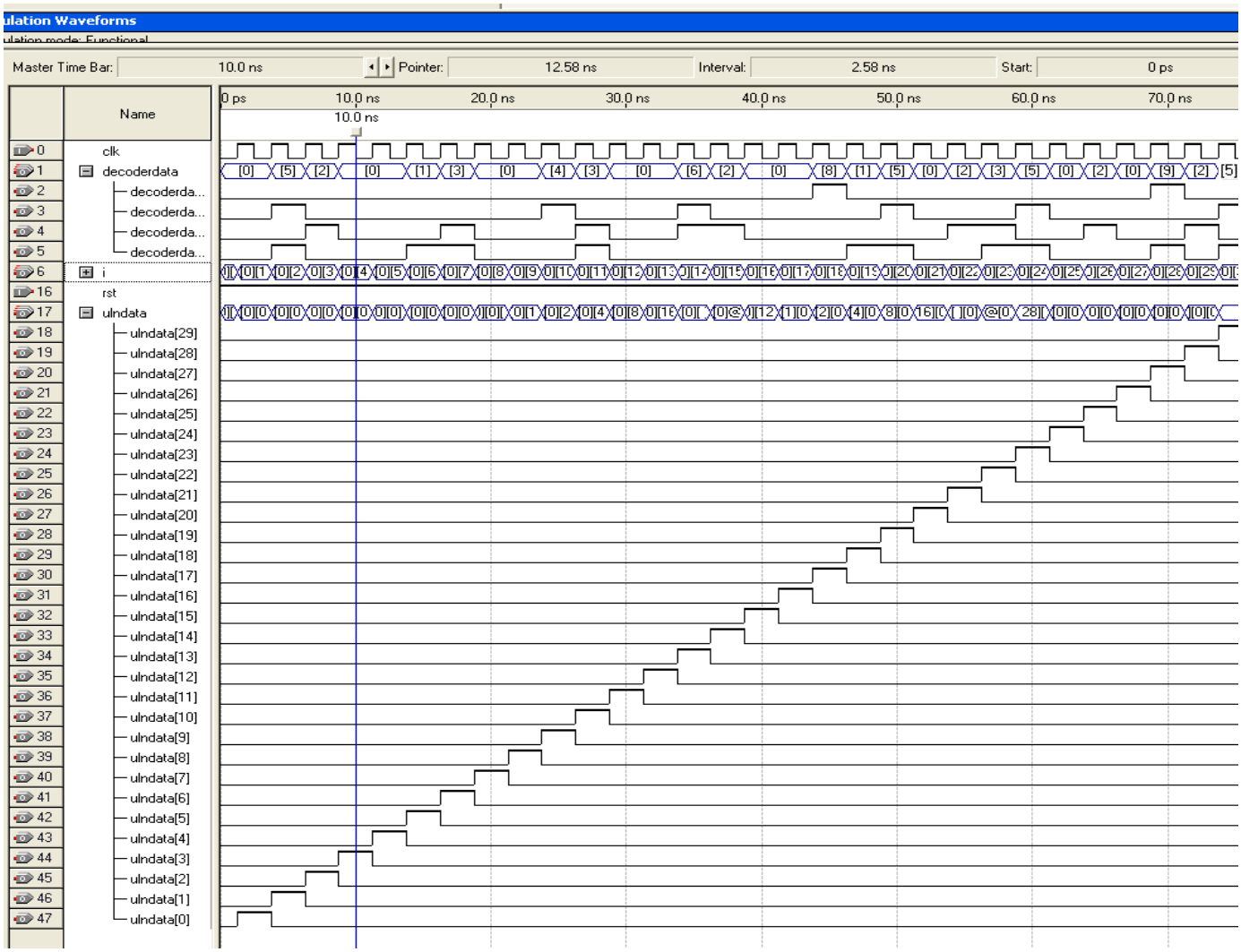


Figure 4.4 Overall Simulation

It can be seen from the simulation that when one ULN passes signal LOW to the common cathode of a particular seven segment display at that time all other ULN passes the HIGH signal to the common cathode of the others seven segment display. When a seven segment display gets LOW signal in Common Cathode then it show the corresponding data passes by

seven segment decoder. Seven segment decoder gets input from FPGA controller and passes to seven segment display. Seven segment display is controlled by ULN.

The simulation result shows that when ULN no. 01 passes 0 signal to Display unit no. 01 then only Display unit no. 01 will be ‘ON’ in that time all other display unit will be ‘OFF’. For a particular signal of ULN a particular data is shown in a corresponding seven segment display unit.

4.3 Artview of the Proposed System

After putting all the components on the circuit board and programming the FPGA the whole system looks like Figure 4.5.

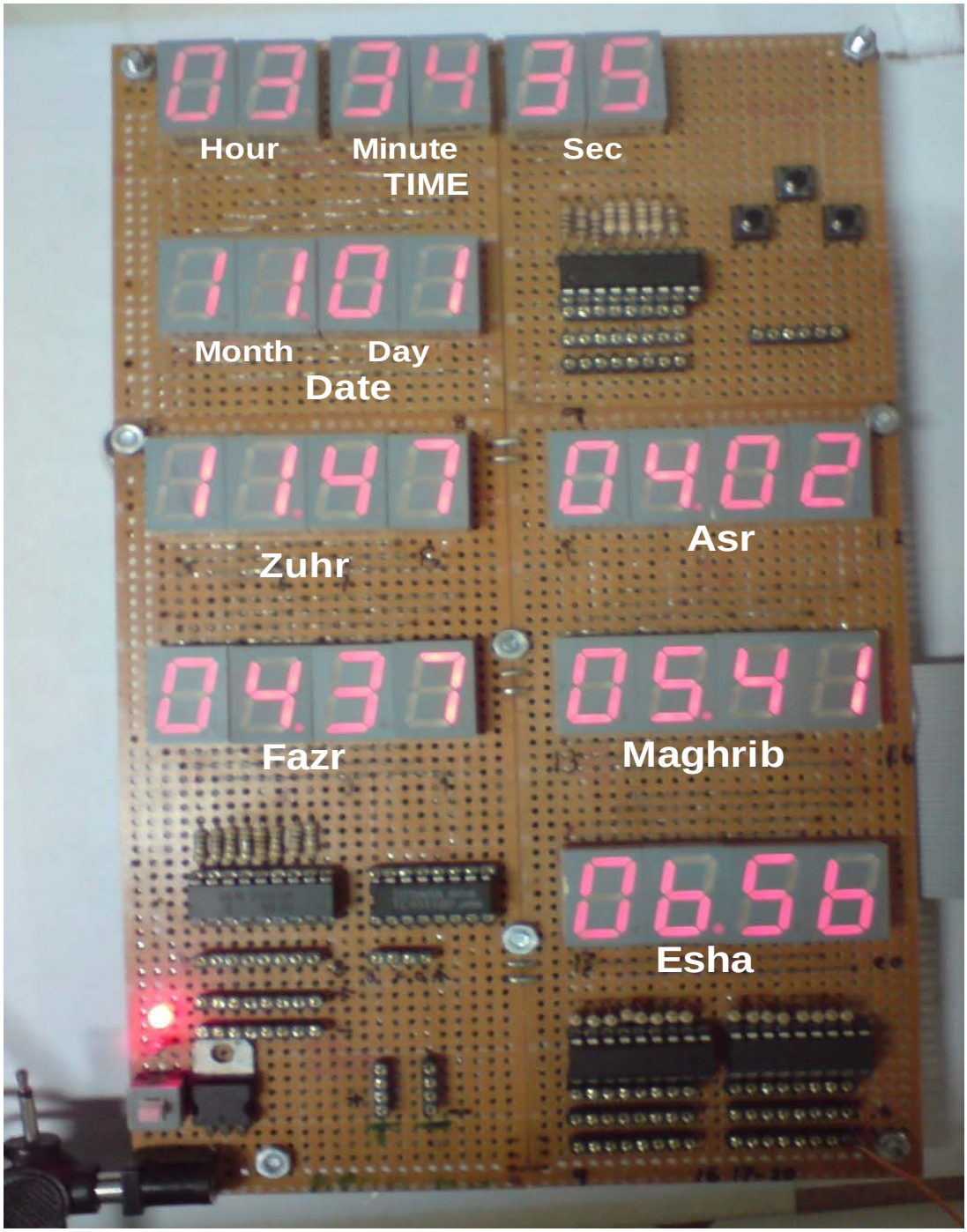


Figure 4.5 Artview of the Proposed System

Here the first block of six (top) seven segment displays shows time. First two units of this block represent hour while second two represent minute. The third two represents seconds. The second block of four (top Left corner) seven segment displays represents date where the first and second two units of the block represent month and day respectively. The remaining five blocks of four seven segment displays represent the prayer times of five salat; Fazr, Zurh, Asr, Maghrib and Esha as labeled in Figure 4.5. Here each first two units of each block represent hour and the remaining two units of each block represent minute.

4.4 Full Artview of the Proposed System (With FPGA)

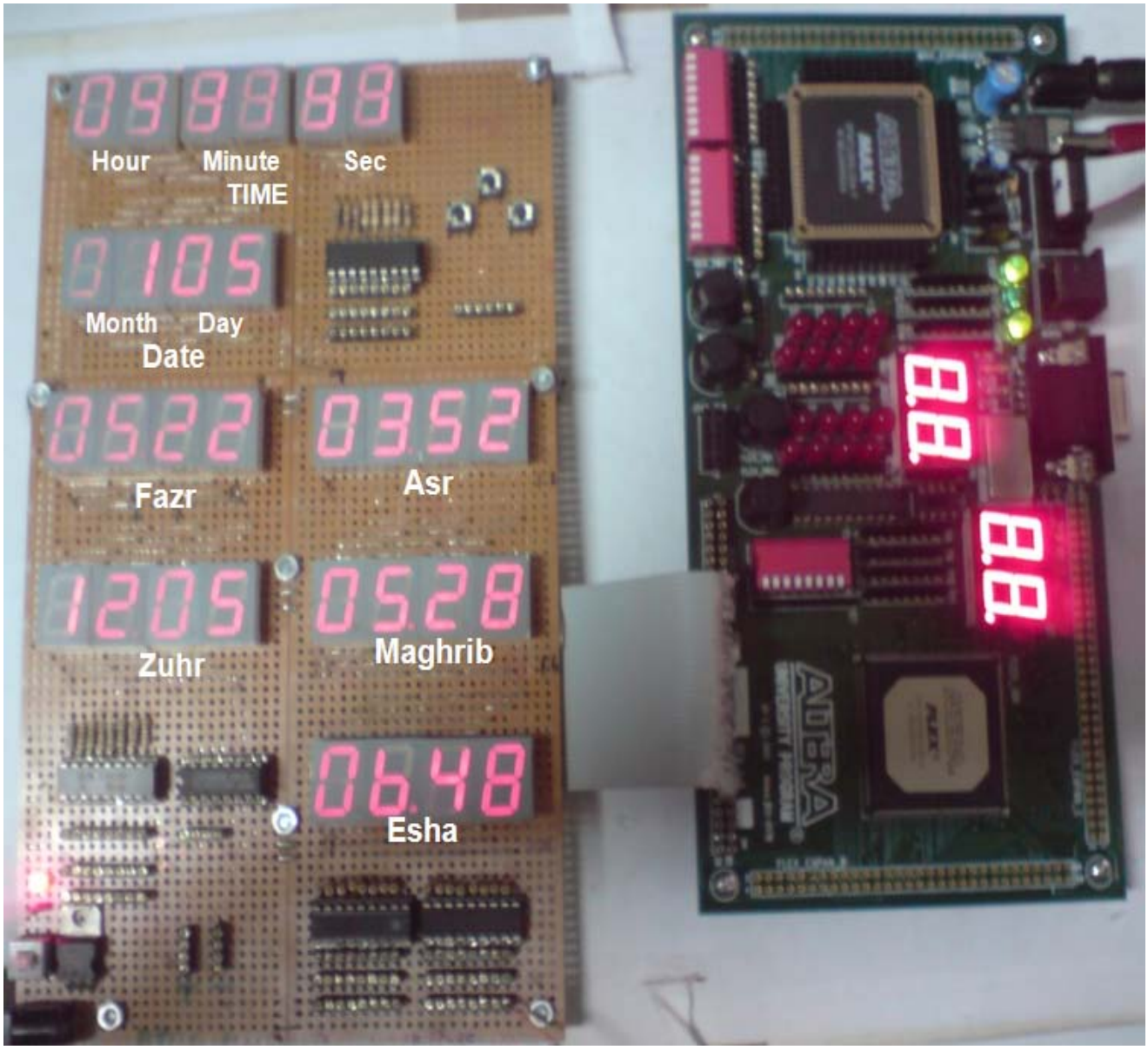


Figure 4.6 Full Artview of the System with FPGA

It is observed from Figure 4.6 that, there are two parts in full artview. Left part is developed circuit board and the right side part is Flex 10k FPGA. The whole circuit board is controlled by FPGA. There are 30 seven segment display unit which are enlighted for different prayer times and also for date and time.

4.5 Result and Performance

The design of the proposed system implementing scanning technique is coded using Verilog HDL. Altera's FLEX 10K FPGA is used as hardware platform. In this project work the used device is EPF10K70RC2404 from Flex 10K family. The compilation results of the proposed design are as follows:

Device family:	FLEX10K
Device:	EPF10K70RC2404
Timing Model:	Final
Met timing requirement:	Yes
Total Logic elements:	2175 (Out of 3744), used 58% of total logic elements
Total Pin:	40 (Out of 189), used 21% of total pins
Clock Period:	191.00ns
Frequency;	5.24Mhz
LC Registers:	207
LUT -Only LCs:	1968(671)
Register- only LCs:	49(49)
LUT/Register LCs:	158(127)
Carry Chain Lcs:	719(7)

4.6 Power and Current Consumption

Scanning technique is a power aware solution for message display. Here we have used 30 seven segment display units. But as scanning technique is applied, at any given time only one seven segment display is 'ON'. So, theoritically the system should consume $^{1}/_{30}$ times less power than it would take to make all the seven segment displays 'ON' at a time.

0.56• (14.2 mm) seven segment display from Agilent Technologies have been used to develop the proposed system. Current consumption per segment is 15 mA. So the total current consumption of a seven segment display will be 105 mA in the worst case when all of its segments are enlighted. As 30 seven segment display unit were used in the project so

theoretically total current required (without scanning technique) is 3150 mA in worst case when all the segments are enlighted.

Power Analysis of the FPGA based system has been carried out using the power analyzer tool of the Quartus II. It shows that the required power for design is 33.25 mW. In the laboratory an experiment has been conducted to measure the current and power for the developed system. To find the current and power consumption of the system in the worst case without scanning technique all the segments have been enlighted and then power and current are measured. In the similar way the current and power for the system with scanning technique are also measured for the worst case situation. Table 4.1 shows the experimental results.

Table 4.1 Current and Power Consumption of the system

	With Scanning Technique	Without Scanning Technique
Current Consumption	0.008 A	0.070 A
Power Consumption	0.070 W	0.383 W

It is observed from Table 4.1 that it is possible to reduce 88% less current and 82% less power consumption using FPGA based scanning technique.

4.7 Comparison of the Present Work with the Other Research

The present work consumes less current and less power than the work of Khan, M. R.[15]. A comparison study between present research and previous research is presented in Table 4.2.

Table 4.2 Comparison of Current and Power Consumptin with the work of Khan, M. R.[15]

	Current Consumption	Power Consumption
Khan, M. R.[15]	0.018 A	0.090 watt
Present Work	0.008 A	0.070 watt

It can be observed from Table 4.2 that the present work requires 55% less current consumption and 22% less power than the work of Khan, M. R.[15].

Since the present work is FPGA based design, it removes the pin limitation problem of microcontroller based design. It is also more secured than microcontroller based design. Moreover in the FPGA based system all the discrete digital components can be put in the single chip and so PCB area for the system will be less than the microcontroller based system which turn will reduce the system size, weight and cost.

Chapter 5
Conclusion and Future Works

5.1 Conclusion

Low power embedded system offers a lot of advantages such as portability, longer battery life and compact size. To overcome the limitations of a previously developed microcontroller based embedded system for Muslim Calender was the core objective of this research. The limitations were limited number of pins, lower processing speed, lower physical security and power consumption. To improve the overall performance of the system, scanning technique has been implemented in the FPGA platform. FPGA technology eliminates limitations of pins, offers higher processing speed and physical security over microcontroller based system. EDA simulation result shows the proper functionality of the system. The laboratory test result of the system proves the significant improvement in power reduction over the existing approach. The system will be reduced in size and cost effective. It is also capable of driving larger number of display units.

5.2 Suggestions for Future Works

The author recommends the following suggestion to improve the proposed work.

1. The work presented in this project, Altera's Flex 10K FPGA has been used. Currently the FPGA vendors have cool runner version of FPGA for low power applications. It can be used to reduce the total power of the proposed system.
2. The Muslim Calendar in this project only displays the prayer time at Dhaka city. It can be improved to make it universal so that it can be able to display the prayer time of any important city in this world.

References:

[1] Islam, M.M., Hossain, M. K., Hasan, K.S., and Haque, A.L., “A 7-Segment Display for Bangla, English and Other Indian Numerals”, Proceedings of International Conference on Electrical and Computer Engineering (ICECE), Dhaka, Bangladesh, 2008.

[2] Arefin, M. S., Dewan, M. A. A., Khan, M. I., and Islam, M. S., "Designing a 24-Segment Display for Bengali Numerical Digits and Characters", Proceedings of International Conference on Electrical and Computer Engineering (ICECE), Dhaka, Bangladesh, pp:549-552, 2004.

[3] Ruckmongathan, T. N., “A Successive Approximation Technique for Displaying Gray Shades in Liquid Crystal Displays (LCDs)”, IEEE Trans. On Image Process., vol. 16 (2), pp. 554–561, 2007.

[4] Azad, M. A. K., Sharmeen, R., Ahmed, S., and Kamruzzaman, S. M., "A Unique 10 Segment Display for Bangla Numerals", Proceedings of International Conference on Computer and Information Technology (ICCIT), 2005.

[5] Karri, R., and Orailoglu, A., “Standard Seven Segmented Display for Burmese Numerals”, Proceedings of Consumer Electronics, IEEE Transactions, vol-36, issue: 4, pp. 959-961, 1990.

[6] Islam, R., Alam, M. G. R., and Uddin, M. N., “An 8-Segment Display for Simple and Accurate Representation of Bangla Numerals”, Proceedings of International Conference on Electrical and Computer Engineering (ICECE), 2006.

[7] Rabbi, F., Hossain, M. K., and Ahmed, M., “An 8-Segment Display for both English and Bangla Digits”, Proceedings of International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, pp. 338-341, 2003.

[8] Rahman, T., Khan, T., Ahmed, S.S., and Karmakar, C. K., "N-Segmented Display of Bangla Numerals", Proceedings of International Conference on Computer and Information Technology (ICCIT), Dhaka, Bangladesh, pp. 118-124, 2005.

[9] Bhunia, S., Mahmoodi, H., Mukhopadhyay, S., Ghosh, D., and Roy, K., “A Novel Low-Power Scan Design Technique Using Supply Gating”, Proceedings of the IEEE International Conference on Computer Design (ICCD’04) , 2004.

[10] Moshnyaga, V.G. and Morikawa, E., “LCD Display Energy Reduction by User Monitoring”, Proceedings of IEEE International Conference on Computer Design: VLSI in Computers and Processors , pp. 94 – 97, 2005.

[11] Ruckmongathan, T. N., Govind, M., and Deepak, G., “Reducing Power Consumption in Liquid Crystal Displays”, IEEE Trans. on Electron Devices, vol. 53(7), pp. 1559–1566, 2006.

[12] Marks, B. W., “Power Reduction in Liquid Crystal Display Modules”, IEEE Trans. on Electron Devices, vol. ED-29(12), pp. 1884–1886, 1982.

[13] Aerts, W. F., Verlaak, S. and Heremans P., “Design of an Organic Pixel Addressing Circuit for an Active-Matrix OLED Display”, Proceedings of IEEE Transactions on Electron Devices, vol-49(12), pp 2124-2130, 2002.

[14] Kimmel, J., Hautanen, L. and Levola, T., “Display Technologies for Portable Communication Devices”, Proceedings of the IEEE, vol-90(4), pp 581-590, 2002.

[15] Rahman, M., “Development of an Embedded System for Low Power Message Display using Scanning Technique”, Thesis, IICT, BUET, Dhaka, Bangladesh, 2008.

[16] http://www.cypress.com/?rID=3063; last access on 20.03.10

[17] http://www.fordata.cn/; last access on 20.03.10

Appendix A

/*Project Coding*/

/Code for Simulation*/

```
module test(rst,decoderdata,ulndata,clk,i);

input rst;
input clk;
output [3:0] decoderdata;

output [29:0] ulndata;

reg [3:0] decoderdata;
reg [29:0] ulndata;

wire [120:0]data[3:0];

output [8:0] i;
reg [8:0] i;
integer j=0;

  assign
data[0]=120'b0101001010010000001000000101001100100000010100011000000000000010011000000000001101000000000000011000100000000000010010100000;

always @( posedge clk   )
begin
  if (rst==0)
     i=0;
   else
    begin
     if (clk==1)
       i=i+1;
       if (i==127)
         begin
          i=0;
```

```
        end
      else
        begin
        decoderoutput(i,decoderdata);
        ulnoutput(i, ulndata);
        ulndata=ulndata;
        decoderdata=decoderdata;
      end
  end

end

always @( posedge clk   )
 begin
   if (i==128)
     j=j+1;
   if (j==5)
     j=0;
  end

task decoderoutput(input integer indexno , output [3:0] namajtime );
 begin
   case (indexno)
                    1: namajtime= {data[j][3],data[j][2],data[j][1],data[j][0]};
                    2: namajtime= {data[j][7],data[j][6],data[j][5],data[j][4]};
                    3: namajtime= {data[j][11],data[j][10],data[j][9],data[j][8]};
                    4: namajtime= {data[j][15],data[j][14],data[j][13],data[j][12]};
                    5: namajtime= {data[j][19],data[j][18],data[j][17],data[j][16]};
                    6: namajtime= {data[j][23],data[j][22],data[j][21],data[j][20]};
                    7: namajtime= {data[j][27],data[j][26],data[j][25],data[j][24]};
                    8: namajtime= {data[j][31],data[j][30],data[j][29],data[j][28]};
                    9: namajtime= {data[j][35],data[j][34],data[j][33],data[j][32]};
                    10: namajtime= {data[j][39],data[j][38],data[j][37],data[j][36]};
                    11: namajtime= {data[j][43],data[j][42],data[j][41],data[j][40]};
                    12: namajtime= {data[j][47],data[j][46],data[j][45],data[j][44]};
                    13: namajtime= {data[j][51],data[j][50],data[j][49],data[j][48]};
                    14: namajtime= {data[j][55],data[j][54],data[j][53],data[j][52]};
                    15: namajtime= {data[j][59],data[j][58],data[j][57],data[j][56]};
                    16: namajtime= {data[j][63],data[j][62],data[j][61],data[j][60]};
```

```
                17: namajtime= {data[j][67],data[j][66],data[j][65],data[j][64]};
                18: namajtime= {data[j][71],data[j][70],data[j][69],data[j][68]};
                19: namajtime= {data[j][75],data[j][74],data[j][73],data[j][72]};
                20: namajtime= {data[j][79],data[j][78],data[j][77],data[j][76]};
                21: namajtime= {data[j][83],data[j][82],data[j][81],data[j][80]};
                22: namajtime= {data[j][87],data[j][86],data[j][85],data[j][84]};
                23: namajtime= {data[j][91],data[j][90],data[j][89],data[j][88]};
                24: namajtime= {data[j][95],data[j][94],data[j][93],data[j][92]};
                25: namajtime= {data[j][99],data[j][98],data[j][97],data[j][96]};
                26: namajtime= {data[j][103],data[j][102],data[j][101],data[j][100]};
                27: namajtime= {data[j][107],data[j][106],data[j][105],data[j][104]};
                28: namajtime= {data[j][111],data[j][110],data[j][109],data[j][108]};
                29: namajtime= {data[j][115],data[j][114],data[j][113],data[j][112]};
                30: namajtime= {data[j][119],data[j][118],data[j][117],data[j][116]};

   endcase
 end
endtask

task ulnoutput(input integer ledno, output [29:0]  lightonoff  );
 begin
   case (ledno)
     1: lightonoff=30'b00000000000000000000000000000001;
     2: lightonoff=30'b00000000000000000000000000000010;
     3: lightonoff=30'b00000000000000000000000000000100;
     4: lightonoff=30'b00000000000000000000000000001000;
     5: lightonoff=30'b00000000000000000000000000010000;
     6: lightonoff=30'b00000000000000000000000000100000;
     7: lightonoff=30'b00000000000000000000000001000000;
     8: lightonoff=30'b00000000000000000000000010000000;
     9: lightonoff=30'b00000000000000000000000100000000;
    10: lightonoff=30'b00000000000000000000001000000000;
    11: lightonoff=30'b00000000000000000000010000000000;
    12: lightonoff=30'b00000000000000000000100000000000;
    13: lightonoff=30'b00000000000000000001000000000000;
    14: lightonoff=30'b00000000000000000010000000000000;
    15: lightonoff=30'b00000000000000000100000000000000;
    16: lightonoff=30'b00000000000000001000000000000000;
    17: lightonoff=30'b00000000000000010000000000000000;
```

```
    18: lightonoff=30'b00000000000010000000000000000;
    19: lightonoff=30'b00000000000100000000000000000;
    20: lightonoff=30'b00000000001000000000000000000;
    21: lightonoff=30'b00000000010000000000000000000;
    22: lightonoff=30'b00000000100000000000000000000;
    23: lightonoff=30'b00000001000000000000000000000;
    24: lightonoff=30'b00000010000000000000000000000;
    25: lightonoff=30'b00000100000000000000000000000;
    26: lightonoff=30'b00001000000000000000000000000;
    27: lightonoff=30'b00010000000000000000000000000;
    28: lightonoff=30'b00100000000000000000000000000;
    29: lightonoff=30'b01000000000000000000000000000;
    30: lightonoff=30'b10000000000000000000000000000;
  endcase
 end
endtask

endmodule
```

/* Main Program module coding*/

```
module namaj(decoderdata,ulndata,clk , sw1, sw2, sw3, sw4, chkcounter );
input sw1;
input sw2;
input sw3;
input sw4;

input clk;

input chkcounter ;
output [3:0] decoderdata;
output [29:0] ulndata;

reg [3:0] decoderdata;
reg [29:0] ulndata;

reg [79:0]data[52:0];

reg [11:0] j=0;
reg [11:0] l=1;

reg [11:0] a1=0;

reg [11:0] sec = 0 ;   // 0
reg [11:0] min = 30 ;   //30 ;
reg [11:0] hr =  3 ;   //   3 ;
reg [11:0] dhr = 0 ;   //0
reg [11:0] dd = 1 ;
reg [11:0] mm = 12 ;
reg [11:0] yy =  10 ;
reg [11:0] ss = 0 ;
reg [11:0] nchange = 0;
reg [24:0] msec = 0;

reg  [11:0]  k = 0  ;
reg  rst;

//reg  chkcounter ;
```

```
wire [11:0] ddcounter ;
wire [11:0] mmcounter ;
wire [11:0] mincounter;
wire [11:0] hrcounter;

switch1 sss1   (sw1 , chkcounter , ddcounter) ;
switch2 sss2   (sw2 , chkcounter , mmcounter) ;
switchmin sss3 (sw3 , chkcounter , mincounter );
switchhr sss4  (sw4 , chkcounter , hrcounter );

initial
 begin

data[0]=80'b0100010001100000001100100101000001110100001100000010000000100001000000100101000 0;
data[1]=80'b1000010001100000100000100101000000100101001100000010100000010000100100010010100 00;
data[2]=80'b0010010101100000001100110101000001110101001100001000000000100001010000100101000 0;
data[3]=80'b0111010101100000100000110101000000100000010000000000000100100001010000100101000 0;
data[4]=80'b0001000001110000001101000101000001110000010000000010000100100001001100100101000 0;
data[5]=80'b0101000001110000011101000101000000010001010000000011000100100001000100100101000 0;
data[6]=80'b1000000001110000001001010101000001100001010000000011000100100001011100010101000 0;
data[7]=80'b0010000101110000011001010101000010010001010000000010000100100001001100010101000 0;
data[8]=80'b0101000101110000000000000110000000100010010000000010000100100001100000000101000 0;
data[9]=80'b0111000101110000001000000110000001000010010000000001000100100001010000000101000 0;
data[10]=80'b0000001001110000010100000110000001100010010000001001000000100001100001010100000 0;
data[11]=80'b0011001001110000100000000110000001110010010000000111000000100001000101010100000 0;
data[12]=80'b0110001001110000000100010110000010010010010000000101000000100001010001000100000 0;
data[13]=80'b0000001101110000010000010110000010010010010000000011000000100001011100110100000 0;
data[14]=80'b0011001101110000011000010110000000000001101000000000100000010000110010010010000 0;
data[15]=80'b0111001101110000100100010110000000010011010000001001010100010001001000100100000 0;
data[16]=80'b0001010001110000001000100110000000010011010000000111010100010001010000010100000 0;
data[17]=80'b0101010001110000010100100110000000010011010000000110010100010001011100000100000 0;
data[18]=80'b0000010101110000100000100110000000010001101000000010101010001000100010000010000 0;
data[19]=80'b0101010101110000001000110110000000110011010000000101010100010001011001010011000 0;
data[20]=80'b1001010101110000010100110110000001000011010000000101010100010001000101010011000 0;
data[21]=80'b0100000010000000010010011011000000101001101000000010101010001000101110100001100 0;
data[22]=80'b1000000010000000001001000110000001100011010000000110010100010001010101000011000 0;
data[23]=80'b0010000110000000010101000110000010000011010000001000010100010001010001000011000 0;
```

```
data[24]=80'b0101000110000000011101000110000010010011010000001001010100010001010001000010000110000;
data[25]=80'b0110000110000000100101000110000000010100010000000001000000100001010101000010000110000;
data[26]=80'b0111000110000000100101000110000000100100010000000010000000100001011101000010000110000;
data[27]=80'b0110000110000000100101000110000000110100010000000011000000100001000001010010000110000;
data[28]=80'b0100000110000000100101000110000001000100010000000010000000100001010001010010000110000;
data[29]=80'b0001000110000000011101000110000001000100010000000010100000010000110000101001100000110000;
data[30]=80'b0111000010000000010001000110000000110100010000000010100000010000100100000010000000000000;
data[31]=80'b0010000010000000000000100011000000010010001000000001010000001000010111000000100000000000;
data[32]=80'b0110010101110000010100110110000000000100010000000100000000100001000100010100000000000000;
data[33]=80'b1001010001110000000000110110000001110011010000000010000000100001010100010100000000000000;
data[34]=80'b0010010001110000010000100110000000110011010000000001000000100001100100010100000000000000;
data[35]=80'b0100001101110000011100010110000010010010010000001001010100010001001100100100000000000000;
data[36]=80'b0110001001110000000000010110000001010010010000000110010100010001011000100100000000000000;
data[37]=80'b1001000101110000001100000110000010010001010000000100010100010001100100100100000000000000;
data[38]=80'b0001000101110000011001010101000001000001010000000001010100010001001000110100000000000000;
data[39]=80'b0011000001110000100001000101000010000000010000001001010000010001010000110100000000000000;
data[40]=80'b0110010101100000000101000101000000100000001000000011101000001000101110011010000000000000;
data[41]=80'b0000010101100000010100110101000001110101001100000101010000010001100100110100000000000000;
data[42]=80'b0100010001100000100100100101000000100101001100000011010000010001001001000100000000000000;
data[43]=80'b0100000001100000001100100101000001110100001100000010010000010001010101000100000000000000;
data[44]=80'b0110001101100000100100010101000000110100001100000010010000010001100001000100000000000000;
data[45]=80'b0011001101100000010100010101000010010011001100000010010000010001001001010100000000000000;
data[46]=80'b0001001101100000001100010101000001110011001100000011010000010001011001010100000000000000;
data[47]=80'b0000001101100000000100010101000001010011001100000101010000010001000000000010100000000000;
data[48]=80'b0001001101100000000100010101000001010011001100000011101000001000101000000001010000000000;
data[49]=80'b0010001101100000001000010101000001100011001100000000001010001000110000000001010000000000;
data[50]=80'b0101001101100000010000010101000010000011001100000101001100010001001000010101000000000000;
data[51]=80'b1000001101100000011100010101000000010100001100000011101010001000101100001010100000000000;
data[52]=80'b0010010001100000000100100101000001010100001100000000000000010000110010001010100000000000;

rst =1'b1 ;
end

always @( posedge clk)
begin
        ulndata=0;
        decoderdata=0;
```

```
        if (l<=20)
          begin
           decoderoutput( l, decoderdata );
          end
         else
          begin
            hrminsec( l, decoderdata );
          end
         ulnoutput (l , ulndata  );

         ulndata=ulndata;
         decoderdata=decoderdata;

end

always @(posedge clk)
 begin
    j=j+1;
     if (j==2001)
      begin
      j=0;
      l=l+1;
      if (l==31)
        l=1;
      end
end

always @(posedge clk)
 begin

    if (chkcounter==1'b1)
          begin
          dd = ddcounter;
          mm = mmcounter;
          min = mincounter;
          hr = hrcounter;
         end
```

```
  msec = msec + 1 ;
//  if (msec ==25'b11000000000010001111011000)   // 1 sec
  if (msec ==25'b00000000000000001111011000)   //Fast
 //   if (msec ==25'b00000000000000000011011000)   // Very Fast

   begin
     sec = sec +1;
     if (sec == 60)
       begin
        min = min +1 ;
         if (min==60)
          begin
            hr = hr +1;
           dhr = dhr +1;
            if (hr == 13)
             begin
              hr = 1 ;
             end
            if (dhr==24)
             begin

                                    if ((dd== 28 ) && ( mm == 2 ) )
                                              begin
                                               ss = yy % 4 ;
                                              if (ss > 0)
                                                begin
                                                 dd = 1;
                                                 mm = mm +1 ;
                                                 end
                                               else
                                                begin
                                                  dd= dd + 1 ;
                                                  k = k +1;
                                                end
                                              end
                                         else if ((dd== 29 ) && ( mm == 2 ))
                                             begin
                                              dd=1 ;
                                              mm = mm +1 ;
```

```
        end

    else if ((dd== 30 ) && ( mm == 4 ) )
        begin
          dd= 1 ;
          mm = mm +1 ;

         end
      else if ((dd== 30 ) && ( mm == 6 ) )
        begin
          dd= 1 ;
          mm = mm +1 ;

        end
      else if ((dd== 30 ) && ( mm == 9 ) )
        begin
          dd= 1 ;
          mm = mm +1;

        end
      else if ((dd== 30 ) && ( mm == 11 ) )
        begin
          dd= 1 ;
          mm = mm +1;

        end
      else if (dd == 31 )
        begin
         dd = 1;
         mm = mm +1;

        end
      else
        begin
         dd = dd + 1;
         k = k +1;
        end

if (mm==13)
```

```
                                begin
                                 yy = yy +1 ;
                                 mm=1;
                                end

                dhr=0 ;
               end
              min=0;
             end
            sec=0;
           end
         msec=0;
        end

if (rst ==1)
 begin
  if (mm==1)
   begin
   a1 = dd ;
   end
  else if (mm==2)
   begin
    a1 = dd + 31 ;
   end
  else if (mm==3)
   begin
    a1 = dd + 59 ;
   end
  else if (mm==4)
   begin
    a1 = dd + 90 ;
   end
  else if (mm==5)
   begin
    a1 = dd + 121 ;
   end
  else if (mm==6)
   begin
    a1 = dd + 151 ;
```

```
    end
   else if (mm==7)
    begin
     a1 = dd + 182 ;
    end
  else if (mm==8)
    begin
     a1 = dd + 213;
    end
 else if (mm==9)
    begin
     a1 = dd + 243 ;
    end
 else if (mm==10)
    begin
     a1 = dd + 274 ;
    end
 else if (mm==11)
    begin
     a1 = dd + 304 ;
    end
 else if (mm==12)
    begin
     a1 = dd + 334 ;
    end

   k = a1/7;
   //rst = 1'b0 ;
  end
 else
   begin
   rst = 1'b0 ;
   end

end
```

```
//Task for Date and Time
task hrminsec (input integer indexno , output [4:0] daytime);
reg [5:0] secl;
reg [5:0] sech;
reg [5:0] minl;
reg [5:0] minh;
reg [5:0] hrl;
reg [5:0] hrh;
reg [5:0] ddl;
reg [5:0] ddh;
reg [5:0] mml;
reg [5:0] mmh;
begin
  sech = sec / 10 ;
  secl = sec % 10 ;
  minh = min / 10 ;
  minl = min % 10 ;
  hrh = hr / 10 ;
  hrl = hr % 10 ;
  ddh  = dd / 10 ;
  ddl  = dd % 10 ;
  mmh  = mm / 10 ;
  mml  = mm % 10 ;

  case (indexno)
       21: daytime= {secl[3] ,secl[2] , secl[1] , secl[0]};
      22: daytime= {sech[3] ,sech[2] , sech[1] , sech[0]};
      23: daytime= {minl[3] ,minl[2] , minl[1] , minl[0]};
      24: daytime= {minh[3] ,minh[2] , minh[1] , minh[0]};
      25: daytime= {hrl[3] ,hrl[2] , hrl[1] , hrl[0]};
      26: daytime= {hrh[3] ,hrh[2] , hrh[1] , hrh[0]};
      27: daytime= {ddl[3] ,ddl[2] , ddl[1] , ddl[0]};
      28: daytime= {ddh[3] ,ddh[2] , ddh[1] , ddh[0]};
      29: daytime= {mml[3] ,mml[2] , mml[1] , mml[0]};
      30: daytime= {mmh[3] ,mmh[2] , mmh[1] , mmh[0]};

 endcase
end
endtask
```

//Task for decoder output

```
task decoderoutput(input integer indexno , output [4:0] namajtime );
 begin
   case (indexno)

                1: namajtime= {data[k][3],data[k][2],data[k][1],data[k][0]};
                2: namajtime= {data[k][7],data[k][6],data[k][5],data[k][4]};
                3: namajtime= {data[k][11],data[k][10],data[k][9],data[k][8]};
                4: namajtime= {data[k][15],data[k][14],data[k][13],data[k][12]};
                5: namajtime= {data[k][19],data[k][18],data[k][17],data[k][16]};
                6: namajtime= {data[k][23],data[k][22],data[k][21],data[k][20]};
                7: namajtime= {data[k][27],data[k][26],data[k][25],data[k][24]};
                8: namajtime= {data[k][31],data[k][30],data[k][29],data[k][28]};
                9: namajtime= {data[k][35],data[k][34],data[k][33],data[k][32]};
                10: namajtime= {data[k][39],data[k][38],data[k][37],data[k][36]};
                11: namajtime= {data[k][43],data[k][42],data[k][41],data[k][40]};
                12: namajtime= {data[k][47],data[k][46],data[k][45],data[k][44]};
                13: namajtime= {data[k][51],data[k][50],data[k][49],data[k][48]};
                14: namajtime= {data[k][55],data[k][54],data[k][53],data[k][52]};
                15: namajtime= {data[k][59],data[k][58],data[k][57],data[k][56]};
                16: namajtime= {data[k][63],data[k][62],data[k][61],data[k][60]};
                17: namajtime= {data[k][67],data[k][66],data[k][65],data[k][64]};
                18: namajtime= {data[k][71],data[k][70],data[k][69],data[k][68]};
                19: namajtime= {data[k][75],data[k][74],data[k][73],data[k][72]};
                20: namajtime= {data[k][79],data[k][78],data[k][77],data[k][76]};


   endcase
 end
endtask
```

```
//Task for ULN output
task ulnoutput(input integer ledno, output [30:0]  lightonoff  );
 begin
   case (ledno)
      1: lightonoff=30'b000000000000000000000000000001;
      2: lightonoff=30'b000000000000000000000000000010;
      3: lightonoff=30'b000000000000000000000000000100;
      4: lightonoff=30'b000000000000000000000000001000;
      5: lightonoff=30'b000000000000000000000000010000;
      6: lightonoff=30'b000000000000000000000000100000;
      7: lightonoff=30'b000000000000000000000001000000;
      8: lightonoff=30'b000000000000000000000010000000;
      9: lightonoff=30'b000000000000000000000100000000;
     10: lightonoff=30'b000000000000000000001000000000;
     11: lightonoff=30'b000000000000000000010000000000;
     12: lightonoff=30'b000000000000000000100000000000;
     13: lightonoff=30'b000000000000000001000000000000;
     14: lightonoff=30'b000000000000000010000000000000;
     15: lightonoff=30'b000000000000000100000000000000;
     16: lightonoff=30'b000000000000001000000000000000;
     17: lightonoff=30'b000000000000010000000000000000;
     18: lightonoff=30'b000000000000100000000000000000;
     19: lightonoff=30'b000000000001000000000000000000;
     20: lightonoff=30'b000000000010000000000000000000;
     21: lightonoff=30'b000000000100000000000000000000;
     22: lightonoff=30'b000000001000000000000000000000;
     23: lightonoff=30'b000000010000000000000000000000;
     24: lightonoff=30'b000000100000000000000000000000;
     25: lightonoff=30'b000001000000000000000000000000;
     26: lightonoff=30'b000010000000000000000000000000;
     27: lightonoff=30'b000100000000000000000000000000;
     28: lightonoff=30'b001000000000000000000000000000;
     29: lightonoff=30'b010000000000000000000000000000;
     30: lightonoff=30'b100000000000000000000000000000;
        endcase
 end
endtask

endmodule
```

//Switch module for adjusting Day

```
module switch1 (sw , rst , counter );

input sw ;
input rst;

output [11:0] counter;
reg [11:0]  counter ;

always @(negedge sw )

 begin
   if (rst==1'b1)
     begin
       if (counter > 12'b000000011110)
         begin
          counter = 12'b000000000000;
         end

        counter = counter + 12'b000000000001  ;

     end

    else
     begin
       counter = 0;
     end

   counter = counter ;
 end
endmodule
```

// Switch module for adjusting month

```
module switch2 (sw , rst , counter );

input sw ;
input rst;

output [11:0] counter;
reg [11:0]  counter ;

always @(posedge sw )

 begin
   if (rst==1'b1)
     begin
       if (counter > 12'b000000001011)
         begin
          counter = 12'b000000000000;
         end

        counter = counter + 12'b000000000001  ;

     end

    else
     begin
       counter = 0;
     end

   counter = counter ;
 end
endmodule
```

// Switch module for adjusting minutes

```
module switchmin (sw , rst , counter );

input sw ;
input rst;

output [11:0] counter;
reg [11:0]  counter ;

always @(posedge sw )
 begin

  if (rst==1'b1)
    begin
      if (counter > 12'b000000111011)
        begin
         counter = 12'b000000000000;
        end

       counter = counter + 12'b000000000001  ;

    end

   else
    begin
      counter = 0;
    end

  counter = counter ;
 end
endmodule
```

```
// Switch module for adjusting Hour
module switchhr (sw , rst , counter );
input sw ;
input rst;

output [11:0] counter;
reg [11:0]  counter ;


always @(posedge sw )
 begin
   if (rst==1'b1)
     begin
       if (counter > 12'b000000001011)
         begin
          counter = 12'b000000000000;
         end

        counter = counter + 12'b000000000001  ;

     end

    else
     begin
       counter = 0;
     end

   counter = counter ;
 end
endmodule
```

Appendix B

Altera FLEX Expansion Slots

There are Three Expansion slots in FLEX 10K device, FLEX_EXPAN_A, FLEX_EXPAN_B & FLEX_EXPAN_C. Each Expansion slot is dual rows of 0.1-inch spaced holes for accessing signal I/O pins and global signals on the FLEX 10K device, power, and ground. Figure A-B-1 shows the numbering convention for these holes.

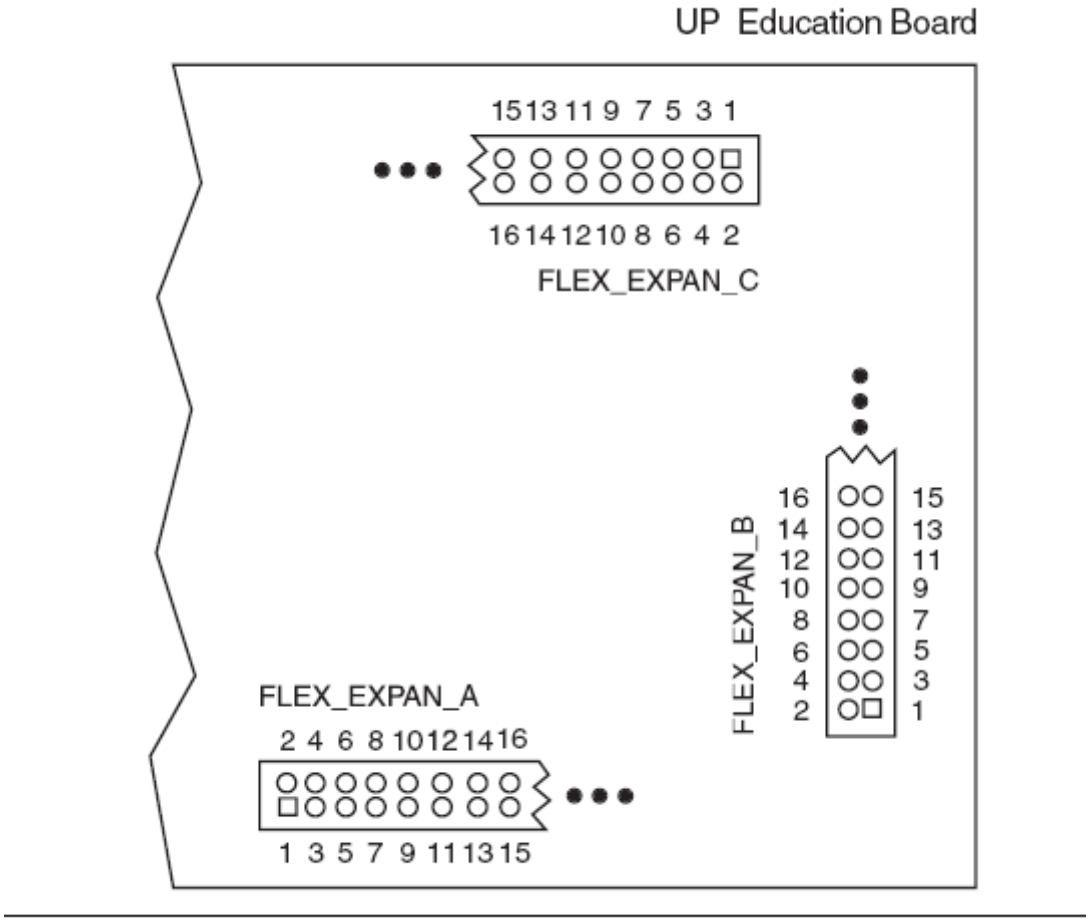


Figure A-B-1 FLEX_EXPAN_A, FLEX_EXPAN_B & FLEX_EXPAN_C Numbering Convention

There are 240 pin in Flex 10K devices. The description about pins of FLEX expansion slot is given in Table A-B-1, A-B-2 and A-B-3.

Table A-B-1 FLEX_EXPAN_A Signal Names & Device Connections

Hole Number	Signal/Pin	Hole Number	Signal/Pin
1	RAW	2	GND
3	VCC	4	GND
5	VCC	6	GND
7	No Connect	8	DI1/90
9	DI2/92	10	DI3/210
11	DI4/212	12	DEV_CLR/209
13	DEV_OE/213	14	DEV_CLK2/211
15	45	16	46
17	48	18	49
19	50	20	51
21	53	22	54
23	55	24	56
25	61	26	62
27	63	28	64
29	65	30	66
31	67	32	68
33	70	34	71
35	72	36	73
37	74	38	75
39	76	40	78
41	79	42	80
43	81	44	82
45	83	46	84
47	86	48	87
49	88	50	94
51	95	52	97
53	98	54	99
55	100	56	101
57	VCC	58	GND
59	VCC	60	GND

Table A-B-2 FLEX_EXPAN_B Signal Names & Device Connections

Hole Number	Signal/Pin	Hole Number	Signal/Pin
1	RAW	2	GND
3	VCC	4	GND
5	VCC	6	GND
7	No Connect	8	DI1/90
9	DI2/92	10	DI3/210
11	DI4/212	12	DEV_CLR/209
13	DEV_OE/213	14	DEV_CLK2/211
15	109	16	110
17	111	18	113
19	114	20	115
21	116	22	117
23	118	24	119
25	120	26	126
27	127	28	128
29	129	30	131
31	132	32	133
33	134	34	136
35	137	36	138
37	139	38	141
39	142	40	143
41	144	42	146
43	147	44	148
45	149	46	151
47	152	48	153
49	154	50	156
51	157	52	158
53	159	54	161
55	162	56	163
57	VCC	58	GND
59	VCC	60	GND

Table A-B-3 FLEX_EXPAN_C Signal Names & Device Connections

Hole Number	Signal/Pin	Hole Number	Signal/Pin
1	RAW	2	GND
3	VCC	4	GND
5	VCC	6	GND
7	No Connect	8	DI1/90
9	DI2/92	10	DI3/210
11	DI4/212	12	DEV_CLR/209
13	DEV_OE/213	14	DEV_CLK2/211
15	175	16	181
17	182	18	183
19	184	20	185
21	186	22	187
23	188	24	190
25	191	26	192
27	193	28	194
29	195	30	196
31	198	32	199
33	200	34	201
35	202	36	203
37	204	38	206
39	207	40	208
41	214	42	215
43	217	44	218
45	219	46	220
47	221	48	222
49	223	50	225
51	226	52	227
53	228	54	229
55	230	56	231
57	VCC	58	GND
59	VCC	60	GND

Appendix C

Using Quartus II

To implement the circuits that will be designed on the CPLD there are few key steps.

1. Write a program using Verilog HDL.
2. Compile the code.
3. Correct any syntax errors.
4. Simulate the circuit to make sure that you are getting the behavior you expect.
5. Download the program onto the CPLD.
6. Test the operation of circuit.

Quartus II helps to implement all of the above easily. The following sections describe how to do those basics

Steps:

1.1 Start New Project

1. Select File > New Project Wizard.
2. Set the directory name. Make sure that you create a new project for each project and do not just copy the directory over.
3. Set the name of the project. It will be simple if you name it by the lab name, e.g., lab1.
4. Click Yes to create the directory if it does not exist.
5. You can add existing files if you have already them, otherwise select Next.
6. Next you need to specify the device that you are using. Set the Device Family to MAX II and select EPM2210F324C3.
7. Press Next.
8. Press Finish.

1.2 Writing the Code

1. Select File > New.
2. Choose Verilog HDL File.
3. Click Ok.
4. Select File > Save As. For one file project name, the name of the file should be the same as the project. In addition, the module name should be the same as the filename
5. Choose Save as type, and select Verilog HDL File.
6. Put a check-mark in the box Add file to current project. Unless the file is part of the project you won't be able to proceed. If you don't add the file now you cal latter added by selecting Project > Add/Remove Files in Project.
7. Click Save.
8. Now you are ready to type in your program.

1.3 Compiling

1. Select Processing > Start Compilation, or click on the play icon on the toolbar.
2. Click on Processing > Compilation Report to see the results of the compilation.
3. If there are no errors, then program is correct from the syntax point of view. Still the program may have some logic errors that the compiler will not be able to detect.

1.4 Pin Assignment

Now it is necessary to specify which pins of the CPLD are connected to which inputs and which outputs. Some pins have already been wired to the LEDs and the push buttons. A list of those pins are provided in Table 1.

1. Select Assignments > Assignment Editor.
2. Under Category select Pin.
3. Double click on <<new>>, a drop-down menu will appear, select the input or output as required.

4. In the column labeled Location select the required pin. Table 1 shows the locations of hardwire for evaluation board.
5. Repeat steps 3 and 4 to assign all the inputs and outputs of your circuit.
6. The Altera default is that all unused pins should be assigned “As outputs driving ground”. This is a good choice for pins not connected to anything (it reduces power and noise), but is not good for pins which may be connected to, say, a clock input – then have both the clock and the Altera chip trying to drive this input. A safer choice is to define all unused pins As input tri-stated with weak pull-up resistor. To do this,
    a. Go to Assignments > Device
    b. Click on Device and Pin Options
    c. Select the Unused Pins tab
    d. From the Reserve all unused pins: drop-down menu, select As input tri-stated with weak pull-up resistor.

Table A-C-1 Pin Assignments for the LEDs, buttons, and clock input

Signal Name	CPLD Pin No.	Description
LED[0]	PIN_U13	Blue LED
LED[1]	PIN_V13	Green LED
LED[2]	PIN_U12	Yellow LED
LED[3]	PIN_V12	Red LED
LED[4]	PIN_V5	Blue LED
LED[5]	PIN_U5	Green LED
LED[6]	PIN_V4	Yellow LED
LED[7]	PIN_U4	Red LED
KEY[0]	PIN_U15	Button1
KEY[1]	PIN_V15	Button2
KEY[2]	PIN_U14	Button3
KEY[3]	PIN_V14	Button4
CLOCK_50	PIN_J6	50 MHz clock input

1.5 Simulating the Designed Circuit

When simulating a circuit it is necessary to figure out the waveforms for the inputs that will make us confident that our circuit works. If we have a simple circuit, we can easily test all the possibilities. As the circuit gets more and more complicated we will need to figure out a scheme to verify its operation. In simulating our circuit there are three main steps.

1. Create a waveform file.
2. Select your inputs and outputs.
3. Create a waveform for each input.
4. Run the simulation to generate the output for verification with your expected results.

A detailed explanation of the above steps are described below.

1. Select File > New.
2. Click on Vector Waveform File.
3. Click Ok.
4. Save the file using some meaningful name, filename.vwf.
5. Set the desired simulation time by selecting Edit > End.
6. Select View > Fit in Window.
7. Select the inputs and outputs to observe by clicking Edit > Insert > Insert Node or Bus.
8. Click on Node Finder.
9. Click on the input or output to observe and click on the > sign. Repeat this process for all inputs and outputs.
10. The next step is to specify the logic value of each of the inputs which are selected and the duration of that value.
11. Save the file, e.g. lab2.vwf.
12. After the waveforms have been defined, we can simulate our circuit. There are two types of modes that we are concerned with.
    i. Functional: we are not worried about the delays and we are interested to make sure that logically the circuit is working;

ii. Timing: we simulate the circuit and include the delays in all the gates. First perform the functional simulation and then perform the timing. To select the mode of the simulation:
    a. Select Assignments > Settings.
    b. Click on Simulator Settings.
    c. Choose Functional or Timing. For now choose Functional unless otherwise instructed.

13. Create the required Netlist that the waveform file will be applied to by selecting Processing > Generate Functional Simulation Netlist.
14. Run the simulation by clicking Processing > Start Simulation, or by clicking on the play icon in the simulation waveform window.

1.6 Programming the CPLD

The final step is to program the CPLD with your designed circuit.

1. Select Tools > Programmer.
2. Select JTAG in the Mode box.
3. If USB-Blaster is not chosen in the box next to the Hardware Setup, selected by clicking on the Hardware Setup.
4. Now we can see a file listed with extension .pof / .sof, if not add it.
5. Finally, press Start. The program will download on respective board and once it is finished then it can be test your circuit in hardware.