FPGA BASED SUPERVISORY CONTROL AND DATA ACQUISITION SYSTEM

A DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF THE DEGREE OF

MASTER OF ENGINEERING

IN

CONTROL & INSTRUMENTATION

SUBMITTED BY

SHWETA SHARMA

(Roll NO. 3344)

UNDER THE ESTEEMED GUIDANCE

OF

Dr. PARMOD KUMAR

(PROFESSOR & HEAD)

DEPARTMENT OF ELECTRICAL ENGINEERING
DELHI COLLEGE OF ENGINEERING
UNIVERSITY OF DELHI
2004-2005

CERTIFICATE

It is certified that Shweta Sharma, Roll No.3344, student of M.E, Control and Instrumentation, Department of Electrical Engineering, Delhi College of Engineering, has submitted the dissertation entitled **"FPGA based Supervisory Control and Data Acquisition System"**, under my guidance towards partial fulfilment of the requirements for the award of the degree of Master of Engineering (Control & Instrumentation Engineering).

This dissertation is a bonafide record of project work carried out by her under my guidance and supervision. Her work is found to be excellent and her discipline impeccable during the course of the project.

I wish her success in all her endeavors.

Date:

(Dr. Parmod Kumar)
Professor & Head
Deptt. of Electrical Engineering
Delhi College of Engineering
Delhi -110042

ACKNOWLEDGEMENTS

Any accomplishment requires the efforts of many people and this work is no exception. I appreciate the contribution and support, which various individuals have provided for the successful completion of this dissertation. It may not be possible to mention all by name but the following were singled out for their exceptional help.

My primary thanks goes to my project guide **Dr. Parmod Kumar**, Professor & Head, Department of Electrical Engineering, DCE, who provided me an opportunity to work under his guidance. His scholastic guidance and sagacious suggestions helped me to complete the project in this advanced field. His immense generosity and affection bestowed on me goes beyond his formal obligations as guide.

I would like to thank **Mr. Vishal Verma**, Assistant professor, Department of Electrical Engineering, DCE for his perpetual encouragement, generous help and inspiring guidance.

I express my gratitude to **Mrs. Rajeshwary Pandey**, Lecturer, Department of Electronics and Communication, DCE for her kind help and cooperation.

My sincere thanks to **Mr. Ajay J.**, Customer Support Engineer, Silicon Micro Systems (SIMS), for his initial help that eventually led to this project.

I am grateful to **Mr. Rajen Bhatt**, Research Scholar, IIT Delhi for his kind cooperation and guidance, whenever needed.

I want to express my regards to **Dr. Ashok De**, Head, Computer Centre, DCE for providing the internet access, without which everything could have been difficult.

I am thankful to **Mr. R.K Shukla**, Librarian, DCE, for facilitating me unconditionally with various literary resources.

I also want to say thanks to **Mr. Karan Singh**, Laboratory Assistant, Department of Electrical Engineering, DCE for his kind cooperation during the development of project.

I would like to extend my sincere appreciation to **Ms. Beena Antony** for reviewing this report and providing valuable comments and thoughtful criticism, which have resulted in an improved version.

I want to say thanks to my friends **Sandeep Sharma**, **Zelalem Girma**, **Sunita Verma**, **Supriya Sharma**, **Shraddha Singhai**, **Payal Singla and Anshu Dev** for their support in all my endeavors.

There are times in a project when the clock beats our time and we run out of energy, wishing to finish it once and forever. **My family** made me endure such times with their unconditional support, love and unfailing humour.

Date: **(Shweta Sharma)**

ABSTRACT

A prototype Supervisory Control and Data Acquisition (SCADA) System has been designed and programmed into the ACEX50K FPGA chip. The designed system provides the basic facilities of SCADA along with the advantages of great speed, high accuracy, negligible & predictable delay, no mechanical components, purely digitalized system facilitated by the FPGA. FPGA has excellent logic capabilities, enormous processing resources, and very high clock speed, therefore suitable for implementing data capture system.

The system is implemented using UVLSI-201 trainer kit and a general-purpose input-output board. The analog data is acquired from eight multiplexed external channels provided on the GPIO board. Selecting a particular channel and addressing it for reading data, is the responsibility of FPGA. As FPGA can deal only with digital information, the analog data is converted to the digital form by ADC 0808 provided on the GPIO board. This ADC is FPGA controlled and the digital data is stored in FPGA by programming the device.

The data is displayed on the GPIO board in hexadecimal form using multiplexed 4 digits, seven segment display. Data is processed and checked for any limit violation within the chip itself. If any limit violation is there, output LED glows, indicating a fault in the system. Also, a 5-volt analog control signal is generated to check the fault.

The system has been modeled using VHDL, a hardware description language**.**

CONTENTS

Page No.

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Introduction

SCADA (Supervisory Control and Data Acquisition) system is an intelligent system, which provides the facility of continuously monitoring, supervising and controlling the process plant. The main components of the SCADA system are [32]:

(1) Master Computer Station or Master Terminal Unit (MTU)

(2) Remote Terminal Unit (RTU)

(3) Communication Media

(4) Human Machine Interface (HMI)

Each component is discussed briefly below:

(1) Master Computer Station or Master Terminal Unit (MTU)

The main incentive for the process control is the optimization of the plant's economic performance. For performance analysis on the process plant, the information from the distributed RTUs should reach a central location where it can be consolidated and analyzed to generate the reports on the plant performance. The analysis may include histogram generation, standard deviation calculation, plotting one parameter with respect to another and so on. Software can be written depending on the type of analysis required.

Many times the human operator cannot find the best operating policy for a plant, which will minimize the operating cost. This deficiency is due to the enormous complexity of a typical process plant. Therefore, to analyze the situation and find out the best policy, the speed and the programmed intelligence of the digital computer is used; this computer is called Master Computer Station or Master Terminal Unit. It monitors, controls and coordinates the activities of various RTUs and provides the supervisory control facility to the process plant. The MTU is located at the operator's central control facility and provides a man-machine software interface, two-way data communication.

(2) Remote Terminal Unit (RTU)

RTU is a field interface device, which collects information from the machine that is to be monitored. The RTUs are basically nodes of the distributed SCADA

system that are located at a remote site to gather data from field devices like pumps, valves, alarms etc. They should be rugged and able to work unattended for a long duration. Since these RTUs have to operate for a long duration unattended, the basic requirements would be that they consume minimum power and have considerable self diagnostic capability. There are two modes in which remote terminal units work: Under command from central computer and stand alone mode.

The RTUs should have some special software facilities, which are mentioned below:

(i) Quiescent Mode Operation: Since the transmitter consumes maximum power in RTU, especially in terrestrial and satellite communications, it is switched on only when the RTU has some information packets ready for sending. The RTU receives all the information from the central computer. Since the receiver is kept on all the time, this information is received and proper action is initiated. The quiescent mode saves considerable amount of power for RTUs.

(ii) Downloading of Limits from Central Computer: Generally the RTUs behave much like stand alone SCADA. They collect the data from various sensors, perform signal conditioning, filtering, conversion to engineering units and store them in the memory. They also perform the limit checking on these values and inform central computer on violation of limits, if any, immediately. Since these RTUs are at remote locations it should be possible to change these limits remotely from central computer. This is called downloading of limits. The central computer makes a special request to the RTU to change the limits. The RTU then enters in a special mode for the change of basic parameters and performs the function in an interactive way.

(iii) Exceptional Reporting: The RTUs normally have intelligence to perform all the functions including limit checking and when the limits are violated the central computer is informed. The other message that goes regularly is regarding all well condition of RTUs. Thus RTUs perform self diagnosis by executing different diagnostic software.

(3) Communication Media

Communication Media is an important component of the SCADA system and has the interface available with 2-wire/4-wire communication line. The communications media transmits the information from RTU to central computer or in the reverse direction. The way the MTU/RTU transmission network or topology is set up can vary, but the system must feature uninterrupted, bi-directional communication in order to properly function. Methods to accomplish this include private medium, where the end user owns, operates, licenses and services the medium or public medium, where the customer pays for a monthly, per time or volume use [32]. Following are the basic communication strategies that are used depending on the application need:

(i) Wireline Communications: The wireline communication may have a number of options and these options can be selected depending upon the distance between central computer and RTU. It is usually limited to low bandwidth applications. These options are enlisted below:

RS232C/442: RTU can support communication via standard RS232C/442. The I/O ports can select the average levels as well as the baud rates.

Switch Line Modem: When the user wants to use the existing telephone lines for communication, the switch line modem can be effective. In such cases, RTUs contain the facilities like auto answer, auto dial and auto select baud rates. The modem is ideal for data networks configured in time or event reporting RTUs.

2-Wire or 4-Wire Communication: The modem residing in the RTU can be configured to 2 or 4-wire communication on dedicated lines.

(ii) Wireless Communications:

UHF/VHF radio: The RTU may support a complete line of UHF/VHF terrestrial radios. UHF/VHF radio is an electromagnetic transmission with frequencies of 175MHz-450MHz-900MHz received by special antennas. Its coverage is limited to special geographical boundaries.

The communication protocol is transparent to the user and supports CRC intelligence, error checking, and packet protocol for error free data transmission.

Microwave Radio: Microwave Radio transmits at high frequencies through parabolic dishes mounted on towers or on top of buildings. This media uses point-

to-point, line-of-sight technology and communication may become interrupted sometimes due to misalignment and/or atmospheric conditions.

Satellite Communications: In the applications where wireline and terrestrial radio communications are impossible or cost prohibitive, the satellite communication may be desirable.

Some of the RTUs provide the facility to be interfaced to one-way or two way satellite communication using Very Small Aperture Terminal (VSAT). These terminals use one meter antennas and have data rates from 50 to 60 kbps.

Fiber-Optic Communications: For applications where electromagnetic interferences or hazardous electrical potentials exist, the RTUs can be networked using fiber-optic cables.

(4) Human Machine Interface (HMI)

For efficient process monitoring and control, effective communication is necessary between the process operator and the process to be automated [4]. The man-machine dialogue between the process operator and the automation system is carried out with the Human Machine Interface. HMI allows operators to view the state of any part of the plant equipment.

The employment of an easy-to-use SCADA software package on PC, known as the human machine interface, provides a reliable representation of the real system at work [32]. An HMI allows the operator to view virtually all system alerts, warnings, urgent messages and functions as well as change set points and analyze, archive or present data trends. Some common HMI software packages are Cimplicity (GE-Fanuc), RSView (Rockwell Automation), IFIX (Intellution) and InTouch (Wonderware). Most of these software packages use standard data manipulation/presentation tools for reporting and archiving and integrate well with Microsoft Excel, Access and Word.

1.2 Benefits of SCADA

SCADA is an industrial measurement and control system and has become the backbone for monitoring, controlling and meeting the desired objectives of the process plant. The major productivity issue facing the process industries today is plant automation i.e. the development of coordinated plant control system.

The process industries are going for automation to maintain their competitive edge. Some of the benefits provided with the SCADA are given below [26]:

- A properly designed SCADA system saves time and money by eliminating the need for service personnel to visit each site for inspection, data collection/ logging or make adjustments.
- Provides the facility of real-time monitoring, system modifications, troubleshooting, automatic report generating
- Reduces operational costs
- Provides immediate knowledge of system performance
- Improves system efficiency and performance
- Increases equipment life and reduces costly repairs
- Reduces the number of man-hours (labor costs) required for troubleshooting or service and frees up personnel for other important tasks
- Facilitates compliance with regulatory agencies through automated report generation

1.3 Introduction to FPGA

FPGAs are one of today's most important digital logic implementation options. An FPGA is a general purpose, multilevel, programmable logic device that is customized in the package by the end users. An FPGA consists of an array of programmable logic blocks and a programmable routing network. The programmable interconnect between blocks allows users to implement multi level logic, removing many of the size limitations of the PLD derived two level logic structure. This extensible architecture can currently support thousands of logic gates at system speed in the tens of megahertz [13].

The size, structure, number of logic blocks and connectivity of the interconnect vary considerably among the architectures. This difference in architectures is driven by different programming technologies and different target applications of the parts.

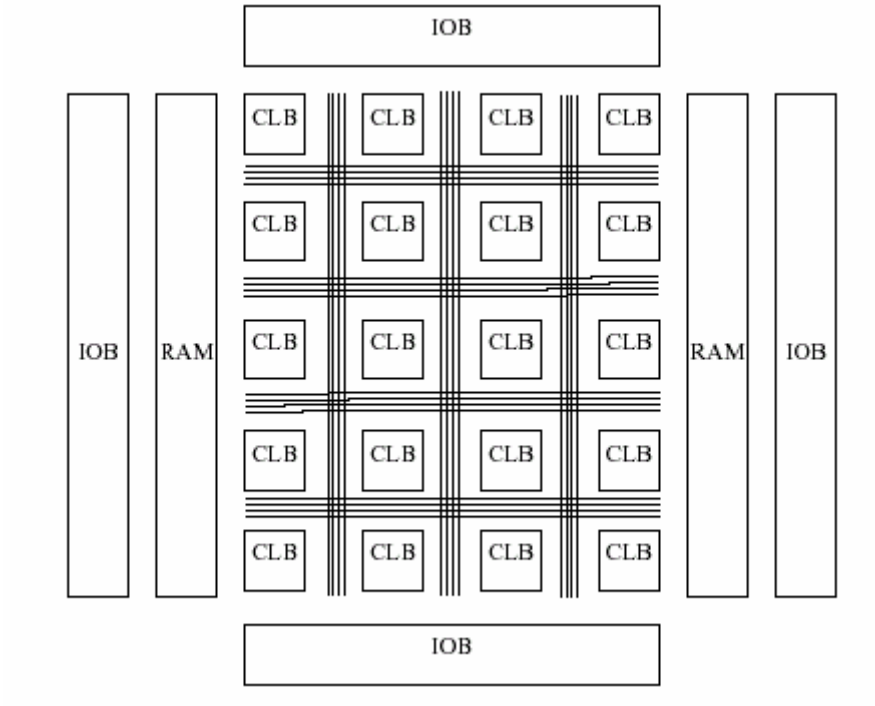


Figure1.1. Basic Architecture of FPGA

FPGAs offer the benefits of both programmable logic arrays and gate arrays. They implement thousands of gates of logic in a single integrated circuit. FPGAs are programmable by designers at their site, eliminating the long delays and tooling costs. These advantages have made FPGAs very popular [13].

1.4 Advantages of FPGAs

1) Low Tooling Costs

There is no custom tooling required for an FPGA, so there are no associated tooling costs, making FPGA cost effective for most logic designs.

2) Rapid Turnaround

An FPGA can be programmed in a few minutes. On an FPGA, a modification to correct a design flaw or to address a late specification change can be made quickly and cheaply. Faster design turnaround leads to faster product development and shorter time to market for new FPGA products.

3) Low Risks

The benefits of low initial Non Recurring Engineering (NRE) charges and rapid turnarounds mean that design iteration due to an error incurs neither a large expense nor a long delay. Low cost encourages early system integration and

prototyping. The low cost of error encourages more aggressive logic design, which may yield better performance and more cost effective designs.

4) Effective Design Verification

Instead of simulating large amounts of time, FPGA user may choose to use in circuit verification. Designers can implement the design and can use any functioning part as a prototype. The prototype operates at full speed and with excellent timing accuracy. A prototype can be inserted into the system to verify functionality of the system as a whole, eliminating a class of system errors early.

5) Low Testing Cost

All ICs must be tested to verify proper manufacturing and packaging. The test program for FPGAs is the same for all design and test the FPGA for all users of the part. Because there is only one test program, it is reasonable to invest a considerable amount of effort in it and it can be continually improved over the lifetime of the FPGA. The resulting test program achieves excellent test coverage leading to high quality ICs. The manufacturer's test program verifies that every FPGA will be functional for all possible designs that may be implemented on it. FPGA users are not required to write design specific test for their designs. Therefore, designers need not built the testability into the design eliminating "design for testability" and the design effort and overhead associated with it.

6) Life Cycle Advantages

The cost effectiveness of FPGAs in low volume and the flexibility provided by field programmability provide advantages over all phases of product lifetime. When introducing a product, an FPGA user may order a few parts at a time while testing the design for functionality and the product for market viability. During production, the FPGA user can accommodate rapid changes in sales easily because long lead times are not required. An FPGA user can make enhancements by shipping an upgraded design on the same FPGA device. This upgrade requires no inventory changes, no new hardware and does not interrupt production.

1.5 Scope of FPGA in SCADA

Development in the field of FPGAs has provided great logic capabilities, enormous processing resources, significant on-chip independent RAM banks and very high clock speeds. FPGAs are therefore suitable for implementing data

capture systems [23]. With the basic understanding of FPGA capabilities, I consider utilizing an FPGA based SCADA system [32] over a basic RTU or a proprietary system for the following reasons:

- FPGAs, like Master Terminal Units can continuously collect, process and store data, operating independently from the MTU through intelligent programming.
- FPGAs can provide security and monitoring of door switches, heat and motion detectors. Managers/operators can be informed 24 hours a day through automatic e-mail, paging and dial-up call features.
- Multiple users can easily be added and, if open architecture protocol is used, future equipment can easily be integrated. Since FPGAs have no moving parts, they are extremely reliable.
- FPGA based SCADA system can reduce the number of man-hours needed for on-site visual inspections, adjustments, data collection and logging. Continually monitoring and troubleshooting potential problems, increases equipment life, reduces service calls, reduces customer complaints and increases system efficiency. Simply put, FPGA based SCADA systems are an excellent means for process control facilities to save time and money.
- Redesigning of the SCADA system can be done easily according to the needs, as FPGA is reprogrammable.
- FPGA based SCADA system mainly consists of digital components and therefore it is more accurate and reliable.

Engineered SCADA systems today not only control processes but are also used for measuring, forecasting, billing, analyzing and planning [32]. Today's SCADA system must meet a whole new level of control automation while interfacing with yesterday's obsolete equipment yet remain flexible enough to adapt to tomorrow's developments.

1.6 FPGA based SCADA System Overview

FPGA are revolutionizing the way system designers implement logic [3]. FPGAs provide a new capability that facilitates simple and trouble free implementation of digital systems.

An FPGA based SCADA system has been designed in this project. The system utilizes the great logic capability and re-programmability of the FPGA to design the SCADA system. The project is divided into three parts; Data Acquisition, Processing and Control.

The system has been designed and implemented using the UVLSI 201 trainer. This universal PLD kit is an ideal trainer to implement and test simple and complex designs. It is possible to execute and verify digital experiments on this kit using VHDL, Verilog, AHDL, the standard hardware description languages. It is an assembled ready for various interfaces that include ADC/DAC, display, keyboard, serial communication, VGA, PS/2.

For data acquisition, general purpose input-output board is used. It has almost all the primary interfaces that a PLD may be used for. This board is designed to interface PLDs of any company, any gate count and any package. Input switches are provided to give steady state inputs and LEDs can indicate high or low outputs.

GPIO board has eight channels to communicate with outside world. Channels, from which data is to be captured, are interfaced with GPIO. FPGA is a digital device, it can handle only digital data; therefore the analog data is converted into the digital form by using a successive approximation ADC. SAR ADC0808 is provided on the GPIO board. It receives all its control signals e.g. start conversion, address latch enable etc. from the FPGA. The digital data is stored in the FPGA chip.

The data is processed for control purpose, in the chip itself. Software is written depending upon the type of analysis required in the system. Any fault in the system e.g. limit violation is indicated by an LED. FPGA is programmed using the VHDL, a hardware description language.

1.7 Objectives of the Study

The present study sets the following objectives:

- An exhaustive study of FPGA
- Scope of FPGA in SCADA
- Designing of FPGA based SCADA system

1.8 Dissection of Dissertation

Chapter 2 presents the literature review. It explains the developments and advancements in SCADA systems in chronological order. Then it describes how FPGAs emerged and became the greatest logic implementation device.

Chapter 3 describes the different types of ICs i.e. Standard Logic ICs, Application Specific ICs, PLDs and compares their architecture, functionality and applications. PLDs have been further classified into SPLDs, CPLDs, FPGAs and each is briefly covered. Advantages and disadvantages of PLDs are listed.

Chapter 4 considers the FPGA in detail. Essential characteristics, different technologies and applications of FPGAs are explained.

ACEX50K device, used in this project is explained. Salient features, architecture, functionality, input-output capabilities, operating conditions, power consumption and pin configuration have been considered. Detailed functional description of Logic Array Block, Embedded Array Block, Logic Element, I/O Element is presented.

Chapter 5 deals with the Altera QuartusII software used for programming the ACEX device. Steps to use this software for implementing any digital logic are defined. Salient features, design process and need of VHDL are explained.

Chapter 7 introduces the features of UVLSI 201 trainer kit, which make it extremely convenient for testing and implementing the VLSI designs.

Chapter 8 describes the features of the general-purpose input-output board (GPIO) and its capabilities to communicate with outside world.

Chapter 9 A newer approach to the design and implementation of the Supervisory Control and Data Acquisition system has been discovered. It uses an FPGA for data acquisition, processing and control. The details of each stage; data acquisition, processing and control are presented.

Results and Discussion

Conclusion

Scope for Future Work

Appendix: VHDL source code to implement the FPGA based SCADA system

References

CHAPTER 2

Literature Review

Supervisory Control and Data Acquisition (SCADA) is a process control system that enables a site operator to monitor and control processes distributed among various remote sites.

A properly designed SCADA system saves time and money by eliminating the need for service personnel to visit each site for inspection, data collection/logging or make adjustments. Real-time monitoring, system modifications, troubleshooting, increased equipment life and automatic report generating are just a few of the benefits that come with the SCADA systems [26].

2.1 SCADA: A brief history

SCADA began in the early 1960s as an electronic system operating as input/output transmissions between a master station and a remote station. The master station would receive the data from remote station through a telemetry network and then store the data on mainframe computers[32].

In the early 1970s, distributed control systems (DCS) were developed to control separate remote subsystems. They have similar functions to SCADA systems, but the field data gathering or control units are usually located within a more confined area. Communications may be via a local area network (LAN), normally reliable and high speed.

In 1977, John Muench, Chairman and Chief Executive Officer, Advanced Control Systems, delivered the industry's first microprocessor based master station and first microprocessor based RTU[28].

In the 1980s, with the development of the microcomputer, process control could be distributed among remote sites. Further development enabled DCS to use programmable logic controllers (PLC), which have the ability to control sites without taking direction from a master.

In the late 1990s, SCADA systems were built with DCS capabilities and systems were customized based on certain proprietary control features built in by the designer. With the internet being utilized more as a communication tool, SCADA and telemetry systems are using automated software with certain portals to download information or control a process.

In 1997, Advanced Control Systems delivered the first SCADA system with DNP implemented in Master Station and RTUs [28].

In 2000, F Morgan, T Bennett, A Shearer, M Redfern, Communications and Signal Processing Research Unit, Department of Electronic Engineering, National University of Ireland, Galway, implemented an "FPGA-based Time Resolved Data Acquisition System for Astronomical and Other Applications".

They described a programmable FPGA-based high-resolution, time resolved photon image capture system which supported current and future generations of astronomical photometry, biological and a range of SCADA applications. The system recorded and time stamped photon data arriving from a number of detectors and transmitted this data to an archive device for post processing. Time resolved data acquisition functionality was implemented and verified using the RC1000-PP Xilinx FPGA-based development platform and Handel-C programming environment [23].

2.2 FPGA: A brief history

One of the most significant components in early digital computers was the magnetic core [16]. This tiny doughnut shaped ferrite material was used from the 1950s through the 1970s to construct the main memory of large computers. Each of these cores could store a binary bit of information by using the direction of magnetization of the core to indicate a 0 or a 1. The direction of the magnetic field inside the core could be changed by controlling the direction of current through the wires that were wound around a portion of the core. For many years, magnetic core storage was the dominant type of main memory for the computer.

As the technology for core storage was improved, the price continued to drop and some impressive computers became available. The IBM system 360 appeared in 1965 with one scientific model capable of storing about 64 million bits in its main memory. This system was sold for a price that varied between $1,000,000 and $2,000,000, depending on several options. The magnetic core cost about 1 to 2 cents per bit, wired into the memory. The 64-megabit storage system added approximately $7,000,000 to the cost of the system. It was obvious that a reduction of the main memory costs would greatly reduce the overall cost of a computer. Furthermore, the core storage system required a set of high current

driver circuits that lead to high power dissipation and expensive circuit components.

Ironically, this same year (1965) saw the first proposal to use semiconductor memory. The obvious size benefits of integration led some engineers to believe that perhaps the integrated circuit might be used to produce low cost storage components. The first IC memories were more expensive and had much less storage capacity than the core memory and did not immediately replace this workhorse of the computer industry. One of the first commercial uses of a small semiconductor main memory was in IBM 360/85 in 1969.

As IC fabrication and design techniques improved over the years, the semiconductor memory became smaller and cheaper, leading to the demise of core storage. Without this development, it would be difficult to produce the highly capable personal computers and workstations that are now available.

Before the semiconductor memory was made large enough to replace the main memory of the computer, it became obvious that the small IC memory would be useful in circuit applications. Several companies implemented small memories such as 64 bit devices that were targeted for use in digital circuits rather than in computer memories. One of the first such devices was the read only memory (ROM). Small IC read write memories, called semiconductor RAMs also appeared at the same time. As the price dropped and the size increased, semiconductor memories began replacing core memories. In the late 1970s, the semiconductor memory was used almost exclusively in the personal computer. By the early 1980s, even large mainframe computers were produced with exclusively semiconductor main memories.

It became obvious in the late 1970s that the ROMs were also useful in logic function realization. As small ROMs were used for this purpose, the combinational PLA and PAL chips were developed to reduce the number of devices needed on a chip. Fabrication methods improved to allow the inclusion of the flip-flops on PLA and PAL chips in the 1980s. As industry looked for faster methods of developing digital products, the registered PLA and PAL, the PLS and the FPGA was conceived. In 1985, Xilinx company introduced the first FPGA. After this many companies like Actel, Altera launched their FPGAs in the market.

These devices became very popular in the late 1980s and continue to be significant in digital system logic design.

Today the worldwide market for programmable logic devices is about $3.5 billion, according the market researcher Gartner/Dataquest. The market for fixed logic devices is about $12 billion. However, in recent years, sales of PLDs have outpaced those of fixed logic devices built with older gate array technology. The high performance FPGAs, made with the more advanced standard cell technology are now beginning to take market share from fixed logic devices [35].

CHAPTER 3

PROGRAMMABLE LOGIC DEVICES

An electronic system designer has several options for implementing digital logic [13]. These options include Integrated Circuits (ICs), which can be broadly classified in the following categories:

- **Standard logic ICs**
- **Application Specific ICs**
- **Programmable Logic Devices**

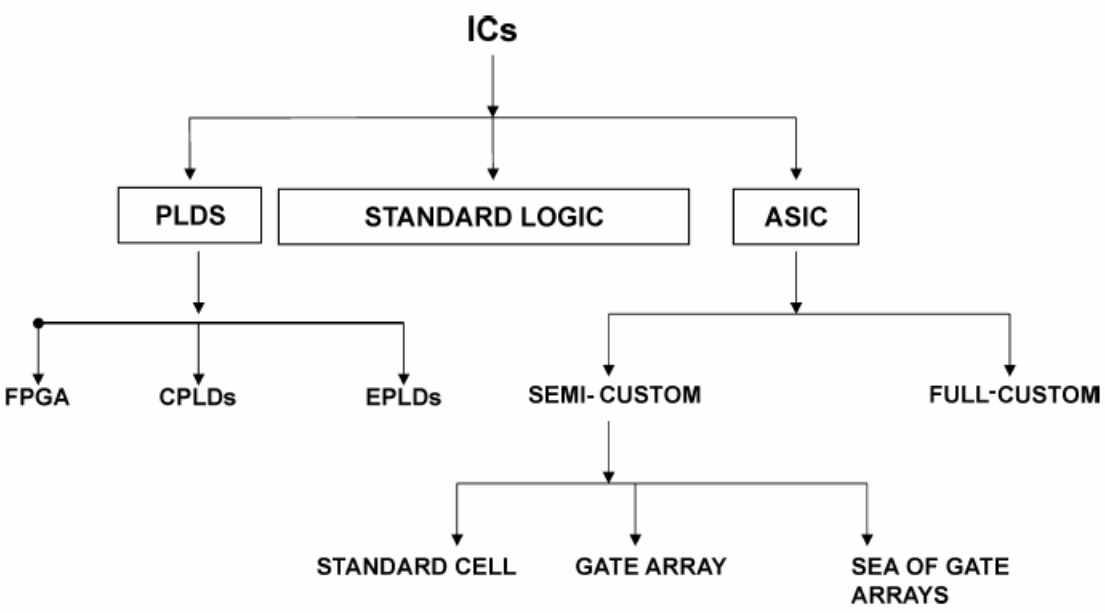


Figure3.1. ICs Classification

3.1 Standard Logic ICs

Standard logic ICs have permanent circuits built in them, they perform one specific function or set of specified functions. Once manufactured, the function of standard ICs cannot be changed. A specific logic is contained in the IC package when it is purchased and it can never be changed. The operation of Standard Logic devices depends entirely on the IC chips used and the electrical connections between chips. The designer has no access to the internal interconnections of the IC chips. In designing a digital system we must specify each IC to be used and

indicate a wiring diagram to show how each circuit is to be connected. Once the design is completed, the system performs the function intended. If it is desired to modify the function of the circuit, the design must be modified. New circuits may be needed and some connections will certainly require changes. This type of system is often referred to as a Hardwired System [16].

Examples of standard ICs include ROMs, DRAM, SRAM, Microprocessors.

With fixed logic devices, the time required to go from design, to prototypes, to a final manufacturing run can take from several months to more than a year, depending on the complexity of the device. And, if the device does not work properly, or if the requirements change, a new design must be developed. The up-front work of designing and verifying fixed logic devices involves substantial "non-recurring engineering" costs, or NRE. NRE represents all the costs customers incur before the final fixed logic device emerges from a silicon foundry, including engineering resources, expensive software design tools, expensive photolithography mask sets for manufacturing the various metal layers of the chip, and the cost of initial prototype devices. These NRE costs can run from a few hundred to several million dollars [35].

3.2 Application Specific Integrated Circuits (ASICs)

ASICs are the integrated circuits that are customized or tailored to a particular system or application rather than using standard ICs alone. These ASICs are specially designed to perform a function that cannot be done using standard components (Standard ICs). Microelectronic system design then can be done by implementing some functions using standard ICs and the remaining logic functions using one or more custom ICs [11].

Examples of ASICs include a chip for a toy bear that talks, a chip for a satellite, a chip designed to handle the interface between memory and a microprocessor for a workstation CPU and a chip containing microprocessor as a cell together with other logic.

ASICs are used in system design to improve the performance of a circuit, to reduce the volume, weight and power requirements so that it increases the reliability of a system by integrating a large number of functions on a single chip.

ASICs are classified into two types: Full Custom ASICs and Semi Custom ASICs. A full custom IC includes possibly all logic cells that are customized and all mask layers that are customized. A microprocessor is an example of a full custom IC. For semi custom ASICs all of the logic cells are pre designed and some (possibly all) of the mask layers are customized. Using pre-designed cells from a cell library makes design much easier [11].

There are many situations in which it is not appropriate to use a custom IC for each and every part of a microelectronic system. For example, if a large amount of memory is needed, it is still best to use standard memory ICs, either DRAM or SRAM, in conjunction with custom ICs.

3.3 Programmable Logic Devices (PLDs)

Programmable Logic Devices (PLDs) consist of an array of identical function cells. The cell array usually contains an AND-OR network and often includes a flip-flop. Some PLDs can perform only combinational logic functions, others can perform combinational and sequential functions.

In Programmable Logic Devices (PLDs), logic function is programmed by the user and in some cases, can be reprogrammed many times. Such a device includes array of logic elements on a chip and allows the user to specify or program many internal connections between the components on the chip. The logic elements could be various gates, inverters, buffers, flip-flops. A system configuration can be created on the chip simply by programming the chip or telling the chip where the interconnections are to be made.

A PLD can be defined as: A PLD is an IC chip that includes arrays of logic elements and allows a user to specify the connections among many of these elements [16].

Figure 3.2 shows the basic architecture of PLD.

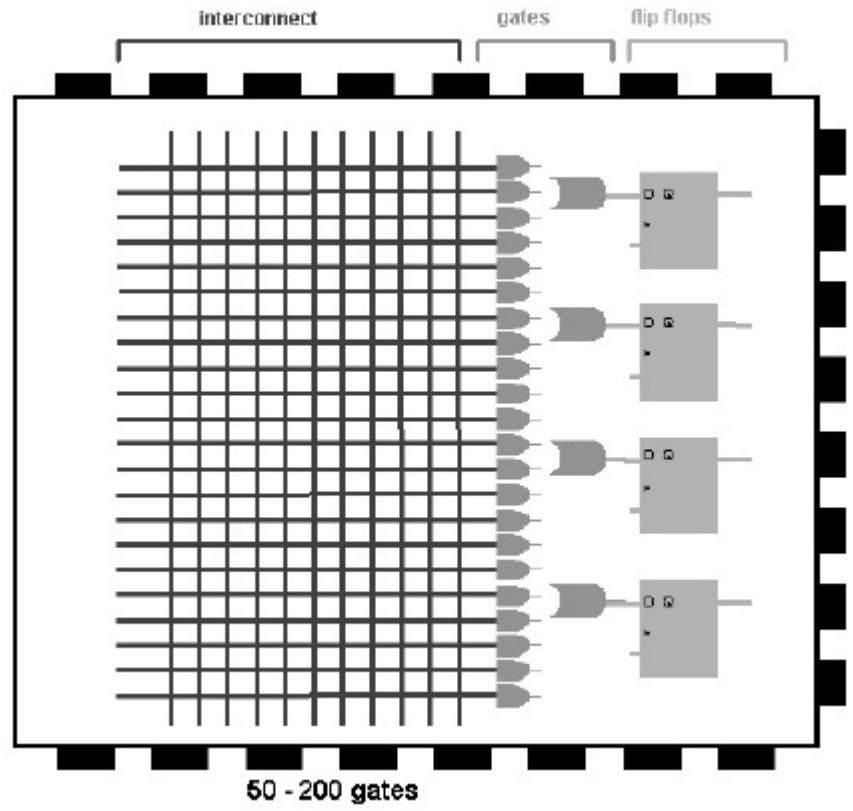


Figure3.2. Programmable Logic Device

With programmable logic devices, designers use inexpensive software tools to quickly develop, simulate, and test their designs. Then, a design can be quickly programmed into a device, and immediately tested in a live circuit. The PLD that is used for this prototyping is the exact same PLD that will be used in the final production of a piece of end equipment, such as a network router, a DSL modem, a DVD player, or an automotive navigation system. There are no NRE costs and the final design is completed much faster than that of a custom, fixed logic device [12].

3.3.1) Types of PLDs

Programmable logic devices are divided into three broad categories [5]:

3.3.1.1) Simple Programmable Logic Devices (SPLDs)

These are the least complex form of PLDs. An SPLD can replace several fixed function SSI or MSI devices and their interconnections. A few categories of SPLD are listed below:

- PAL (Programmable Array Logic)
- GAL (Generic Array Logic)

- PLA (Programmable Logic Array)
- PROM (Programmable Read only Memory)

3.3.1.2) Complex Programmable Logic Devices (CPLDs)

These have a much higher capacity than SPLDs, permitting more complex logic circuits to be programmed into them. A typical CPLD is equivalent of from 2 to 64 SPLDs. CPLDs generally come in 44-pin to160-pin packages depending on the complexity [5].

CPLDs offer logic up to about 10,000 gates. CPLDs offer very predictable timing characteristics and are therefore ideal for critical control applications. CPLDs also require extremely low amounts of power and are very inexpensive, making them ideal for cost-sensitive, battery-operated, portable applications such as mobile phones and digital handheld assistants.

3.3.1.3) Field Programmable Gate Arrays (FPGAs)

These are different from SPLDs and CPLDs in their internal organization and have greatest logic capability. FPGAs consist of an array of anywhere from 64 to thousands of logic gates groups that are called logic blocks.

Although the generalized architecture of the simpler PLDs is fairly standardized, that of FPGAs and CPLDs continues to evolve and differs considerably from one manufacturer to other [12]. The architecture has a mesh of horizontal and vertical interconnect tracks. At each junction, there is a fuse. With the aid of software tools, the user can select which junctions will not be connected, by blowing all unwanted fuses. This is done by a device programmer.

Input pins are connected to the vertical interconnect and the horizontal tracks are connected to AND-OR gates, also called product terms. These in turn connect to dedicated flip-flops whose outputs are connected to output pins.

All FPGA contain a regular structure of programmable basic logic cells surrounded by programmable interconnect. The exact type, size and the number of programmable basic logic cells varies tremendously.

3.3.2) Advantages & Disadvantages of PLDs

Fixed logic devices and PLDs both have their advantages and disadvantages. Fixed logic devices, for example, are often more appropriate for large volume applications because they can be mass-produced more economically. For certain

applications where the very highest performance is required, fixed logic devices may be the best choice [35].

However, programmable logic devices offer a number of important advantages over fixed logic devices [12]. It is desirable to use PLDs for the following reasons:

1. To decrease PC board cost by reducing the package count; 15 to 20 SSI packages can be replaced by a single package. Many more logic circuits can be stuffed into a smaller area with PLDs.
2. To improve reliability, fewer packages mean less interconnection and thus greater reliability.
3. To allow design changes as reprogramming the PLDs is less time consuming than re-designing a complete PC board using random logic or MSI logic devices.
4. During the design phase customers can change the circuitry as often as they want until the design operates to their satisfaction. That is because PLDs are based on re-writable memory technology; to change the design, the device is simply reprogrammed.
5. PLDs offer customers much more flexibility during the design cycle because design iterations are simply a matter of changing the programming file, and the results of design changes can be seen immediately in working parts.
6. To shorten design time; PLDs do not require long lead times for prototypes or production parts. The PLDs are already on a distributor's shelf and ready for shipment [12].
7. PLDs do not require customers to pay for large NRE costs and purchase expensive mask sets. PLDs suppliers incur those costs when they design their programmable devices and are able to amortize those costs over the multi-year lifespan of a given line of PLDs.
8. PLDs allow customers to order just the number of parts they need, when they need them, allowing them to control inventory. Customers who use fixed logic devices often end up with excess inventory which must be scrapped, or if demand for their product surges, they may be caught short of parts and face production delays.

9. PLDs can be reprogrammed even after a piece of equipment is shipped to a customer. Because of programmable logic devices, it is very easy to add new features or upgrade products that already are in the field. To do this, simply upload a new programming file to the PLD, via the internet, creating new hardware logic in the system.
10. Advanced process technologies help PLDs in a number of key areas: faster performance, integration of more features, reduced power consumption, and lower cost.
11. PLDs now have a growing library of intellectual property (IP) or cores. These are predefined and tested software modules that customer can use to create system functions instantly inside the PLD. Cores include everything from complex digital signal processing algorithms and memory controllers to bus interfaces and full-blown software-based microprocessors. Such cores save customers a lot of time and expense.

The disadvantage of PLDs is that the interconnections between elements on the chip must be specified or programmed [16]. Unlike conventional circuits, even after PLDs are wired into the system, they will not function properly unless they have been programmed.

CHAPTER 4

Field Programmable Gate Array (FPGA)

Introduction

Field Programmable Gate Array provides the next step in the Programmable Logic Devices hierarchy. The word Field in the name refers to the ability of the gate array to be programmed for a particular function by the user instead of by the manufacturer of the device [11]. The word array is used to denote a series of columns and rows of gates that can be configured by the end user.

All FPGAs contain a regular structure of programmable basic logic cells surrounded by programmable interconnects. The exact type, size and the number of programmable basic logic cells vary tremendously.

Essential characteristics of FPGAs:

1. None of the mask layers are customized.
2. A method for programming the basic logic cells and the interconnect is required.
3. The core is a regular array of programmable basic logic cells that can implement combinational as well as sequential logic.
4. A matrix of programmable interconnects surrounds the basic logic cells.
5. Programmable I/O cells surround the core.

4.2 Technologies Used

In all FPGAs, the interconnections and how they are programmed vary. Depending upon the application, one FPGA technology may have features desirable for that application. Currently there are following technologies in use [12]:

4.2.1) Static RAM Technology

In the Static RAM FPGAs, programmable connections are made using pass transistors, transmission gates or multiplexer that are controlled by SRAM cells. The advantage of this technology is that it allows fast in-circuit re-configuration. The major disadvantage is the size of the chip required by the RAM technology. The FPGAs are customized by loading configuration data into the internal memory cells. The FPGA can be programmed an unlimited number of times and

supports system clock rates up to 50 MHz. In the SRAM logic cell, instead of conventional gates there is a look-up-table (LUT), which determines the output based on the values of the inputs.

4.2.2) Anti-Fuse Technology

An anti-fuse resides in a high-impedance state, and can be programmed into low impedance or fused state. A less expensive than the RAM technology, this device is a program once device.

4.2.3) EPROM /EEPROM Technology

This method is the same as used in the EPROM memories. One advantage of this technology is that it can be reprogrammed without external storage of configuration, though the EPROM transistors cannot be re-programmed in-circuit. There are two basic types of FPGAs: SRAM based re-programmable and One-time programmable (OTP). These two types of FPGAs differ in the implementation of the logic cell and the mechanism used to make connections in the device.

4.3 Applications

Applications of FPGAs are so varied in embedded systems that it is impossible to generalize. Following are some instances [29]:

4.3.1) Prototyping

Many times FPGAs are used in a prototype system. A small device may be present to allow the designers to change a board's glue logic more easily during product development and testing. Or a large device may be included to allow prototyping of a system-on-a-chip design that will eventually find its way into an ASIC. Either way, the basic idea is the same, allow the hardware to be flexible during product development. When the product is ready to ship in large quantities, the programmable device will be replaced with a less expensive, though functionally equivalent, hard-wired alternative.

4.3.2) Embedded Cores

More and more vendors are selling or giving away their processors and peripherals in a form that is ready to be integrated into a programmable logic based design. They either recognize the potential for growth in the system-on-a-chip area and want a piece of the royalties or want to promote the use of their

particular FPGA by providing libraries of ready-to-use building blocks. Either way, lower system costs and faster time-to-market are achieved. We can buy an equivalent piece of virtual silicon, so there is no need to develop own hardware.

The Intellectual Property (IP) market is growing rapidly. It is common to find microprocessors and microcontrollers for sale in this form, as well as complex peripherals like PCI controllers. Many of the IP cores are even configurable. We can find the entire usual supporting cast of simple peripherals like serial controllers and timer/counter units are available as well.

4.3.3) Hybrid Chips

There is also been some movement in the direction of hybrid chips, which combine a dedicated processor core with an area of programmable logic. According to the vendors of hybrid chips, a processor core embedded within a programmable logic device will require far too many gates for typical applications. So they have created hybrid chips that are part fixed logic and part programmable logic. The fixed logic contains a fully functional processor and perhaps even some on-chip memory. This part of the chip also interfaces to dedicated address and data bus pins on the outside of the chip. Application-specific peripherals can be inserted into the programmable logic portion of the chip, either from a library of IP cores or the customer's own designs.

4.3.4) Reconfigurable Computing

As mentioned earlier, an SRAM-based programmable device can have its internal design altered on-the-fly. This practice is known as reconfigurable computing. Though originally proposed in the late 1960's by a researcher at UCLA, this is still a relatively new field of study. The decades-long delay had mostly to do with a lack of acceptable reconfigurable hardware. On-the-fly reprogrammable logic chips have only recently reached gate densities making them suitable for anything more than academic research. But the future of reconfigurable computing is bright and it is already finding a niche in high-end communications, military, and intelligence applications.

4.3.5) Other Applications

Application Specific Integrated Circuits (ASICs)

Implementation of random logic

Replacement of SSI for random logic

Onsite configuration of hardware

4.4 ACEX50K Device

Altera is a leading company in the production and advancements of PLDs. Altera ACEX EP1K50, 144 pin, TQFP FPGA has been used in the development of this project. The general and functional characteristics of this device are discussed in detail below.

4.4.1) Salient Features

1. ACEX50K FPGAs provide low cost System-On-Programmable-Chip (SOPC) integration in a single device.
    - Enhanced embedded array for implementing megafunctions such as efficient memory and specialized logic functions
    - Dual-port capability with up to 16-bit width per embedded array block (EAB)
    - Logic array for implementing general logic functions

2. High density
    - 10,000 to 100,000 typical gates
    - Up to 49,152 RAM bits, 4,096 bits per EAB, all of which can be used without reducing logic capacity

3. Cost-efficient programmable architecture for high-volume applications
    - Cost-optimized process
    - Low cost solution for high-performance communications applications

4. System-level features
    - Multi Volt TM I/O pins can drive or be driven by 2.5V, 3.3V, or 5.0V devices
    - Low power consumption
    - Bidirectional I/O performance up to 250 MHz
    - Fully compliant with the peripheral component interconnect

5. Extended temperature range

Table4.1 ACEX Device Features

Features	EP1K10	EP1K30	EP1K50	EP1K100
Typical gates	10,000	30,000	50,000	100,000
Max System Gates	56,000	119,000	199,000	257,000
Logic elements	576	1,728	2,880	4,992
EABs	3	6	10	12
Total RAM bits	12,288	24,576	40,960	49,152
Maximum user I/O pins	136	171	249	333

6. Fully compliant with the peripheral component interconnect Special Interest Group (PCI SIG) PCI Local Bus Specification.

- -1 speed grade devices are compliant with PCI Local Bus Specification
- Built-in Joint Test Action Group (JTAG) boundary scan test (BST) circuitry compliant with IEEE Std. 1149.1-1990, available without consuming additional device logic
- Operate with a 2.5V internal supply voltage
- In-circuit reconfigurability (ICR) via external configuration devices, intelligent controller or, JTAG port
- ClockLockTM and ClockBoostTM options for reduced clock delay, clock skew, and clock multiplication
- Built-in, low-skew clock distribution trees
- 100% functional testing of all devices, so test vectors or scan chains are not required

7. Flexible interconnect

- FastTrack Interconnect continuous routing structure for fast, predictable interconnect delays
- Dedicated carry chain that implements arithmetic functions such as fast adders, counters, and comparators, that are automatically used by software tools and megafunctions
- Dedicated cascade chain that implements high-speed, high-fan-in logic functions, automatically used by software tools and megafunctions
- Tri-state emulation that implements internal tri-state buses

- Up to six global clock signals and four global clear signals

8. Powerful I/O pins
    - Individual tri-state output enable control for each pin
    - Open-drain option on each I/O pin
    - Programmable output slew-rate control to reduce switching noise
    - Clamp to $V_{CCI/O}$ user-selectable on a pin-by-pin basis
    - Supports hot-socketing

9. Software design support and automatic place-and-route provided by Altera development systems for Windows-based PCs and Sun SPARCstation, and HP 9000 Series 700/800 workstations.

10. Additional design entry and simulation support provided by EDIF 200 and 300 netlist files, library of parameterized modules (LPM), DesignWare components, Verilog HDL, VHDL, and other interfaces to popular EDA tools from manufacturers such as Cadence, Exemplar Logic, Mentor Graphics, OrCAD, Synopsys, Synplicity, and Viewlogic [18].

4.4.2) General Description

Altera ACEX50K devices provide a die-efficient, low-cost architecture by combining look-up-table (LUT) architecture with EABs. LUT-based logic provides optimized performance and efficiency for data-path, register intensive, mathematical, or digital signal processing (DSP) designs, while EABs implement RAM, ROM, dual-port RAM, or first-in first-out (FIFO) functions. These elements make ACEX50K suitable for complex logic functions and memory functions such as digital signal processing, wide data path manipulation, data transformation and microcontrollers, as required in high-performance communications applications.

Based on reconfigurable CMOS SRAM elements, the ACEX50K architecture incorporates all features necessary to implement common gate array megafunctions, along with a high pin count to enable an effective interface with system components. The advanced process and the low voltage requirement of the 2.5V core allow ACEX 50K devices to meet the requirements of low cost, high-volume applications ranging from DSL modems to low-cost switches.

The ability to reconfigure ACEX50K devices enables complete testing prior to shipment and allows the designer to focus on simulation and design verification. Reconfigurability eliminates inventory management for gate array designs and test vector generation for fault coverage.

Table 4.2 shows ACEX50K device performance for some common designs. Special design techniques are not required to implement the applications, the designer simply infers or instantiates a function in a Verilog HDL, VHDL, Altera Hardware Description Language (AHDL), or schematic design file.

Table4.2 ACEX50K Performance

Application	**Resources Used**		**Performance Speed Grade**			
	LEs	**EABs**	**-1**	**-2**	**-3**	**Units**
16 Bit loadable Counter	16	0	285	232	185	(MHz)
16 Bit Accumulator	16	0	285	232	185	(MHz)
16 to 1 Multiplexer	10	0	3.5	4.5	6.6	(ns)
256*16 RAM read cycle speed	0	1	278	196	143	(MHz)
256*16 RAM write cycle speed	0	1	185	143	111	(MHz)

Each ACEX50K device contains an embedded array and a logic array. The embedded array is used to implement a variety of memory functions or complex logic functions, such as digital signal processing (DSP), wide data-path manipulation, microcontroller applications, and data transformation functions. The logic array performs the same function as the sea-of-gates in the gate array and is used to implement general logic such as counters, adders, state machines, and multiplexers. The combination of embedded and logic arrays provides the high performance and high density of embedded gate arrays, enabling designers to implement an entire system on a single device.

ACEX50K devices are configured at system power-up with data stored in an Altera serial configuration device or provided by a system controller. Altera offers EPC16, EPC2, EPC1, and EPC1441 configuration devices, which configure ACEX50K devices via a serial data stream. Configuration data can also be

downloaded from system RAM via the Altera MasterBlasterTM, ByteBlasterMVTM, or BitBlasterTM download cables. After an ACEX50K device has been configured, it can be reconfigured in-circuit by resetting the device and loading new data. Because reconfiguration requires less than 40 ms, real-time changes can be made during system operation [18].

ACEX50K devices are supported by Altera development systems, which are integrated packages that offer schematic, text (including AHDL), and waveform design entry, compilation and logic synthesis, full simulation and worst-case timing analysis, and device configuration. The Altera software works easily with common gate array EDA tools for synthesis and simulation. For example, the Altera software can generate Verilog HDL files for simulation with tools such as Cadence Verilog XL. Additionally, the Altera software contains EDA libraries that use device specific features such as carry chains, which are used for fast counter and arithmetic functions. The Altera development systems run on Windows-based PCs and Sun SPARCstation, and HP 9000 Series 700/800 workstations.

4.4.3) Functional Description

Each ACEX50K device contains an enhanced embedded array that implements memory and specialized logic functions, and a logic array that implements general logic. The embedded array consists of a series of EABs. When implementing memory functions, each EAB provides 4,096 bits, which can be used to create RAM, ROM, dual-port RAM, or first-in first-out (FIFO) functions. When implementing logic, each EAB can contribute 100 to 600 gates towards complex logic functions such as multipliers, microcontrollers, state machines and DSP functions. EABs can be used independently or multiple EABs can be combined to implement larger functions.

The logic array consists of logic array blocks (LABs). Each LAB contains eight LEs and a local interconnect. An LE consists of a 4-input LUT, a programmable flip-flop, and dedicated signal paths for carry and cascade functions. The eight LEs can be used to create medium sized blocks of logic such as 8-bit counters, address decoders, or state machines or combined across LABs to create larger logic blocks. Each LAB represents about 96 usable logic gates.

Signal interconnections within ACEX50K devices (as well as to and from device pins) are provided by the FastTrack Interconnect routing structure, which is a series of fast, continuous row and column channels that run the entire length and width of the device. Each I/O pin is fed by an I/O element (IOE) located at the end of each row and column of the FastTrack Interconnect routing structure. Each IOE contains a bidirectional I/O buffer and a flip-flop that can be used as either an output or input register to feed input, output, or bidirectional signals. When used with a dedicated clock pin, these registers provide exceptional performance. As inputs, they provide setup times as low as 1.1 ns and hold times of 0 ns. As outputs, these registers provide clock-to-output times as low as 2.5 ns. IOEs provide a variety of features such as JTAG BST support, slew-rate control, tri-state buffers, and open-drain outputs.

Figure 4.1 shows the block diagram of the ACEX50K device.

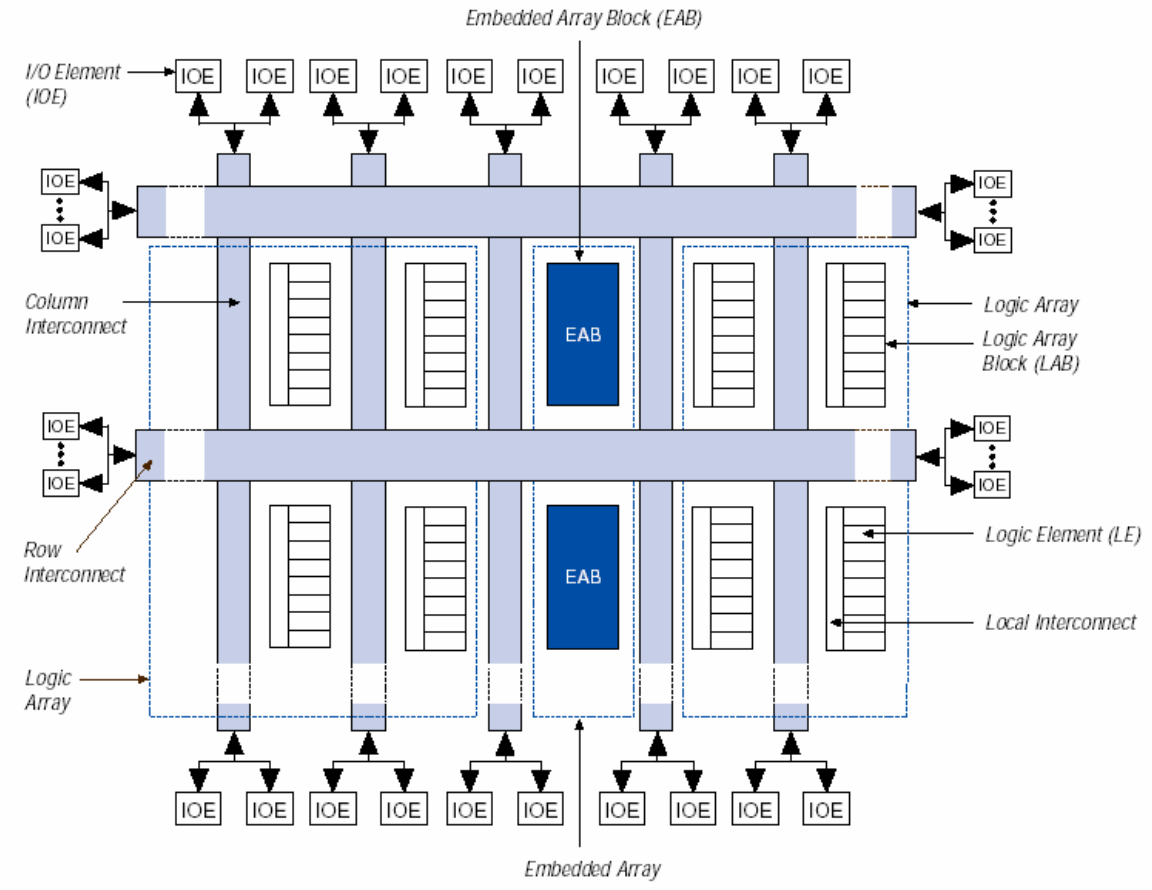


Figure4.1 ACEX50K Device Block Diagram

Each group of LEs is combined into an LAB; groups of LABs are arranged into rows and columns. Each row also contains a single EAB. The LABs and EABs are interconnected by the FastTrack Interconnect routing structure. IOEs are located at the end of each row and column of the FastTrack Interconnect routing structure. ACEX50K devices provide six dedicated inputs that drive the flip-flop's control inputs and ensure the efficient distribution of high-speed, low skew (less than 1.0 ns) control signals. These signals use dedicated routing channels that provide shorter delays and lower skews than the FastTrack Interconnect routing structure. Four of the dedicated inputs drive four global signals. These four global signals can also be driven by internal logic, providing an ideal solution for a clock divider or an internally generated asynchronous clear signal that clears many registers in the device.

4.4.3.1) Embedded Array Block (EAB)

The EAB is a flexible block of RAM, with registers on the input and output ports, that is used to implement common gate array megafunctions. Because it is large and flexible, the EAB is suitable for functions such as multipliers, vector scalars, and error correction circuits. These functions can be combined in applications such as digital filters and microcontrollers.

Logic functions are implemented by programming the EAB with a read only pattern during configuration, thereby creating a large LUT. With LUTs, combinatorial functions are implemented by looking up the results rather than by computing them. This implementation of combinatorial functions can be faster than using algorithms implemented in general logic, a performance advantage that is further enhanced by the fast access times of EABs.

The large capacity of EABs enables designers to implement complex functions in a single logic level without the routing delays associated with linked LEs. For example, a single EAB can implement any function with 8 inputs and 16 outputs. The ACEX50K enhanced EAB supports dual-port RAM. The dual-port structure is ideal for FIFO buffers with one or two clocks. The EAB can also support up to 16 bit wide RAM blocks. It can act in dual-port or single-port mode. When in dual-port mode, separate clocks may be used for read and write sections, allowing the EAB to be written and read at different rates. The EAB can also be used for

bidirectional, dual-port memory applications where two ports read or write simultaneously. To implement this type of dual-port memory, two EABs are used to support two simultaneous reads or writes. Alternatively, one clock and clock enable can be used to control the input registers of the EAB, while a different clock and clock enable control the output registers.

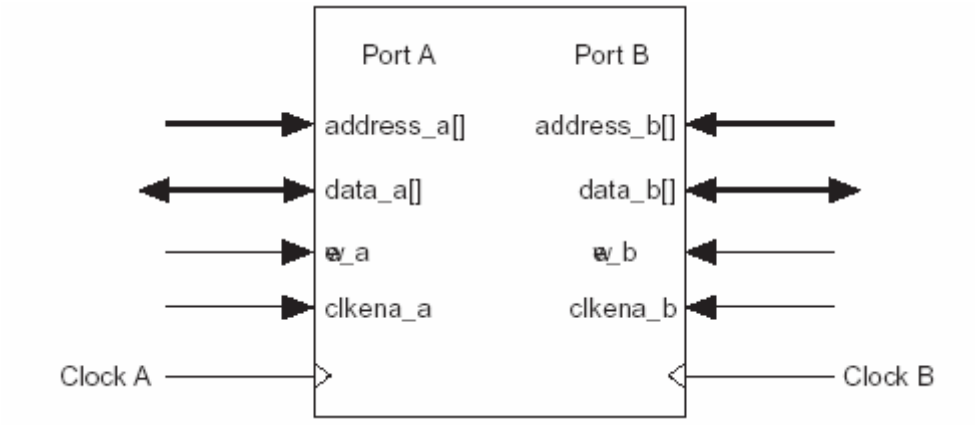


Figure4.2 ACEX50K EAB in Dual-Port RAM Mode

EABs can be used to implement synchronous RAM, which is easier to use than asynchronous RAM. A circuit using asynchronous RAM must generate the RAM write enable signal, while ensuring that its data and address signals meet setup and hold time specifications relative to the write enable signal. In contrast, the EAB's synchronous RAM generates its own write enable signal and is self-timed with respect to the input or write clock. When used as RAM, each EAB can be configured in any of the following sizes: 256 × 16; 512 × 8; 1,024 × 4; or 2,048 × 2. Figure 4.3 shows the ACEX50K EAB memory configurations.

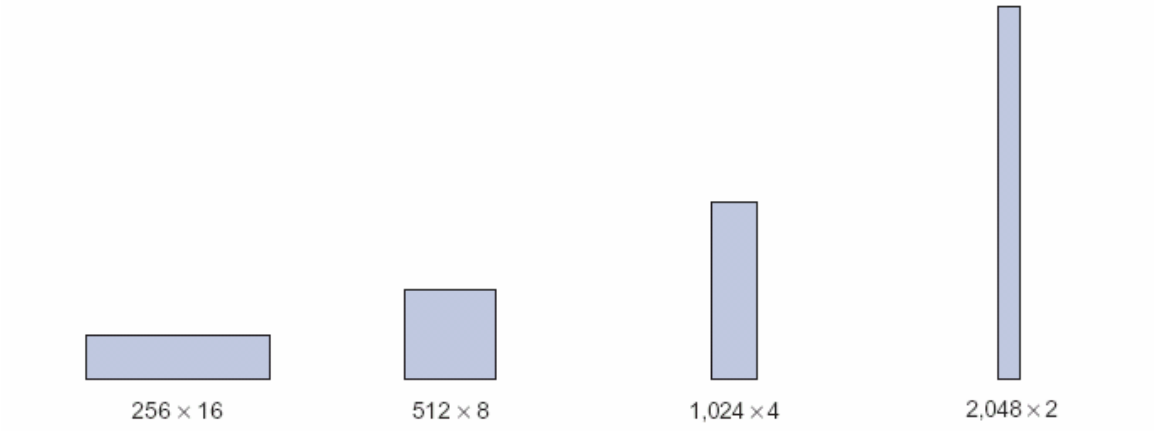


Figure4.3 ACEX50K EAB Memory Configurations

Larger blocks of RAM are created by combining multiple EABs. For example, two 256 × 16 RAM blocks can be combined to form a 256 × 32 block, and two 512 × 8 RAM blocks can be combined to form a 512 × 16 block Figure 4.4 shows examples of multiple EAB combination.

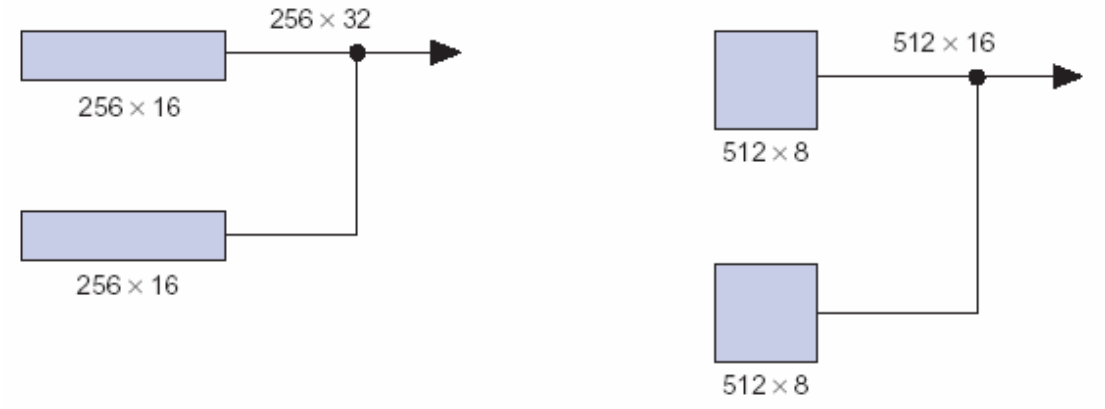


Figure4.4 Examples of Combining EABs

Altera software automatically combines EABs to meet the designer's RAM specifications. An EAB is fed by a row interconnect and can drive out to row and column interconnects. Each EAB output can drive up to two row channels and up to two column channels; the unused row channel can be driven by other LEs. This feature increases the routing resources available for EAB outputs [18].

4.4.3.2) Logic Array Block (LAB)

An LAB consists of eight LEs, their associated carry and cascade chains, LAB control signals, and the LAB local interconnect. The LAB provides the coarse-grained structure to the ACEX50K architecture, facilitating efficient routing with optimum device utilization and high performance.

Each LAB provides four control signals with programmable inversion that can be used in all eight LEs. Two of these signals can be used as clocks; the other two can be used for clear/preset control. The LAB clocks can be driven by the dedicated clock input pins, global signals, I/O signals, or internal signals via the LAB local interconnect. The LAB preset and clear control signals can be driven by the global signals, I/O signals, or internal signals via the LAB local interconnect. The global control signals are typically used for global clock, clear, or preset signals because they provide asynchronous control with very low skew across the device. If logic is required on a control signal, it can be generated in

one or more LEs in any LAB and driven into the local interconnect of the target LAB. In addition, the global control signals can be generated from LE outputs.

Figure 4.5 shows theACEX50K Logic Array Block.

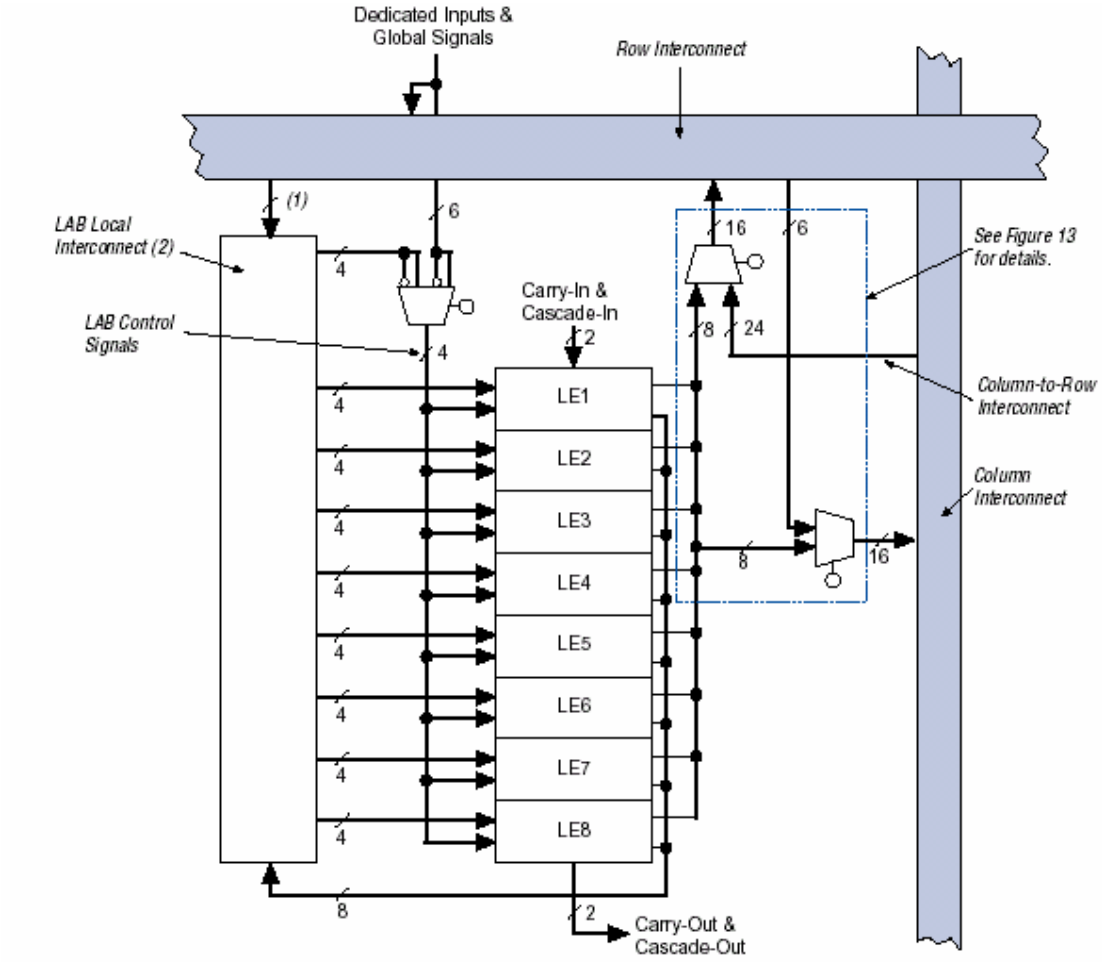


Figure4.5 ACEX50K Logic Array Block

4.4.3.3) Logic Element

The LE, the smallest unit of logic in the ACEX50K architecture, has a compact size that provides efficient logic utilization. Each LE contains a 4-input LUT, which is a function generator that can quickly compute any function of four variables. In addition, each LE contains a programmable flip-flop with a synchronous clock enable, a carry chain, and a cascade chain. Each LE drives both the local and the FastTrack Interconnect routing structure.

The programmable flip-flop in the LE can be configured for D, T, JK, or SR operation. The clock, clear and preset control signals on the flip-flop can be driven

by global signals, general-purpose I/O pins, or any internal logic. For combinatorial functions, the flip-flop is bypassed and the LUT's output drives the LE's output. The LE has two outputs that drive the interconnect: one drives the local interconnect, and the other drives either the row or column FastTrack Interconnect routing structure. The two outputs can be controlled independently. For example, the LUT can drive one output while the register drives the other output. This feature, called register packing, can improve LE utilization because the register and the LUT can be used for unrelated functions.

Figure 4.6 shows the ACEX50K Logic Element.

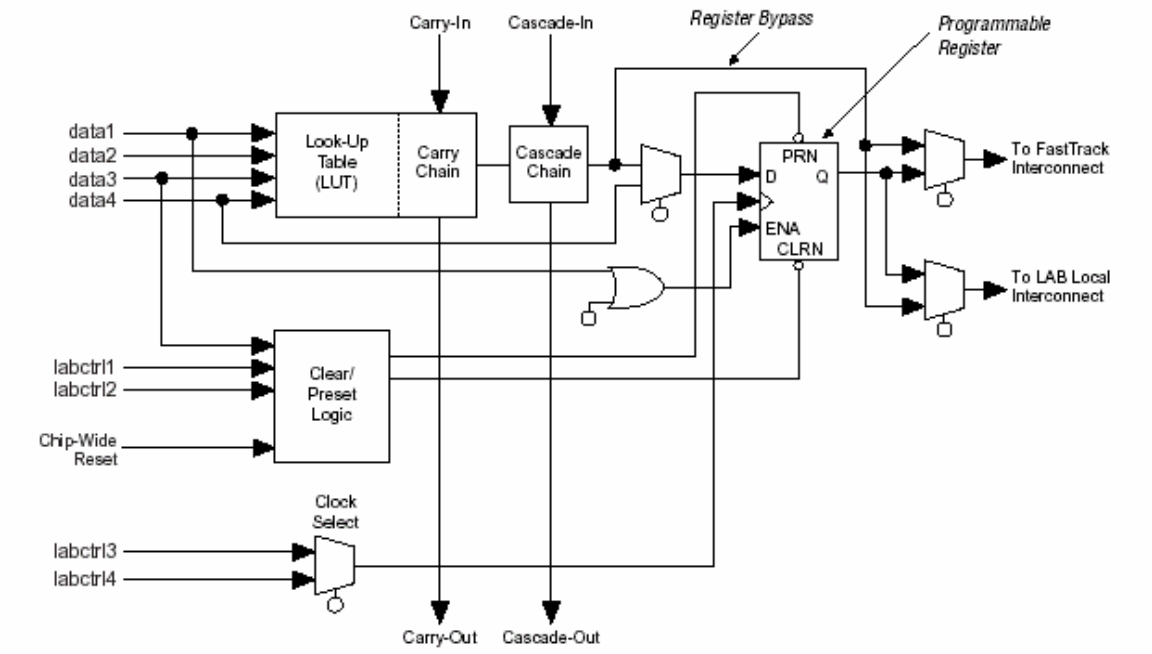


Figure4.6 ACEX50K Logic Element

The ACEX50K architecture provides two types of dedicated high-speed data paths that connect adjacent LEs without using local interconnect paths: carry chains and cascade chains. The carry chain supports high speed counters and adders, and the cascade chain implements wide-input functions with minimum delay. Carry and cascade chains connect all LEs in a LAB and all LABs in the same row. Intensive use of carry and cascade chains can reduce routing flexibility. Therefore, the use of these chains should be limited to speed-critical portions of a design.

1) Carry Chain

The carry chain provides a very fast (as low as 0.2 ns) carry-forward function between LEs. The carry-in signal from a lower-order bit drives forward into the higher-order bit via the carry chain, and feeds into both the LUT and the next portion of the carry chain. This feature allows the ACEX50K architecture to efficiently implement high-speed counters, adders, and comparators of arbitrary width.

Carry chains longer than eight LEs are automatically implemented by linking LABs together. A carry chain longer than one LAB skips either from even-numbered LAB to even-numbered LAB, or from odd numbered LAB to odd-numbered LAB. The carry chain does not cross the EAB at the middle of the row. For instance, in the EP1K50 device, the carry chain stops at the eighteenth LAB and a new carry chain begins at the nineteenth LAB.

Figure 4.7 shows how an n-bit full adder can be implemented in n + 1 LEs with the carry chain. One portion of the LUT generates the sum of two bits using the input signals and the carry-in signal; the sum is routed to the output of the LE. Another portion of the LUT and the carry chain logic generates the carry-out signal, which is routed directly to the carry-in signal of the next-higher-order bit. The final carry-out signal is routed to an LE, where it can be used as a general-purpose signal.

2) Cascade Chain

With the cascade chain, the ACEX50K architecture can implement functions that have a very wide fan-in. Adjacent LUTs can be used to compute portions of the function in parallel; the cascade chain serially connects the intermediate values. The cascade chain can use a logical AND or logical OR to connect the outputs of adjacent LEs. With a delay as low as 0.6 ns per LE, each additional LE provides four more inputs to the effective width of a function.

Cascade chains longer than eight bits are implemented automatically by linking several LABs together. For easier routing, a long cascade chain skips every other LAB in a row. A cascade chain longer than one LAB skips either from even-numbered LAB to even-numbered LAB, or from odd-numbered LAB to odd-numbered LAB (for example, the last LE of the first LAB in a row cascades to the

first LE of the third LAB). The cascade chain does not cross the center of the row. The cascade chain stops at the eighteenth LAB and a new one begins at the nineteenth LAB. This break is due to the EAB's placement in the middle of the row.

Figures 4.8 and 4.9 show how the cascade function can connect adjacent LEs to form functions with a wide fan-in. These examples show functions of 4n variables implemented with n LEs. The LE delay is 1.3 ns, the cascade chain delay is 0.6 ns. With the cascade chain, decoding a 16-bit address requires 3.1 ns.

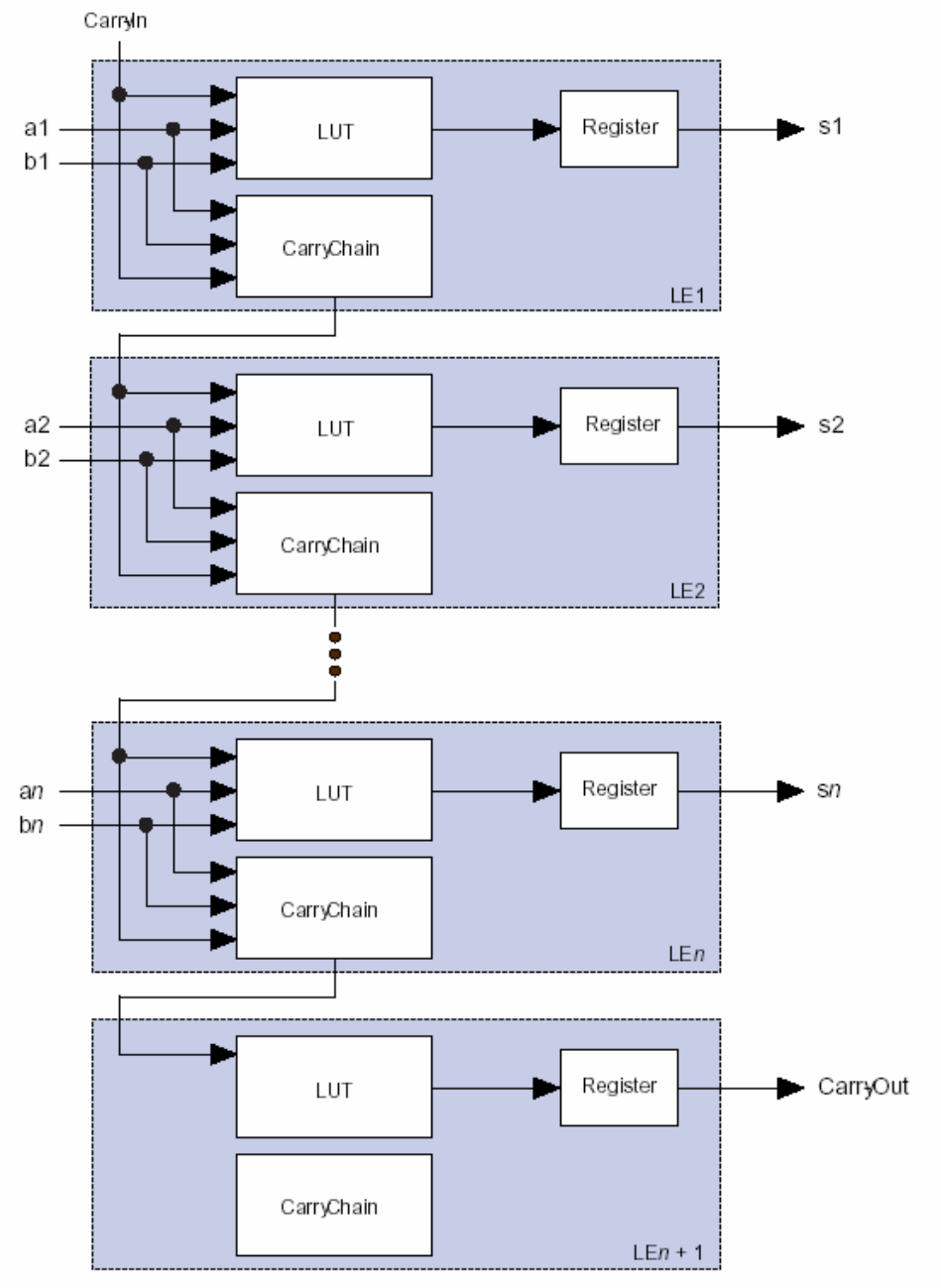


Figure4.7 ACEX50K Carry Chain Operation (n-Bit Full Adder)

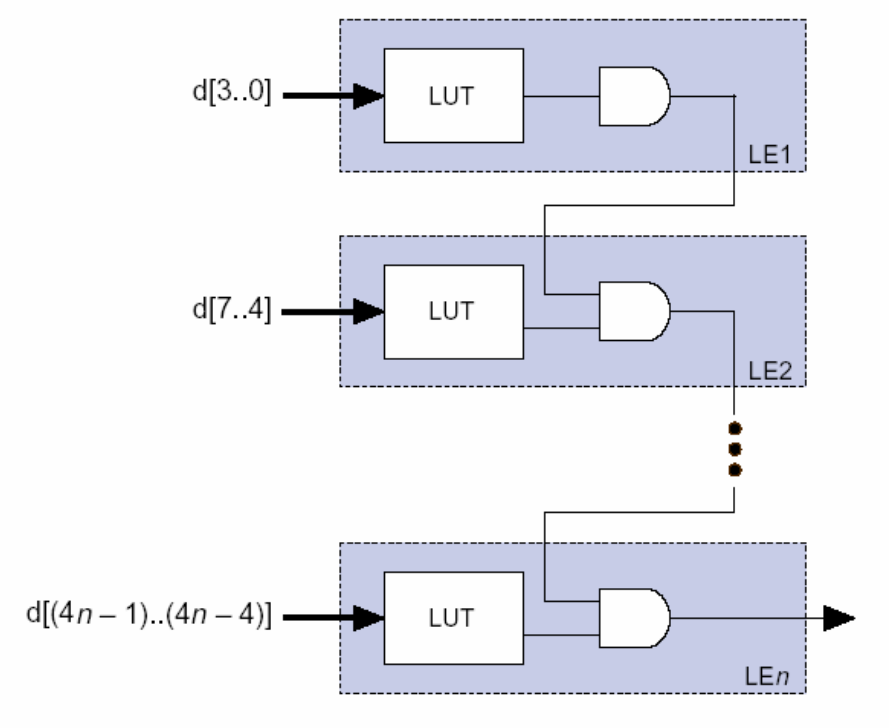


Figure4.8 ACEX50K AND Cascade Chain Operation

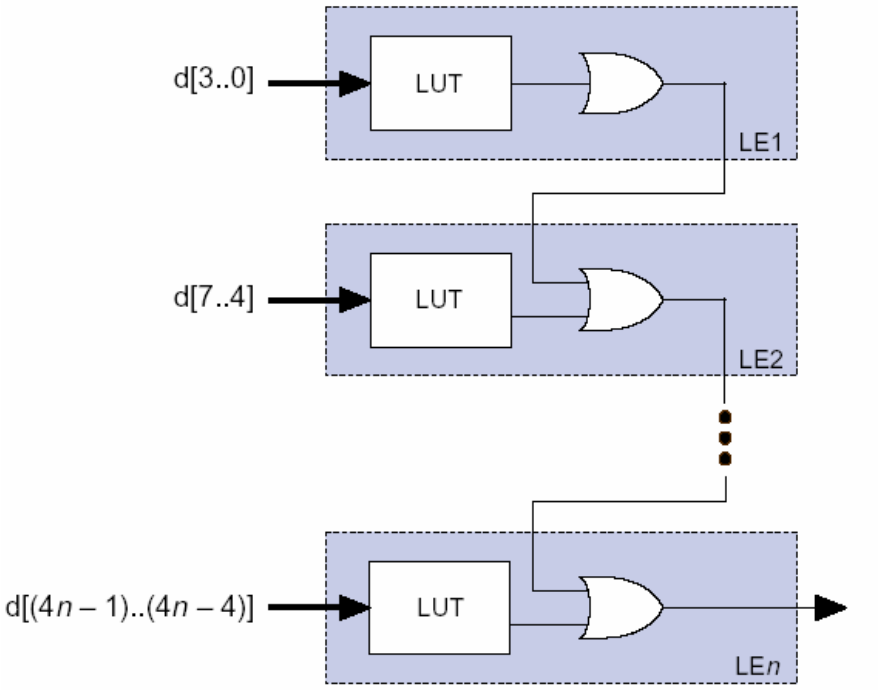


Figure4.9 ACEX50K OR Cascade Chain Operation

4.4.3.4) FastTrack Interconnect Routing Structure

In the ACEX50K architecture, connections between LEs, EABs, and device I/O pins are provided by the FastTrack Interconnect routing structure, which is a series of continuous horizontal and vertical routing channels that traverse the device. This global routing structure provides predictable performance, even in complex designs. In contrast, the segmented routing in FPGAs requires switch matrices to connect a variable number of routing paths, increasing the delays between logic resources and reducing performance. The FastTrack Interconnect routing structure consists of row and column interconnect channels that span the entire device.

Each row of LABs is served by a dedicated row interconnect. The row interconnect can drive I/O pins and feed other LABs in the row. The column interconnect routes signals between rows and can drive I/O pins. Row channels drive into the LAB or EAB local interconnect. The row signal is buffered at every LAB or EAB to reduce the effect of fan-out on delay. A row channel can be driven by an LE or by one of three column channels. These four signals feed dual 4-to-1 multiplexers that connect to two specific row channels. These multiplexers, which are connected to each LE, allow column channels to drive row channels even when all eight LEs in a LAB drive the row interconnect [18].

Each column of LABs or EABs is served by a dedicated column interconnect. The column interconnect that serves the EABs has twice as many channels as other column interconnects. The column interconnect can then drive I/O pins or another row's interconnect to route the signals to other LABs or EABs in the device. A signal from the column interconnect, which can be either the output of a LE or an input from I/O pin, must be routed to the row interconnect before it can enter a LAB or EAB. Each row channel that is driven by an IOE or EAB can drive one specific column channel. Access to row and column channels can be switched between LEs in adjacent pairs of LABs. For example, a LE in one LAB can drive the row and column channels normally driven by a particular LE in the adjacent LAB in the same row and vice versa. This flexibility enables routing resources to be used more efficiently.

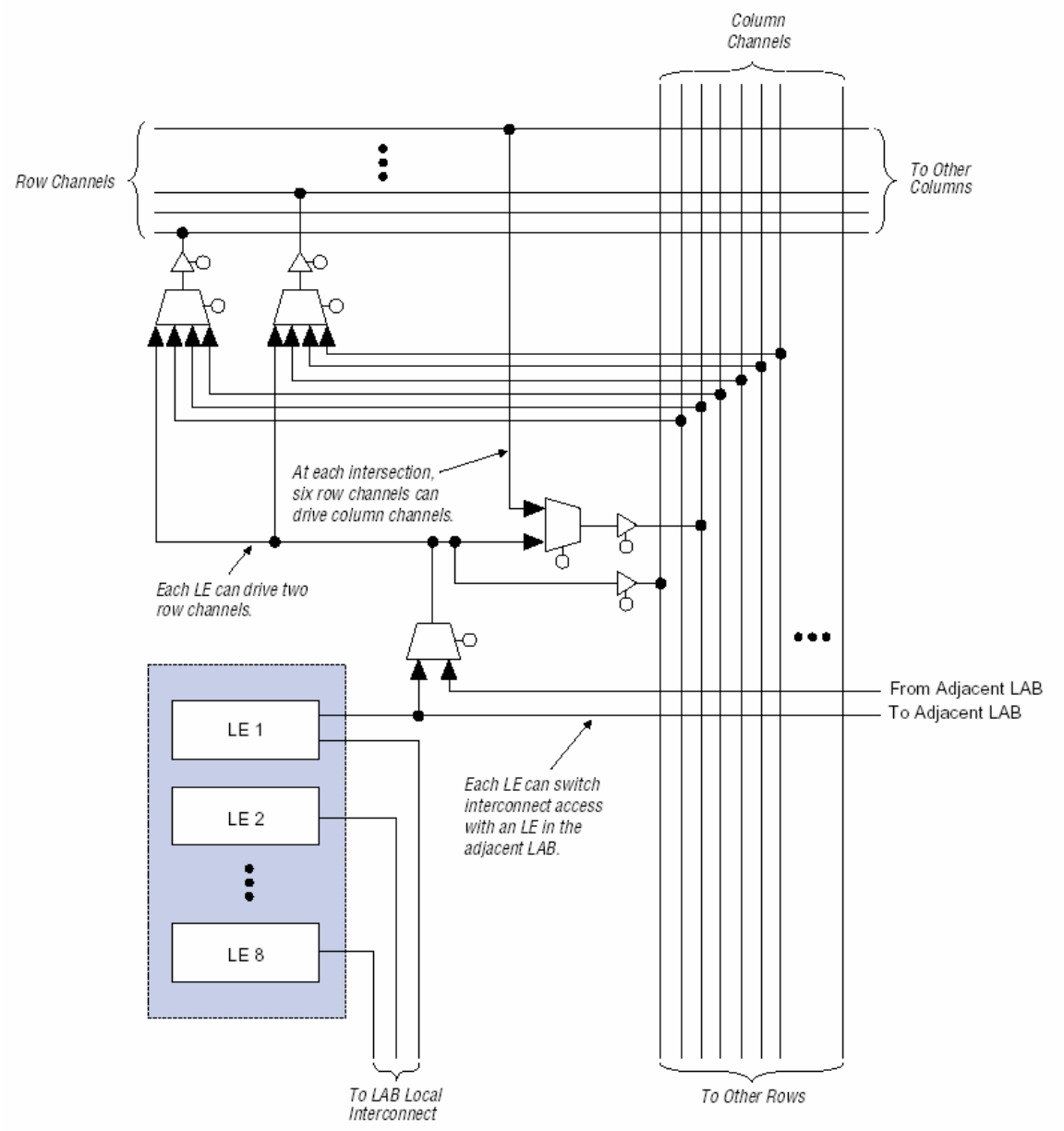


Figure4.10 ACEX50K LAB Connections to Row & Column Interconnect

For improved routing, the row interconnect consists of a combination of full-length and half-length channels. The full-length channels connect to all LABs in a row; the half-length channels connect to the LABs in half of the row. The EAB can be driven by the half-length channels in the left half of the row and by the full-length channels. The EAB drives out to the full-length channels. In addition to providing a predictable, row-wide interconnect, this architecture provides increased routing resources. Two neighboring LABs can be connected using a

half-row channel, thereby saving the other half of the channel for the other half of the row.

In addition to general-purpose I/O pins, ACEX50K devices have six dedicated input pins that provide low-skew signal distribution across the device. These six inputs can be used for global clock, clear, preset, and peripheral output-enable and clock-enable control signals. These signals are available as control signals for all LABs and IOEs in the device. The dedicated inputs can also be used as general-purpose data inputs because they can feed the local interconnect of each LAB in the device.

Table 4.3 summarizes the FastTrack Interconnect routing structure resources available in each ACEX device.

Table4.3 ACEX FastTrack Interconnect Resources

Device	Rows	Channels per Row	Columns	Channels per Column
EP1K10	3	144	24	24
EP1K30	6	216	36	24
EP1K50	10	216	36	24
EP1K100	12	312	52	24

4.4.3.5) I/O Element

An IOE contains a bidirectional I/O buffer and a register that can be used either as an input register for external data that requires a fast set-up time or as an output register for data that requires fast clock-to-output performance. For bi-directional registered I/O implementation, the output register should be in the IOE and the data input and output enable registers should be LE registers, placed adjacent to the bidirectional pin. On all ACEX50K devices, the input path from the I/O pad to the FastTrack Interconnect has a programmable delay element that can be used to guarantee a zero hold time. Depending on the placement of the IOE relative to what it is driving, the designer may choose to turn on the programmable delay to ensure a zero hold time or turn it off to minimize setup time. This feature is used to reduce setup time for complex pin-to register paths, for example, PCI designs. Each IOE selects the clock, clear, clock enable, and output enable controls from a

network of I/O control signals called the peripheral control bus. The peripheral control bus uses high-speed drivers to minimize signal skew across devices and provides up to 12 peripheral control signals.

Row-to-IOE Connections

When an IOE is used as an input signal, it can drive two separate row channels. The signal is accessible by all LEs within that row. When an IOE is used as an output, the signal is driven by a multiplexer that selects a signal from the row channels. Up to eight IOEs connect to each side of each row channel.

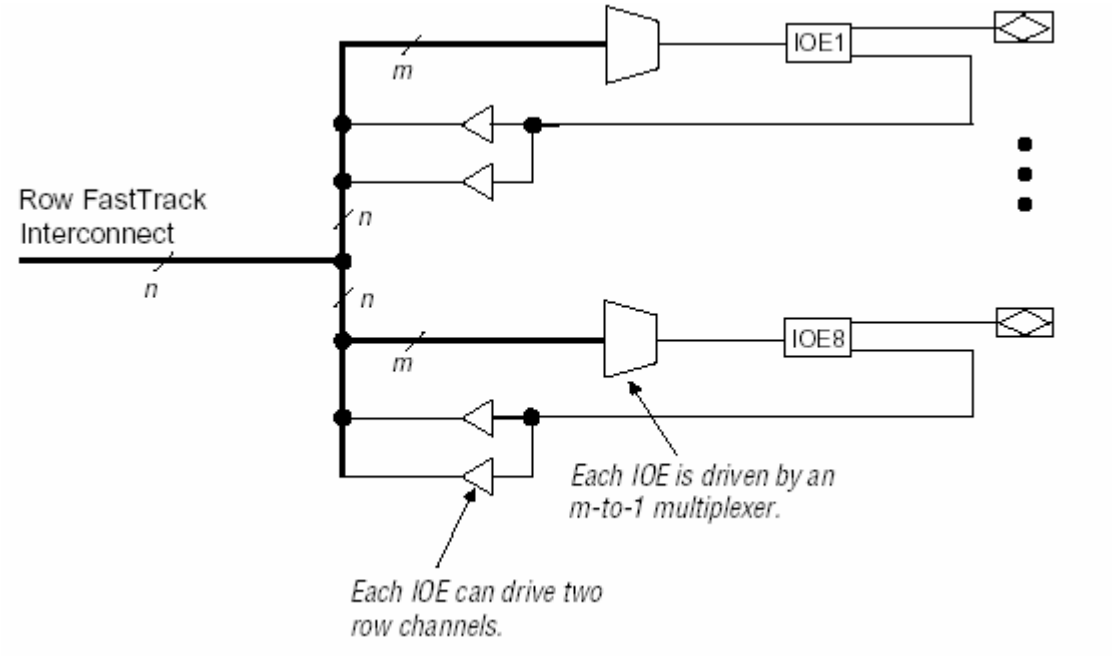


Figure4.11 ACEX50K Row-to-IOE Connections

Column-to-IOE Connections

When an IOE is used as an input, it can drive up to two separate column channels. When an IOE is used as an output, the signal is driven by a multiplexer that selects a signal from the column channels. Two IOEs connect to each side of the column channels. Each IOE can be driven by column channels via a multiplexer. The set of column channels is different for each IOE. Figure 4.12 shows the column to IOE connections in ACEX50K device.

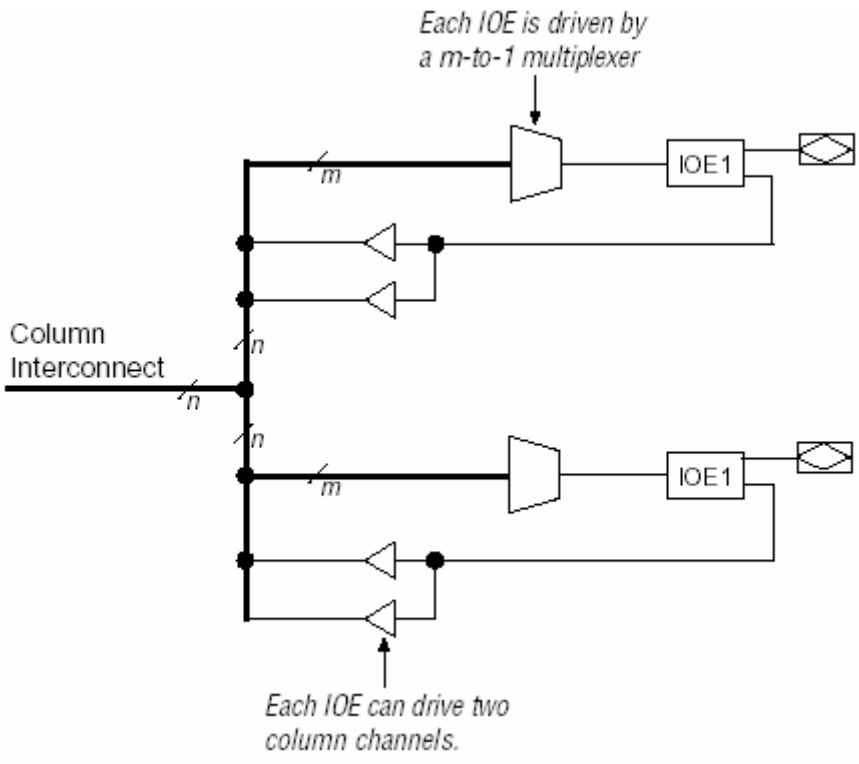


Figure4.12 ACEX50K Column-to-IOE Connections

4.4.4) ClockLock & ClockBoost Features

To support high-speed designs, ACEX50K devices offer ClockLock and ClockBoost circuitry containing a phase-locked loop (PLL) that is used to increase design speed and reduce resource usage. The ClockLock circuitry uses a synchronizing PLL that reduces the clock delay and skew within a device. This reduction minimizes clock-to-output and setup times while maintaining zero hold times. The ClockBoost circuitry, which provides a clock multiplier, allows the designer to enhance device area efficiency by sharing resources within the device. The ClockBoost feature allows the designer to distribute a low-speed clock and multiply that clock on-device. Combined, the ClockLock and ClockBoost features provide significant improvements in system performance and bandwidth [18].

The ClockLock and ClockBoost features in ACEX50K devices are enabled through the Altera software. External devices are not required to use these features. The ClockLock and ClockBoost circuitry lock onto the rising edge of the incoming clock.

4.4.5) JTAG Boundary-Scan Support

All ACEX50K devices provide JTAG BST circuitry that complies with the IEEE Std. 1149.1-1990 specification. ACEX50K devices can also be configured using the JTAG pins through the ByteBlasterMV or BitBlaster download cable or via hardware that uses the JamTM Standard Test and Programming Language (STAPL), JEDEC standard JESD-71. ACEX50K devices support the JTAG instructions. The instruction register length of ACEX50K devices is 10 bits. The USERCODE register length in ACEX50K devices is 32 bits; 7 bits are determined by the user, and 25 bits are pre-determined.

4.4.6) Operating Conditions

Table 4.4 provides the information on absolute maximum ratings for 2.5V ACEX50K device.

Table4.4 ACEX50K Absolute Maximum Rating

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CCINT}	Supply Voltage	With respect to ground	-0.5	3.6	V
V_{CCIO}			-0.5	4.6	V
V_I	DC input voltage		-2.0	5.75	V
I_{OUT}	DC output current		-25	25	mA
T_{STG}	Storage temperature	No bias	-65	150	Cels
T_{AMB}	Ambient temperature	Under bias	-65	135	Cels
T_J	Junction temperature	PQFP, TQFP & BGA packages, under bias		135	Cels

4.4.7) Timing Model

The continuous, high-performance FastTrack Interconnect routing resources ensure accurate simulation and timing analysis as well as predictable performance. This predictable performance contrasts with that of other FPGAs, which use a segmented connection scheme and therefore, have an unpredictable performance. Device performance can be estimated by following the signal path from a source, through the interconnect, to the destination. For example, the registered performance between two LEs on the same row can be calculated by adding the following parameters:

- LE register clock-to-output delay (t_{CO})
- Interconnect delay ($t_{SAMEROW}$)
- LE look-up table delay (t_{LUT})
- LE register setup time (t_{SU})

The routing delay depends on the placement of the source and destination LEs. A more complex registered path may involve multiple combinatorial LEs between the source and destination LEs. Timing simulation and delay prediction are available with the simulator and timing analyzer. The Simulator offers both pre-synthesis functional simulation to evaluate logic design accuracy and post-synthesis timing simulation with 0.1 ns resolution. The Timing Analyzer provides point-to-point timing delay information, setup and hold time analysis, and device-wide performance analysis. Figure 4.13 shows the overall timing model, which maps the possible paths to and from the various elements of the ACEX50K device.

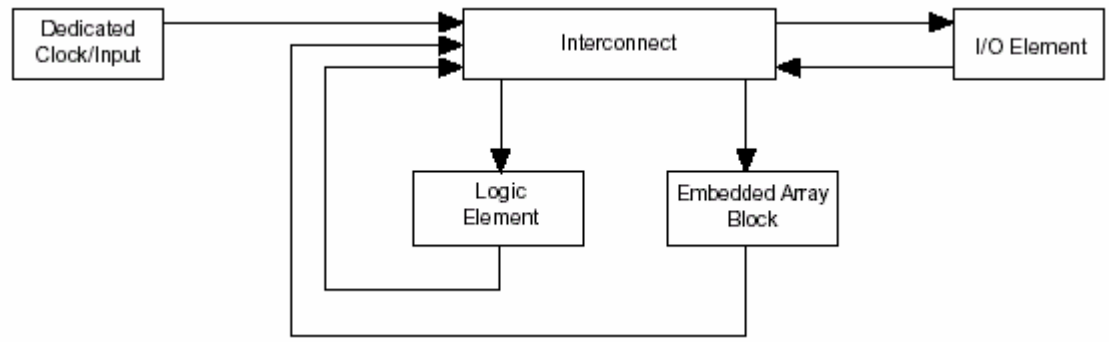


Figure4.13 ACEX50K Device Timing Model

4.4.8) Power Consumption

The supply power (P) for ACEX50K devices can be calculated with the following equation:

$P = P_{INT} + P_{IO} = (I_{CCSTANDBY} + I_{CCACTIVE}) \times V_{CC} + P_{IO}$

The $I_{CCACTIVE}$ value depends on the switching frequency and the application logic. This value is calculated on the basis of amount of current that each LE typically consumes. The P_{IO} value depends on the device output load characteristics and switching frequency. Compared to the rest of the device, the embedded array

consumes a negligible amount of power. Therefore, the embedded array can be ignored when calculating supply current.

The $I_{CCACTIVE}$ value can be calculated with the following equation:

$I_{CCACTIVE} = K \times f_{MAX} \times N \times toglc\ (\mu A)$,

Where, f_{MAX} = Maximum operating frequency in MHz

N = Total number of LEs used in the device

toglc = Average percent of LEs toggling at each clock (typically 12.5%)

K = Constant (For EP1K50 device, value of K is 4.5)

This supply power calculation provides an I_{CC} estimate based on typical conditions with no output load. The actual I_{CC} should be verified during operation because this measurement is sensitive to the actual pattern in the device and the environmental operating conditions. To better reflect actual designs, the power model for continuous interconnect ACEX50K devices assumes that LEs drive FastTrack Interconnect channels. In contrast, the power model of segmented FPGAs assumes that all LEs drive only one short interconnect segment. This assumption may lead to inaccurate results when compared to measured power consumption for actual designs in segmented FPGAs.

4.4.9) Configuration & Operation

The ACEX50K architecture supports several configuration schemes. This section summarizes the device operating modes and available device configuration schemes.

4.4.9.1) Operating Modes

The ACEX50K architecture uses SRAM configuration elements that require configuration data to be loaded every time the circuit powers up. The process of physically loading the SRAM data into the device is called configuration. Before configuration, as V_{CC} rises, the device initiates a Power-On Reset (POR). This POR event clears the device and prepares it for configuration. The ACEX50K POR time does not exceed 50 µs. During initialization, which occurs immediately after configuration, the device resets registers, enables I/O pins, and begins to operate as a logic device. Before and during configuration, all I/O pins (except dedicated inputs, clock, or configuration pins) are pulled high by a weak pull-up

resistor. Together, the configuration and initialization processes are called command mode; normal device operation is called user mode.

SRAM configuration elements allow ACEX50K devices to be reconfigured in-circuit by loading new configuration data into the device. Real-time reconfiguration is performed by forcing the device into command mode with a device pin, loading different configuration data, re-initializing the device, and resuming user-mode operation. The entire reconfiguration process requires less than 40 ms and can be used to reconfigure an entire system dynamically.

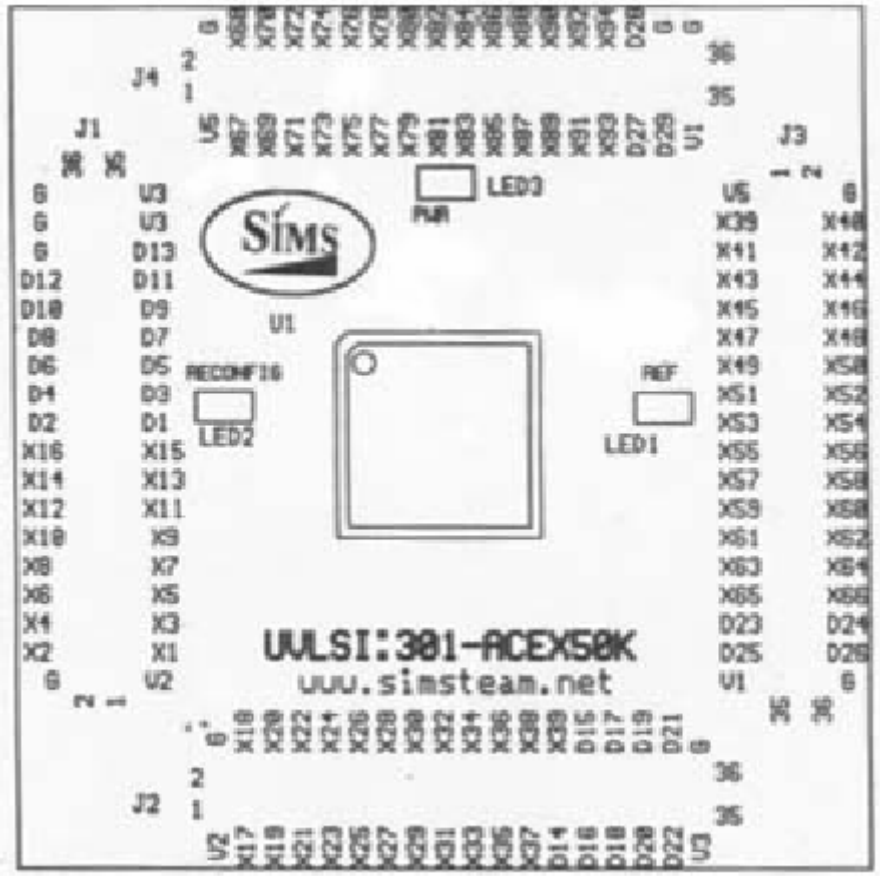
4.4.9.2) Configuration Schemes

The configuration data for an ACEX50K device can be loaded with one of five configuration schemes, chosen on the basis of the target application. An EPC16, EPC2, EPC1 or EPC1441 configuration device, intelligent controller, or the JTAG port can be used to control the configuration of an ACEX device, allowing automatic configuration on system power-up.

Table4.5 Data Sources for ACEX Configuration

Configuration Scheme	Data Source
Configuration device	EPC16, EPC2, EPC1 or EPC1441 devices
Passive serial (PS)	BitBlaster or ByteBlasterMV download cables or serial data source
Passive parallel asynchronous	Parallel data source
Passive parallel synchronous	Parallel data source
JTAG	BitBlaster or ByteBlasterMV download cables or Microprocessor with a Jam STAPL file

4.4.10) ACEX50K Device Pin-out



Switch on UVLSI201	Device Pin No	Property	FPGA Signal
RESET1	56	GCLK1	CLRN
RESET2	124	DED I/P3	RESET_N
	126	DED I/P4	RESET
Reference LED	87	I/O	
External Clock Input	125	DED CLK2	CLK2
On Board Clock Input	55	DED CLK1	CLK1

Connector	Device Pin	Property	Signal
P14/1	128	I/O	EXT I/O 1
P14/2	122	I/O	EXT I/O 2
P14/3	121	I/O	EXT I/O 3
P14/4	120	I/O	EXT I/O 4
P14/5	119	I/O	EXT I/O 5
P14/6	118	I/O	EXT I/O 6
P14/7	117	I/O	EXT I/O 7
P14/8	116	I/O	EXT I/O 8
P14/9	5V	Vcc	Vcc
P14/10	GND	Ground	Ground
P15/1	26	Dedicated O	Buffered O/P 9
P15/2	23	Dedicated O	Buffered O/P 10
P15/3	22	Dedicated O	Buffered O/P 11
P15/4	21	Dedicated O	Buffered O/P 12
P15/5	20	Dedicated O	Buffered O/P 13
P15/6	19	Dedicated O	Buffered O/P 14
P15/7	18	Dedicated O	Buffered O/P 15
P15/8	17	Dedicated O	Buffered O/P 16
P15/9	5V	Vcc	Vcc
P15/10	GND	Ground	Ground
P16/1	36	Dedicated O	Buffered O/P 1
P16/2	33	Dedicated O	Buffered O/P 2
P16/3	32	Dedicated O	Buffered O/P 3
P16/4	31	Dedicated O	Buffered O/P 4
P16/5	30	Dedicated O	Buffered O/P 5
P16/6	29	Dedicated O	Buffered O/P 6
P16/7	28	Dedicated O	Buffered O/P 7
P16/8	27	Dedicated O	Buffered O/P 8
P16/9	5V	Vcc	Vcc
P16/10	GND	Ground	Ground

Connector	Device Pin	Property	Signals from GPIO Board
P17/1	9	INPUT 1	SW1 digital input I16
P17/2	8	INPUT 2	SW1 digital input I15
P17/3	7	INPUT 3	SW1 digital input I14
P17/4	144	INPUT 4	SW1 digital input I13
P17/5	143	INPUT 5	SW1 digital input I12
P17/6	142	INPUT 6	SW1 digital input I11
P17/7	141	INPUT 7	SW1 digital input I10
P17/8	140	INPUT 8	SW1 digital input I9
P17/9	138	INPUT 9	SW1 digital input I8
P17/10	137	INPUT 10	SW1 digital input I7
P17/11	136	INPUT 11	SW1 digital input I6
P17/12	135	INPUT 12	SW1 digital input I5
P17/13	133	INPUT 13	SW1 digital input I4
P17/14	132	INPUT 14	SW1 digital input I3
P17/15	131	INPUT 15	SW1 digital input I2
P17/16	130	INPUT 16	SW1 digital input I1
P17/17	110	OUTPUT16	Output LED O16
P17/18	109	OUTPUT15	Output LED O15
P17/19	102	OUTPUT14	Output LED O14
P17/20	101	OUTPUT13	Output LED O13
P17/21	100	OUTPUT12	Output LED O12
P17/22	99	OUTPUT11	Output LED O11
P17/23	98	OUTPUT10	Output LED O10
P17/24	97	OUTPUT 9	Output LED O9
P17/25	96	OUTPUT 8	Output LED O8
P17/26	95	OUTPUT 7	Output LED O7
P17/27	92	OUTPUT 6	Output LED O6
P17/28	91	OUTPUT 5	Output LED O5
P17/29	90	OUTPUT 4	Output LED O4
P17/30	89	OUTPUT 3	Output LED O3

Connector	Device Pin	Property	Signals from GPIO Board
P17/31	88	OUTPUT 2	Output LED O2
P17/32	86	OUTPUT 1	Output LED O1
P17/33	83	SEG A	7 Segment O/P => ‘a’
P17/34	82	SEG B	7 Segment O/P => ‘b’
P17/35	81	SEG C	7 Segment O/P => ‘c’
P17/36	80	SEG D	7 Segment O/P => ‘d’
P17/37	79	SEG E	7 Segment O/P => ‘e’
P17/38	78	SEG F	7 Segment O/P => ‘f’
P17/39	73	SEG G	7 Segment O/P => ‘g’
P17/40	72	SEG DP	7 Segment O/P => ‘dp’
P17/41	36	DISP 1	Digit 0 select o/p
P17/42	33	DISP 2	Digit 1 select o/p
P17/43	32	DISP 3	Digit 2 select o/p
P17/44	31	DISP 4	Digit 3 select o/p
P17/45	114	KEY 1	Key k1
P17/46	113	KEY 2	Key k2
P17/47	112	KEY 3	Key k3
P17/48	111	KEY 4	Key k4
P17/49	5V	Vcc	Vcc
P17/50	GND	Ground	Ground
P18/1	37	DAC0	DAC data0 o/p from FPGA
P18/2	38	DAC1	DAC data1 o/p from FPGA
P18/3	39	DAC2	DAC data2 o/p from FPGA
P18/4	41	DAC 3	DAC data3 o/p from FPGA
P18/5	42	DAC4	DAC data4 o/p from FPGA
P18/6	43	DAC5	DAC data5 o/p from FPGA
P18/7	44	DAC6	DAC data6 o/p from FPGA
P18/8	46	DAC7	DAC data7 o/p from FPGA
P18/9	47	ADC_D0	Data 0 from ADC 0808
P18/10	48	ADC_D1	Data 1 from ADC 0808

Connector	Device Pin	Property	Signals from GPIO Board
P18/11	49	ADC_D2	Data 2 from ADC 0808
P18/12	51	ADC_D3	Data 3 from ADC 0808
P18/13	54	ADC_D4	Data 4 from ADC 0808
P18/14	59	ADC_D5	Data 5 from ADC 0808
P18/15	60	ADC_D6	Data 6 from ADC 0808
P18/16	62	ADC_D7	Data 7 from ADC 0808
P18/17	63	ADC_A0	ADC Channel Select Bit
P18/18	67	ADC_START	Write for ADC (SOC)
P18/19	64	ADC_A1	ADC Channel Select Bit
P18/20	68	ADC_ALE	ADC ALE Signal
P18/21	65	ADC_A2	ADC Channel select Bit
P18/22	69	ADC_EOC	Interrupt Signal from ADC (EOC)
P18/23	125	EXT CLK	555 Frequency o/p to FPGA
P18/24	70	ADC_OE	ADC Output Enable (OE)
P18/25	5V	Vcc	Vcc
P18/26	GND	Ground	Ground

CHAPTER 5

SOFTWARE

Introduction

This chapter is intended to become familiar with the VHDL for specifying programmable logic design. For serious work, use of EDA Tools like Altera is essential because PLDs contain many thousands of programmable fuses. The process of producing fuse maps is therefore highly impossible to manage by hand. The purpose of EDA tool is to interpret the logic design and convert it into a format which may be loaded in the PLD directly, called In-System-Programming (ISP), or indirectly via a separate device programmer.

5.2 Altera Quartus II Software

The Altera Quartus II design software provides a complete, multi platform design environment that easily adapts to our specific design needs. It is a comprehensive environment for system on a programmable chip (SOPC) design. This software includes solutions for all phases of FPGA and CPLD design.

In addition, Quartus II software allows us to use the graphical user interface, EDA tool interface or command line interface for each phase of the design flow. We can use one of these interfaces for the entire flow or we can use different options at different phases of the design flow.

Graphical user interface has been used for this project. We will study it in detail.

5.2.1) Graphical User Interface Design Flow

We can use the Quartus II software to perform all stages of the design flow. It is a complete, easy to use, stand alone solution [12]. The Quartus II graphical user interface provides the following features for each stage of design flow.

(1) Design Entry

- Text Editor
- Block and Symbol Editor
- MegaWizard Plug In Manager
- Assignment Editor
- Floor plan Editor

(2) Synthesis

- Analysis and Synthesis
- VHDL, Verilog HDL & AHDL
- Design Assistant

(3) Place and Route

- Fitter
- Assignment Editor
- Floor plan Editor
- Chip Editor
- Report Window
- Incremental Fitting

(4) Timing Analysis

- Timing Analyzer
- Report Window

(5) Simulation

- Simulator
- Waveform Editor

(6) Programming

- Assembler
- Programmer
- Convert programming Files

5.2.2) Procedure

The following steps describe the basic design flow for the Quartus II graphical user interface:

1. Create a new project and specify a target device or device family by using the New Project wizard.
2. Create a VHDL, Verilog HDL or AHDL design by using the text editor. We can use the Block Editor to create a block diagram with symbols that represent other design files or to create a schematic.
3. Specify initial design constraints using the assignment editor, the setting dialogue box and the floor plan editor.
4. Create a system level design by using the SOPC builder or DSP builder.

5. Create software and programming files for Excalibur device processors by using the software builder.
6. Synthesize the design by using Analysis and Synthesis.
7. Perform functional simulation on the design by using the Simulator.
8. Perform place and route on the design by using the fitter. For a small change to the source code we can also use incremental fitting.
9. Perform timing analysis on the design by Timing Analyzer.
10. Perform timing simulation on the design by using the simulator.
11. Make timing improvements to achieve timing closure by using physical synthesis, the timing closure floor plan, the setting dialogue box and the assignment Editor.
12. Create programming files for the design by using the assembler.
13. Program the device by using programming files, the Programmer and Altera hardware or convert programming files to other file formats for use by other systems.
14. Debug the design by using the Signal Tap II logic Analyzer, the signal probe feature or the chip editor.
15. Manage engineering changes by using the chip editor, the resource property editor and the change manager (optional).

Figures 5.1 to 5.8 show the steps to use the QuartusII software.

Figure5.1 Using Wizard Create New Project

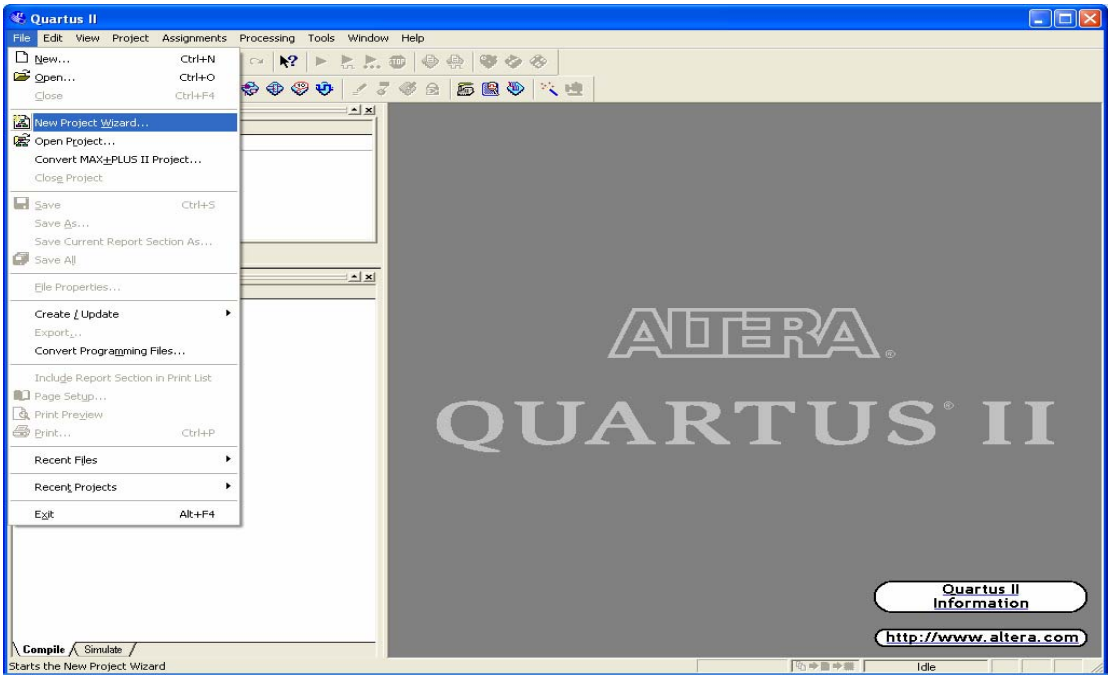


Figure5.2 Create Directory and Project Name

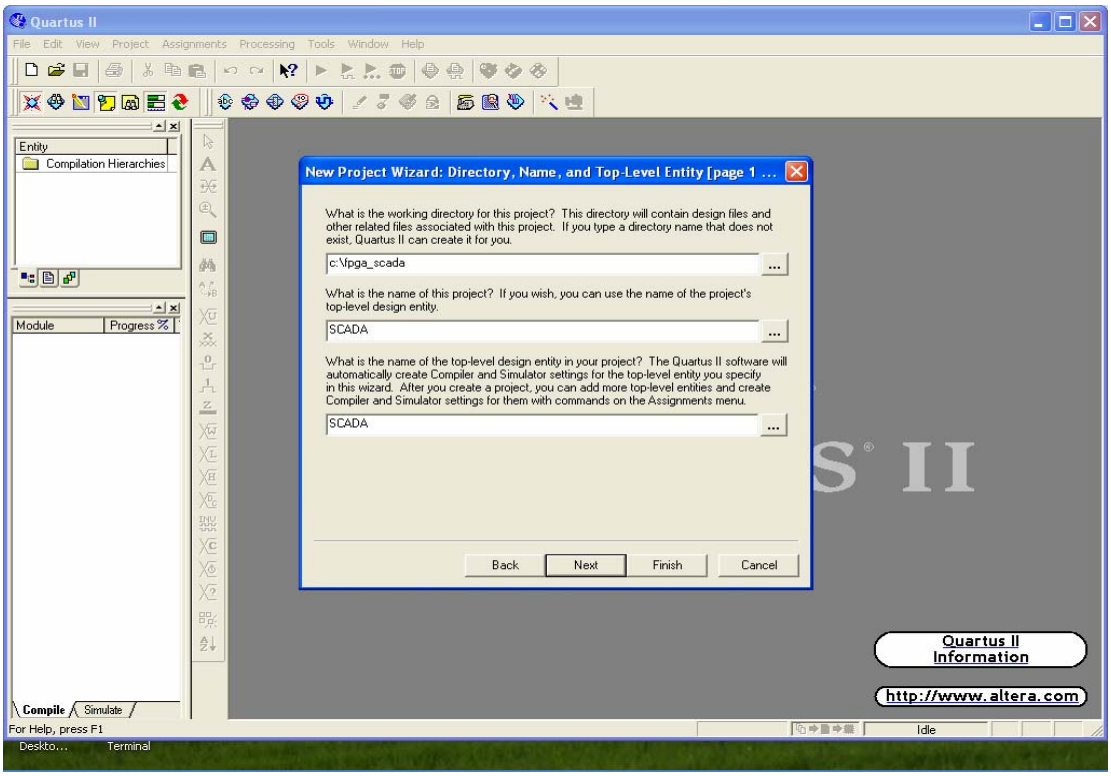


Figure5.3 Select the Family of the Device

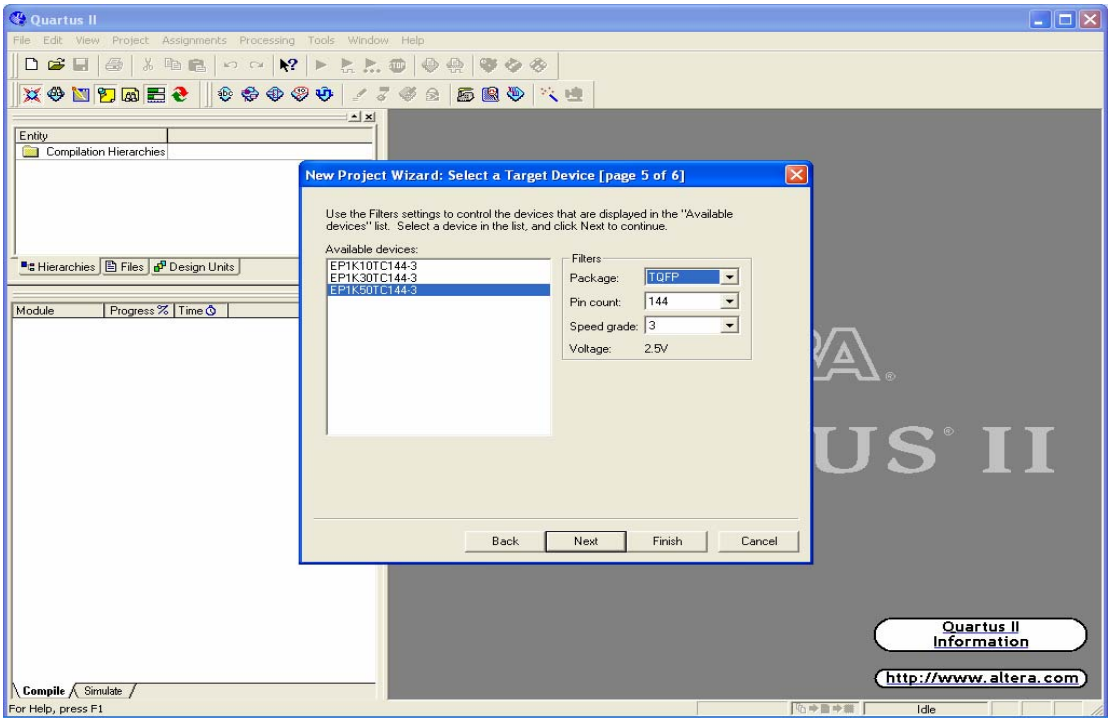


Figure 5.4 Summary of the New Project Wizard

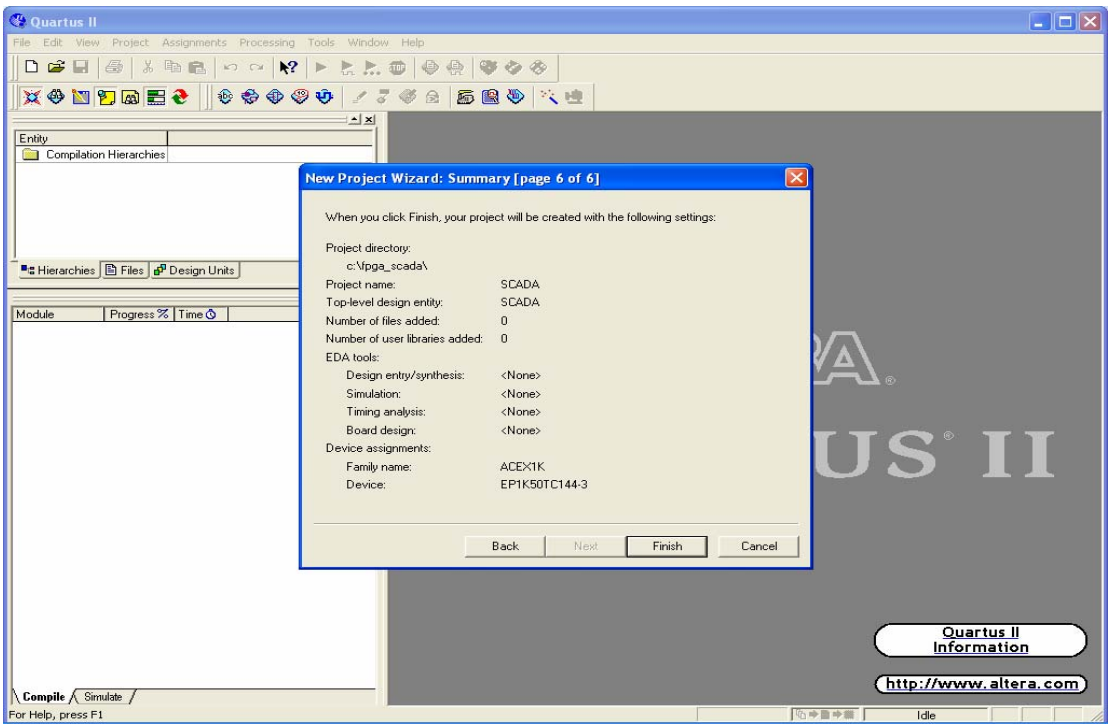


Figure5.5 Select the VHDL file

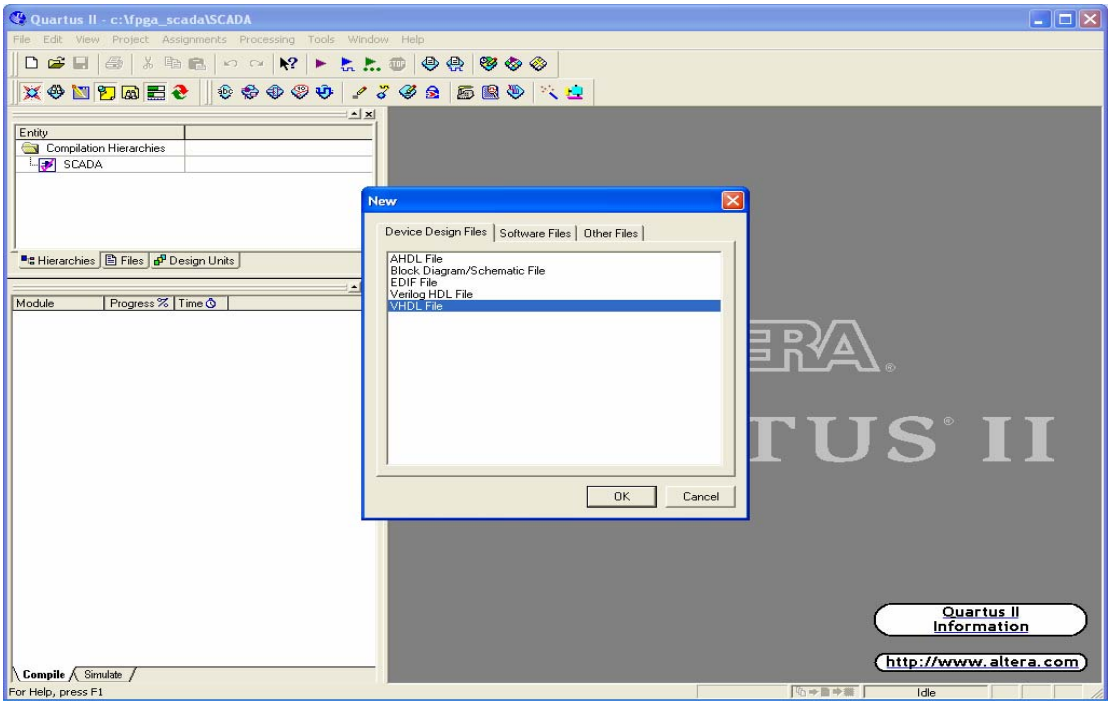


Figure5.6 Write the VHDL Code

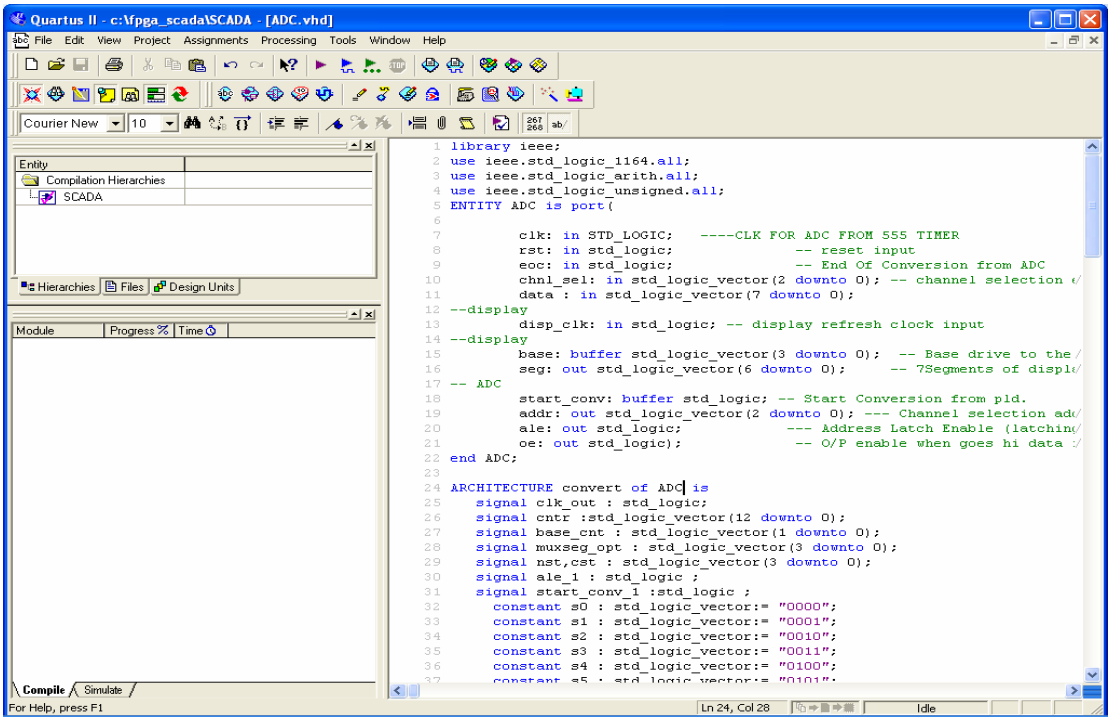


Figure5.7 Compilation Process

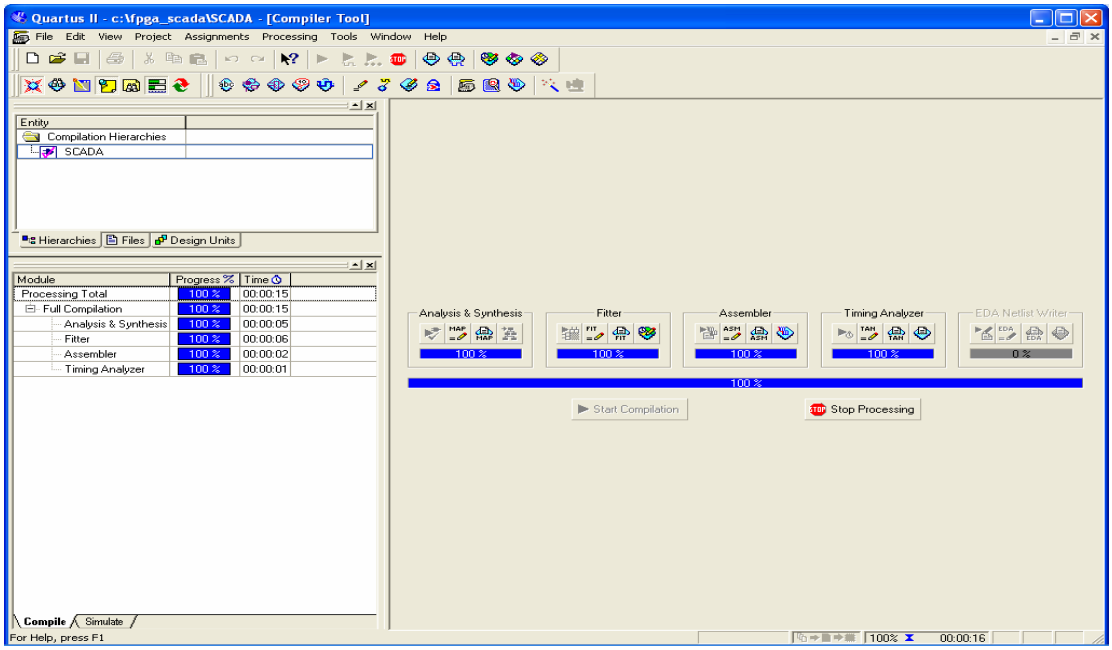
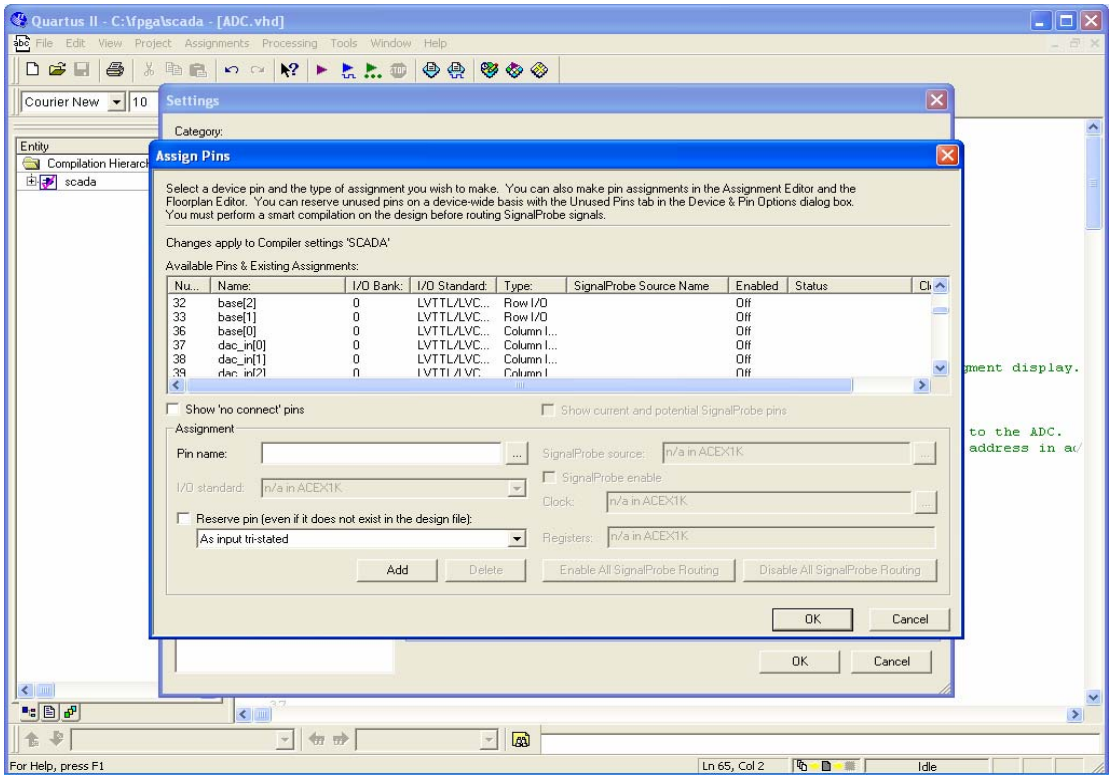


Figure5.8 Assignment of Pins for the Device



5.3 VHDL

Introduction

VHDL is a Hardware Description Language that can be used to model a system. The digital system can be as simple as a logic gate or as complex as a complete electronics system [2]. VHDL is an acronym for VHSIC Hardware Description Language; VHSIC is an acronym for Very High Speed Integrated Circuits. It can be used to model a digital system at many levels of abstraction, ranging from the algorithmic level to the gate level. The digital system can also be described hierarchically. Timing can also be explicitly modeled in the same description. The VHDL language can be regarded as an integrated amalgamation of the many languages.

VHDL = Sequential language + Concurrent language + Net list language + Timing specifications + Waveform generation language

Therefore, the language has constructs that enable us to express the concurrent or sequential behavior of a digital system with or without timing. The language not only defines the syntax but also defines very clear simulation semantics for each language construct. Therefore, models written in this language can be verified using a VHDL simulator.

It is a strongly typed language and is often verbose to write. It inherits many of its features, especially the sequential language part from the Ada Programming language. Because VHDL provides an extensive range of modeling capabilities, it is often difficult to understand. Fortunately, it is possible to quickly assimilate a core subset of the language that is both easy and simple to understand without learning the more complex features. This subset is usually sufficient to model most applications. The complete language has sufficient power to capture the descriptions of the most complex chips to a complete electronic system.

5.3.2) Salient Features

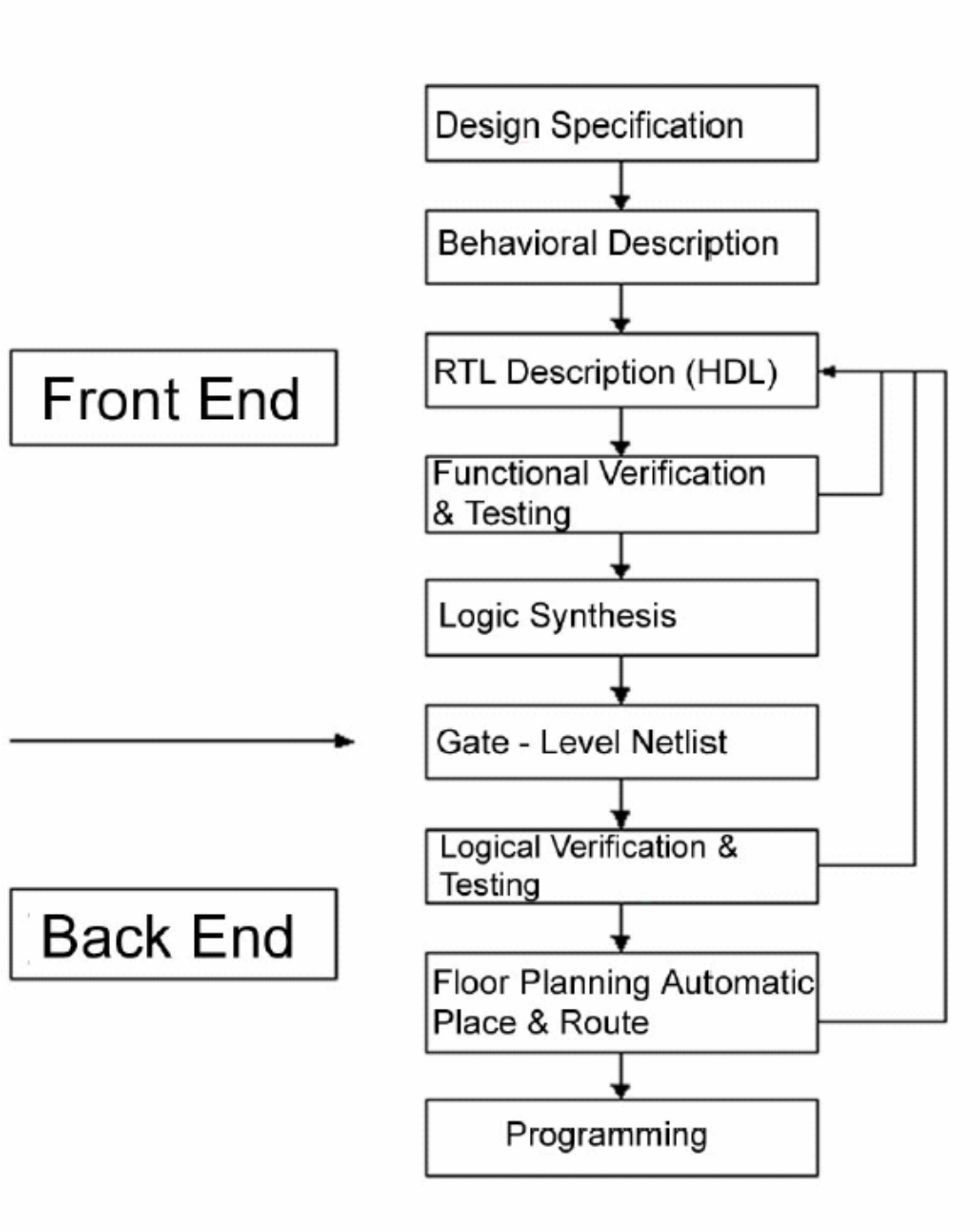
The following are the major capabilities that the language provides along with the features that differentiate it from other hardware description languages.

- The language can be used as an exchange medium between chip vendors and CAD tool users. Different chip vendors can provide VHDL descriptions of their components to system designers. CAD tool user can

use it to capture the behavior of the design at a high level of abstraction for functional simulation.

- The language can also be used as a communication medium between different CAD and CAE tools. For example, a schematic capture program may be used to generate a VHDL description for the design, which can be used as an input to a simulation program [2].
- The language supports hierarchy, that is, a digital system can be modeled as a set of interconnected components, and each component can be modeled as a set of interconnected subcomponents.
- The language supports flexible design methodologies; top down, bottom up or mixed.
- The language is not technology specific, but is capable of supporting technology specific features. It can also support various hardware technologies, for example we may define new logic types and new components.
- Various digital modeling techniques, such as finite state machine descriptions, algorithmic descriptions and boolean equations can be modeled using the language.
- The language is publicly available, human readable, machine readable and above all, it is not proprietary.
- It is an IEEE and ANSI standard; therefore models described using this language are portable.
- Test benches can be written using the same language to test other VHDL models.
- The capability of defining new data types provides the power to describe and simulate a new design technique at a very high level of abstraction without any concern about the implementation details.

Figure5.9 VLSI Design Flow



CHAPTER 6

UVLSI TRAINER

Introduction

Programmable Logic Devices are playing major role in system design due to their flexible architecture, re-programmability and fast time to market resulting in a smaller design-cycle period. Also lower design risk is involved with the use of PLDs [12].

This Universal PLD kit is an ideal trainer to implement and test the designs. This kit makes it possible to execute and verify basic digital experiments using VHDL and Verilog, the standard Hardware Description Languages. VHDL code can be written and the results can be verified on this kit using FPGA or CPLD. We can verify various experiments involving combinational and sequential logic using this kit. It is assembled ready for various interfaces that include ADC/DAC, display, keyboard, serial communication, VGA, PS2 etc.

The Universal PLD Trainer System consists of

• Power Supply Unit

• Hardware Access Unit

• Connecting cables

• Universal Board (UVLSI 201)

• FPGA Daughter Board

Each is discussed in brief below:

(1) Power Supply Unit

SMPS power supply: Input 100V to 300V, 50 Hz and Output 9.6 V dc, 1.2A

(2) Hardware Access Unit

This is security unit used to configure the devices on the UVLSI 201. This unit detects following things.

- Card in: When daughter card is inserted this led will glow.
- Altera LED: When Altera daughter board is inserted this led will glow.
- Altera SPROM: This led will glow, when Altera SPROM is being programmed.
- Xilinx LED: When Xilinx daughter board is inserted this led will glow.

- Xilinx SPROM - This led will glow, when Xilinx SPROM is being programmed.

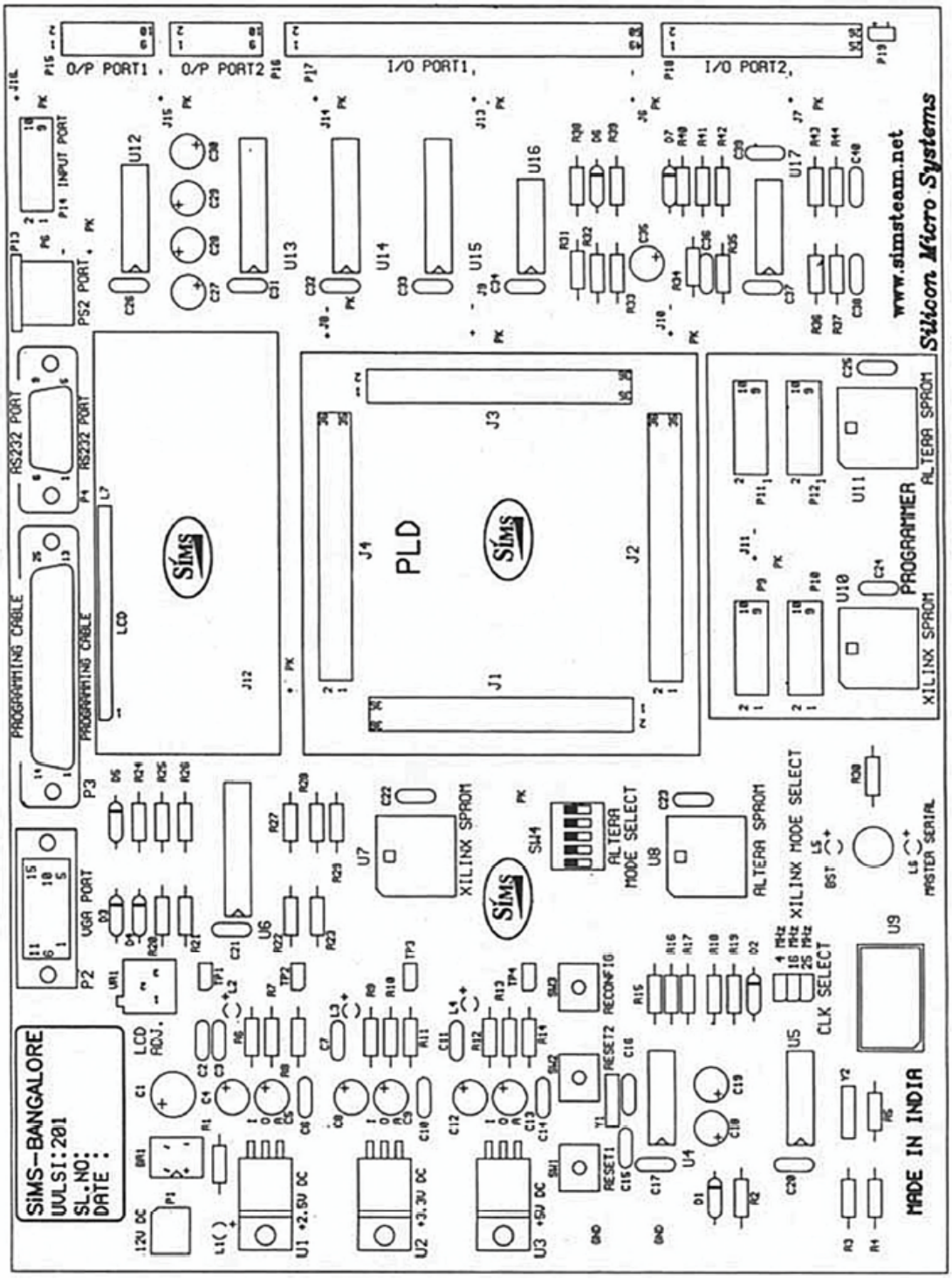


Figure6.1 Layout of UVLSI 201

(3) List of cables

- A set of cables is provided to connect the UVLSI 201 board to the PC through the hardware access unit and to the interfacing units.
- Parallel Port of the PC to Hardware Access Unit (25 pin FRC M-M)
- Hardware Access Unit to UVLSI 201 (25 pin FRC F-F)
- 10 Pin FRC Cable to program the configuration devices
- 50 and 26 Pin FRC cables to interface with the GPIO 401A module

6.2 Salient Features of Universal Board

The printed circuit board assembled in the enclosure Universal Board contains all the devices available for interfacing, assembled with the supporting hardware and the connectors for interfacing to the PLD board.

6.2.1) Connectors

(1) Input Port (P14)

This is 10-pin FRC header with 8 I/O lines and Vcc [+5V] (pin 9), GND (pin 10). This port can be configured as input or as output port.

(2) Output Port (P15)

This is 10-pin FRC header with 8 I/O lines and Vcc [+5V] (pin 9), GND (pin 10). This port is a dedicated output port with buffers (74LS245).

(3) Output Port2 (P16)

This is 10-pin FRC header with 8 I/O lines and Vcc [+5V] (pin 9), GND (pin 10). This port is a dedicated output port with buffers (74LS245).

(4) I/O Port1 (P17)

This is 50 pin header with 48 I/O lines and Vcc [+5V] (pin 49), GND (pin50). This port can be configured as input or output.

(5) I/O Port2 (P18)

This is 26 pin header with 24 I/O lines and Vcc [+5V] (pin 25), GND (pin26). This port can be configured as input or output.

(6) PS2 Port (P13)

This is used to interface a PS2 standard keyboard or a mouse.

(7) Serial Port (P4)

This is a RS-232 standard serial communication port.

(8) Programming Cable (P3)

This is D type 25-pin male, used to configure the PLDs and to program the configuration devices.

(9) VGA Port (P2)

This is used to interface VGA standard graphics devices.

(10) SPROM Programmer Connectors

Xilinx SPROM: Connect P9 and P10 through a 10-pin FRC cable when programming the Xilinx Configuration devices.

Altera SPROM: Connect P11 and P12 through a 10-pin FRC cable when programming the Altera Configuration devices.

6.2.2) Switches

(1) Altera Mode Select: This switch is used when configuring the Altera FPGAs, through the configuration Device.

(2) Xilinx Mode Select: This switch is used to select the mode when configuring the Xilinx FPGAs.

6.2.3) LCD Display

UVLSI 201 supports on board 16X1 characters LCD display. Data has to be sent nibble by nibble from the PLD to the LCD module on its MS byte.

6.2.4) Daughter Board Connectors

Connectors J1, J2, J3 and J4 are provided to accommodate the daughter boards of various vendors.

6.2.5) Jumpers

Clock Select: This can be used to select different on board clock frequencies 4MHz, 16MHz, 25MHz.

6.2.6) On Board Programmer

UVLSI 201 features Onboard Programmer to program the Altera (EPC2) and Xilinx (XC18V01) Configuration devices.

6.2.7) RS-232 Connector

RS-232 interface standard is provided for implementing serial communication to and from computer. DB9 connector is used for connection of RS-232 interface.

CHAPTER 7

GPIO BOARD

Introduction

As the name says this is a Multi purpose I/O board i.e. it has almost all the primary interfaces that a PLD may be used for. This board is designed to interface PLDs of any make (Xilinx, Altera, Lattice, Actel etc), any gate count and any package. It has all input and output interfaces brought on to the two connectors on the board. The FPGA can be configured to fit on these connectors to complete the interface circuit [12]. The interfaces depend on the device gate count and the number of I/Os used in the design.

7.2 Details of GPIO 401A Board

The General Purpose I/O Board provides following interfaces.

- 16 digital inputs: Two 8 way DIP Switch with 3mm LED indication
- 16 digital outputs: 16 output LEDs
- 4 key switches: 4 tactile key switches
- 4 digits Multiplexed 7 Segment display (common anode type)
- 8 Channel 8 Bit ADC (Analog to Digital Converter)
- Single channel 8 bit DAC (Digital to Analog Converter)

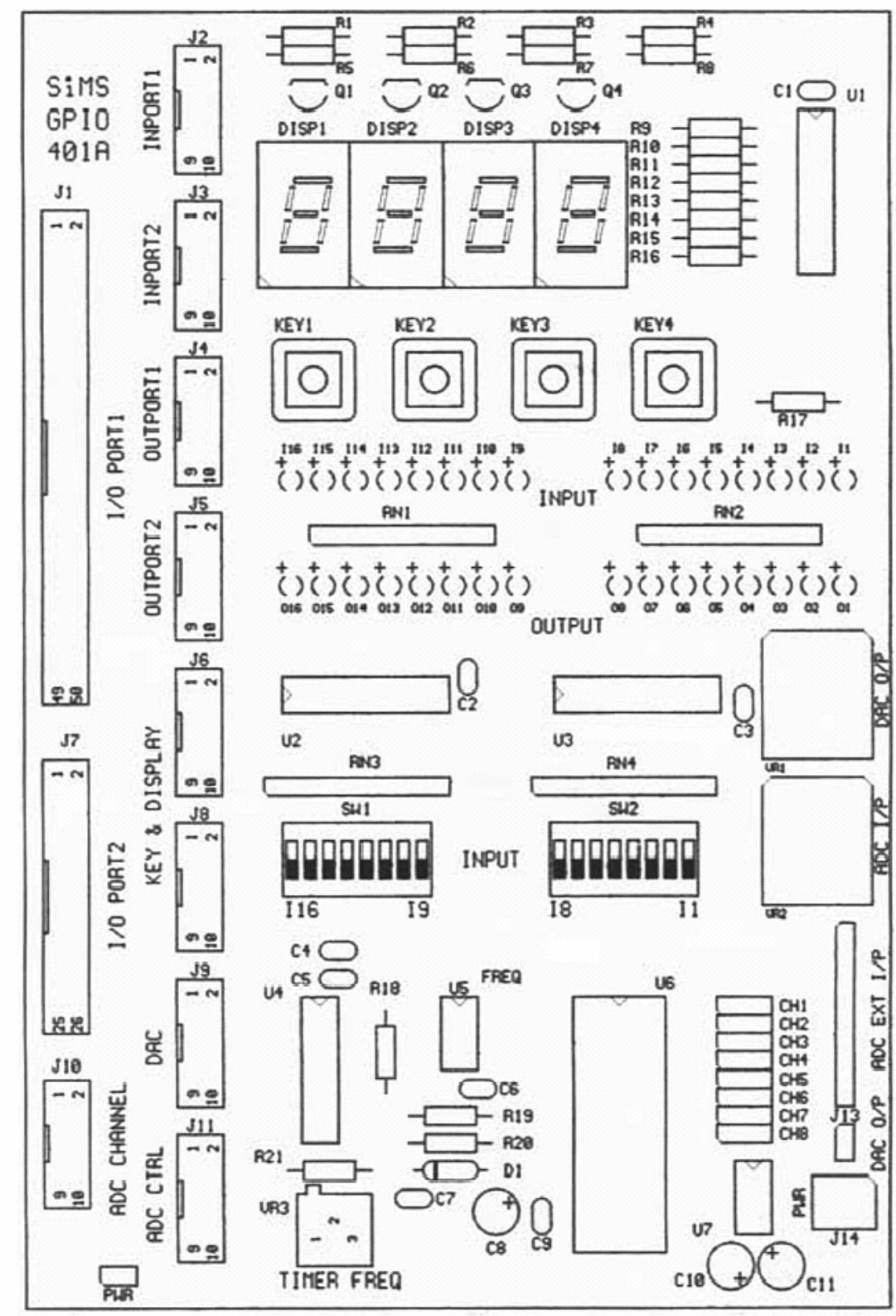


Figure 7.1 GPIO 401 Board Lay out

7.2.1) 16 DIP Switch Inputs

8 + 8 DIP switches, indented as (I16 ….I9, I8 ….I1) are available to give steady state inputs. They are all active high switches. The LEDs indicate the position of individual switches i.e. ON or OFF. The LED glows when the switch is ON.

Figure7.2 Input DIP Switches

7.2.2) 16 Output LEDs

16 Red LEDs, indented as (O16…O9, O8…O1) are available to indicate the steady state outputs. A high voltage level makes the LED glow.

Figure7.3 Output LEDs

7.2.3) 4 Key Interfaces

4 robust keys (K1 to K4) are arranged.

7.2.4) Multiplexed (4 digits) 7 Segment Display

7 segment display is available for multiplexed interface selectable through connectors I/O port1. All the 4 digits can be used for multiplexed interface connecting I/O port 1 from UVLSI 201 to I/O port 1 on the GPIO 401A.

A high output lights the segment. Similarly a high output also selects a digit. SEG A to SEG DP and DISP1 to DISP4 are total 12 pins of the PLD which are used for the 7 segment display function. For multiplexed displays, SEG A to SEG DP are connected to segments of all the 4 displays and DISP1 TO DISP4 is connected to the transistor base of display DISP1. Similarly DISP3 to DISP4 are connected to other transistor bases respectively.

7.2.5) 8-Bit ADC Interface

To provide exposure to the projects of analog world an Analog to Digital converter device interface is provided.

ADC0808 has been selected for this purpose. This is an 8-bit, successive approximation ADC with tri-state outputs. Successive approximation register (SAR) analog to digital converters (ADCs) are frequently used for medium-to-high-resolution applications with sample rates under 5 mega samples per second (Msps). SAR ADCs most commonly range in resolution from 8 to 16 bits and provide low power consumption as well as a small form factor. This combination makes them ideal for a wide variety of applications, such as portable/battery-powered instruments, pen digitizers, industrial controls, and data/signal acquisition.

The ADC channel selection and the control signals have to come from the FPGA. Analog input can be given from on board source and its value can be changed through the potentiometer (ADC I/P). External Analog input 0 to +5V can be given through the connector ADC EXT I/P with appropriate jumper settings. The Clock for ADC operation is generated on board using a 555 timer, which can be varied using the POT (timer frequency).

7.2.6) 8-Bit DAC Interface

Digital-to-Analog converter device interface is also provided on the GPIO to have interface with analog world. DAC0800 is used for this purpose. DAC 0800 is a single channel 8-bit DAC. The analog output of the DAC can be observed at pin no.1 of connector J13.

CHAPTER 8

FPGA BASED SCADA SYSTEM

Introduction

SCADA is an acronym for Supervisory Control and Data Acquisition. SCADA system is an intelligent system, which provides the facility of continuously monitoring, supervising and controlling any process. It is the first step towards automation. The SCADA system has become the backbone for monitoring, controlling and meeting the desired objectives of the process plant.

The functioning of a SCADA system consists of three stages:

- **Data Acquisition**: Data Acquisition is scanning of the channels in the specified order and at the specified frequency to acquire the data. There are many ways in which various channels can be addressed to read the data [4]. The analog data is converted to the digital form for processing and control purposes in the FPGA chip.
- **Data Processing:** The output of ADC is converted to the equivalent engineering units before any analysis is done. An ADC output value will correspond to a particular engineering value based on calibration of transmitter, ADC mode and digital output line. The data read from the ADC output for various channels is processed by the FPGA to carry out limit checking. For limit checking, limits for the channels are set using software. When any limit is violated, appropriate indicator like LED, alarm etc is activated.
- **Analysis and Control:** If the set limit is violated by the process output variable, some corrective actions should be taken to keep the desired performance. The system performance can also be analyzed. This analysis will enable us to visualize the problems in the system, and to take decisions regarding system modification or alternate operational strategy to increase the system performance. The software for analysis and control can be written depending on the type of analysis required.

The proposed scheme implements the SCADA system using UVLSI 201 kit, which includes an Altera FPGA and a general purpose input-output board.

VHDL programming environment is used to model the system. Details regarding each stage are presented below.

8.2 Data Acquisition

8.2.1) Channel Scanning

The FPGA scans the multiplexed channels continuously to capture the data. There are many ways in which FPGA can address the various channels and read the data. The most commonly used method is polling. In polling, the action of selecting a channel and addressing it is the responsibility of FPGA.

The channel selection may be sequential or in any particular order decided by the designer. It is also possible to assign priority to some channels over others i.e. some channels can be scanned more frequently than others. It is also possible to offer this facility of selecting the order of channel addressing and channel priorities to the operator level i.e. make these facilities as dynamic [4]. The FPGA may scan the channels continuously in the particular order or the channels may be scanned after every fixed time period.

In the proposed scheme, analog data is acquired from two multiplexed channels provided on the GPIO board. The FPGA scans these two channels, channel number 0 and 1, at the fixed time interval of two seconds, to acquire the data. To introduce the delay of two seconds, a counter circuit is implemented using VHDL. FPGA sends the address of the selected channel to the multiplexer to interface this channel to the ADC.

8.2.2) Analog to Digital Conversion

The captured analog data is converted to the digital form using ADC0808 interface, provided on the GPIO board. This requires the following actions to be taken by the FPGA [7]:

- Sending the channel address to the multiplexer
- Sending start convert pulse to ADC
- Reading the digital data at ADC output

ADC0808 is an 8 bit, successive approximation type ADC with tri-state outputs. Channel selection and control signals come from FPGA on the UVLSI 201. Hence, ADC is controlled by the FPGA. The clock for ADC operation is generated on board using a 555 timer, which can be varied using the POT [12].

Address of the selected channel is sent to the ADC0808 and signal 'ale' is sent to latch this address in ADC. A pulse to the 'start' pin of the ADC is required to start the conversion process and to disable the tri-state output buffer. At the end of the conversion period, 'end of conversion' pin becomes active and the digital output is made available at the output buffer. To read the digital data at ADC output, the end of conversion signal of ADC chip is read by the FPGA and when it is 'high', digital data is read and stored in the chip.

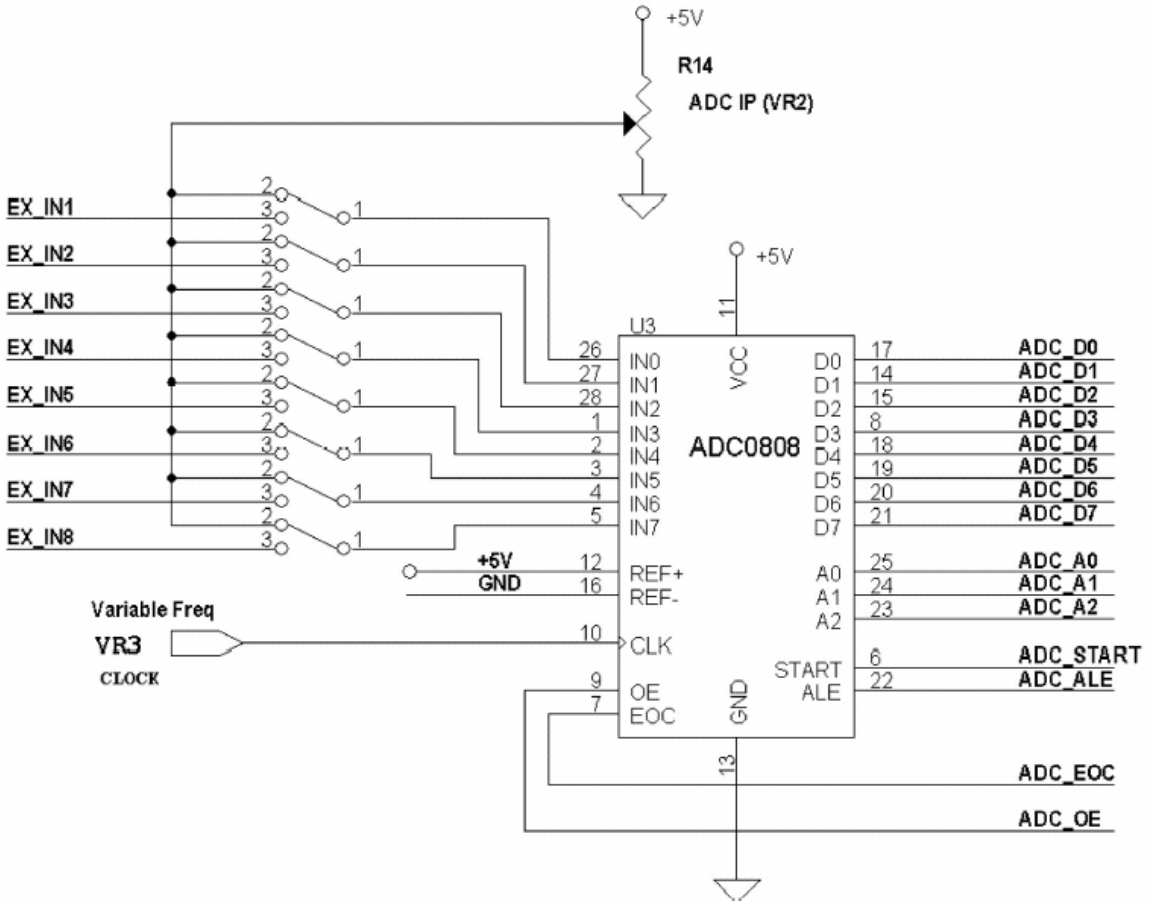


Figure8.1 Block Diagram of ADC0808

8.2.3) Seven Segment Display

7 segment displays are available for multiplexed interface selectable through connectors I/O port1. All the 4 digits can be used for multiplexed interface connecting I/O port1 from UVLSI 201 to I/O port1 on the GPIO 401A.

The displays are common anode type. A high output lights the segment. Similarly, a high output also selects a digit. SEG A to SEG DP and DISP1 to DISP4 are total 12 pins of the FPGA, which are used for the seven segment display function.

For multiplexed displays, SEG A to SEG DP are connected to segments of all the 4 displays and DISP1 is connected to the transistor base of display DISP1. Similarly DISP3 to DISP4 are connected to other transistor bases respectively [12].

To select a particular digit for display, its base drive is made low through programming. For example, to display '0' on digit 1, DISP1 is made low and segments "gfedcba" are set to ""0111111".

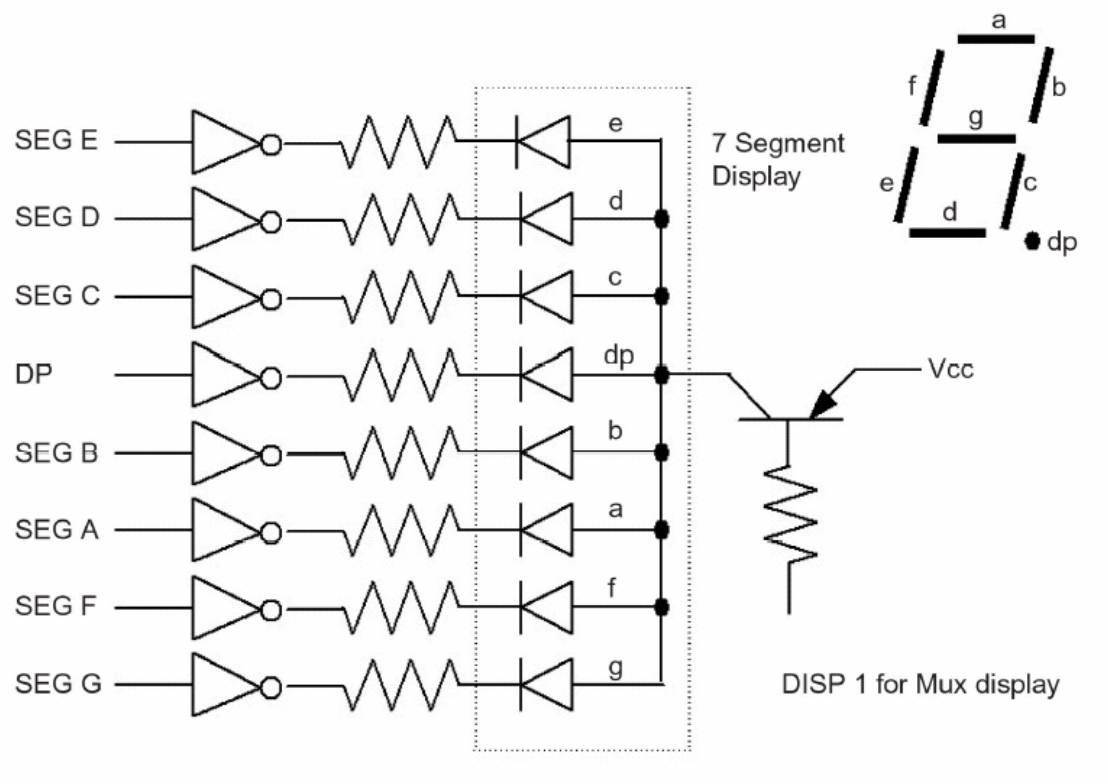


Figure8.2 Seven Segment Display

Address of the selected channel is displayed on the first digit and the data acquired is displayed in hexadecimal form using third and fourth digit. Second digit has not been used.

The kit has an on-board clock of 4 MHz and above. It is not possible for human eyes to observe and notice the changes at such high frequency. So, a clock divider circuit is implemented, which generates another clock of less frequency. This circuit divides the frequency of the on board clock by any desired number and the low frequency clock is used for implementing the display logic.

8.3 Data Processing

The FPGA reads and stores the captured data in digital form. To check whether the data is within the specified limits or not, it is processed by the FPGA.

In the interfaced process, if the captured data is less than or equal to the set limit, it means that the process is working in the desired manner. But if the data crosses the set limit, it means that the system is behaving in an unacceptable manner and it need to be manipulated by some means.

To apply this logic, a comparator is implemented in VHDL which compares the process output variable with the set limit. Whenever it becomes more than the set limit, red LED glows. The glowing LED indicates that the process behavior is not in accordance to our needs and it should be controlled by some means. In addition to limit checking, the system performance may also be analyzed. This will help us to visualize the problems in the system.

8.4 Analysis and Control

Whenever the captured data goes beyond the specified limit, process needs to be manipulated and some corrective actions should be taken. For example, in a temperature control system, if the temperature becomes more than the set limit, fans should be switched on, or steam supply to the heater should be reduced. In a level control system, if level becomes more than the set point, inlet valve opening should be reduced or outlet valve opening should be increased.

To actuate these mechanical devices or to take any other corrective measure, a 5-volt analog control signal is generated by the FPGA and DAC0800. DAC 0800 is a single channel, 8-bit digital-to-analog converter provided on the GPIO to have interface with analog world. Figure 8.3 shows the block diagram of the DAC0800.

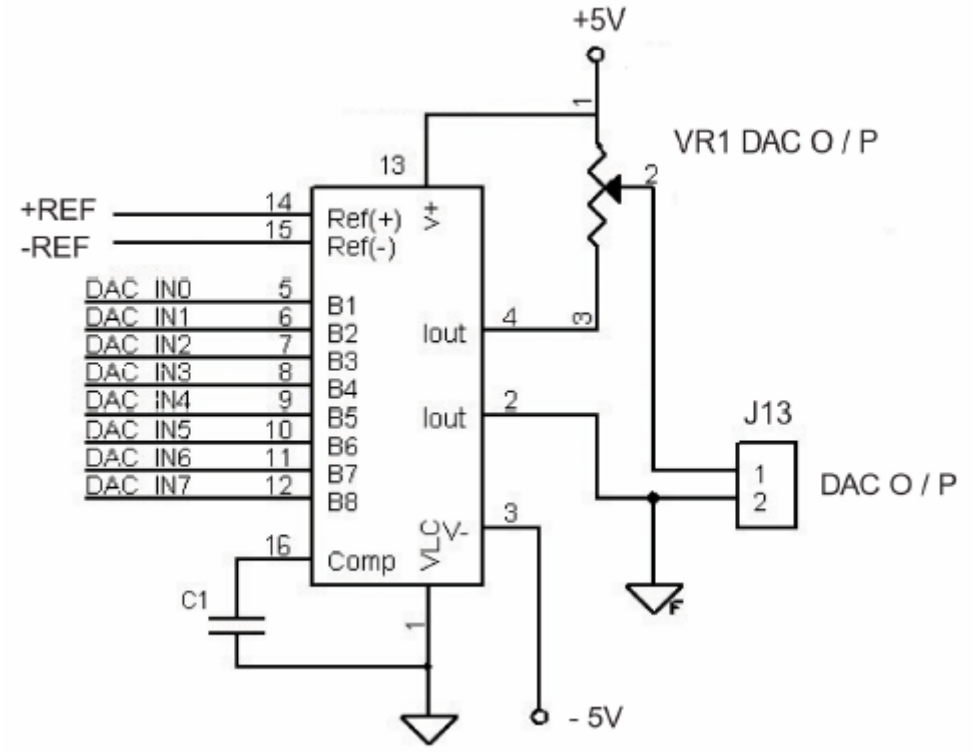


Figure8.3 Block Diagram of DAC0800

If the system is not performing in the desired way, the FPGA generates a 5-volt digital control signal by applying signal "11111111" at DAC inputs. DAC 0800 is a single channel, 8-bit DAC and converts this digital signal to analog form; this 5-volt analog signal is available at the DAC output port. It can be used to manipulate the process, so that the system behaves in the desired way.

8.5 Pin Locking in FPGA

ACEX50K FPGA has 144 pins that can be used for different functions. In this project, 38 pins have been used; 1 output pin to indicate fault, 1 reset, 2 clocks, 8 ADC outputs, 7 ADC control signals, 8 DAC inputs, 11 for 4-digits, seven segment display.

PIN NAME	DEVICE PIN NO	PROPERTY	SIGNALS FROM GPIO BOARD
clk	125	DED I/P 4	RESET
rst	126	DED CLK2	CLK2
ADC			
data[0]	47	ADC_D0	Data 0 from ADC 0808
data[1]	48	ADC_D1	Data 1 from ADC 0808
data[2]	49	ADC_D2	Data 2 from ADC 0808
data[3]	51	ADC_D3	Data 3 from ADC 0808
data[4]	54	ADC_D4	Data 4 from ADC 0808
data[5]	59	ADC_D5	Data 5 from ADC 0808
data[6]	60	ADC_D6	Data 6 from ADC 0808
data[7]	62	ADC_D7	Data 7 from ADC 0808
eoc	69	ADC_EOC	Interrupt signal from ADC 0808
start_conv	67	ADC_START	SOC for ADC
ale	68	ADC_ALE	ADC ALE signal
oe	70	ADC_OE	ADC output enable
addr[0]	63	ADC_A0	ADC channel select bit
addr[1]	64	ADC_A1	ADC channel select bit
addr[2]	65	ADC_A2	ADC channel select bit

PIN NAME	DEVICE PIN NO	PROPERTY	SIGNALS FROM GPIO BOARD
DISPLAY			
disp_clk	55	DED CLK1	CLK1
base[0]	36	DISP1	Digit 0 select o/p
base[1]	33	DISP 2	Digit 1 select o/p
base[2]	32	DISP 3	Digit 2 select o/p
base[3]	31	DISP 4	Digit 3 select o/p
seg[0]	83	SEG A	Segment ‘a’
seg[1]	82	SEG B	Segment ‘b’
seg[2]	81	SEG C	Segment ‘c’
seg[3]	80	SEG D	Segment ‘d’
seg[4]	79	SEG E	Segment ‘e’
seg[5]	78	SEG F	Segment ‘f’
seg[6]	73	SEG G	Segment ‘g’
DAC			
dac_in[0]	37	DAC0	DAC data0 from FPGA
dac_in[1]	38	DAC1	DAC data1 from FPGA
dac_in[2]	39	DAC2	DAC data2 from FPGA
dac_in[3]	41	DAC3	DAC data3 from FPGA
dac_in[4]	42	DAC4	DAC data4 from FPGA
dac_in[5]	43	DAC5	DAC data5 from FPGA
dac_in[6]	44	DAC6	DAC data6 from FPGA
dac_in[7]	46	DAC7	DAC data7 from FPGA
fault	110	O/P	Output pin

Results & Discussion

The designed SCADA system provides the facility of monitoring, processing and controlling any process. The functioning of this system consists of three stages: Data Acquisition, Data Processing and Analysis & Control. These three stages have been implemented using UVLSI201 trainer kit, which includes an FPGA and a general purpose input-output board.

FPGA scans two external channels, channel number 0 and 1 at fixed time interval of two seconds. FPGA sends the address of the selected channel and control signals for the ADC. ADC converts the captured analog data to the digital form. The acquired data is displayed in the hexadecimal form using the multiplexed 4 digits, seven segment display.

The digital data is analyzed and processed by the FPGA to check any limit violation. If the data crosses the specified limit, red LED glows indicating the unacceptable behavior of the process. The process needs to be manipulated to achieve the desired performance. A 5-volt analog signal is generated which may control a valve, switch etc. to make the process behavior in accordance to our needs.

The designed system has been implemented in the VHDL programming environment. The results obtained for the designed FPGA based SCADA system are presented in two stages; first ACEX device performance is considered and after that the performance of SCADA system is analyzed.

1. ACEX50K Device Performance

Figures 1 to 8 show the performance of ACEX50K device in concern with the project. Figure 1 shows the Compilation process. Compiler tool consists of mainly four modules; Analysis & Synthesis, Fitter, Timing Analyzer and Assembler.

- Analysis & Synthesis module checks the code for any syntax error and converts it into technology specific netlist.
- Fitter performs the mapping between logical description (netlist) and physical description (floorplanning).

- The Timing Analyzer provides point-to-point timing delay information, setup and hold time analysis.
- Assembler converts the VHDL code into binary file, which is downloaded or programmed into the FPGA.

Time elapsed during each stage of compilation is shown in figure 2. The SCADA code compilation process took only 11 seconds.

Floorplanning of the device is shown in figure 3. ACEX EP1K50 device has 10 rows (A,B…J) and 36 columns (1,2…36). Figure 4 shows the translation of logic design into the physical design and the routing of interconnections between various LABs. Placing and routing of the nodes e.g. seg, eoc, oe, addr, base etc. is shown clearly.

To make the data display steady, frequency of the onboard clock is reduced. This approach requires a counter implementation and carry chains are used for it. Figure 5, Carry Chain Usage shows the length of carry chain versus number of carry chains. Figure 6 illustrates the LABs usage in this project. It shows the number of logic elements versus number of logic array blocks.

ACEX device has 360 logic array blocks, 10 embedded array blocks, 2880 logic elements, 3468 registers, 102 general purpose input-output pins, 40960 memory bits. Figure 7 gives the resource usage summary for this project. It shows that only 133 logic elements and 38 input-output pins are used for the designed system. Hence the great logic capability of the FPGA is proved.

The final output of compilation process is in the binary form, which is downloaded or programmed in the FPGA. This process is called programming of FPGA and is shown in figure 8. Passive serial configuration scheme is used for programming and ByteBlasterMV download cables act as data source for FPGA.

Figure1 Compilation Process

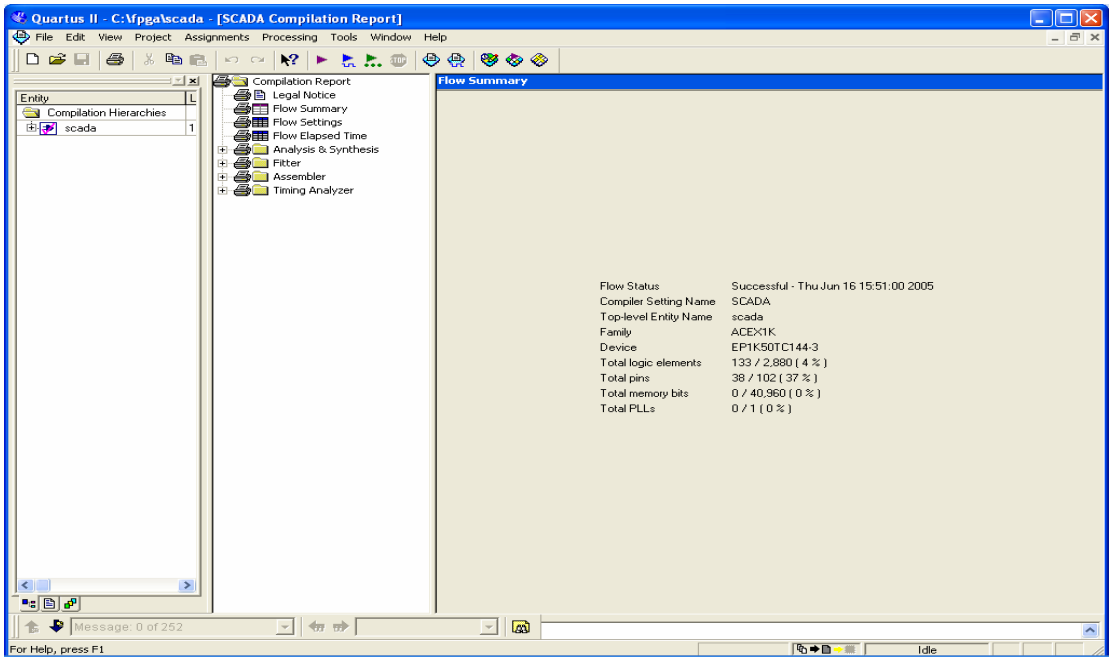


Figure2 Flow Elapsed Time

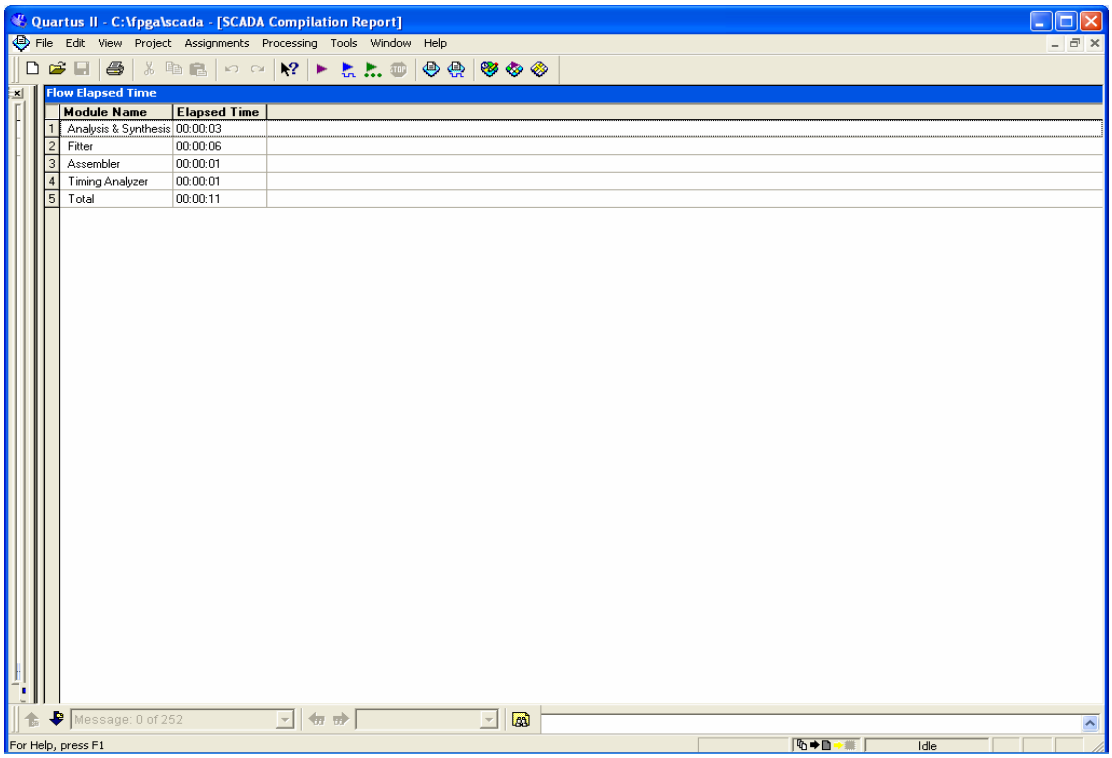


Figure3 Floor Planning

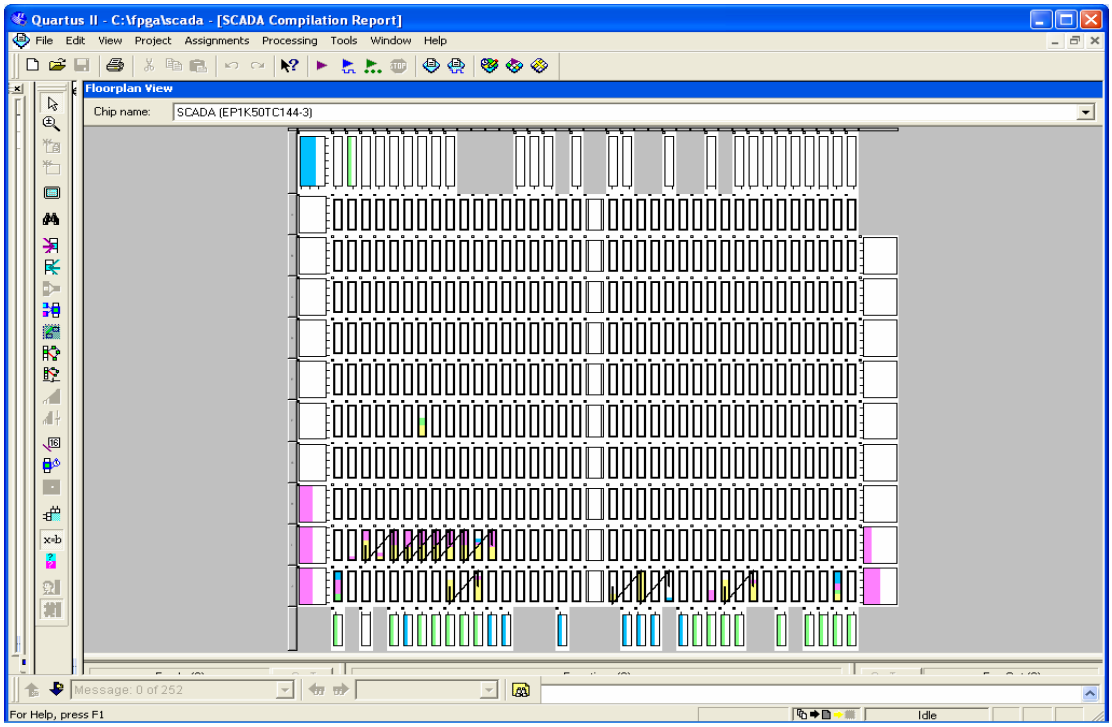


Figure4 Internal view of LABs

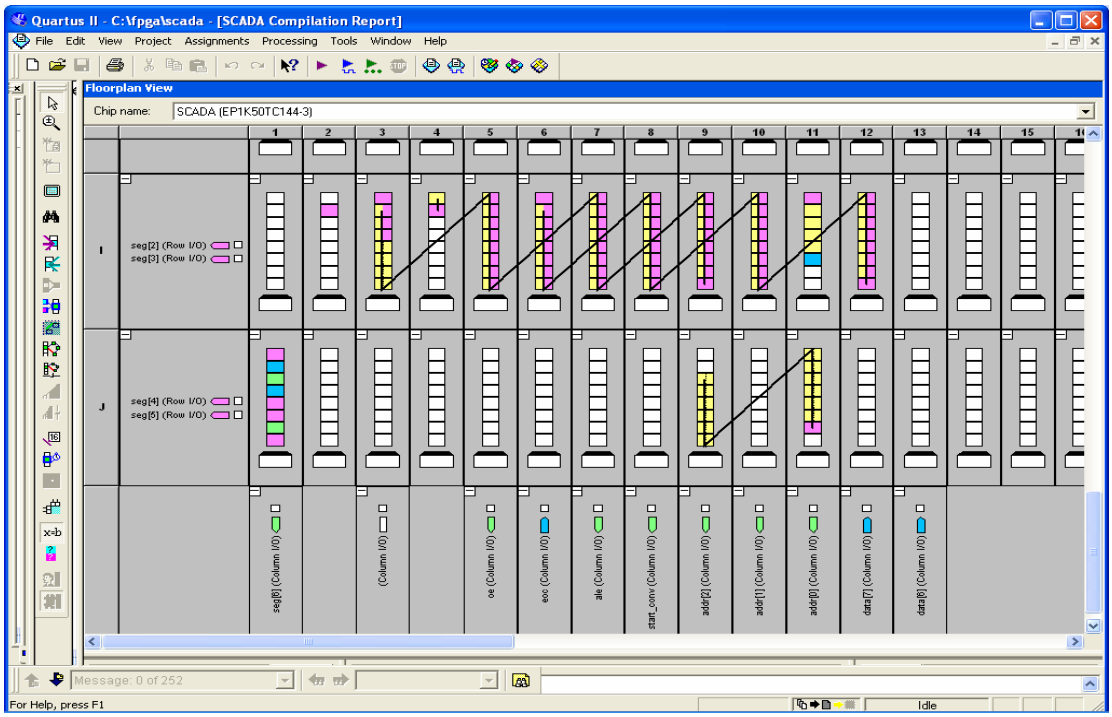


Figure5 Carry Chain Usage

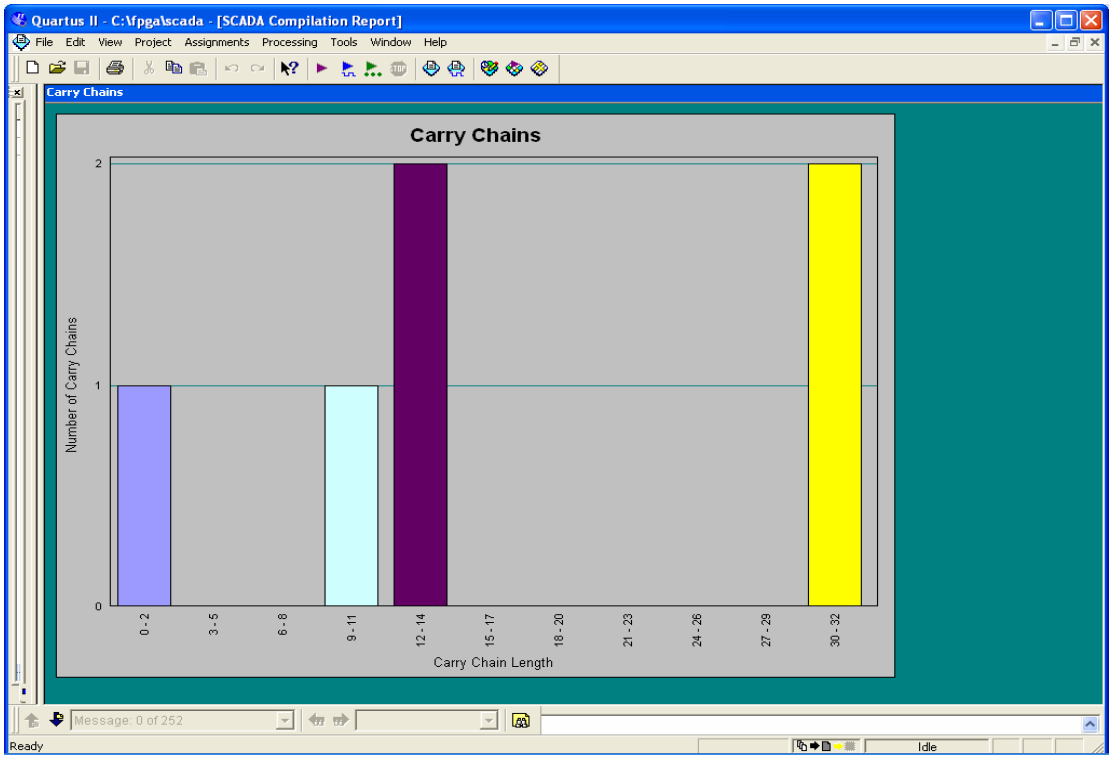


Figure6 Logic Array Blocks Usage

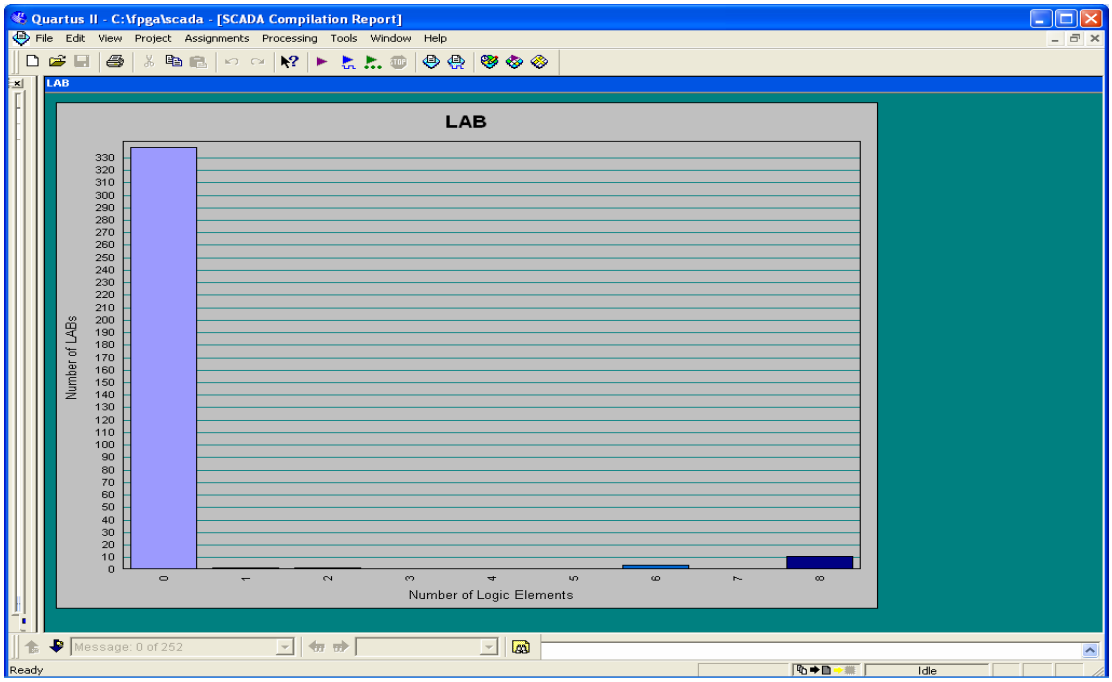


Figure7 Resource Usage Summary

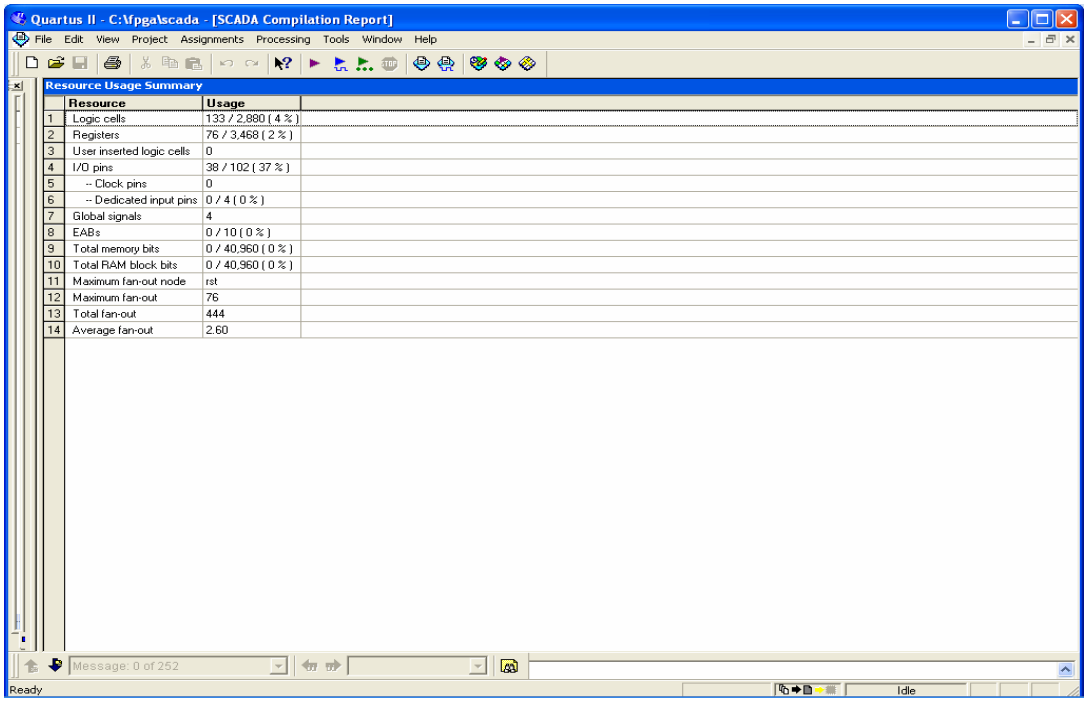
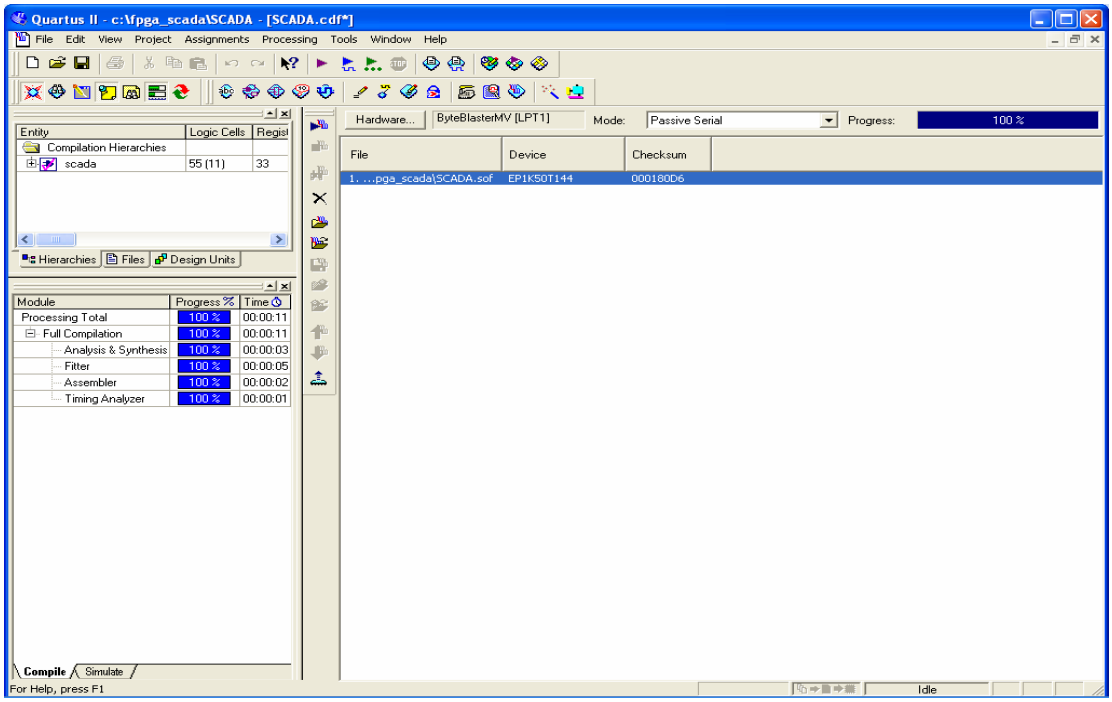


Figure8 Programming of FPGA



2. SCADA System Performance

The data is acquired from external channels at the fixed interval of two seconds and processed within the FPGA to check any limit violation. The set limit is 3 volts i.e. 99h. Whenever the data goes beyond the set limit, red LED glows indicating a fault in the process. First digit displays the channel number; third and fourth digits display the acquired data in the hexadecimal form.

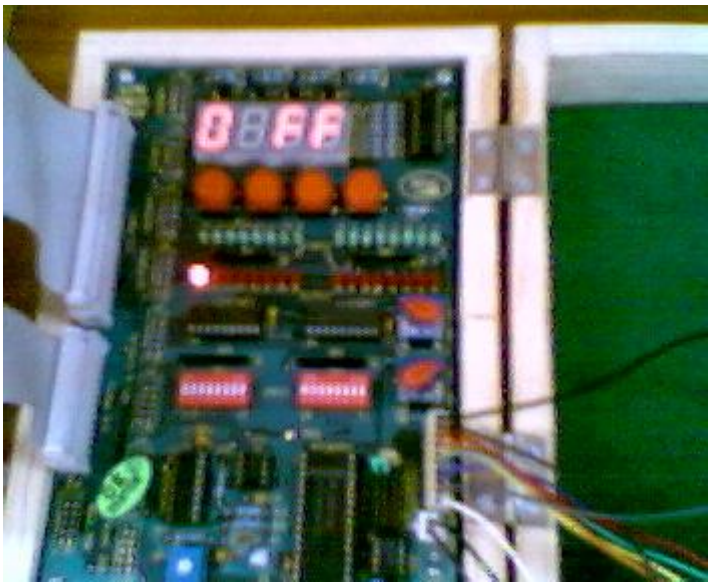
Figure9 Channel number 0 is scanned and the acquired data is 97h, which is less than the set limit 99h i.e. 3 volts. So, no fault is indicated.

Figure10 Channel number 1 is scanned and the acquired data is 8Ah, which is less than the set limit 99h i.e. 3 volts. So, no fault is indicated.

Figure11 Channel number 0 is scanned and the acquired data is 9dh, which is greater than the set limit 99h i.e. 3 volts. So, LED glows indicating a fault.

Figure12 Channel number 1 is scanned and the acquired data is 9Ah, which is greater than the set limit 99h i.e. 3 volts. So, LED glows, indicating a fault.

Figure13 Channel number 0 is scanned and the acquired data is FFh, which is greater than the set limit 99h i.e. 3 volts. So, LED glows, indicating a fault.



It is clear from the above illustrations that FPGA has excellent logic capabilities, enormous processing resources, negligible & predictable delay and very high clock speed. So, it is suitable for real time data capturing and even a complex digital system can be easily implemented with it. FPGA is reprogrammable and can be reprogrammed in a few milliseconds, so the designed system can be modified according to the requirements.

Conclusion

An FPGA based Supervisory Control and Data Acquisition system is successfully designed and tested for the desired performance. The designed system provides the facilities of data acquisition, processing and control along with the advantages provided by the FPGA. FPGA based SCADA system is an excellent mean for process control facilities to save time and money.

FPGA has the greatest logic capabilities, enormous processing resources, high clock speed and negligible delay, so it is suitable for real system data capturing, processing and control. The designed system used only 4% Logic Elements, 2% registers, 37% Input/Output pins, 4 global signals, 0% Embedded Array Blocks, 0% Memory bits of the ACEX50K FPGA, which proves that the FPGA is an excellent option to implement any digital circuit. Even complex circuits can be easily implemented using FPGA. Redesigning of the SCADA system can be done easily according to the needs, as FPGA is reprogrammable. The designed system mainly consists of digital components; it is more accurate and reliable.

Scope for Further Work

The designed FPGA based SCADA system provides the facility of ON-OFF control. Any real time application like temperature control system, level control system can be interfaced with this system to maintain the temperature or level at the desired set point. The system can be extended to provide the facility of PID control. PID control can be implemented with VHDL and then, can be programmed into the FPGA.

The designed SCADA system has interface with only eight external channels. In any application, if the number of channels is quite large, then to interface them to the FPGA we have to use multiplexers at different levels. To interface 256 channels, we will have to use 17 multiplexers of 16 channels each. This approach will suit the processes which are basically slow. Even if a channel is scanned only once in every scan it will be only after 255 channels have been scanned, limit checking and analysis have been performed, a particular channel will be addressed again. This may be acceptable in many processes.

However, many processes are quite fast and thus only alternative is to use more than one SCADA system and distribute the channels among them. For performance analysis on the process plant it is mandatory that the data from various channels should reach a central location where it can be consolidated and analyzed to generate the reports on plant performance.

Appendix: SOURCE CODE

1. ADC (ANALOG TO DIGITAL CONVERTER)

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

ENTITY ADC is port (
                              //ADC
            clk: in STD_LOGIC;                    //ADC clock input
            rst: in STD_LOGIC;                       // reset input
            eoc: in STD_LOGIC;       // End of Conversion from ADC
            data: in STD_LOGIC_VECTOR (7 downto 0);
                                      //Data available from ADC
                        // DISPLAY
            disp_clk: in STD_LOGIC;
            base: buffer STD_LOGIC_VECTOR (3 downto 0);
                        //Base drive to the seven segment display
            seg: out STD_LOGIC_VECTOR (6 downto 0);
                                        //7 segments of display
                              //ADC control
            start_conv: buffer STD_LOGIC;
                                     //Start conversion from PLD
            addr: out STD_LOGIC_VECTOR (2 downto 0);
                           //Channel selection address to the ADC
            ale: out STD_LOGIC;            // Address Latch Enable
            oe: out STD_LOGIC );                   // Output enable
END ADC;
ARCHITTECTURE convert of ADC is
            SIGNAL clk_out: STD_LOGIC;
            SIGNAL clk_chnl: STD_LOGIC;
            SIGNAL cntr2 :STD_LOGIC_VECTOR(12 downto 0);
            SIGNAL cntr: STD_LOGIC_VECTOR (12 downto 0);
            SIGNAL base_cnt: STD_LOGIC_VECTOR (1 downto 0);
            SIGNAL muxseg_opt: STD_LOGIC_VECTOR (3 downto 0);
            SIGNAL nst, cst: STD_LOGIC_VECTOR (3 downto 0);
BEGIN
              //Divider counter
              PROCESS (disp_clk, rst)
```

```
            BEGIN
                  IF (rst = ‘0’) THEN
                           cntr <= (others => 0 );
                  ELSIF (disp_clk’ event and disp_clk = ‘1’)THEN
                           cntr <= cntr + 1 ;
                  ENDIF;
            END PROCESS;
// Divided clock o/p from 4 KHz
            clk_out <= cntr(11) ;

//Generating clk_chnl of 0.5 MHz
            PROCESS (clk_out, rst)
            BEGIN
                   IF (rst = '0') THEN
                           cntr2 <= (others => '0');
                   ELSIF (clk_out 'event and clk_out ='1')THEN
                           cntr2 <= cntr2 + 1;
                   END IF;
            END PROCESS;
         clk_chnl <= cntr2(10);

            PROCESS (clk_chnl, rst)
            VARIABLE count: Natural: = 0;
            BEGIN
                  IF (rst = '0') THEN
                           count:= 0;
                  ELSIF(clk_chnl 'event and clk_chnl = '1')THEN
                           chnl_sel <= "000";
                           count: = count + 1;
                           IF (count = 2) THEN
                               chnl_sel <= "001";
                               count: =0;
                           END IF;
                   END IF;
            END PROCESS;
            PROCESS (clk_out, rst)
            BEGIN
                   IF (rst = ’0’) THEN
                           base_cnt <= “00”;
                   ELSIF (clk_out’ event and clk_out = ‘1’)THEN
```

```
                        base_cnt <= base_cnt + 1;
                ENDIF;
        END PROCESS;

        PROCESS (base, data, chnl_sel)
        BEGIN
                CASE base is
                WHEN “1110” =>
                        muxseg_opt (2 downto 0) <= chnl_sel
                        muxseg_opt (3) <= ‘0’;
                WHEN “1011” =>
                        muxseg_opt <= data (7 downto 4);
                WHEN “0111” =>
                        muxseg_opt <= data (3 downto 0);
                WHEN others =>
                        muxseg_opt <= “0000”;
                END CASE;
        END PROCESS;

base <= “1110” WHEN base_cnt = “00” ELSE
     <= “1101” WHEN base_cnt = “01” ELSE
                                        //Digit 3 not selected
     <= “0111” WHEN base_cnt = “11” ELSE
        “1111”;

seg   <=  “0111111”  WHEN  muxseg_opt  =  “0000”  ELSE
seg   <=  “0000110”  WHEN  muxseg_opt  =  “0001”  ELSE
seg   <=  “1011011”  WHEN  muxseg_opt  =  “0010”  ELSE
seg   <=  “1001111”  WHEN  muxseg_opt  =  “0011”  ELSE
seg   <=  “1100110”  WHEN  muxseg_opt  =  “0100”  ELSE
seg   <=  “1101101”  WHEN  muxseg_opt  =  “0101”  ELSE
seg   <=  “1111101”  WHEN  muxseg_opt  =  “0110”  ELSE
seg   <=  “0000111”  WHEN  muxseg_opt  =  “0111”  ELSE
seg   <=  “1111111”  WHEN  muxseg_opt  =  “1000”  ELSE
seg   <=  “1100111”  WHEN  muxseg_opt  =  “1001”  ELSE
seg   <=  “1110111”  WHEN  muxseg_opt  =  “1010”  ELSE
seg   <=  “1111100”  WHEN  muxseg_opt  =  “1011”  ELSE
seg   <=  “0111001”  WHEN  muxseg_opt  =  “1100”  ELSE
seg   <=  “1011110”  WHEN  muxseg_opt  =  “1101”  ELSE
seg   <=  “1111001”  WHEN  muxseg_opt  =  “1110”  ELSE
```

```
    seg    <=  "1110001"  WHEN  muxseg_opt  =  "1111"  ELSE
               "0000000";

addr <= chnl_sel;
oe <= '1';

              // ADC section
              PROCESS (cst, eoc)
                   CONSTANT s0: STD_LOGIC_VECTOR:= "0000";
                   CONSTANT s1: STD_LOGIC_VECTOR:= "0001";
                   CONSTANT s2: STD_LOGIC_VECTOR:= "0010";
                   CONSTANT s3: STD_LOGIC_VECTOR:= "0011";
                   CONSTANT s4: STD_LOGIC_VECTOR:= "0100";
                   CONSTANT s5: STD_LOGIC_VECTOR:= "0101";
                   CONSTANT s6: STD_LOGIC_VECTOR:= "0110";
                   CONSTANT s7: STD_LOGIC_VECTOR:= "0111";
              BEGIN
                   SIGNAL ale: STD_LOGIC;
                   SIGNAL start_conv_1: STD_LOGIC;
              CASE cst is
              WHEN s0 => nst <= s1;
              WHEN s1 => ale_1 = '1'; nst <= s2;
              WHEN s2 => ale_1 <= '1'; nst <= s3;
              WHEN s3 => start_conv_1 <= '1'; nst <= s4;
              WHEN s4 => nst <= s5;
              WHEN s5 => IF (eoc = '1') THEN   nst <= s6;
                                       ELSE   nst <= s5;
                         ENDIF;
              WHEN s6 => nst <= s7;
              WHEN others => nst <= "0000";
              END CASE;
              END PROCESS;

              PROCESS (clk, rst)
              BEGIN
                      IF (rst = '0') THEN
                             cst <= "0000";
                      ELSIF (clk' event and clk = '1') THEN
                             cst<= nst;
                      ENDIF;
```

```
        END PROCESS;

        //Registered output
        PROCESS (clk, rst)
        BEGIN
                IF (rst = '0') THEN
                        ale <= '0';
                        start_conv <='0';
                ELSIF (clk' event and clk = '1') THEN
                        ale <= ale_1;
                        start_conv <= start_conv_1;
                ENDIF;
        END PROCESS;
END convert;
```

2. Data Processing

```
library ieee;
use ieee.std_logic_1164.all;

ENTITY Data_Process is port (
            rst: in STD_LOGIC;
            disp_clk: in STD_LOGIC;
            data: in STD_LOGIC_VECTOR(7 downto 0);
            fault: out STD_LOGIC;
            dac_in: out STD_LOGIC_VECTOR( 7 downto 0)
            );
END Data_Process;

ARCHITECTURE BEHAVIOR of Data_Process is
      Signal cntr1: STD_LOGIC_VECTOR(13 downto 0);
      Signal clk_out1: STD_LOGIC;
BEGIN
      PROCESS (disp_clk, rst)
      BEGIN
              IF (rst = '0') THEN
                cntr1 <= (others => '0');
              ELSIF (disp_clk 'event and disp_clk ='1') THEN
                cntr1 <= cntr1 + 1;
              END IF;
      END PROCESS;
        clk_out1 <= cntr1(11);
      PROCESS (clk_out1, data)
          CONSTANT LIMIT: STD_LOGIC_VECTOR(7 downto 0):=
"10011001";
      BEGIN
          IF (clk_out1 'event and clk_out1 ='1') THEN
               IF (data > limit)THEN
                  fault <= '1';
                  dac_in <= “11111111”;
               ELSE fault <= '0';
               END IF;
          END IF;
      END PROCESS;
      END BEHAVIOR;
```

3. SCADA SYSTEM

```
library ieee;
use ieee.std_logic_1164.all;
ENTITY SCADA is port(
                    //Data Process Signals
          data: in STD_LOGIC_VECTOR (7 downto 0);
          fault: out STD_LOGIC;
          dac_in: out STD_LOGIC_VECTOR( 7 downto 0);

                    //ADC clock input
          clk: in STD_LOGIC;        //CLK for ADC from 555 timer
          rst: in STD_LOGIC;                    // reset input
          eoc: in STD_LOGIC;        //End Of Conversion from ADC
          chnl_sel: in STD_LOGIC_VECTOR(2 downto 0);
                      //channel selection external switch
                    //display
          disp_clk: in STD_LOGIC;  //display refresh clock input
          base: out STD_LOGIC_VECTOR(3 downto 0);
                    //Base drive to the 7 segment display
          seg: out STD_LOGIC_VECTOR(6 downto 0);
                                  //7 Segments of display
                     //ADC
          start_conv: out STD_LOGIC; //Start Conversion from PLD
          addr: out STD_LOGIC_VECTOR(2 downto 0);
                   //Channel selection address to the ADC
          ale: out STD_LOGIC;            //Address Latch Enable
          oe: out STD_LOGIC);
        //O/P enable when goes hi data is available at data lines
END SCADA;

ARCHITECTURE SCADA of SCADA is
     COMPONENT Data_Process port (
                    rst: in STD_LOGIC;
                    disp_clk: in STD_LOGIC;
                    data: in STD_LOGIC_VECTOR (7 downto 0);
                    fault: out STD_LOGIC;
                    dac_in: out STD_LOGIC_VECTOR(7 downto 0));
     END COMPONENT;
```

```
COMPONENT ADC port (
                    //ADC clock input
        clk: in STD_LOGIC;
        rst: in STD_LOGIC;                          //reset input
        eoc: in STD_LOGIC;        //End Of Conversion from ADC
        chnl_sel: in STD_LOGIC_VECTOR (2 downto 0);
                  //channel selection through external switch
        data: in STD_LOGIC_VECTOR(7 downto 0);
                                  //Data available from ADC

                        //display
        disp_clk: in STD_LOGIC;  //display refresh clock input
        base: out STD_LOGIC_VECTOR(3 downto 0);
                        //Base drive to the 7 segment display
        seg: out STD_LOGIC_VECTOR(6 downto 0);
                                     //7 Segments of display
                          //ADC
        start_conv: out STD_LOGIC; //Start Conversion from PLD
        addr: out STD_LOGIC_VECTOR(2 downto 0);
                       //Channel selection address to the ADC
        ale: out STD_LOGIC;             //Address Latch Enable
        oe: out STD_LOGIC);
              //O/P enable when goes high data is available at
              //data lines.
END COMPONENT;


BEGIN
ADC1: ADC port map (clk, rst, eoc, chnl_sel, data, disp_clk, base,
seg, start_conv, addr, ale, oe);
Data_Process1: Data_Process port map (rst, disp_clk, data, fault,
dac_in);
END SCADA;
```

References

[1] Barr, Michael "Programmable Logic: What's it to Ya?", June 1999, pp. 75-84.

[2] Bhasker J., "A VHDL Primer", Pearson Education Pte. Ltd., 3rd Ed., 2004.

[3] Brown D. Stephen, Francis J. Robert, Rose J., Vranesic G. Zvonko, "Field Programmable Gate Arrays", Kluwer Academic Publishers, 1997.

[4] Kant Krishan, "Computer Based Industrial Control", ISTE Learning Materials Centre, First Ed., 2001.

[5] Floyd L. Thomas "Digital Fundamentals", Pearson Education Publications, 8th Ed., 2004.

[6] Floyd L. Thomas "Electronic Devices", Pearson Education Publications, 6th Ed., 2003.

[7] Gaonkar S. Ramesh, "Microprocessor Architecture, Programming & Applications with the 8085", Penram International Publishing, 4th Ed., 2000.

[8] Perry, D., "VHDL", McGraw Hill Publications, 1991.

[9] Rabaey, Chandrakasan, Nikolic, "Digital Integrated Circuits: A Design Perspective", Pearson Education Pte. Ltd., 2nd Ed., 2003.

[10] Razavi, Behzad, "Principles of Data Conversion System Design", IEEE Press, 1995

[11] Sebastian Smith J. Michael, "Application Specific Integrated Circuits", Pearson Education Pte. Ltd., 9th Ed., 2004.

[12] Silicon Micro Systems, "UVLSI-201 Technical Reference Manual".

[13] Trimberger M. Stephen, McCarty D., Whitney T., Hartmann R., "Field Programmable Gate Array Technology" Kluwer Academic Publishers, 4th Ed., 1999.

[14] Plassche D. Van, Rudy, "Integrated Analog-to-Digital and Digital-to-Analog Converters", Kluwer Academic Publishers, 1994.

[15] Yarbrough M. John, "Digital Logic Applications and Design", PWS Publishing Company, 1997.

[16] Wakerly F. John, "Digital Design, Principles and Practices", PHI Publications, 3rd Ed., 2003.

[17] www.actel.com/products

[18] www.altera.com/literature/lit-acx.jsp

[19] www.altera.com/products/devices/dev.index.jsp

[20] www.angelfire.com/electronics/in/vlsi/books.html

[21] www.autosoln.com

[22] www.beyondlogic.org/serial/serial1.htm

[23] www.celoxica.com/techlib/files/CEL-W0307171HR6-FPGA

[24] www.eedesign.com

[25] www.eg3.com/fpga

[26] www.epgco.com

[27] www.iclinks.com

[28] www.nationjob.com/company/acns

[29] www.netrino.com/articles/programmableLogic

[30] www.ref.web.cern.ch/ref/CERN/CNL/2000/003/scada

[31] www.simsteam.com

[32] www.sensiblesoftware.com/articles

[33] www.sss-mag.com/scada.html

[34] www.webopedia.com/TERMS/S/SCADA.html

[35] www.xilinx.com/company/about/programmable.html