CERTIFICATE

This is to certify that this report entitled," **"TEMPREATURE SENSING BY USING GSM MODULE WITH THE HELP OF FPGA"**, submitted by, Pallavi Verma Roll No-12239 in the partial fulfilment of the requirement for the award of the degree of **Master of Engineering** in **Control and Instrumentation** ,Electrical Engineering Department and 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 endeavours.

Date:

(Dr. Parmod Kumar)
Professor & Head
Deptt. Of Electrical Engg
Delhi College of Engineering
Delhi -1100042

ACKNOWLEDGEMENT

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.

I would, first of all, like to thank **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. I express my deepest sense of gratitude for his valuable support in this regard.

I am also indebted to all my Teachers and Professors whose technical support was always there, whenever it was required.

I also want to say thanks to **Mr. Karan Singh**, Laboratory Assistant, and Department of Electrical Engineering.

I want to say thanks to my friends ***B.Lavanya, Abhishek Pathak, Hemant Yadav, Nishant Gautam, Vaibhav Gupta, Parul Singh, Sonal Bramh, Gunjan Thakur, Brijendar Sanger, Ajayendra Singh*****, and Gurjinder Singh,** for their support in all my endeavours.

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: **(Pallavi Verma)**

ABSTRACT

FPGAs (Filed programmable Gate Array) have become a competitive alternative for high performance DSP applications, previously dominated by general purpose DSP and ASIC (Application Specific Integrated Circuit) devices. In this design based on using FPGA for the hardware implementation of the controller circuit and GSM (Global System for Mobile) for remote monitoring. The controller circuit has been described using VHDL (VHSIC Hardware Description Language). The design has been simulated using ModelSim from Model Technology and implemented using Xilinx ISE software tools. FPGA Spartan2 starter kit from Digilent has been used for the hardware implementation of the controller circuit. The system offers low cost and user friendly way of 24hours real-time remote monitoring for temperature using SMS (Short Messaging Service) messages.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction

This chapter includes the evolution of programmable devices, classifications of ICs. Features of ASICs, CPLDs, PALs, PLAs and types of PLDs. And introduction of FPGA advantages of FPGA and introduction of GSM.and why we are using FPGA. and Dissection of Dissertation

1.2 The Evolution of Programmable Devices

Programmable devices have gone through a long evolution to reach the complexity that they have today. An electronic system designer has several options for implementing digital logic. These options include Integrated Circuits (ICs), which can be broadly classified in the following categories:

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

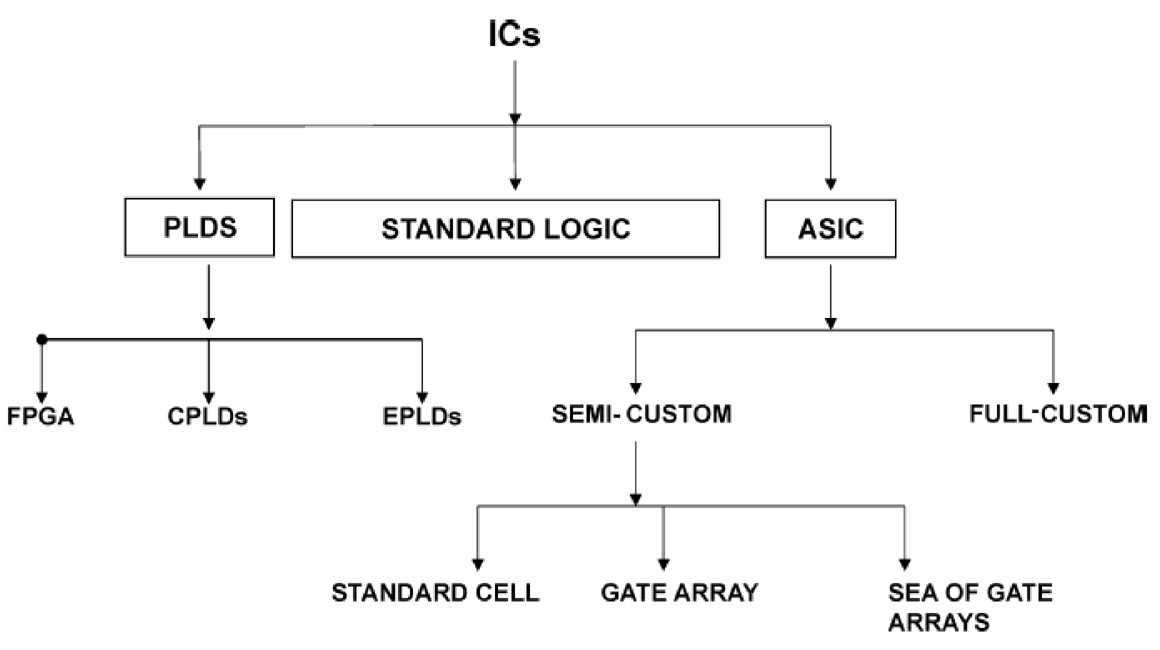


Figure1. ICs Classification

1.2.2 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..

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..FPGAs are usually slower than their application-specific integrated circuit (ASIC) counterparts, cannot handle as complex a design, and draw more power (for any given semiconductor process).

1.2.3 Programmable Logic Devices (PLDs)

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. A PLD is a general-purpose device for implementing logic circuits. It contains a collection of logic gates (elements) and customizable pathways

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.Other similar devices include PAL, GAL and PLA.

1.2.3.1) Types of PLDs

Programmable logic devices are divided into three broad categories.

1.2.3.1.1) 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)

Programmable Logic Arrays (PLAs)

Programmable Logic Arrays (PLAs) were a solution to the speed and input limitations of PROMs. PLAs consist of a large number of inputs connected to an AND plane, where different combinations of signals can be logically AND together according to how the part is programmed. The outputs of the AND plane go into an OR plane, where the terms are OR together in different combinations and finally outputs are produced. At the inputs and outputs there are typically inverters so that logical NOTs can be obtained. These devices can implement a large number of combinatorial functions, though not all possible combinations like a PROM can. However, they generally have many more inputs and are much faster.

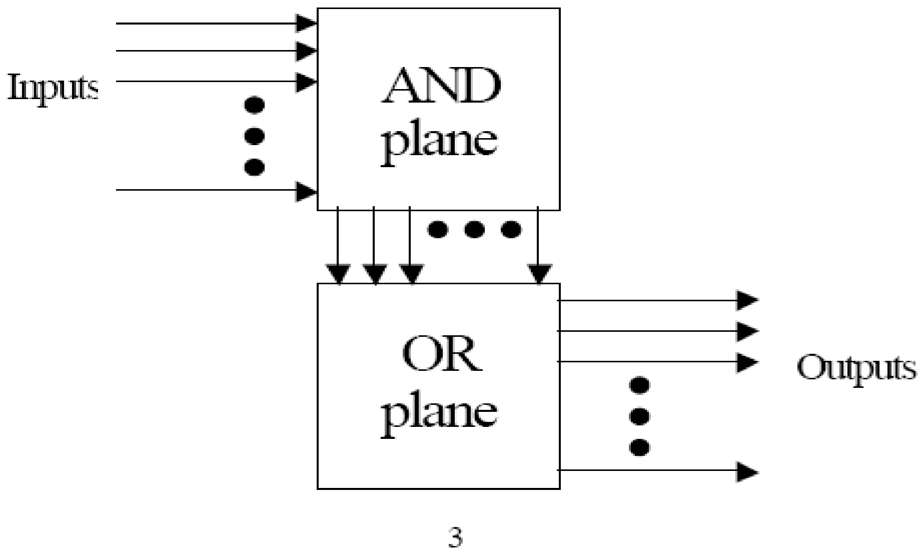


Figure 2- PLAs Architecture

Programmable Array Logic (PALs)

The Programmable Array Logic (PAL) is a variation of the PLA. Like the PLA, it has a wide, programmable AND plane for ANDing inputs together. However, the OR plane is fixed, limiting the number of terms that can be ORed together. Other basic logic devices, such as multiplexers, exclusive ORs, and latches are added to the inputs and outputs. Most importantly, clocked elements, typically flip-flops, are included. These devices are now able to implement a large number of logic functions including clocked sequential logic need for state machines. This was an important development that allowed PALs to replace much of the standard logic in many designs. PALs are also extremely fast.

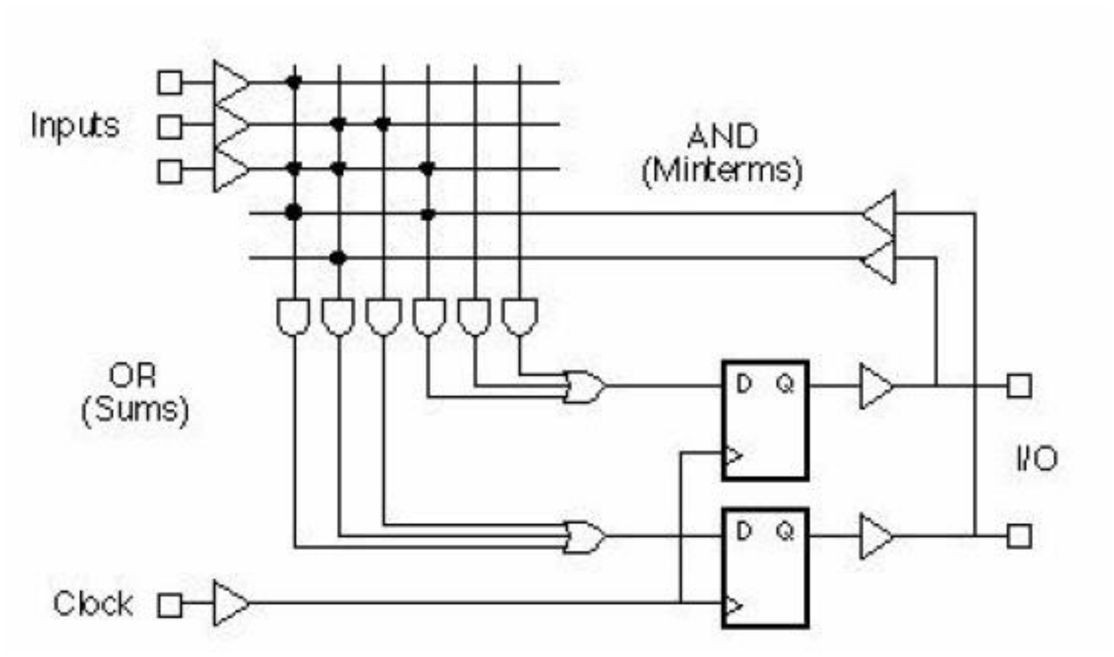


Figure 3-. PAL Architecture

1.2.4 Complex Programmable Logic Devices (CPLDs).

Complex Programmable Logic Devices (CPLDs) are exactly what they claim to be. Essentially they are designed to appear just like a large number of PALs in a single chip, connected to each other through a crosspoint switch they use the same development tools and programmers, and are based on the same technologies, but they can handle much more complex logic and more of it. Most complex programmable logic devices contain macro cells with a sum-of-product combinatorial logic function and an optional flip-flop CPLDs is available with different amounts of memory and different types of memory support. Typically, memory is expressed in bits or megabits. Memory support includes read-only memory (ROM), random access memory (RAM), and dual-port RAM. It also includes content addressable memory (CAM) as well as first-in, first-out (FIFO) memory and last-in, last-out (LIFO) memory. 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. 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.

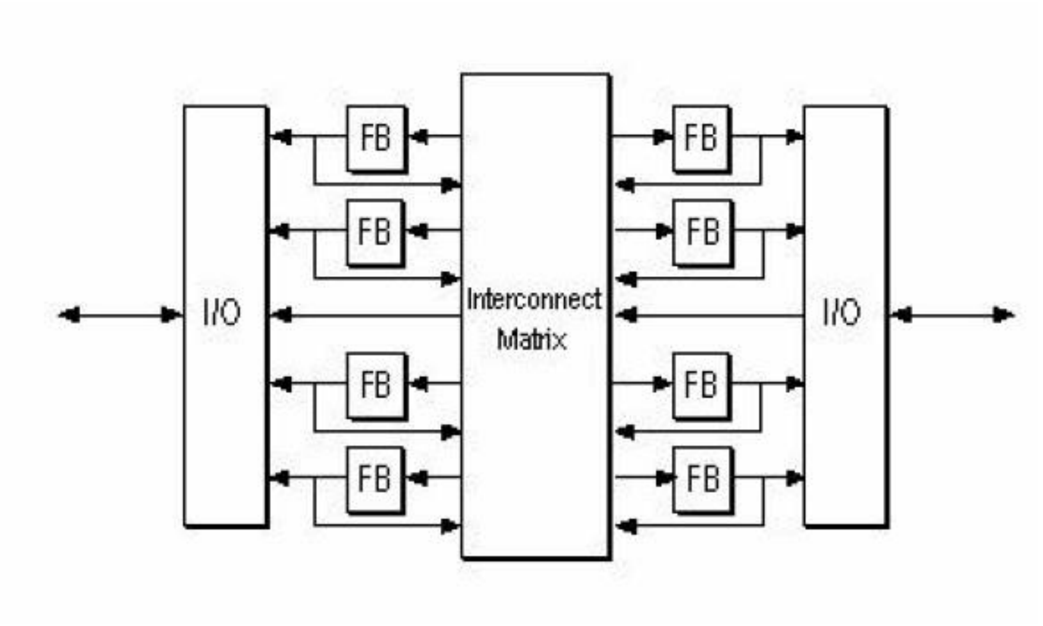


Figure 4- CPLD Architecture

1.3 Introduction to Field Programmable Gate Array

Field-Programmable Gate Arrays (FPGAs) are a revolutionary new type of user-programmable integrated circuits that provide fast, inexpensive access to customized VLSI. 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.

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. An FPGA consists of an array of logic cells that can be interconnected via programmable routing switches, where the routing structures are sufficiently general to allow the configuration of multiple levels of the FPGA’s logic cells. FPGAs represent a combination of the features of Mask Programmable Gate Arrays (MPGAs) and Programmable Logic Devices (PLDs). From MPGAs, FPGAs have adopted a two-dimensional array of logic cells, and from PLDs the user-programmability. The work reported in this thesis is focused on FPGA based system design for temperature sensing by using GSM modem.

Following their introduction in 1985, by the Xilinx Company, FPGAs have evolved considerably as various new devices have been developed. FPGAs have quickly gained widespread use, which can be attributed to the reduced manufacturing time and relatively low costs of these large-capacity users- programmable devices. As an implementation medium for customized VLSI circuits, FPGAs offer following unique advantages over the alternative technologies (MPGAs, standard cells, and full custom design):

(1) FPGAs provide a reduction in the cost of manufacturing a customized VLSI circuit from tens of thousands of dollars to about one hundred dollars.

(2) FPGAs reduce the manufacturing time from months to minutes.

These advantages, which are attributable to the user-programmability of FPGAs, provide a faster time-to-market and less pressure on designers, because multiple design iterations can be done quickly and inexpensively. However, user-programmability also has drawbacks: the logic density and speed performance of FPGAs is considerably lower than those of the alternatives. While developments over the last few years have shown significant improvements in FPGAs, much research is still needed before the best FPGA designs are discovered.

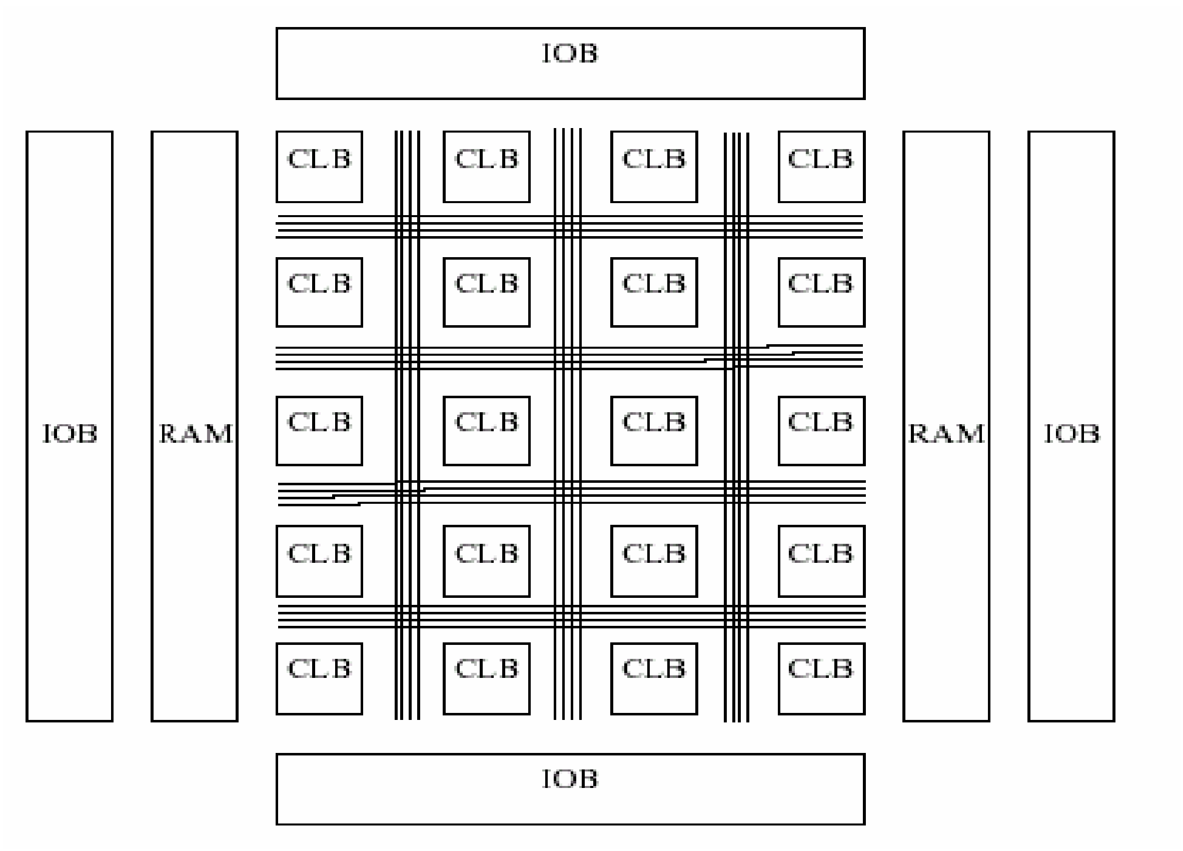


Figure 5-. Basic Architecture of FPGA

1.2.1 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.

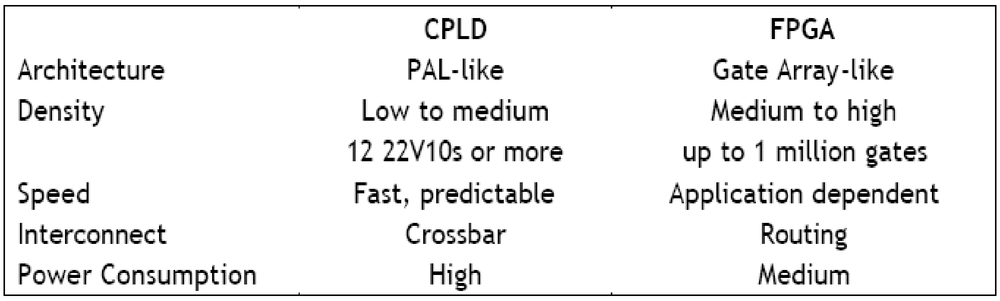
6) Life Cycle Advantages

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.

1.4 Choosing Between CPLDs and FPGAs

Choosing between a CPLD and an FPGA will depend on the characteristics and requirements of the project. A summary of the characteristics of each is



	CPLD	FPGA
Architecture	PAL-like	Gate Array-like
Density	Low to medium	Medium to high
	12 22V10s or more	up to 1 million gates
Speed	Fast, predictable	Application dependent
Interconnect	Crossbar	Routing
Power Consumption	High	Medium

Figure 6- CPLDs vs. FPGAs

Complex programmable logic devices (CPLDs) are integrated circuits (ICs) or chips that application designers configure to implement digital hardware such as mobile phones. CPLDs can handle significantly larger designs than simple programmable logic devices (SPLDs), but provide less logic than field programmable gate arrays (FPGAs) Complex programmable logic devices also vary in terms of logic gates and shift registers. For this reason, CPLDs with a large number of logic gates may be used in place of FPGAs. Another CPLD specification denotes the number of product terms that a macrocell can manage. Product terms are the product of digital signals that perform a specific logic function.

1.5 Introduction to GSM (GLOBAL SYSTEM OF MOBILE)

The first GSM network was launched in 1991 by Radiolinja in Finland .GSM is Global System for Mobile Communication .It is the technology of voice and data transmission with the help of authenticated SIM card .This is widely used on end user side in telecommunication networks. Probably the most useful thing to know about the Global System for Mobile communications (GSM) is that it is an international standard. Instead of using analog service, The GSM net used by cell phones provides a low cost, long range, wireless communication channel for applications that need connectivity rather than high data rates. Machinery such as industrial refrigerators and freezers, HVAC, vending machines, vehicle service etc. could benefit from being connected to a GSM system. GSM was developed as a digital system using TDMA technology. Using TDMA, a narrow band that is 30 kHz wide and 6.7 milliseconds long is split time-wise into three time slots. Narrow band means channels in the traditional sense. Each conversation gets the radio for one-third of the time. This is possible because voice data that has been converted to digital information is compressed so that it takes significantly less transmission space. The modulation used in GSM is Gaussian minimum shift keying (GMSK), a kind of continuous-phase frequency shift keying. In GMSK, the signal to be modulated onto the carrier is first smoothed with a Gaussian low-pass filter prior to being fed to a frequency modulator, which greatly reduces the interference to neighbouring channels (adjacent channel interference). GSM is a cellular network, which means that mobile phones connect to it by searching for cells in the immediate vicinity. GSM networks operate in four different frequency ranges.

1.6 Dissection of Dissertation

Chapter 2

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

Chapter 3

Considers the FPGA in detail. Essential characteristics, applications of FPGAs are explained. Device FPGA, GSM, LM35, ADC, MAX232, 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 4

Deals with Steps to use software for implementing any digital logic are defined. Salient features, design process and need of VHDL are explained.

Chapter 5

Deals with the Hardware description for FPGA based Temperature sensing system, UVLSI user kit, ADC.Max232 and temperature sensing unit.

Conclusion Scope for Future Work

Appendix: VHDL source code to implement the FPGA based temperature sensing using GSM. Some Data sheets.

References

CHAPTER 2

Literature Review

2.1 Introduction

This chapter includes theory and detailed information about Global System of Mobile (GSM), and SIM. And frequency ranges of GSM and brief history of FPGA and Modern developments.

2.2 Global System of Mobile. (GSM)

The first GSM network was launched in 1991 by Radiolinja in Finland. GSM is Global System for Mobile Communication .It is the technology of voice and data transmission with the help of authenticated SIM card .This is widely used on end user side in telecommunication networks .The Uplink Frequency of the GSM system is 890MHz – 915 MHz (25 MHz).The Downlink Frequency of the system is 935MHz – 960 MHz (25 MHz). Frequency difference between Uplink-Downlink Frequency is 45 MHZ. .Probably the most useful thing to know about the Global System for Mobile communications (GSM) is that it is an international standard. Instead of using analog service, GSM was developed as a digital system using TDMA technology. Using TDMA, a narrow band that is 30 kHz wide and 6.7 milliseconds long is split time-wise into three time slots. Narrow band means channels in the traditional sense. Each conversation gets the radio for one-third of the time. This is possible because voice data that has been converted to digital information is compressed so that it takes significantly less transmission space. Therefore, TDMA has three times the capacity of an analog system using the same number of channels. The transmission power in the handset is limited to a maximum of 2 watts in GSM850/900 and 1 watt in GSM1800/1900. The modulation used in GSM is Gaussian minimum shift keying (GMSK), a kind of continuous-phase frequency shift keying. In GMSK, the signal to be modulated onto the carrier is first smoothed with a Gaussian low-pass filter prior to being fed to a frequency modulator, which greatly reduces the interference to neighbouring channels (adjacent channel interference).

2.2.1 SUBSCRIBER IDENTITY MODULE (SIM):

One of the key features of GSM is the **Subscriber Identity Module (SIM),** commonly known as a **SIM** card.

- The SIM is a detachable smart card containing the user's subscription information and phonebook. This allows the user to retain his or her information after switching handsets.

- The user can also change operators while retaining the handset simply by changing the SIM. Some operators will block this by allowing the phone to use

Only a single SIM, or only a SIM issued by them; this practice is known as **SIM locking**, and is illegal in some countries

GSM is a cellular network, which means that mobile phones connect to it by searching for cells in the immediate vicinity. GSM networks operate in four different frequency ranges. Most GSM networks operate in the ***900 MHz or 1800 MHz*** bands. Time division multiplexing is used to allow eight full-rate or sixteen half-rate speech channels per radio frequency channel. There are eight radio timeslots (giving eight burst periods) grouped into what is called a TDMA frame. Half rate channels use alternate frames in the same timeslot.

2.3 FPGA: A brief history

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.

The historical roots of FPGAs are in complex programmable logic devices (CPLDs) of the early to mid 1980s. A Xilinx co-founder invented the field programmable gate array in 1984. In 1985, Xilinx Company introduced the first FPGA. After this many companies like Actel, Altera launched their FPGAs in the market CPLDs and FPGAs include a relatively large number of programmable logic elements. CPLD logic gate densities range from the equivalent of several thousand to tens of thousands of logic gates, while FPGAs typically range from tens of thousands to several million. The primary differences between CPLDs and FPGAs are architectural. A CPLD has a somewhat restrictive structure consisting of one or more programmable sum-of-products logic arrays feeding a relatively small number of clocked registers. The result of this is less flexibility, with the advantage of more predictable timing delays and a higher logic-to-interconnect ratio. The FPGA architectures, on the other hand, are dominated by interconnect. This makes them far more flexible (in terms of the range of designs that are practical for implementation within them) but also far more complex to design for. Another notable difference between CPLDs and FPGAs is the presence in most FPGAs of higher-level embedded functions (such as adders and multipliers) and embedded memories. Some FPGAs have the capability of partial re-configuration that lets one portion of the device be re-programmed while other portions continue running. The high performance FPGAs, made with the more advanced standard cell technology are now beginning to take market share from fixed logic devices.

2.3.1 Modern developments

A recent trend has been to take the coarse-grained architectural approach a step further by combining the logic blocks and interconnects of traditional FPGAs with embedded microprocessors and related peripherals to form a complete "system on a programmable chip". Examples of such hybrid technologies can be found in the Xilinx Virtex-II PRO and Virtex-4 devices, which include one or more PowerPC processors embedded within the FPGA's logic fabric. The Atmel FPSLIC is another such device, which uses an AVR processor in combination with Atmel's programmable logic architecture. An alternate approach to using hard-macro processors is to make use of "soft" processor cores that are implemented within the FPGA logic. Many modern FPGAs have the ability to be reprogrammed at "run time," and this is leading to the idea of reconfigurable computing or reconfigurable systems — CPUs that reconfigure themselves to suit the task at hand. It does not however support dynamic reconfiguration at runtime, but instead adapts itself to a specific program.

CHAPTER 3

Field Programmable Gate Array (FPGA)

3.1 Introduction

Field Programmable Gate Array provides the next step in the Programmable Logic Devices hierarchy. Field Programmable Gate Arrays are called this because rather than having a structure similar to a PAL or other programmable device, they are structured very much like a gate array ASIC. This makes FPGAs very nice for use in prototyping ASICs, or in places where and ASIC will eventually be used. An FPGA is similar to a CPLD but has more logic blocks and other components. Other components include PLL,memory blocks, pre-defined hardware multipliers, DACs/ADCs, processors and more.High density of gates and logic elements. The exact type, size and the number of programmable basic logic cells vary tremendously. We can download FPGAs as many time as we want - no limit - with different functionalities every time if we want. If we make a mistake in your design, just fix our "logic function", re-compile and re-download it. No PCB, solder or component to change. The designs can run much faster than if you were to design a board with discrete components, since everything runs within the FPGA, on its silicon die.

Essential characteristics of FPGAs:

1. Programmable I/O cells surround the core.

2. The core is a regular array of programmable basic logic cells that can implement combinational as well as sequential logic.

3. Besides logic, the other key feature that characterizes an FPGA is its interconnecting structure.

4. FPGAs loose their functionality when the power goes away (like RAM in a computer that looses its content). We have to re-download them when power goes back up to restore the functionality.

3.2 FPGA Architectures

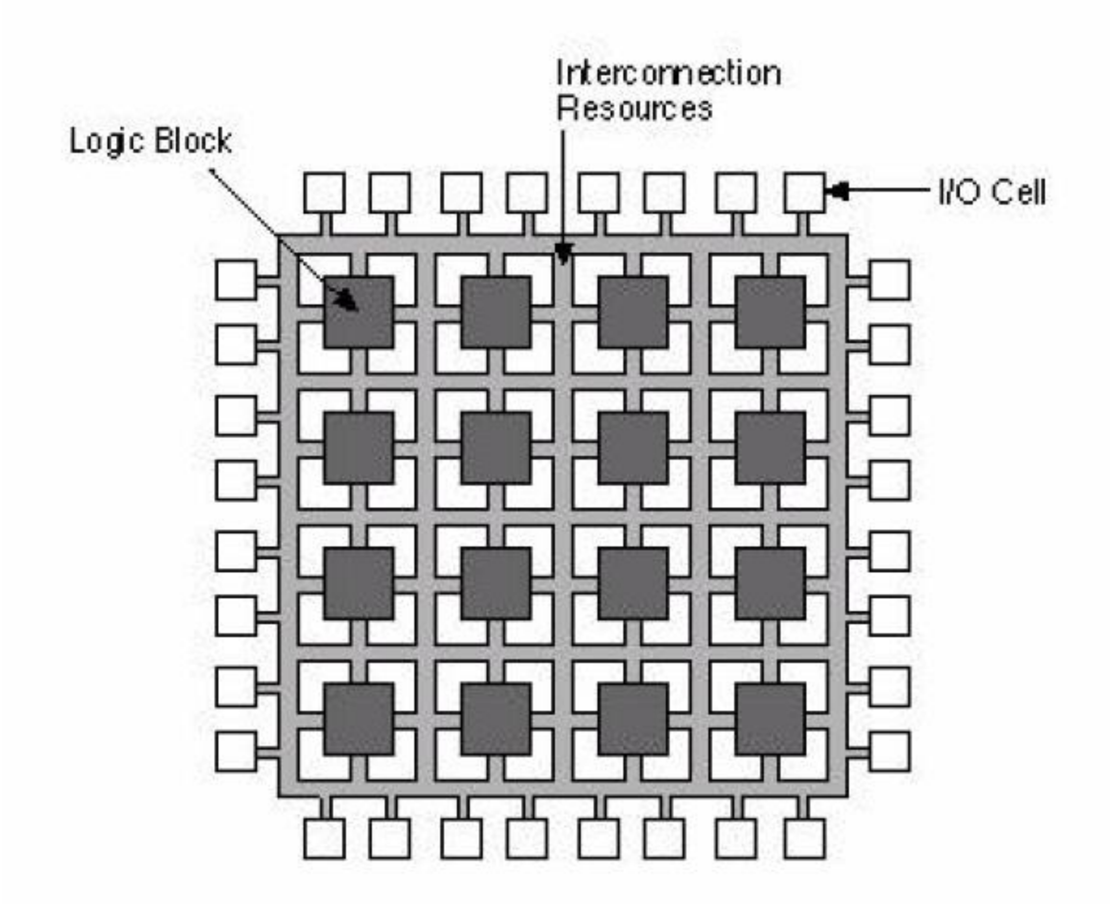


Figure 7- FPGA Architecture

In general terms they are all a variation of that shown in Figure 6. The architecture consists of configurable logic blocks, configurable I/O blocks, and programmable interconnect. Also, there will be clock circuitry for driving the clock signals to each logic block, and additional logic resources such as ALUs, memory, and decoders may be available. The two basic types of programmable elements for an FPGA are Static RAM and anti-fuses.

3.2.1. Configurable I/O Blocks

A Configurable I/O Block, shown in Figure 10, is used to bring signals onto the chip and send them back off again. It consists of an input buffer and an output buffer with three state and open collector output controls. Typically there are pull up resistors on the outputs and sometimes pull down resistors The polarity of the output can usually be programmed for active high or active low output and often the slew rate of the output can be programmed for fast or slow rise and fall times. In addition, there is often a flip-flop on outputs so that clocked signals can be output directly to the pins without encountering significant delay. It is done for inputs so that there is not much delay on a signal before reaching a flip-flop which would increase the device hold time requirement.

3.3 Applications of FPGAs

FPGAs have gained rapid acceptance and growth over the past decade because they can be applied to a very wide range of applications. A list of typical applications includes: random logic, integrating multiple SPLDs, device controllers, communication encoding and filtering, small to medium sized systems with SRAM blocks, and many more. Another promising area for FPGA application, which is only beginning to be developed, is the usage of FPGAs as custom computing machines. This involves using the programmable parts to "execute" software, rather than compiling the software for execution on a regular CPU.

3.3.1) Prototyping

Prototyping of designs later to be implemented in gate arrays, and also emulation of entire large hardware systems. The former of these applications might be possible using only a single large FPGA (which corresponds to a small Gate Array in terms of capacity), and the latter would entail many FPGAs connected by some sort of interconnect; for emulation of hardware, Quick Turn [Wolff90] (and others) has developed products that comprise many FPGAs and the necessary software to partition and map circuits. 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.

3.3.2) 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. 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.

3.4 Devices

There are two basic categories of FPGAs on the market today: 1. SRAM-based FPGAs and 2.antifuse-based FPGAs. In the first category, Xilinx and Altera are the leading manufacturers in terms of number of users, with the major competitor being AT&T. For antifuse-based products, Actel, Quick logic and Cypress, and Xilinx offer competing products.

3.4.1 Xilinx SRAM-based FPGAs

The basic structure of Xilinx FPGAs is *array-based*, meaning that each chip comprises a two dimensional array of logic blocks that can be interconnected via horizontal and vertical routing channels. An illustration of this type of architecture was shown in Figure 2. Xilinx introduced the first FPGA family, called the XC2000 series, in about 1985 and now offers three more generations: XC3000, XC4000, and XC5000. Although the XC3000 devices are still widely used, we will focus on the more recent and more popular XC4000 family. We note that XC5000 is similar to XC4000, but has been engineered to offer similar features at a more attractive price, with some penalty in speed. We should also note that Xilinx has recently introduced an FPGA family based on anti-fuses, called the XC8100. The XC8100 has many interesting features, but since it is not yet in widespread use, we will not discuss it here. The Xilinx 4000 family devices range in capacity from about 2000 to more than 15,000 equivalent gates. The XC4000 features a logic block (called a *Configurable Logic Block* (CLB) by Xilinx) that is based on look-up tables (LUTs). A LUT is a small one bit wide memory array, where the address lines for the memory are inputs of the logic block and the one bit output from the memory is the LUT output. A LUT with K inputs would then correspond to a 2_K x 1 bit memory, and can
realize any logic function of its K inputs by programming the logic function's truth table directly into the memory. The XC4000 CLB contains three separate LUTs, in the configuration shown in Figure 18. There are two 4-input LUTS that are fed by CLB inputs, and the third LUT can be used in combination with the other two. This arrangement allows the CLB to implement a wide range of logic functions of up to

nine inputs, two separate functions of four inputs or other possibilities. Each CLB also contains two flip-flops.

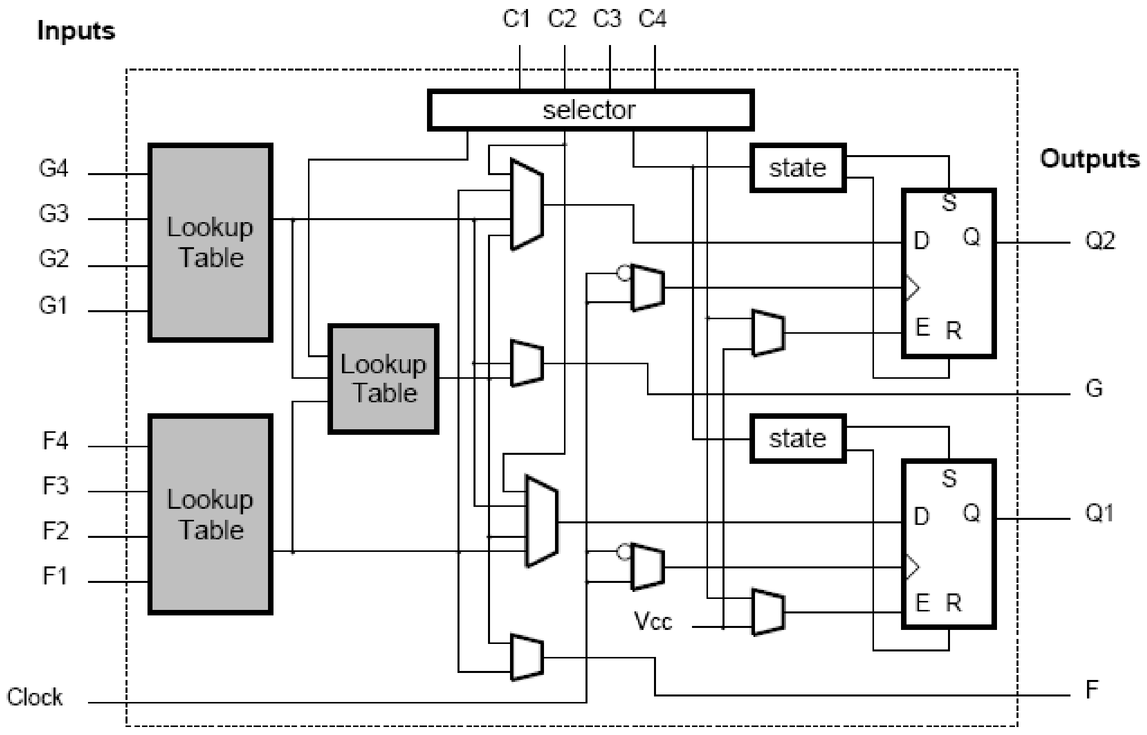


Figure 8 - Xilinx XC4000 Configurable Logic Block (CLB*)*

Toward the goal of providing high density devices that support the integration of entire systems, the XC4000 chips have "system oriented" features. For instance, each CLB contains circuitry that allows it to efficiently perform arithmetic (i.e., a circuit that can implement a fast carry operation for adder-like circuits) and also the LUTs in a CLB can be configured as read/write RAM cells. A new version of this family, the 4000E, has the additional feature that the RAM can be configured as a dual port RAM with a single write and two read ports. In the 4000E, RAM blocks can be synchronous RAM. Also, each XC4000 chip includes very wide AND-planes around the periphery of the logic block array to facilitate implementing circuit blocks such as wide decoders. Besides logic, the other key feature that characterizes an FPGA is it's interconnecting structure. The XC4000 interconnect is arranged in horizontal and vertical channels. Each channel contains some number of short wire segments that span a single CLB (the number of segments in each channel depends on the specific part number), longer segments that span two CLBs and very long segments that span the entire length or width of the chip. Programmable switches are available to connect the inputs and outputs of the CLBs to the wire segments, or to connect one wire segment to another. A small section of a routing channel representative of an XC4000 device appears in Figure 8. The figure shows only the wire segments in a horizontal channel, and does not show the vertical routing channels, the CLB inputs and outputs, or the routing switches. An important point worth noting about the Xilinx interconnect is that signals must pass through switches to reach one CLB from another, and the total number of switches traversed depends on the particular set of wire segments used. Thus, speed-performance of an implemented circuit depends in part on how the wire segments are allocated to individual signals by CAD tools.

3.4.2 Spartan-3 FPGA

The Spartan-3 family of Field-Programmable Gate Arrays is specifically designed to meet the needs of high volume, cost-sensitive consumer electronic applications. The eight-member family offers densities ranging from 50,000 to five million system gates, The Spartan-3 family builds on the success of the earlier Spartan-IIE family by increasing the amount of logic resources, the capacity of internal RAM, the total number of I/Os, and the overall level of performance as well as by improving clock management functions. Numerous enhancements derive from the Virtex®-II platform technology. These Spartan-3 FPGA enhancements, combined with advanced process technology, deliver more functionality and bandwidth per dollar than was previously possible, setting new standards in the programmable logic industry. Because of their exceptionally low cost, Spartan-3 FPGAs are ideally suited to a wide range of consumer electronics applications; including broadband access, home networking, display/projection and digital television equipment. The Spartan-3 family is a superior alternative to mask programmed ASICs. FPGAs avoid the high initial cost, the lengthy development cycles, and the inherent inflexibility of conventional ASICs.

Features

Low-cost, high-performance logic solution for high-volume, consumer-oriented applications

- Densities up to 74,880 logic cells.
- Select IO interface signalling.
- Up to 633 I/O pins.
- 622 Mb/s data transfer rate per I/O.
- 18 single-ended signal standards.
- 8 differential I/O standards including LVDS, RSDS.
- Termination by Digitally Controlled Impedance.
- Signal swing ranging from 1.14V to 3.465V.
- Double Data Rate (DDR) support.
- DDR, DDR2 SDRAM support up to 333 Mbps.

Logic resources

- Abundant logic cells with shift register capability.
- Wide, fast multiplexers.
- Fast look-ahead carry logic.
- Dedicated 18 x 18 multipliers.
- JTAG logic compatible with IEEE 1149.1/1532.
- Select RAM hierarchical memory.
- Up to 1,872 Kbits of total block RAM.
- Up to 520 Kbits of total distributed RAM.

Digital Clock Manager (up to four DCMs)

- Clock skew elimination.
- Frequency synthesis.
- High resolution phase shifting
- Eight global clock lines and abundant routing

3.4.2.1 Configuration

Spartan-3 FPGAs are programmed by loading configuration data into robust, reprogrammable, static CMOS configuration latches (CCLs) that collectively control all functional elements and routing resources. Before powering on the FPGA, configuration data is stored externally in a PROM or some other non-volatile medium either on or off the board. After applying power, the configuration data is written to the FPGA using any of five different modes: Master Parallel, Slave Parallel, Master Serial, Slave Serial, and Boundary Scan (JTAG). The Master and Slave Parallel modes use an 8-bit wide Select MAP port the recommended memory for storing the configuration data is the low-cost Xilinx Platform Flash PROM family, which includes the XCF00S PROMs for serial configuration and the higher density XCF00P PROMs for parallel or serial configuration.

3.4.2.2 Configurable Logic Blocks (CLBs)

A Virtex-5 FPGA CLB resource is made up of two slices. Each slice is equivalent and contains:

• Four function generators
• Four storage elements
• Arithmetic logic gates
• Large multiplexers
• Fast carry look-ahead chain

The function generators are configurable as 6-input LUTs or dual-output 5-input LUTs. SLICEMs in some CLBs can be configured to operate as 32-bit shift registers (or 16-bit x 2 shift registers) or as 64-bit distributed RAM. In addition, the four storage elements can be configured as either edge-triggered D-type flip-flops or level sensitive latches. Each CLB has internal fast interconnect and connects to a switch matrix to access general routing resources.

3.4.2.3 Interconnect

Interconnect (or routing) passes signals among the various functional elements of Spartan-3 devices. There are four kinds of interconnect: Long lines, Hex lines, Double lines, and Direct lines. Long lines connect to one out of every six CLBs (see Figure 9a). Because of their low capacitance, these lines are well-suited for carrying high-frequency signals with minimal loading effects (e.g. skew). If all eight Global Clock Inputs are already committed and there remain additional clock signals to be assigned, Long lines serve as a good alternative. Hex lines connect one out of every three CLBs (see Figure 9b). These lines fall between Long lines and Dou Double lines in terms of capability: Hex lines approach the high-frequency characteristics of Long lines at the same time, offering greater connectivity. Double lines connect to every other CLB (see Figure 9c). Compared to the types of lines already discussed, Double lines provide a higher degree of flexibility when making connections. Direct lines afford any CLB direct access to neighbouring CLBs (see Figure 9d). These lines are most often used to conduct a signal from a "source" CLB to a Double, Hex, or Long line and then from the longer interconnect back to a Direct line accessing a "destination" CLB.

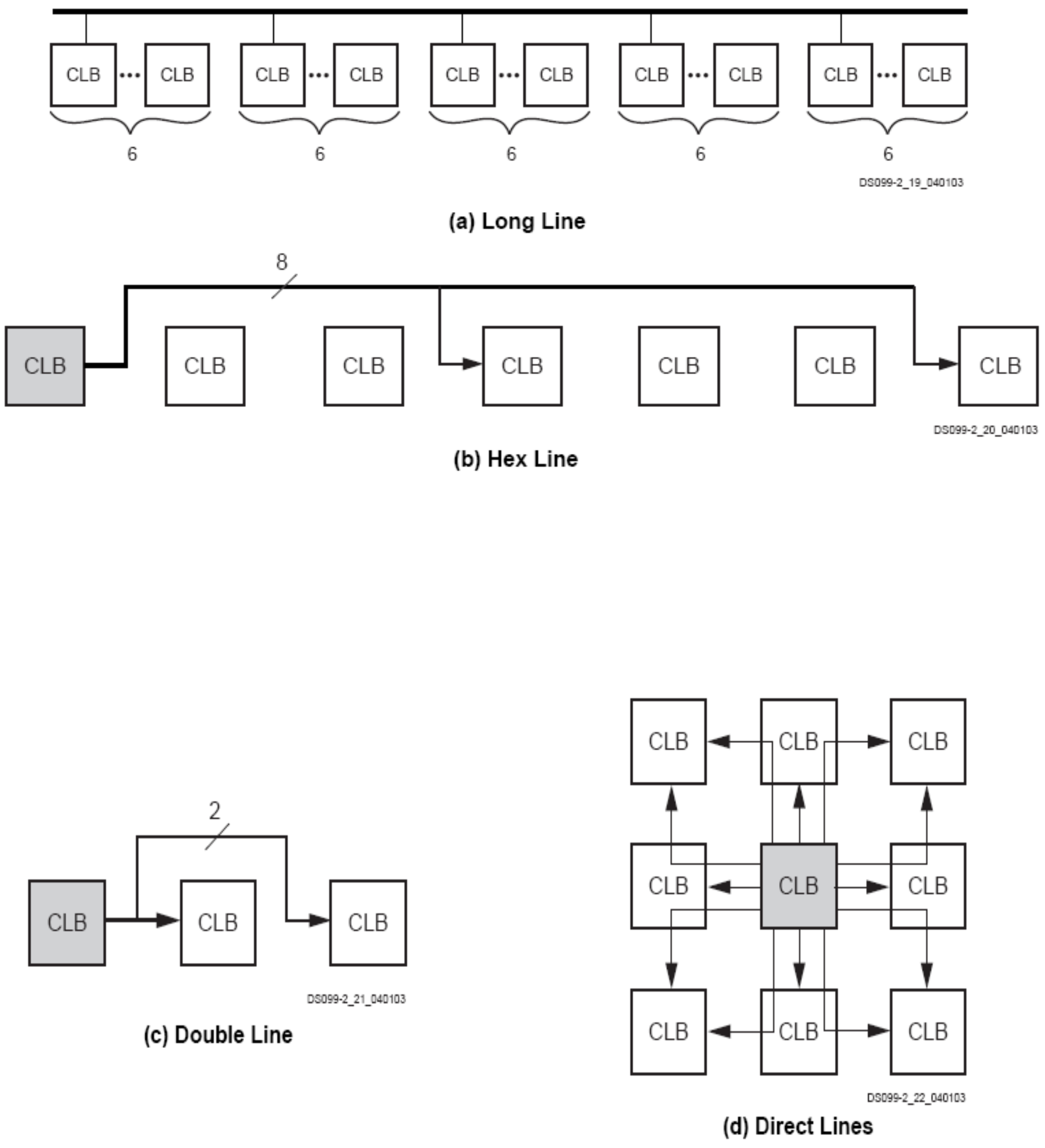


Figure 9- Types of Interconnect

3.4.2.4 Powering Spartan-3 FPGAs

3.4.2.4.1 Voltage Regulators

Various power supply manufacturers offer complete power solutions for Xilinx FPGAs, including some with integrated multi-rail regulators specifically designed for Spartan-3 FPGAs.

3.4.2.4.2 Power-On Behaviour

Spartan-3 FPGAs have a built-in Power-On Reset (POR) circuit that monitors the three power rails required to successfully configure the FPGA. At power-up, the POR circuit holds the FPGA in a reset state until the VCCINT, VCCAUX, and VCCO Bank 4 supplies reach their respective input threshold levels. After all three supplies reach their respective threshold, the POR reset is released and the FPGA begins its configuration process. Because the three supply inputs must be valid to release the POR reset and can be supplied in any order, there are no specific voltages sequencing requirements. However, applying the FPGA's VCCAUX supply before the VCCINT supply uses the least ICCINT current. Once all three supplies are valid, the minimum current required to power-on the FPGA is equal to the worst-case quiescent current. Spartan- 3 FPGAs do not require Power-On Surge (POS) current to successfully configure.

3.4.3. 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. The peripheral control bus uses high-speed drivers to minimize signal skew across devices and provides up to 12 peripheral control signals.

3.4.4 Clocks and Global lines

An FPGA design is usually "synchronous". Simply put, that means that the design is clock based - each clock (rising edge) allows the D-flipflops to take a new state. In a synchronous design, a single clock may drive a lot of flipflops simultaneously. That can cause timing and electrical problems inside the FPGA. To get that working properly, FPGA manufacturers provide special internal wires called "global routing" or "global lines". They allow distributing the clock signal all over the FPGA with a low skew (i.e. the clock signal appears almost simultaneously to all the flipflops). When you feed a clock signal to your FPGA, you shouldn't use any FPGA pin, but use a "dedicated input pin". Usually, only such input pins have the ability to drive a global line. Check the FPGA datasheet to find which pins are the "dedicated inputs". FPGA software are aware of these dedicated inputs, and will automatically assign clocks to them if given the choice.

3.4.4.1 Clock domains

An FPGA can use multiple clocks (using multiple global lines and dedicated input pins). Each clock forms a "clock domain" inside the FPGA.

3.5 Precision Centigrade Temperature Sensors LM35

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ° Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Centigrade scaling. The LM35 does not require any external calibration or trimming to provide typical accuracies. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only 60 mA from its supply, it has very low self-heating, less than 0.1°C in still air. The LM35 is rated to operate over a -55° to +150°C temperature range, while the LM35C is rated for a -40° to +110°C range (-10°Cwith improved accuracy). The LM35 series is available packaged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface mount small outline package and a plastic TO-220 package.

3.5.1 OPERATING PRINCIPLE:

LM35 is a temperature sensor. The pinout is shown in the schematic fig 10. It has a wide temperature range from -40ºC to +110ºC. It gives an output of 10mV per degree centigrade. The hardware for this is fairly simple. A supply of +5V to V_{CC} & a ground signal is to be given and its output in mV is recorded at pin no.2. As the temperature of surroundings rises, the voltage at pin no.2 of LM35 increases by 10mV for every degree rise of temperature which can be detected by an 8-bit ADC. The ADC can be calibrated to the voltage and the temperature can be displayed on the LCD in degree centigrade. This is fairly simple as voltage varies in direct proportion with temperature.

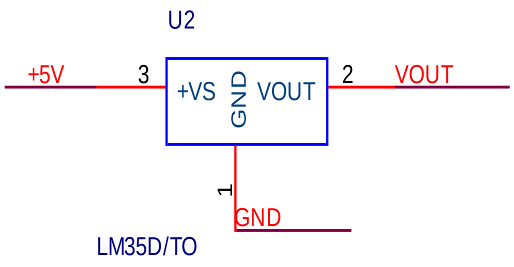


Figure 10-. APPLICATION CIRCUIT of LM35

Features

- Calibrated directly in ° Celsius (Centigrade).
- Linear + 10.0 mV/°C scale factor.
- 0.5°C accuracy guaranteeable (at +25°C).
- Rated for full −55° to +150°C range.
- Suitable for remote applications.
- Low cost due to wafer-level trimming.
- Operates from 4 to 30 volts.

- Less than 60 µA current drain.
- Low self-heating, 0.08°C in still air.
- Nonlinearity only ±1⁄4°C typical.
- Low impedance output, 0.1 W for 1 mA load.

3.5.2 Typical Application

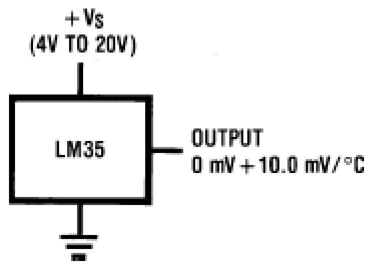
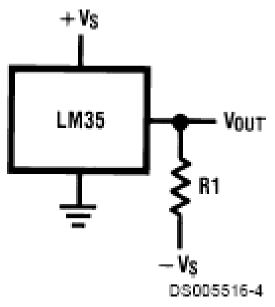


FIGURE 11(a). Basic Centigrade Temperature Sensor (+2°C to +150°C)



Choose $R_1 = -V_S/50\ \mu A$

V_{OUT}=+1,500 mV at +150°C

= +250 mV at +25°C

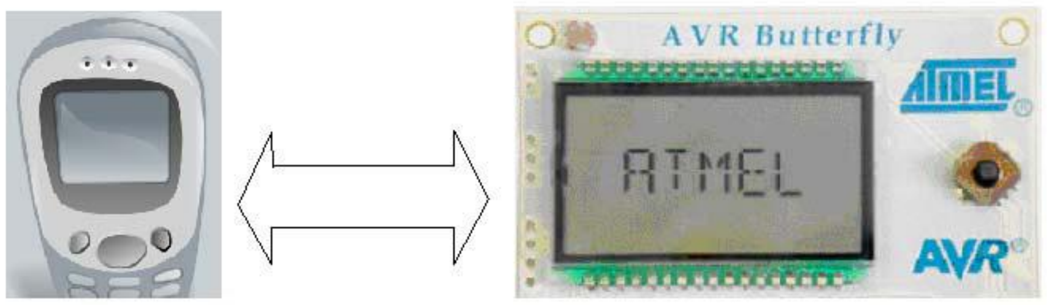
= −550 mV at −55°C

FIGURE 11(b). Full-Range Centigrade Temperature Sensor

3.6. Global System of Mobile (GSM)

The GSM net used by cell phones provides a low cost, long range, wireless communication channel for applications that need connectivity rather than high data rates. Machinery such as industrial refrigerators and freezers, HVAC, vending machines, vehicle service etc. could benefit from being connected to a GSM system.

The customer will benefit from a reliable and well-serviced vehicle at a minimum cost. The garage on the other hand can provide excellent customer support, vehicle statistics, efficient work scheduling, and minimum stocks. This application note describes how to use an AVR to control a GSM modem in a cellular phone. The interface between modem and host is a textual protocol called Hayes AT-Commands. These commands enable phone setup, dialing, text messaging etc. This particular application connects an AVR Butterfly and Siemens® M65 cellular phone using a RS232 based data cable. Most cellular phones could be used, except Nokia® phones using F or M-bus.



3.6.1. Theory of Operation

The protocol used by GSM modems for setup and control is based on the Hayes AT-Command set. The GSM modem specific commands are adapted to the services offered by a GSM modem such as: text messaging, calling a given Phone number, deleting memory locations etc. Since the main objective for this application note is to show how to send and receive text messages, only a subset of the AT-Command set needs to be implemented. The European Telecommunication Standard Institute (ETSI) GSM 07.05 defines the AT-Command interface for GSM compatible modems. From this document some selected commands are chosen, and presented briefly in this section. This command subset will enable the modem to send and receive SMS messages.

3.6.1.1 AT-Command Set

The commands can be tried out by connecting a GSM modem to one of the PC’s COM ports. Type in the test-command, adding CR + LF (Carriage return + Line feed = \r\n) before executing.

Table 3-1. AT-Command set overview

Command	Description
AT	Check if serial interface and GSM modem is working.
ATE0	Turn echo off, less traffic on serial line.
AT+CNMI	Display of new incoming SMS.
AT+CPMS	Selection of SMS memory.
AT+CMGF	SMS string format, how they are compressed.
AT+CMGR	Read new message from a given memory location.
AT+CMGS	Send message to a given recipient.
AT+CMGD	Delete message.

3.6.1.2 Status (AT)

The “AT” command is a status request used for testing if a compatible modem is connected and that the serial interface is working properly.

Table 3-2. AT command and possible responses

Command	Response	Comment
“AT”	“OK”	Connected and working
	“ERROR”	Serial line OK, modem error

3.6.2 Error Code

Many of the commands in the implemented subset can terminate with an error message related to the modem or network. These could be errors such as:

• Memory failure.

• Invalid recipient number.

• Network timeout.

• SIM busy or wrong.

• Operation not allowed.

• No network service.

These error messages can be useful, and could be implemented as a part of the application. It is possible to extend the handling of the error codes, but this is beyond the scope of this application note. We will just catch the ERROR message, and repeat the command.

3.7 MAX-232

MAX-232 converts from RS-232 voltage levels to TTL voltage levels and vice versa. One advantage of MAX-232 chip is that it uses +5V power source ,which is the same as the source voltage for the 8051.that is we can power both 8051 and MAX-232 with single +5V power supply, with no need for the dual poor supplies.

The MAX-232 has two sets of line drivers for transferring and receiving data .The line drivers used for TxD are called T1 and T2.while line drivers for RxD are called R1and R2 out of which only one is used at time . In MAX-232 the T1 line driver has a designation of T1in and T1out on pin no 11&14 respectively. The T1 in pin is TTL side and is connected to TxD of the FPGAs while T1out is connected to RxD pin of RS-232 DB connector. The R1 line driver has a designation of R1in &R1out on pin no 13&12 respectively. The R1in pin is the RS-232 side that is connected to the TxD.

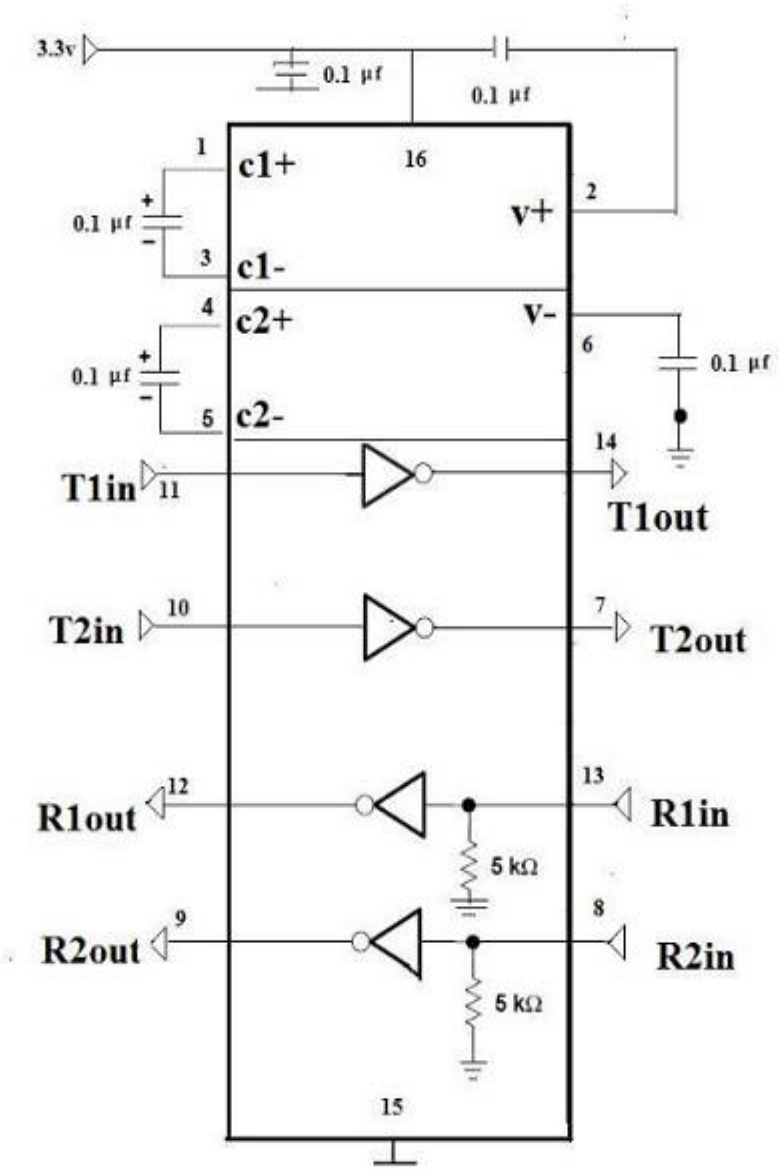


Figure 12-Pin layout of RS-232.

3.7.1 The RS-232 standard includes details of:

- The protocol to be used for data transmission
- The voltages to be used on signal lines.
- The connector to be used to link equipment together.

The overall standard comprehensive and widely used at data transfer rates of up to around 115 or330 Kbits per second. Data transfer can be over distances of 15 meters or more.

3.7.2 Basic RS-232 Protocol:

Rs-232 is a character-oriented protocol. That is it is intended to be used to send single 8-bit blocks of data. To transmit a byte of data over an RS-232 link, we generally encode the information as follows:-

# We send a “start“ bit.

# We send the data.

# We send a “stop” bit.

3.7.3 Asynchronous data transmission and baud rates

RS-232 uses an asynchronous protocol; both ends of the comm. link have an internal clock, running at the same rate. The data (in the case of RS-232, the start bit) is then used to synchronize the clocks, if necessary to ensure successful data transfer.

# RS-232 generally operates at one of a range of baud rates.

# Typically these are 75, 110, 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 33600, 56000, 115000, and 330,000 baud.

# 9600 baud is a very safe choice as it is very widely supported.

3.7.4 RS-232 Voltage levels:

# The threshold levels used by the receiver are +3V and -3V and lines are inverted
# The maximum voltage allowed is +/-15 volt.

# Note that that these voltages cannot be obtained directly from the naked microcontroller port pins some form of interface hardware is required.

# for e.g. The Maxim MAX-232 and MAX-233 are popular and widely-used line driver chips.

CHAPTER 4

SOFTWARE

4.1 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, Xilinx 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.

4.2 Xilinx ISE 6 Software

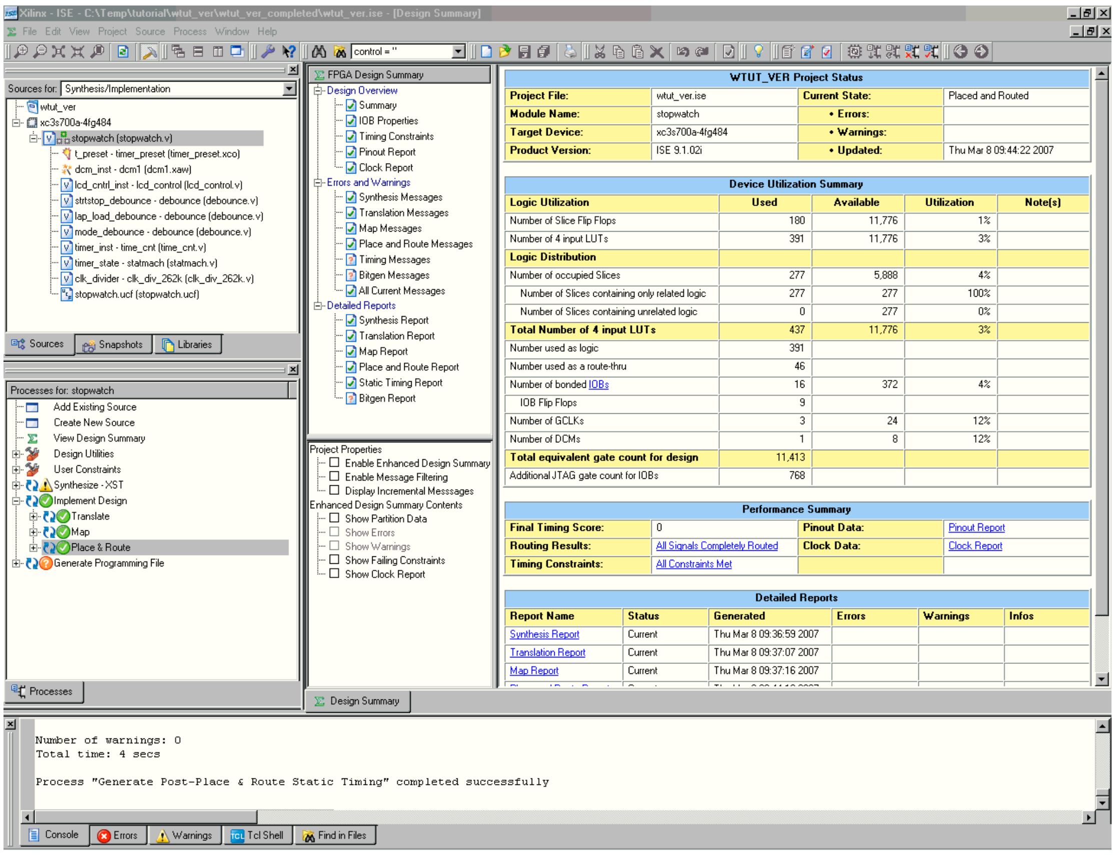
The Xilinx ISE (Integrated Software Environment) 6 design software provides a complete, multi platform design environment that easily adapts to our specific design needs. ISE Provides an overview of the Xilinx Integrated Software Environment (ISE), including design flow information, Explains how to create, define, and compile your FPGA or CPLD design using the suite of ISE tools available from the Project Navigator and also Describes what’s new in the software release and how to migrate past projects to the current software. Explains how to use HDLs to design FPGAs with emphasis on synthesis and simulation. This software includes solutions for all phases of FPGA and CPLD design.

4.2.1 Overview of ISE

ISE controls all aspects of the design flow. Through the Project Navigator interface, we can access all of the design entry and design implementation tools. We can also access the files and documents associated with our project.

Project Navigator Interface

The Project Navigator Interface is divided into four main subwindows, as seen in Figure 13. On the top left is the Sources window which hierarchically displays the elements included in the project. Beneath the Sources window is the Processes window, which displays available processes for the currently selected source. The third window at the bottom of the Project Navigator is the Transcript window which displays status messages, errors, and warnings and also contains interactive tabs for Tcl scripting and the Find in Files function. The fourth window to the right is a multi-document interface (MDI) window referred to as the *Workspace*. It enables you to view html reports, ASCII text files, schematics, and simulation waveforms. Each window may be resized, undocked from Project Navigator or moved to a new location within the main Project Navigator window.



WTUT_VER Project Status

Project File:	wtut_ver.ise	**Current State:**	Placed and Routed
Module Name:	stopwatch	**• Errors:**	
Target Device:	xc3s700a-4fg484	**• Warnings:**	
Product Version:	ISE 9.1.02i	**• Updated:**	Thu Mar 8 09:44:22 2007

Device Utilization Summary

Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	180	11,776	1%	
Number of 4 input LUTs	391	11,776	3%	
Logic Distribution				
Number of occupied Slices	277	5,888	4%	
Number of Slices containing only related logic	277	277	100%	
Number of Slices containing unrelated logic	0	277	0%	
Total Number of 4 input LUTs	437	11,776	3%	
Number used as logic	391			
Number used as a route-thru	46			
Number of bonded IOBs	16	372	4%	
IOB Flip Flops	9			
Number of GCLKs	3	24	12%	
Number of DCMs	1	8	12%	
Total equivalent gate count for design	11,413			
Additional JTAG gate count for IOBs	768			

Performance Summary

Final Timing Score:	0	**Pinout Data:**	Pinout Report
Routing Results:	All Signals Completely Routed	**Clock Data:**	Clock Report
Timing Constraints:	All Constraints Met		

Detailed Reports

Report Name	**Status**	**Generated**	**Errors**	**Warnings**	**Infos**
Synthesis Report	Current	Thu Mar 8 09:36:59 2007			
Translation Report	Current	Thu Mar 8 09:37:07 2007			
Map Report	Current	Thu Mar 8 09:37:16 2007			

```
Number of warnings: 0
Total time: 4 secs

Process "Generate Post-Place & Route Static Timing" completed successfully
```

Figure 13: Project Navigator

Processes Window

This window contains one default tab called the Processes tab. The Processes tab is context sensitive and it changes based upon the source type selected in the Sources tab and the Top-Level Source in your project. From the Processes tab, you can run the functions necessary to define, run and view your design. The Processes tab provides access to the following functions:

• **Add an Existing Source**

• **Create New Source**

• **View Design Summary**

• **Design Utilities**

Provides access to symbol generation, instantiation templates, viewing command line history, and simulation library compilation.

• **User Constraints**

Provides access to editing location and timing constraints.

• **Synthesis**

Provides access to Check Syntax, Synthesis, View RTL or Technology Schematic, and synthesis reports. Available processes vary depending on the synthesis tools you use.

• **Implement Design**

Provides access to implementation tools, design flow reports, and point tools.

• **Generate Programming File**

Provides access to configuration tools and bitstream generation. The Processes tab incorporates automake technology. This enables the user to select any process in the flow and the software automatically runs the processes necessary to get to the desired step. For example, when you run the Implement Design process, Project Navigator also runs the Synthesis process because implementation is dependent on up-to date synthesis results.

4.3 HDL Design Flow

The design is composed of HDL elements and two cores Required Software. To perform this, We must have the following software and software components installed:

- Xilinx Series ISE 9.1i
- Spartan-3A libraries and device files

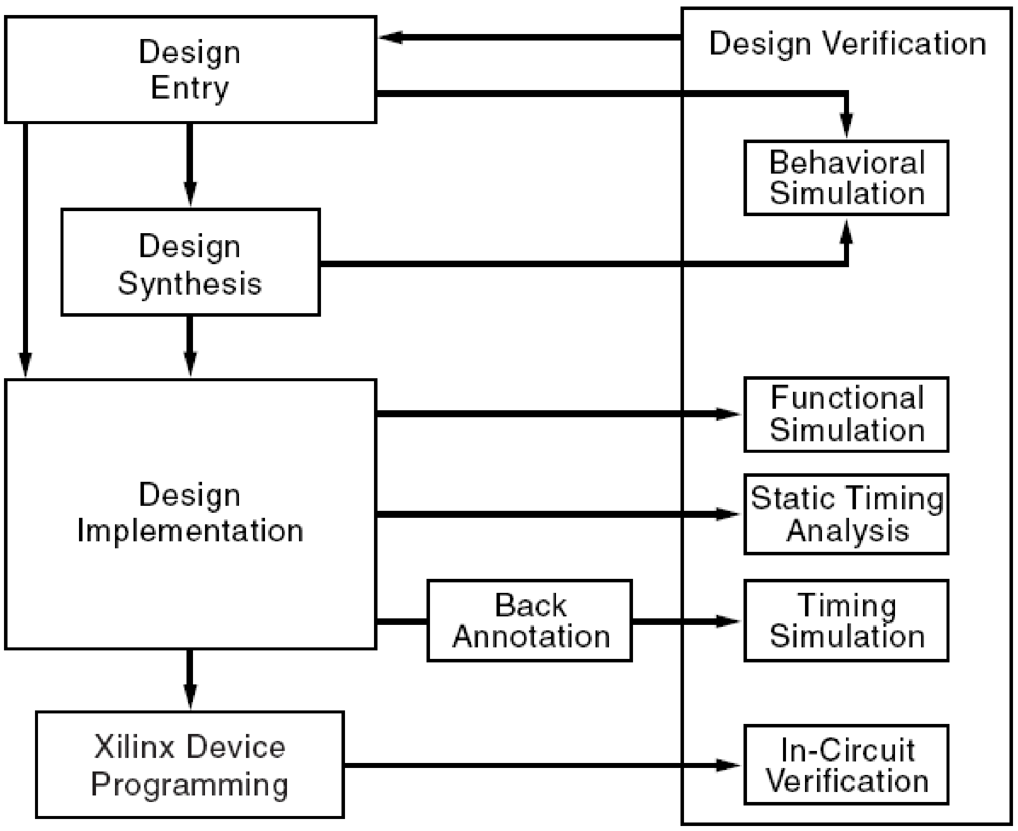


Figure 14- HDL Design Flow

4.3.1. VHDL (VERY HIGH DEFINITION LANGUAGE)

VHDL is an acronym for VHSIC (Very high speed integrated circuit) hardware description language. The language supports flexible design methodologies: top-down, bottom-up or mix. VHDL is a hardware description language that can be used to model a digital system. The language is case insensitive & also in free format. The requirement for the language was first generated in 1981 under the VHSIC program. Version 7.2 of VHDL was developed and released to the public in 1985. VHDL is an acronym for VHSIC (Very high speed integrated circuit) hardware description language. The VHDL language can be regarded as an integrated amalgam of the following languages:

- Sequential language
- Concurrent language
- Net-list language
- Timing specifications
- Waveform generation language

The language supports flexible design methodologies: top-down, bottom-up or mix. VHDL is a hardware description language that can be used to model a digital system. The language is case insensitive & also in free format. The language supports three basic different description styles: Structural, Dataflow, and Behavioural. The digital system can also be described hierarchically. Timing can also be explicitly modelled 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 behaviour 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.

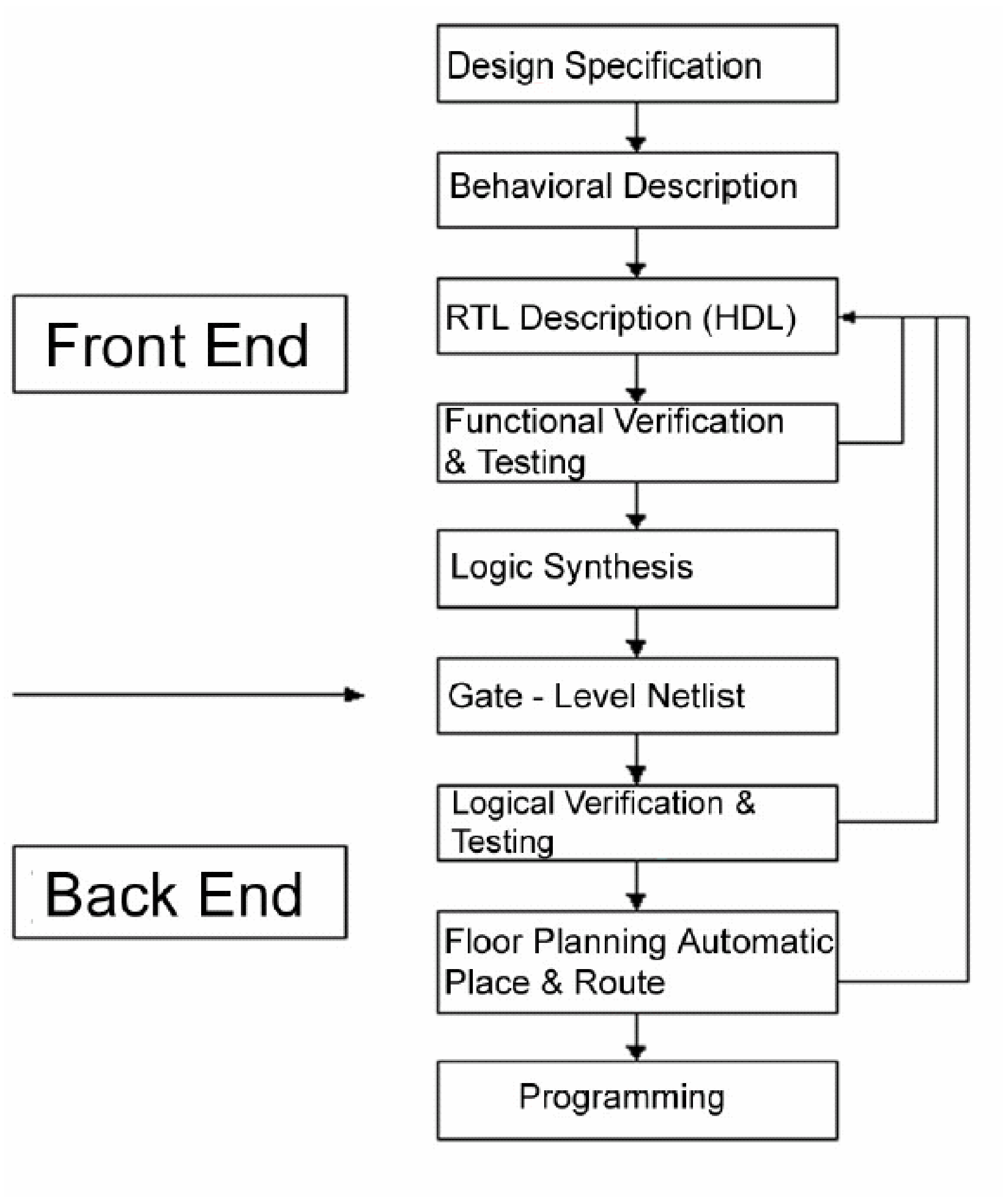


Figure15- VLSI Design Flow

CHAPTER 5

HARDWARE

5.1 Introduction

The Xilinx Spartan-3 FPGA Starter Kit provides a low-cost, easy-to-use development and evaluation platform for Spartan-3 FPGA designs. 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

5.2 Key Components and Features of Xilinx

In Figure The Spartan-3 Starter Kit board, includes the following components and features:

- 200,000-gate Xilinx Spartan-3 XC3S200 FPGA in a 256-ball thin Ball Grid Array package (XC3S200FT256)
- 4,320 logic cell equivalents
- Twelve 18K-bit block RAMs (216K bits)
- Twelve 18x18 hardware multipliers
- Four Digital Clock Managers (DCMs)
- Up to 173 user-defined I/O signals
- 2Mbit Xilinx XCF02S Platform Flash, in-system programmable configuration PROM
- 1Mbit non-volatile data or application code storage available after FPGA configuration
- Jumper options allow FPGA application to read PROM data or FPGA configuration from other source

- 9-pin RS-232 Serial Port
- RS-232 transceiver/level translator
- Uses straight-through serial cable to connect to computer or workstation serial port
- Second RS-232 transmit and receive channel available on board test points.

Figure16- FPGA Kit

5.3 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.

5.3.1) Connectors

(1) Input Port (P14)

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

(4) I/O Port1 (P17)

This is 50 pin headers with 48 I/O lines and Vcc [+5V].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.

5.3.2) 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.

5.3.3) 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. .

5.3.4) Jumpers

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

5.3.5) On Board Programmer

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

5.3.6) 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.

5.4 Temperature Sensor Unit

GENERAL DESCRIPTION

The TMP35, TMP36, and TMP37 are low voltage, precision, centigrade temperature sensors. They provide a voltage output that is linearly proportional to the Celsius (centigrade) tempera-ture. The TMP35/TMP36/TMP37 do not require any external calibration to provide typical accuracies of ±1°C at +25°C and ±2°C over the −40°C to +125°C temperature range.

The low output impedance of the TMP35/TMP36/TMP37 and its linear output and precise calibration simplify interfacing to temperature control circuitry and A/D converters. All three devices are intended for single-supply operation from 2.7 V to 5.5 V maximum. The supply current runs well below 50 µA, providing very low self-heating—less than 0.1°C in still air. In addition, a shutdown function is provided to cut the supply current to less than 0.5 µA.

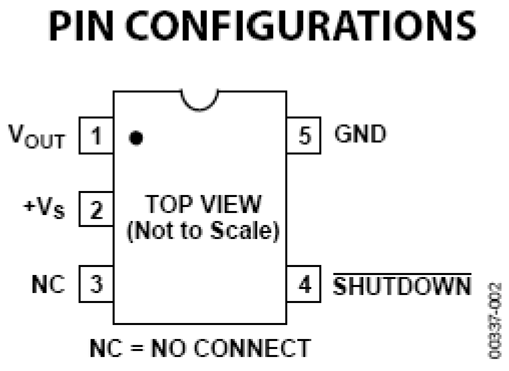


Figure 17- pin configuration

Why Use LM35s to Measure Temperature?

- We can measure temperature more accurately than a using a thermistor.
- The sensor circuitry is sealed and not subject to oxidation, etc.
- The LM35 generates a higher output voltage than thermocouples and may not require that the output voltage be amplified.

5.5 Analog to Digital Converter (ADC)

Analog–to-digital converters are among the most widely used devices for data acquisition. Digital computers use binary (discrete) values, but in the physical world everything is analog (continuous). Temperature, pressure (wind or liquid), humidity, and velocity are a few examples of physical quantities that we deal with every day. A physical quantity is converted to electrical (voltage, current) signals using a device called a transducer. Transducers are also referred to as *sensors.* Sensors for temperature, velocity, pressure, light, and many other natural quantities produce an output that is voltage (or current).

Therefore, we need an analog-to-digital converter to translate the analog signals to digital numbers so that the FPGAs can read and process them. An ADC has *n*-bit resolution where *n* can be 8, 10, 12, 16or even 24 bits. The higher-resolution ADC provides a smaller step size, where step size is the smallest change that can be discerned by an ADC. The ADC converts the analog signal into digital data to be read by the fpga.. In this Vref (+) and Vref (-) set the reference voltage. We choose Vref (-) =gnd and Vref (+) =5V, the step size is 5V/256 =19.53 mV

5.5.1 Pin Diagram

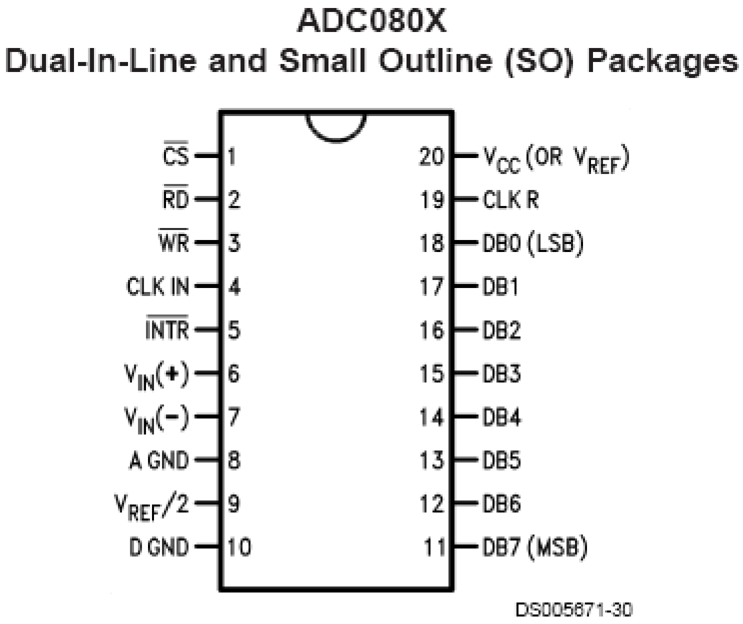


Figure 18: Pin Diagram for ADC0804

5.5.2 FEATURES

- Compatible with most microprocessors
- Differential inputs
- 3-State outputs
- Logic levels TTL and MOS compatible
- Can be used with internal or external clock
- Analog input range 0 V to VCC
- Single 5 V supply
- Guaranteed specification with 1 MHz clock

5.5.3 Applications

- Transducer-to-microprocessor interface
- Digital thermometer
- Digitally-controlled thermostat
- Microprocessor-based monitoring and control systems

5.6 MAX 232

The MAX232 is a dual driver/receiver that includes a capacitive voltage generator to supply TIA/EIA-232-F voltage levels from a single 5-V supply. Each receiver converts TIA/EIA-232-F inputs to 5-V TTL/CMOS levels. These receivers have a typical threshold of 1.3 V, a typical hysteresis of 0.5 V, and can accept 30-V inputs.
Each driver converts TTL/CMOS input levels into TIA/EIA-232-F levels. The driver, receiver, and voltage-generator functions.

5.6.1 Pin Diagram

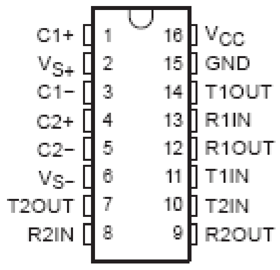


figure 19- Pin diagram of MAX232.

5.6.2 Logic diagram (positive logic)

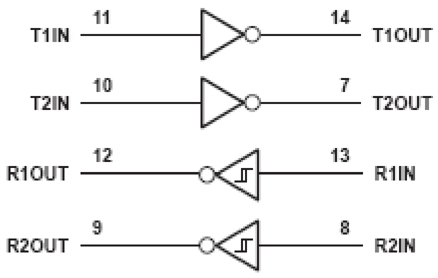


Figure 20- Logic diagram

5.7 FPGA based Temperature Sensing System

The system mainly consists of two units: the system board and the control centre. The control centre in turn consist of two units, The PC and Mobile phone connected together through the serial communication port RS232.the system board consist of three units; the controller unit which has been implemented in Spartan FPGA, the sensor circuit, and the GSM MODEM, the controller connected connected to the GSM Modem through the serial communication port is located in the remote land, where temperature are measured. The main function of system board is continuously measure the temperature and compares the measured values with the threshold level, and sends message through GSM network to the control centre in of high temperature,. The main subunit of the system board is the controller that has been designed using VHDL and using Xilinx Spartan 3 FPGA.

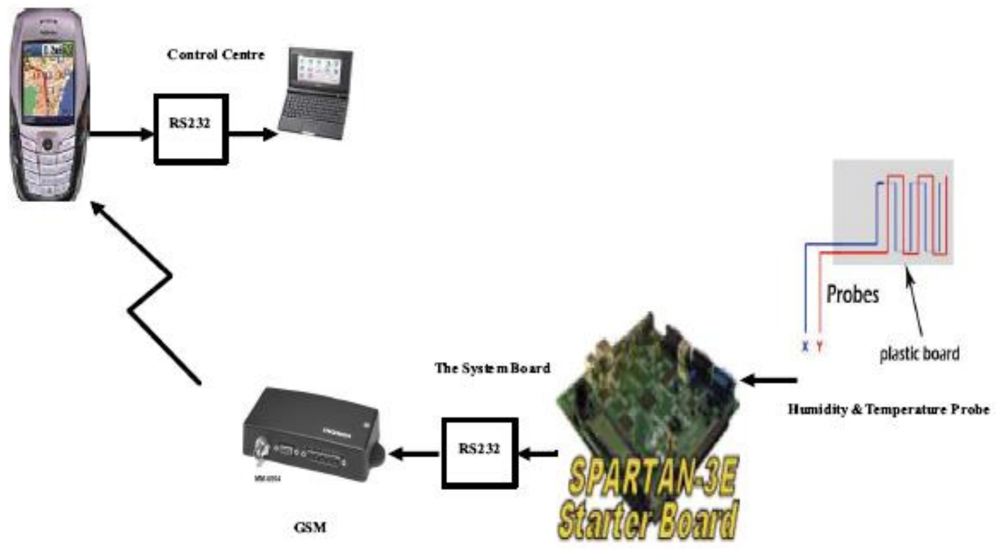


Figure 21 -System View

Conclusion and Scope for Further Work

Conclusion

In the project FPGA is used because of its reprogrammability and flexibility as compared to ASICs. When compared with processors, FPGA are much faster and low power consuming devices and ASICs are faster than processors but they are Very expensive. Also Long production cycles and Upgradeability are major problems with ASICs. So, FPGA offers a better solution to above constraints.

In this project, an FPGA hardware design implementation of remote sensing system for temperature control using GSM has been introduced. FPGAs can be reprogrammed any no. of times with different logic functionalities. A change in design requires only modification of “logic Function”, recompilation and redownloading.

The system is designed using VHDL in a high level design method. Components of the design have been simulated and implemented using Xilinx tools. Because FPGAs are Digital Input, Digital Output devices, hence temperature sensing unit is connected with FPGA using ADC, because parameters of temperature sensing unit are Analog in nature, like temperature, humidity etc. For real time monitoring, GSM is used and it is connected with FPGAs using RS-232 connector.

Scope for Further Work

In this project, an FPGA hardware design implementation of remote sensing system for temperature control using GSM has been introduced. And we can connect many type of sensors to sense many physical parameters with this like humidity, level, pressure and many more. FPGAs are good for multiplexing, demultiplexing, decoding and for all logical functions. And second is to use GPRS (General Packet Radio Service) for remote monitoring instead of GSM.

.

Appendix: Source Code

CODE-1 LED glow

```
-------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    16:04:39 08/07/2008
-- Design Name:
-- Module Name:    led - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--

-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-------------------------------------------------------------------------------

----

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity led is
   Port ( x : in  STD_LOGIC_vector(6 downto 0);
          y : out  STD_LOGIC_vector (6 downto 0));
end led;

architecture Behavioral of led is
signal clk :STD_LOGIC:='0';
begin
process (clk)
```

```
variable c:integer:=0;

begin

if (clk='1' and clk' event) then
c:=c + 1;


if (c=0)then y<="1111110";end if ;
if(c=1)then y<="0110000";end if ;
if (c=2)then y<="1101101";end if ;
if (c=3)then y<="1111001";end if ;
if (c=4)then y<="0110011";end if ;
if (c=5)then y<="1011011";end if ;
if (c=6)then y<="1011111";end if ;
if (c=7)then y<="1110000";end if ;
if (c=8)then y<="1111111";end if ;
if (c=9)then y<="1111011";
end if;
end if;
end process ;

end Behavioral;
```

CODE-2 Interfacing program for ADC (analog to digital converter) with FPGA

```
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity adc is
      port(clk:in std_logic;
           rd,wr:out std_logic;
                intr:in std_logic);
end adc;


architecture Behavioral of adc is
--signal di:integer range 1 to 4;
begin
process (clk)
variable di: integer range 1 to 4;
begin
  if (clk'event and clk='1') then


      --prog for ADC Control
case di is
                    when 1=>
                           if intr='1'      then
                           wr<='0';
                           end if;
                 di:=di+1;
                    when 2 =>
                --if intr='0'    then
                if intr='1'      then
                wr<='1';
                rd<='0';
```

```
            di:=1;
    end if;

    --end if;
         di:=di+1;
                         when 3 =>
     if intr='0'        then
    rd<='1';
            end if;
     di:=di+1;


                         when 4 =>
    rd<='0';
    wr<='0';

    di:=1;
    end case;
       end if;
                        end process;
end Behavioral;
```

Code 3.. ADC Message command to GSM

```
            ----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    17:42:57 06/17/2009
-- Design Name:
-- Module Name:    adcmessa - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;


entity adcmessa is
Port ( clk : in  STD_LOGIC;
          y : out  STD_LOGIC;
          s : in  STD_LOGIC;
          rd,wr:out std_logic;
           intr:in std_logic;
           z:in std_logic_vector(7 downto 0)    );
  end adcmessa;

architecture Behavioral of adcmessa is

signal clock:std_logic:='0';


signal clk1:std_logic:='0';

signal  t:integer range 0 to 40000 :=0;
signal k: integer range 0 to 79000:=0;

constant a: std_logic_vector(9 downto 0):="1000000000";
constant a1: std_logic_vector(19 downto 10):="1011000010";--a
constant a2: std_logic_vector(29 downto 20):="1011101000";--t
constant a3:std_logic_vector(39 downto 30):="1000011010";---enter d
constant a4:std_logic_vector(13010 downto 13001):="1011000010";--a
constant a5:std_logic_vector(13020 downto 13011):="1011101000";--t
constant a7:std_logic_vector(13030 downto 13021):="1001010110";--+
constant a8:std_logic_vector(13040 downto 13031):="1011000110";--c
constant a9:std_logic_vector(13050 downto 13041):="1011011010";--m
constant a10:std_logic_vector(13060 downto 13051):="1011001110";--g
constant a11:std_logic_vector(13070 downto 13061):="1011001100";--f
constant a12:std_logic_vector(13080 downto 13071):="1001111010";--=
constant a13:std_logic_vector(13090 downto 13081):="1001100010";--1
constant a14:std_logic_vector( 13100 downto 13091):="1000011010";---enterd
constant a15:std_logic_vector(26071 downto 26062):="1011000010";--a
constant a16:std_logic_vector(26081 downto 26072):="1011101000";--t
constant a17:std_logic_vector(26091 downto 26082):="1001010110";--+
constant a18:std_logic_vector(26101 downto 26092):="1011000110";--c
constant a19:std_logic_vector(26111 downto 26102):="1011011010";--m
constant a20:std_logic_vector(26121 downto 26112):="1011001110";--g
constant a21:std_logic_vector(26131 downto 26122):="1011100110";--s
constant a22:std_logic_vector(26141 downto 26132):="1001111010";--=
constant a23:std_logic_vector(26151 downto 26142):="1001000100";--"
constant a24:std_logic_vector(26161 downto 26152):="1001110010";--9
constant a25:std_logic_vector(26171 downto 26162):="1001110000";--8
constant a26:std_logic_vector(26181 downto 26172):="1001100010";--1
constant a27:std_logic_vector(26191 downto 26182):="1001101010";--5
constant a28:std_logic_vector(26201 downto 26192):="1001110010";--9
constant a29:std_logic_vector(26211 downto 26202):="1001100100";--2
constant a30:std_logic_vector(26221 downto 26212):="1001101010";--5
constant a31:std_logic_vector(26231 downto 26222):="1001100010";--1
constant a32:std_logic_vector(26241 downto 26232):="1001101010";--5
constant a33:std_logic_vector(26251 downto 26242):="1001110010";--9
constant a34:std_logic_vector(26261 downto 26252):="1001000100";--"
constant a35:std_logic_vector(26271 downto 26262):="1000011010";--enterd
constant a36:std_logic_vector(39242 downto 39233):="1011010000";--h
constant a37:std_logic_vector(39252 downto 39243 ):="1011001010";--e
constant a38:std_logic_vector(39262 downto 39253):="1011011000";--l
constant a39:std_logic_vector(39272 downto 39263):="1011011000";--l
constant a40:std_logic_vector(39282 downto 39273):="1011011110";--od
constant a41:std_logic_vector(39292 downto 39283):="1000110100";--ctrl z


begin
process(clk,s)
variable c:integer range 0 to 8000000:=0;
--variable d:integer range 0 to 8000000:=0;
begin
if s='1' then

if clk'event and clk= '1' then
c:= c+1;
--d:=d+1;
if c=416 and k<78584 then
clock<= not clock;
k<=k+1;
c:=0;
elsif k=78584 then
clock<='Z';
end if;
end if;

--if d=4 then
--clk1<=not clk1;
--d:=0;
--end if;
```

```
elsif s='0' then
k<=0;
end if;
end process;
process(clk,s)
variable di:integer range 1 to 4;
begin
if s='1' then
if clk'event and clk='1' then
case di is
when 1=>
               if intr='1'       then
               wr<='0';
end if;
        di:=di+1;

         when 2 =>
      --if intr='0' then
if intr='1'          then
        wr<='1';
     rd<='0';
     di:=1;
    end if;
           --end if;
     di:=di+1;

when 3 =>
     if intr='0'     then
     rd<='1';
   end if;
       di:=di+1;

when 4 =>
      rd<='0';
            wr<='0';
             di:=1;


            end case;
  end if;
elsif s='0' then
null;
end if;
end process;
```

```
process(clock,z)
begin
--if s='1' then
if z>="00000001" then
if clock'event and clock='1' then


t<=t+1;
if (t<=9) then
y<=a(t);
elsif(t>=10 and t<=19) then
y<=a1(t);
elsif(t>=20 and t<=29)then
y<=a2(t);
elsif(t>=30 and t<=39)then
y<=a3(t);
elsif(t>=40 and t<=13000)then
null;
elsif(t>=13001 and t<=13010) then
y<=a4(t);
elsif(t>=13011 and t<=13020) then
y<=a5(t);
elsif(t>=13021 and t<=13030) then
y<=a7(t);
elsif(t>=13031 and t<=13040) then
y<=a8(t);
elsif(t>=13041 and t<=13050) then
y<=a9(t);
elsif(t>=13051 and t<=13060) then
y<=a10(t);
elsif(t>=13061 and t<=13070) then
y<=a11(t);
elsif(t>=13071 and t<=13080) then
y<=a12(t);
elsif(t>=13081 and t<=13090) then
y<=a13(t);
elsif(t>=13091 and t<=13100) then
y<=a14(t);
elsif(t>=13101 and t<=26061) then
null;
```

```
elsif(t>=26062 and t<=26071) then
y<=a15(t);
elsif(t>=26072 and t<=26081) then
y<=a16(t);
elsif(t>=26082 and t<=26091) then
y<=a17(t);
elsif(t>=26092 and t<=26101) then
y<=a18(t);
elsif(t>=26102 and t<=26111) then
y<=a19(t);
elsif(t>=26112 and t<=26121) then
y<=a20(t);
elsif(t>=26122 and t<=26131) then
y<=a21(t);
elsif(t>=26132 and t<=26141) then
y<=a22(t);
elsif(t>=26142 and t<=26151) then
y<=a23(t);
elsif(t>=26152 and t<=26161) then
y<=a24(t);
elsif(t>=26162 and t<=26171) then
y<=a25(t);
elsif(t>=26172 and t<=26181) then
y<=a26(t);
elsif(t>=26182 and t<=26191) then
y<=a27(t);
elsif(t>=26192 and t<=26201) then
y<=a28(t);
elsif(t>=26202 and t<=26211) then
y<=a29(t);
elsif(t>=26212 and t<=26221) then
y<=a30(t);
elsif(t>=26222 and t<=26231) then
y<=a31(t);
```

```
elsif(t>=26232 and t<=26241) then
y<=a32(t);
elsif(t>=26242 and t<=26251) then
y<=a33(t);
elsif(t>=26252 and t<=26261) then
y<=a34(t);
elsif(t>=26262 and t<=26271) then
y<=a35(t);
elsif(t>=26272 and t<=39232) then--
null;
elsif(t>=39233 and t<=39242) then
y<=a36(t);
elsif(t>=39243 and t<=39252) then
y<=a37(t);
elsif(t>=39253 and t<=39262) then
y<=a38(t);
elsif(t>=39263 and t<=39272) then
y<=a39(t);
elsif(t>=39273 and t<=39282) then
y<=a40(t);
elsif(t>=39283 and t<=39292) then
y<=a41(t);
end if;
end if;
else   null;
end if;
--end if;
end process;
end Behavioral;
```

CODE-4 GSM code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity at is
       port(clk:in std_logic;
            d :in std_logic;
             tx:out std_logic;
              rx:in std_logic;
        d1: out std_logic_vector(15 downto 0));
end at;


architecture Behavioral of at is

signal a: std_logic_vector(19 downto 0);
signal b: std_logic_vector(19 downto 0):="10100111101010010110";
signal t:integer:=0;
signal x:integer:=0;
signal clock: std_logic:='0';
-----------------------------------------------------------------------------------
constant a0:  std_logic_vector (9   downto 0)  :="1000000000"; --00--null

constant a1: std_logic vector (19 downto 10)  :="1011000010"; --61--for=a

constant a2:  std_logic_vector (29    downto 20)    :="1011101000"; --74--for=t


constant a3:  std_logic_vector (39    downto 30)    :="1000011010"; --5c--for=slash 0d

constant a4:  std_logic_vector (49    downto 40)    :="1000010100"; --6e--for=n  0a

begin


       process (clk)

      variable i:integer:=0;

begin
       if (clk='1' and clk'event ) then
        if i=2 then
        clock<= not clock;
```

```
               i:=0;
             else
           i:=i+1;
          end if;
        end if;
     end process;
---------------------------------------------------------
process(clock)
begin
if clock='1' and clock'event then
if(d='1')then
--t<=t+1;
if(t<=9) then
t<=t+1;
tx<=a0(t);
   elsif(t>=10 and t<=19)then
t<=t+1;
tx<=a1(t);
     elsif(t>=20 and t<=29)then
     t<=t+1;
tx<=a2(t);
      elsif(t>=30 and t<=39)then
t<=t+1;
tx<=a3(t);
      elsif(t>=40 and t<=49)then
t<=t+1;
tx<=a4(t);
      elsif(t=50)then
if rx='0' then
t<=t+1;
       else
t<=50;
        end if;
   elsif(t=51)then
t<=t+1;
```

```
d1(0)<=rx;
     elsif(t=52)then
t<=t+1;
d1(1)<=rx;
      elsif(t=53)then
t<=t+1;
d1(2)<=rx;
       elsif(t=54)then
t<=t+1;
d1(3)<=rx;
      elsif(t=55)then
t<=t+1;
d1(4)<=rx;
     elsif(t=56)then
t<=t+1;
d1(5)<=rx;
     elsif(t=57)then
t<=t+1;
d1(6)<=rx;
      elsif(t=58)then
t<=t+1;
d1(7)<=rx;
     elsif(t=59)then
if rx='1' then
t<=t+1;
   --d1(9)<=rx;
       else
t<=50;
           end if;
  elsif(t=60)then
if rx='0' then
t<=t+1;
   else
t<=60;
```

```
end if;
     elsif(t=61)then
t<=t+1;
d1(8)<=rx;
      elsif(t=62)then
t<=t+1;
d1(9)<=rx;
      elsif(t=63)then
t<=t+1;
d1(10)<=rx;
      elsif(t=64)then
t<=t+1;
d1(11)<=rx;
    elsif(t=65)then
t<=t+1;
d1(12)<=rx;
      elsif(t=66)then
t<=t+1;
d1(13)<=rx;
      elsif(t=67)then
t<=t+1;
d1(14)<=rx;
      elsif(t=68)then
t<=t+1;
d1(15)<=rx;
     elsif(t=69)then
if rx='1' then
t<=t+1;
     --d1(10)<=rx;
   else
t<=60;
     end if;
        end if;
            end if;
--tx<=a4(t);

   --end if;
--
--case x is
--
      --when 1=>
--if (rx='0') then
--a(0)<=rx;
--x<=x+1;
--end if;
```

```
--
    --when 2=>
--a(1)<=rx;
--x<=x+1;
--
    --when 3=>
--a(2)<=rx;
--x<=x+1;
--
    --when 4=>
--a(3)<=rx;

--x<=x+1;

--

     --when 5=>
--a(4)<=rx;
--x<=x+1;
--
     --when 6=>
--a(5)<=rx;
--x<=x+1;
--
    --when 7=>
--a(6)<=rx;
--x<=x+1;
--
    --when 8=>
--a(7)<=rx;
--x<=x+1;
--
    --when 9=>
--a(8)<=rx;
--x<=x+1;
--
    --when 10=>
--if(rx='1')then
--a(9)<=rx;
--x<=x+1;
--end if;
--
    --when 11=>
--if(rx='0')then
--a(10)<=rx;
--x<=x+1;
--end if;
--
    --when 12=>
--a(11)<=rx;
--x<=x+1;
--
```

```
     --when 13=>
--a(12)<=rx;
--x<=x+1;
--
     --when 14=>
--a(13)<=rx;
--x<=x+1;
--
    --when 15=>
--a(14)<=rx;
--x<=x+1;
--
     --when 16=>
--a(15)<=rx;
--x<=x+1;
--
    --when 17=>
--a(16)<=rx;
--x<=x+1;
--
    --when 18=>
--a(17)<=rx;
--x<=x+1;
--
    --when 19=>
--a(18)<=rx;
--x<=x+1;
--
    --when others=>
--
--if(rx='1') then
--a(19)<=rx;
----x<=x+1;
        --end if;
--
--
--if(a=b)then
--d1(15 downto 8)<=a(18 downto 11);
--d1(7 downto 0)<=a(9 downto 2);
        --end if;
            --end case;
                      --end if;

end if;
     end process;
           end Behavioral;
```

References

Literature

• Books

❖ S. Kilts, *Advanced FPGA Design: Architecture, Implementation, and Optimization*, John Wiley & Sons, 2007.

❖ U. Meyer-Bäse, *Digital Signal Processing with Field Programmable Gate Arrays* 2nd ed., Springer, 2004.

❖ W. Wolf, *FPGA-Based System Design*, Prentice Hall, 2004.

❖ P.J. Ashenden, *The Designer's Guide to VHDL*, 2nd ed., Morgan Kaufmann, 2002.

❖ Circuit Design With VHDL by Pedroni.

❖ Perry, D., "VHDL", McGraw Hill Publications, 1991.

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

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

• Papers

❖ K. Bondalapati, V.K. Prasanna, Reconfigurable Computing Systems, *Proceedings of the IEEE*, vol. 90, no. 7, Jul. 2002, pp. 1201–1217.

❖ D. Bouldin, Enhancing Electronic Systems with Reconfigurable Hardware, *IEEE Circuits and Devices Magazine*, vol. 22, no. 3, May–Jun. 2006, pp. 32–36.

❖ N. Chakravarthy and X. Jizhong, FPGA-based Control System for Miniature Robots, IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, 2006, pp. 3399-3404

❖ Kuon, J. Rose, Measuring the Gap Between FPGAs and ASICs, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, Feb. 2007, pp.203–215.

❖ T.J. Todman *et al*., Reconfigurable Computing: Architectures and Design Methods, *IEE Proceedings: Computers and Digital Techniques*, vol. 152, no. 2, Mar. 2005, pp. 193–207.

❖ El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. El-Ayat and A. Mohsen, "An Architecture for Electrically Configurable Gate Arrays," Proc. 1988 Custom Integrated Circuits Conference, May 1988, pp. 15.4.1 - 15.4.4.

❖ http://en.wikipedia.org/wiki.
  ▪ Xilinx, www.xilinx.com
  ▪ Altera, www.altera.com

❖ National semiconductor,LM35 precision centigrade temperature sensor. On www.national.com. Nov 2000.

❖ The Programmable Gate Array Data Book, Xilinx Co., 1989.

❖ www.beyondlogic.org/serial/serial1.htm

❖ www.celoxica.com/techlib/files/CEL-W0307171HR6-FPGA

❖ www.eedesign.com

❖ www.eg3.com/fpga

❖ www.xilinx.com/company/about/programmable.html