FPGA BASED SINGLE CHANNEL ANALYZER (SCA) USED FOR NUCLEAR RADIATION COUNTING SYSTEM

A Thesis

Submitted By

Mohaimina Begum

ID: 10371003

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Engineering in Electrical and Electronic Engineering

Department of Electrical and Electronic Engineering

BRAC University

Dhaka-1212, Bangladesh

October 2015

Dedication

This work is dedicated to my parents

The thesis titled, "**FPGA BASED SINGLE CHANNEL ANALYZER (SCA) USED FOR NUCLEAR RADIATION COUNTING SYSTEM**" submitted by **Mohaimina Begum**, Roll No. 10371003, Program-M in EEE, BRAC University has been accepted as satisfactory in partial fulfillment of the requirement for the degree of Master of Engineering in Electrical and Electronic Engineering, M. Engg. (EEE) on ……

Board of Examiners

1.

Dr. Md. Mosaddequr Rahman
Professor
Electrical and Electronic Engineering
BRAC University
Dhaka, Bangladesh.

Chairman & Supervisor

2.

Dr. Mohammed Belal Hossain Bhuian
Associated Professor
Electrical and Electronic Engineering
BRAC University
Dhaka, Bangladesh.

Member

3.

Dr. A. B. M. Harun-Ur-Rashid
Professor
Electrical and Electronic Engineering
BRAC University
Dhaka, Bangladesh.
&
Professor
Electrical and Electronic Engineering
Bangladesh University of Engineering & Technology (BUET)
Dhaka, Bangladesh.

Member (External)

Declaration of Originality

This is certify that this thesis titled “FPGA based Single Channel Analyzer (SCA) used for Nuclear Radiation Counting System” submitted to the Department of EEE, BRAC University is our original work and no part of this work has been submitted to any university or elsewhere for the award of any academic degree.

(Author)
Mohaimina Begum
ID 10371003
M. Engg. (EEE)
Department of Electrical & Electronic Engineering
BRAC University
Bangladesh

Acknowledgement

All the appreciation to Allah, the merciful and beneficial who has enabled me to submit this unassuming work leading to the M. Engineering Degree.

First of all I would like to express my sincere gratitude to my supervisor, Dr. Md. Mosaddequr Rahman, Professor, Department of Electrical & Electronic Engineering, BRAC University, Dhaka, for his invaluable help and steadfast support throughout the period of this research.

I am grateful to Dr. Md. Sayeed Salam, Head, Department of Electrical & Electronic Engineering, BRAC University, Dhaka for providing me with all the facilities of this department to carry out this work.

I thank the staff at the BRAC University, Dhaka for their support and co-operation during this research work.

I am grateful to Director of Atomic Energy Centre, Dhaka for giving me the opportunity to carry out my research work in the laboratory of Electronics Division and providing me with all the necessary instruments and software for my work. I express my sincere gratitude to Engr. Anisa Begum, Head, Electronics Division, Atomic Energy Centre, and Dhaka for her constant guidance and help through my research work. I also express my profound gratitude to Mr. Abdullah Al Mamun and Atiar Rahman, S.S.O. Electronics Division, Atomic Energy Centre Dhaka for their practical help and valuable suggestions in my work.

Finally, I am indebted to my husband, parents, my daughter, son and my extended family and friends for their great support.

Abstract

Hardware configurable Single Channel Analyzer (SCA) is designed and developed to improve the performance and flexibility of the system compared with traditional approaches. In this work, a new technique has been introduced for changing the window between the Upper Level Detection (ULD) and the lower Level Detection (LLD) for SCA by a simple change in software, without any hardware modification. To check this technique, electric pulse generated from the detector conditioned by a preamplifier is fed to ADC through a processing circuit. When the ADC output value is higher than the LLD and lower than the ULD, then counter will count those values over a period of one minute and store the counting value in a register. Finally the stored counting values are given to LCD through other necessary circuits. In addition, maximum peak value, counts per minute and total counting value are also displayed on LCD. Associate firmware has been developed by Xilinx ISE Design suite 9.2 using VHDL code and tested on Xilinx Spartan 3E Starter board. This developed system may be effectively used for radiation monitoring of human body as well as environment.

Table of Contents

Chapter 4 FPGA based Single Channel Analyzer

Chapter 5 Results and discussion

List of Figures

Figure No. & Name **Page No.**

List of Tables

Introduction

1.1 Introduction

Radiation is the emission or transmission of energy in the form of waves or particles through space or through a material medium, often categorized as either ionizing or non-ionizing depending on the energy of the radiated particles. This thesis gives importance on the Ionizing radiation. Ionizing radiation carries more than 10 eV, which is enough to ionize atoms and molecules, and break chemical bonds. This is an important distinction due to the large difference in harmfulness to living organisms. Common source of ionizing radiation are radioactive materials that emit α, β or γ radiation, consisting of helium nuclei, electrons or positrons, and photons, respectively. This radiation also comes from natural sources such as cosmic rays from the universe, the earth, as well as man-made sources (Artificial) such as those from nuclear fuel and medical procedures. Other sources include X-rays from medical radiography examinations, diagnostic imaging, cancer treatment (such as radiation therapy), radiation is also used in many industries including food irradiation, nuclear reactors with neutron fission etc. If these radiations used in a right technique then radiation is a blessing, but without its controlled uses it is very much harmful for creatures and environment. It is harmful when material that contains radioactive atoms is deposited on skin, clothing, or any place where is it not desired. It is important to remember that radiation does not spread or get "on" or "in" people; rather it is radioactive contamination that can spread. A person contaminated with radioactive materials will be irradiated until the source of radiation (the radioactive material) is removed. Basic control methods for external radiation is decrease Time, increase distance and increase shielding, i.e. minimum time of exposure is to minimize total dose, maximum distance is to source to maximize attenuation in air and minimize exposure is by placing absorbing shield between worker and source.

Because of radiation hazards many diseases occur and in the long run leads to death. Therefore to grow awareness in the people about radiation it is needed to develop available facility for radiation detection and monitoring.

The nuclear radiation cannot be detected by human senses, therefore need equipment, so called "Nuclear Counting System" to detect and measure that radiation. In nuclear counting system, Single Channel Analyzer (SCA) is used for measuring radiation. In this work, we attempt to develop a Field Programmable Gate Array (FPGA) based SCA system that can

detect and measure the nuclear radiation by counting the electric pulses, which are produced by the detector. Number of output pulses is proportional to the number of incoming radiation. This system has a lower and an upper level discriminator and produces an output logic pulse whenever an input pulse falls between the discriminator levels. All voltage pulses in a specific range of discriminator can be selected and counted by counter [1] and finally displays these counts in LCD.

Nuclear Counting Systems are employed in nuclear medicine to measure radioactivity for various purposes such as radioimmunoassay and competitive protein binding assay of drugs, hormones and other biologically active compounds; and for radionuclide identification, quality control and radioactivity examines in radio pharmacy and radiochemistry. SCA is an easy and efficient system for radiation detection and counting. The developed system can also be used in Health Physics Division of Atomic Energy Centre, Dhaka for measuring Gamma radiation of background, imported food samples and also vegetables, water & soil samples of different districts of Bangladesh. In future the system can be employed in Rooppur Nuclear Power Plant (RNPP) for environmental radiation monitoring purposes.

Many researchers have reported their design and application on FPGA based system and SCA. A. Ezzatpanah Latifi developed design and construction of an accurate timing Single Channel Analyzer [2]. Amitkumar Singh designed and simulated a system on FPGA based digital multi channel analyzer for nuclear spectroscopy application [3]. Hui Tan reported the design on single channel beta-gamma coincidence detection of radioactive xenon using digital pulse shape analysis of phoswich detector signals [4]. From this scientific information on the use of FPGA based system; the proposed system was focused on a new technology applied in Nuclear Counting System. In existing SCA, hardware modification is necessary to change window between upper level detection (ULD) and lower level detection (LLD) to detect radiation from different radiation sources, which has less flexibility. In our proposed design, an FPGA based SCA nuclear counting system has been completed which does not require hardware modification to change window between ULD and LLD and can be easily performed by software control. As the system can be made possible by a single FPGA chip, it takes less time to modify the design and will also be cost effective but other side the difficulties had to face in my design is unavailability of detector and another is the system works at low range activity. Finally, in this thesis we report the design and development of an FPGA based nuclear counting system which can make reconfigurable hardware by software, has high precision and faster than other microcontroller based system.

1.2 Scope of work

We know radiation is harmful but if it is used in controlled way then it is very much useful in different field. So the main objective of our work is to detect and monitor radiation and grow consciousness in the people. The system is FPGA based so the hardware is reconfigurable by software. XC3S500E FPGA of Xilinx is used in our design because we have Xilinx SPARTAN 3E, Starter board. VHDL and Verilog are used for programming FPGA. In our design VHDL is used for coding. The detector Model 712, LND, INC. (Halogen GAS) which is used in this design it can perform to detect Gamma Ray (γ-ray) and Beta Ray (β-ray). Our designed system offer high precision and is portable and fast which can effectively detect and count activity range for Gamma radiation from .01 to 1000 count per minute (CPM). Consider low activity range from 0.2 to 1.0 Micro Sivert per Hour (µSv $h^{-1)}$ is equivalent to 20 to 100 CPM.

All types of ionizing radiation are controlled by three ways: Time, Distance and Shielding. We have used in this thesis distance parameter for monitoring radiation. A commercial Survey Meter (GAMMA-SCOUT) and our developed FPGA based SCA system was placed at a fixed point and the distance of radioactive point source ^{137}Cs was varied in cm. We also used point source ^{60}Co and ^{131}I for measurement radiation. We have presented data for point source ^{137}Cs in performance evaluation. Difficulty faced in this work, is the unavailability of detector also during measurement of radiation at have to care of experiment time since it is harmful for human and environment. Except for these difficulties, as our system has flexibility to configure hardware and it can replace complex analog nuclear counting circuitry.

Our system can be used for radionuclide identification in nuclear power plants, Health Physics Division of different organizations etc. which may overcome the in availability of effective nuclear counting systems in the market. Also the system will open a new era of radionuclide characterization research in the field of nuclear medicine, radio pharmacy, radiochemistry etc.

1.3 Thesis Organization

The thesis has been divided into six chapters for step by step development.

In Chapter 1, introduction, literature review and scope of work have been stated.

In Chapter 2, radiation & its sources, characteristics of radiation, its energy, radiation detectors, nuclear counting system, different type of nuclear counting system, and details of Single Channel Analyzer have been explained.

Chapter 3 deals with FPGA, its internal architecture, characteristics of FPGA, uses of FPGA, introduction to VHDL, VHDL constructs, basic VHDL programming, programming in Software Xilinx ISE design suite 9.2 and FPGA programming step have been presented.

In Chapter 4, Design scenario for nuclear counting system, specification of the developed FPGA based SCA system, setting of high voltage to detector, detector plateau measurement for different voltage, the proposed FPGA based SCA, gain amplifier and ADC, discriminators, counters, timer, display, software development, view of RTL schematic design, flow diagram of SCA VHDL code, schematic design of SCA, design summery and software development have been described.

In Chapter 5, implementation of FPGA based single channel analyzer, experimental setup for performance evaluation, comparison of developed FPGA based SCA system with commercial survey meter results & its findings and discussion have been presented.

Chapter 6 concludes the work and provides directions for further work.

2.1 Introduction

This chapter describes theoretical consideration of the source of radiation, its characteristics, energy distribution of common isotopes, radiation detectors, different type of nuclear counting system and single channel analyzer etc.

2.2 Radiation & its sources

Generally ionizing radiation is called radiation. Radiation is electromagnetic wave that propagates through matter or space. Radiation is usually classified into non-ionizing (visible light, TV, radio wave) and ionizing radiation. Ionizing radiation has the ability to knock electrons off of atoms changing its chemical properties. When radiation enters a body, it can deposit enough energy that can directly damage DNA. It causes much ionization of atoms in tissues that would eventually cause damage to critical chemical bonds in the body. The effect can be acute (happen right away such as radiation burns, sickness, nausea) or delayed (long-term, such as cancer).

2.3 Characteristics of Radiation

Radiation is characterized by its intensity & energy. The intensity is the number of radiation (photon) per unit time emitted by the source and absorbed by the detector. The intensity of the radiation depends on the activity (in Currie or Becquerel) of the source. Energy is the strength of each radiation emitted by the source. The radiation energy is characteristic to the type of the source. For instance, the isotope ^{137}Cs emits radiation with energy of 662 keV. So, isotope ^{137}Cs with activity of 100 Becquerel (Bq) will emit 100 radiations per second and each radiation has strength of 662 keV. Radiation source that has activity of 100 Bq emits approximately 100 radiations per second. The radiation intensity is random in time, following the Gaussian or normal distribution. So, if we carry out repetitive measurements of radiation intensity with the same condition, we will not get the same result. There will be a fluctuation between those values [5].

2.4 Radiation energy & energy distribution of some common isotopes

The following table shows some common isotopes & their radiation energy distribution which are used for calibration of nuclear counting system [5].

Table 2.1: Radiation energy of some common isotopes

Isotope	Energy (keV)	Relative Intensity(%)
^{137}Cs	662	85
^{60}Co	1173 1332	99 100
^{22}Na	511 1274	200 95

The following Fig. 2.1 shows the ideal distribution of emitted radiation from ^{137}Cs, ^{60}Co and ^{22}Na isotopes. With assumption that all of those isotopes have 100 Bq activity. (1 Bq equal to 1 nuclear disintegration per second).

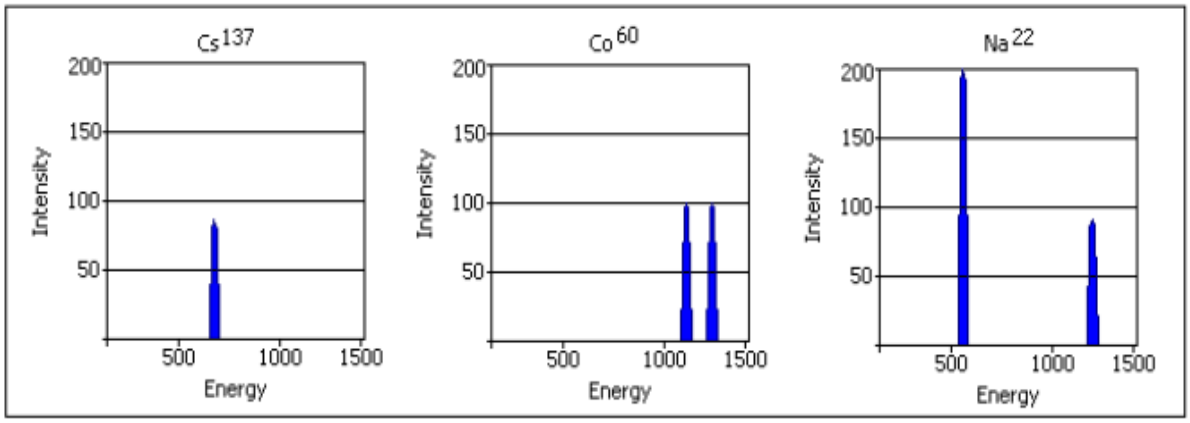


Fig. 2.1: Energy distribution of radiation emitted from some common isotopes

2.5 Radiation Detectors

As radiation is harmful to our health, we need detectors that are capable of sensing the presence and measuring the intensity of radiation, most common radiation detectors are Geiger-Muller (GM) tubes which are gas-filled radiation detectors, useful, cheap and robust. A GM tube basically detects the presence and intensity of radiation. Geiger counters which

use GM tube as a detector are used to detect usually gamma and beta radiation, but some models can also detect alpha radiation.

Ionizing radiation that is associated with radioactivity cannot be directly detected by human senses. Ionization is the process whereby the radiation has sufficient energy to strip electrons away from atoms. The ionization results in the formation of free electrons and an ionized atom that has lost some of its orbital electrons. Examples of ionizing radiation include particles such as alpha and beta particles, and photon radiation such as x-rays and gamma rays. Neutrons and protons can also cause ionizations [5].

The front end of every counting system is a radiation detector, which converts incoming radiation to an electric charge. Ideally, the detector will produce an electric pulse every time a radiation comes in, but this doesn't happen in the real situation. Efficiency is a terminology, which used to compare between the number of electric pulses produced by detector and the number of incoming radiations. Some detectors have efficiency in order of 50% but other detectors have efficiency less than 10%. The most probable interaction that occurs in the detector is the ionization process. In this process, the material absorbs radiation energy and then one or more electrons are ejected from their orbital which is shown in Fig. 2. 2.

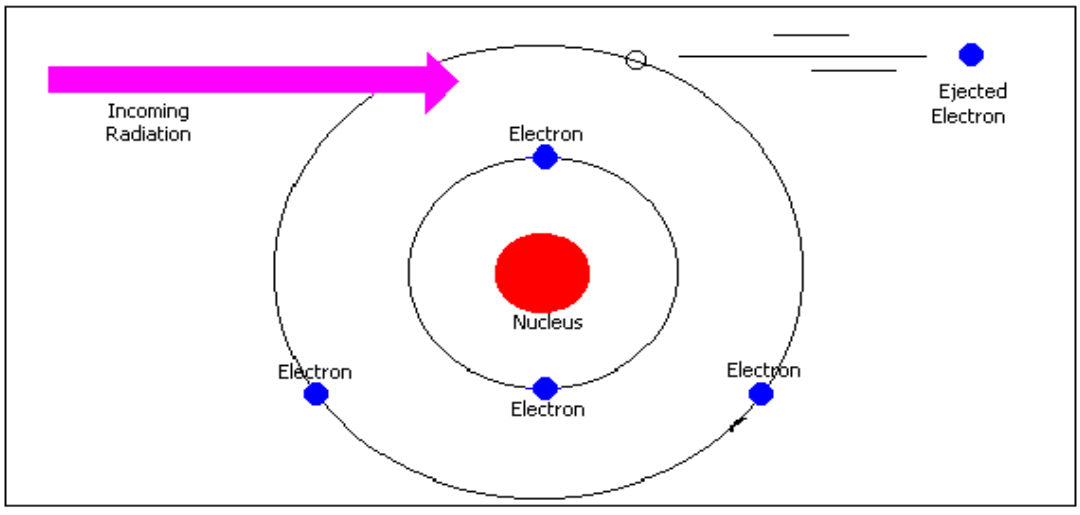


Fig. 2.2: Ionization process

The number of electrons that are ejected from their orbital depends on the energy of incoming radiation. Stronger radiation energy will eject more electrons. In order to produce an electric pulse, those free electrons (electric charge) have to be captured and stored in a

capacitor. Voltage of the output pulse ΔV is equal to the total number of stored electrons Σe^- divided by the capacitance C of detector as shown below, thus the pulse height is correlated linearly to the radiation energy in counting system [5].

$$\mathbf{\Delta V = \frac{\Sigma e^-}{c}} \quad \text{……(2.1)}$$

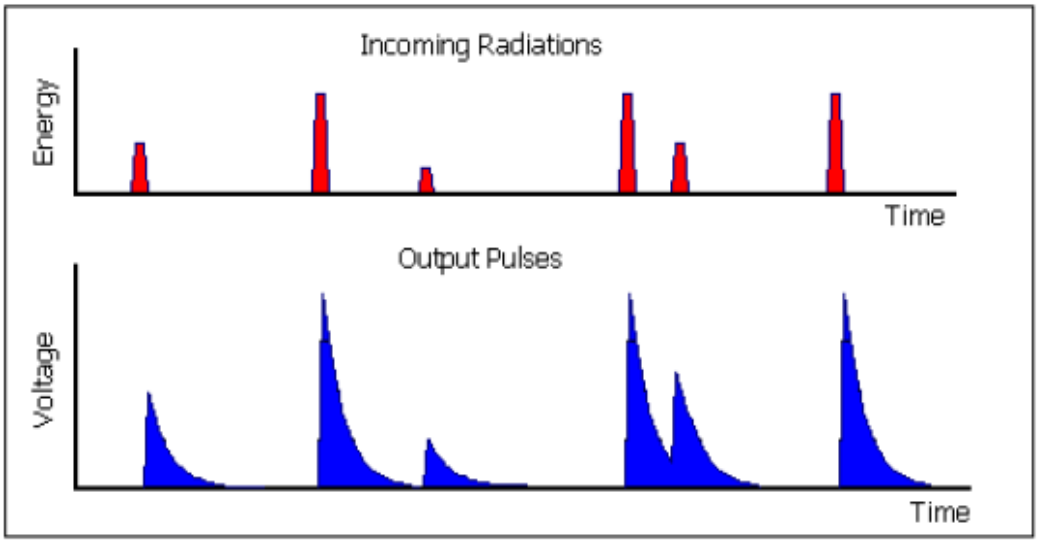


Fig. 2.3: Output pulse from detector

The shape of output pulses from detector is shown in above Fig. 2.3 which is exponential due to discharging capacitor phenomena. The pulse height is proportional to the radiation energy and ideally, each incoming radiation will produce an electric pulse. Sometimes it happens that consecutive radiations come too close (due to random in time characteristic), thus the second pulse will start on the tail of the first pulse (pile up).

There are three types of detector that most frequently used, gas filled detector, scintillation detector and semiconductor detector. The construction of a gas filled detector is very simple, the scintillation detector has very high efficiency, and semiconductor detectors have very high resolution. In my project gas filled (GM) tube detector has been used [5].

2.6 Nuclear Counting System

Based on its application, there are many types of nuclear counting system in the market, starting from very simple and compact equipment such as a survey meter, which is used for

radiation protection (health physics) purpose, to very complex and large scale equipment such as nuclear reactor instrumentation and control unit [5].

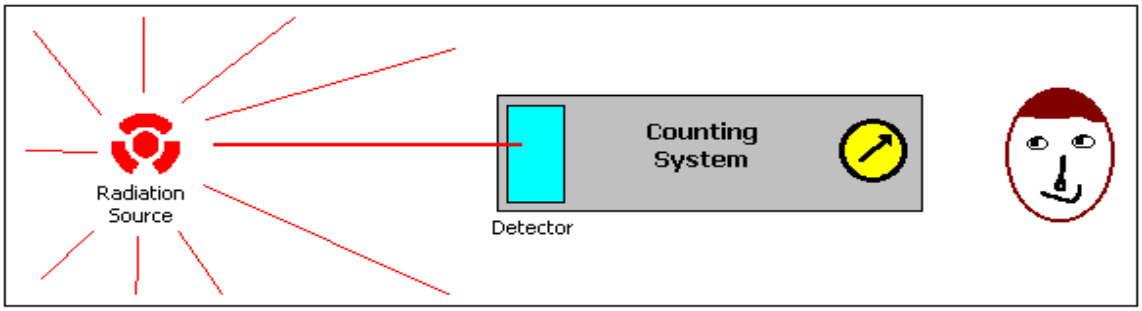


Fig. 2.4: Configuration of measuring the radiation

2.7 Different type of Nuclear Counting System

Basically, all nuclear counting systems have the same principle. When the detector is hit by a radiation, it will convert the radiation energy to be an electronic signal and then those signals are processed by electronic signal and finally can be displayed as an useful information.

Depending on the application, the counting systems can be roughly grouped into

- Single Channel Analyzer (SCA)
- Multi Channel Analyzer (MCA)

The measurements of radiations can be distinguished into two categories, the first is measuring the number of radiations or intensity and the second is measuring the energy distribution. Single Channel Analyzer (SCA) is used for measuring the number of radiations or intensity. A MCA is used for measuring the energy distribution (energy spectrum) of incoming radiation. The spectrum can give information about the intensity at each energy level or on the other hand energy peaks of the incoming radiation can be determined [5].

2.8 Single Channel Analyzer (SCA) System

SCA is used for counting the number of incoming radiation at selected energy range. The SCA has a lower and an upper level discriminator and produces an output logic pulse whenever an input pulse falls between the discriminator levels. With this device, all voltage pulses in a specific range can be selected and counted.

The measurements of radiations can be distinguished into two categories, the first is measuring the number of radiations or intensity and the second is measuring the energy distribution. The intensity of radiation can be measured just by counting the electric pulses, which are produced by the detector. Number of output pulses is proportional to the number of incoming radiation. In my project single channel analyzer system is used to measure the number of radiations or intensity.

When radiation hits the glass window of detector of SCA, detector converts this radiation into electric pulse and amplifier amplifies this pulse and feeds it to discriminator. Discriminator produces TTL logic signal, when the incoming pulse falls between the selected voltage level and counter counts this logic signal from the discriminator for certain interval time. Single Channel Analyzer system consists of some blocks of electronic circuit details has been described below, except the detector, as following Fig. 2.5.

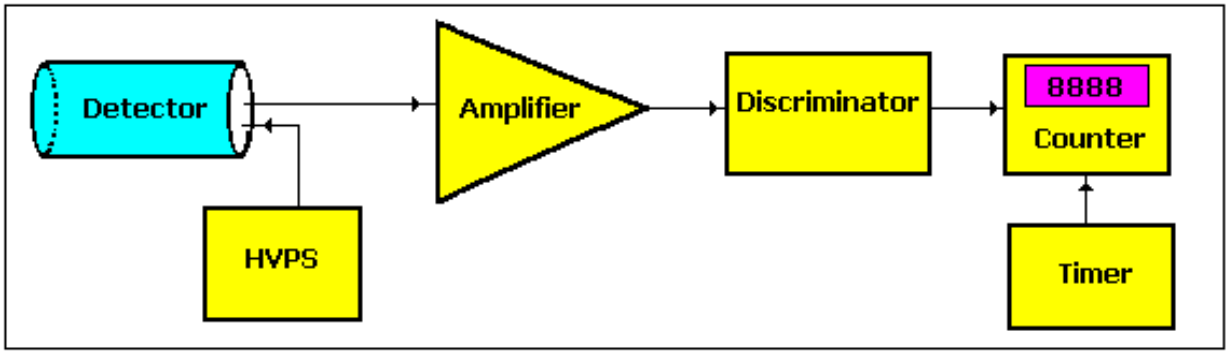


Fig. 2.5: Overview of SCA system

2.8.1 High Voltage Power Supply (HVPS)

HVPS is needed for polarizing the detector. Free electrons which produced by ionization process have to be captured and stored. There must be an electric field in order to push or attract the free electrons to the anode (positive electrode). If there is lack of electric field, the free electrons will move randomly and cannot be captured by the anode.

2.8.2 Amplifier

Amplifier has two functions, shaping and amplifying the electric pulses from detector. The peak of exponential pulses from detector is too sharp to be measured or distinguished and the tail is too long. So, they have to be shaped as Gaussian pulses, which are more flat at the

peak and have not a so long tail in Fig. 2.6 shows the output pulses from detector and amplifier.

The second function of amplifier is to amplify the amplitude of the pulses. Output pulses of detector are in order of mV or even hundreds of μV, so it has to be amplified to few Volts. The amplifier must have facility to change the gain factor of the pulse amplification [5].

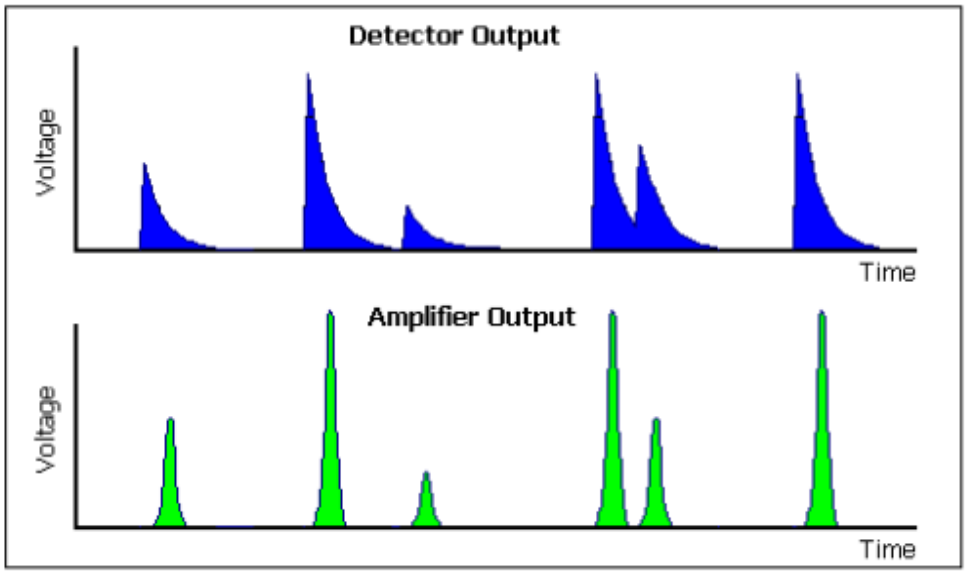


Fig. 2.6: Output pulses from detector and amplifier

2.8.3 Discriminator

Discriminator has a function to discriminate the analog incoming pulses, which comes from the amplifier. The discriminator will produce a TTL logic signal, when the incoming pulse fulfills the energy range criteria, which is defined by the user selectable lower - and upper level shows in Fig. 2.7. Energy range between the red mark lower and upper level is called window of SCA. Only the window voltage is acceptable for counting radiation. Then the lower and upper levels of the discriminator are set at little bit lower and higher than that level [5].

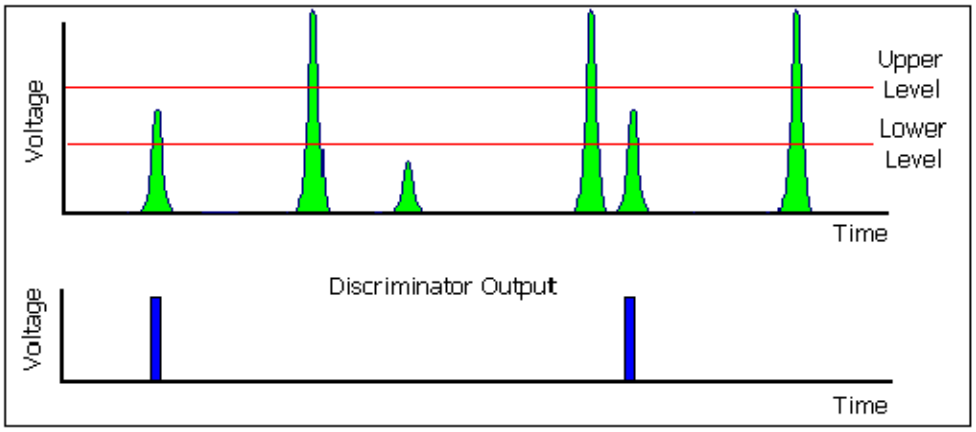


Fig. 2.7: Pulse discrimination process in discriminator [5]

2.8.4 Counters & Timer

Counters & Timer are used for counting the logic signal from the discriminator for certain interval time (counting time). The user sets the counting time through the timer in order of seconds, minutes or hours.

SCA mainly consists of HVPS, detector, amplifier, discriminator, counter and timer block. Without this block some extra circuits are included in SCA design such as preamplifier, latch and driver of LCD etc. Overall objective of this complete system is identification of ionizing radiation.

2.9 Conclusion

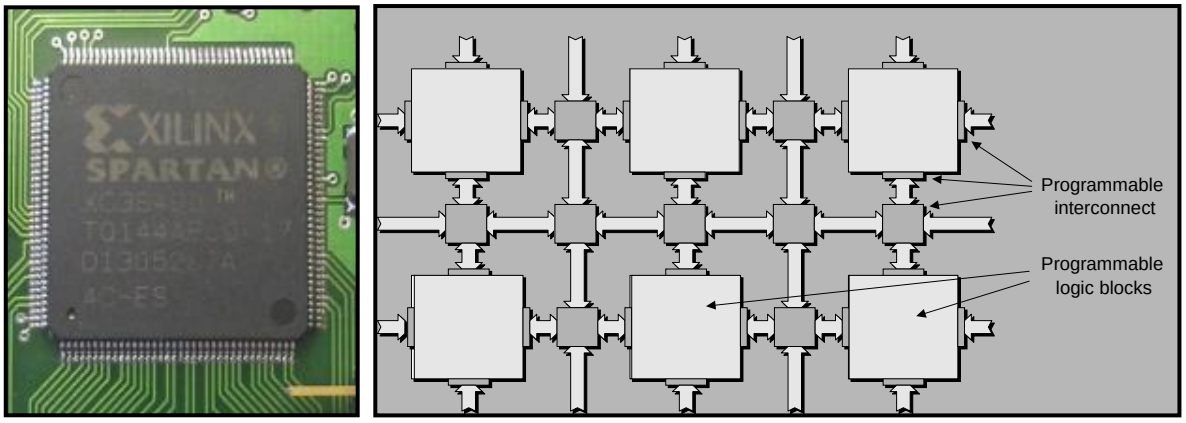
In this chapter we have learnt theoretically about radiation & its background, radiation detector and nuclear counting system which will be helpful to develop nuclear counting system.

3.1 Introduction

This chapter describes about the FPGA, its internal architecture, characteristics, uses and introduction to VHDL, its construct, FPGA programming etc.

3.2 FPGA

FPGA (Field Programmable Gate Array) devices can make reconfigurable hardware which is high precision and faster. It is digital integrated circuit (IC) that contains configurable (programmable) blocks of logic along with configurable (programmable) interconnects between these blocks. Fig. 3.1 illustrates FPGA and its internal blocks.



3.1: FPGA and its Internal Block

In 1984, Xilinx designed this new class of IC: field-programmable gate array (FPGA). “Field programmable” portion of FPGA’s name refers to the fact that its programming takes place “in the field”, which means that FPGA is configured in the laboratory. The first FPGA were based on CMOS and used SRAM cells for configuration purposes. Early design were based on a 3-input Look Up Table (LUT) in the Programmable Logic Block. Depending on the way they are implemented, some FPGAs may only be programmed a single time, while others may be programmed over and over again. Design Engineer configures (programs) this device to perform a tremendous variety of tasks [6].

3.2.1 Internal architecture of FPGA

The core building block in a modern FPGA from Xilinx is called Logic Cell (LC). The Spartan-3 has 4-input LUT.

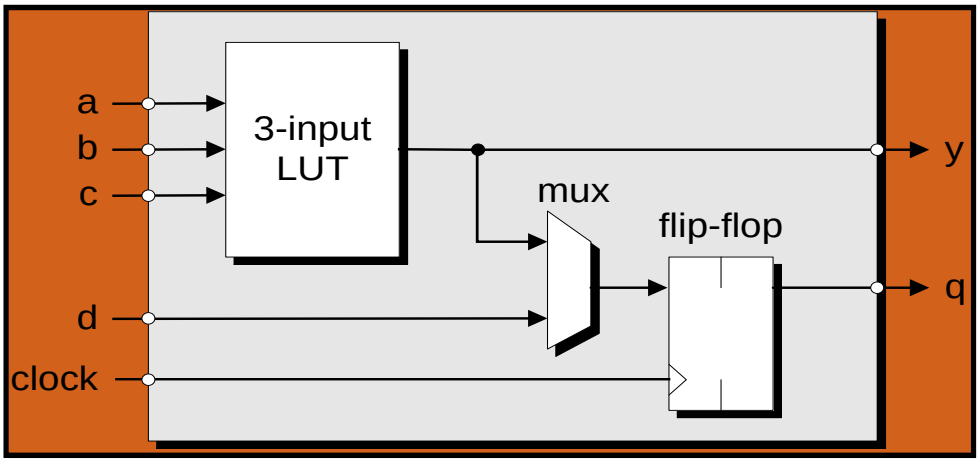


Fig. 3. 2: Simple programmable logic block

A Slice contains two Logic Cells. CLB is a single configurable logic block connected to other CLBs using programmable interconnect. Each CLB can contain two or four slices.

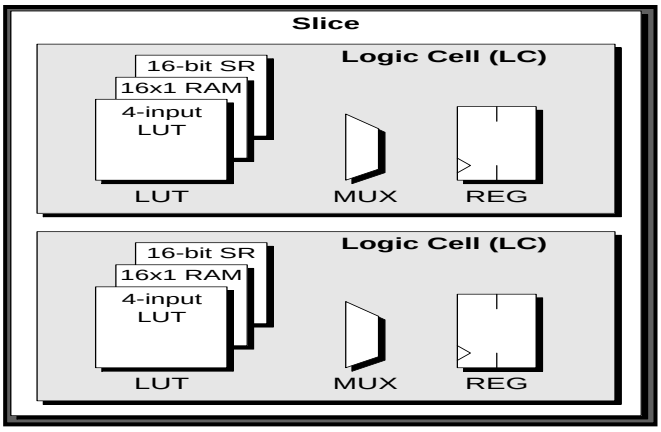


Fig. 3.3: A Slice containing two Logic Cells of Xilinx FPGA

FPGA includes relatively large chunks of embedded RAM called e-RAM or block RAM. The capacity of the block RAM can be varied from few hundred thousand bits to several million bits depending on the chip. The block can be used for a variety of purposes. Some FPGAs provide embedded adder blocks, and it may include embedded MAC (Multiply and Accumulate). Some FPGA also have in addition to RAM, Multipliers, a hard embedded Microprocessor. All synchronous elements inside FPGA need to be driven by an outside clock signal. A clock tree, connect the clock signal to all the registers in the CLBs [6].

3.2.2 Characteristics of FPGA

FPGAs can be specified and compared using the following:

Number of Logic Cells (number of 4-input LUT's and associated flip-flop), Number (and size) of embedded RAM blocks, Number (and size) of embedded Multipliers, Number (and size) of embedded adders, Number (and size) of MACs, Availability of hardware embedded microprocessor cores, Number of I/O pins [6].

3.2.3 Uses of FPGA

As FPGA is a reconfigurable hardware and software control then it is used for various instrument design and control system. FPGAs can contain embedded Multipliers, dedicated arithmetic routines, large amount of on-chip RAM and with all these connected together it can outperform the fastest DSPs. FPGAs are becoming increasingly attractive for embedded control applications such as physical layer communications, FPGAs are used as a glue logic that interfaces the physical layers communication chips and high level networking protocols layers [6].

3.3 Introduction to VHDL

All the components of SCA have been designed by FPGA using VHDL, Xilinx ISE Design suite 9.2.

VHDL means: **VHDL = VHSIC Hardware Description Language & VHSIC = Very High Speed Integrated Circuit.**

VHDL was designed as a general hardware description and simulation language. It has a very complex syntax which includes also all kind of IO operations available on computer systems.
VHDL used for the programming of "Field Programmable Gate Arrays" (FPGA) uses only a subset of the complex VHDL syntax. This subset is called RTL (Register Transfer Logic). VHDL modules using this subset only can be placed and routed into a real hardware FPGA.
It was originally sponsored by the U.S. Department of Defense and later transferred to the IEEE (Institute of Electrical and Electronics Engineers). The language is formally defined by IEEE Standard 1076. The standard was ratified in 1987 (referred to as VHDL 87**)** and revised several times. We use a simple comparator to illustrate the selection of a VHDL program. The description uses only logical operators and represents a gate-level combinational circuit, which is composed of simple logic gates [7].

3.3.1 VHDL Constructs

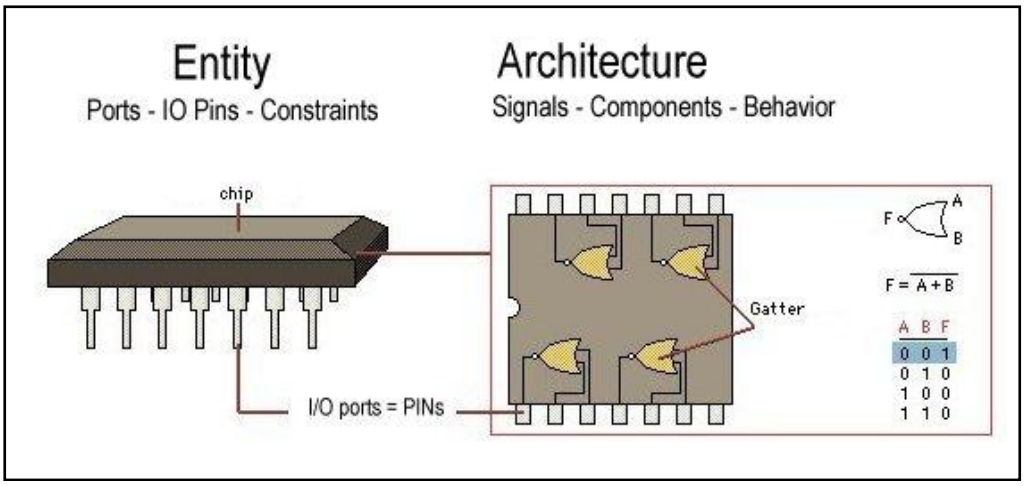


Fig. 3.4: Entity and Architecture of VHDL

3.3.2 Basic VHDL Programming

Entities

The entity describes the ports of the chip under design. The ports are the real-world pins, which connects the FPGA to the external hardware signals.

entity *Chip Name* is port
(
SignalName : IN / OUT / INOUT std_logic (_vector)
);
end *Chip Name*;

IN: An electrical signal comes from an external device into the FPGA

OUT: the FPGA drives a signal out to an external device

INOUT: the signal (data) on this line can flow in both directions.

This is mostly used to create a bus for connection to a Microprocessor.

Architectures

The architecture describes the behaviour of a certain chip. This is where we place the logic equations and where we "program" our chip.

architecture *ArchitectureName* of *ChipName* is
-- declare internal signals and components here
begin
-- describe the chip behaviour here

-- using Processes and logic equations

end *ArchitectureName*; [7]

3.3.3 FPGA Programming step

FPGA Programming steps are as follows:

- Translates register-transfer-level (RTL) design into gate-level netlist
- Restrictions on coding style for RTL model
- Place the required logic in the CLBs
- Generate a programming file

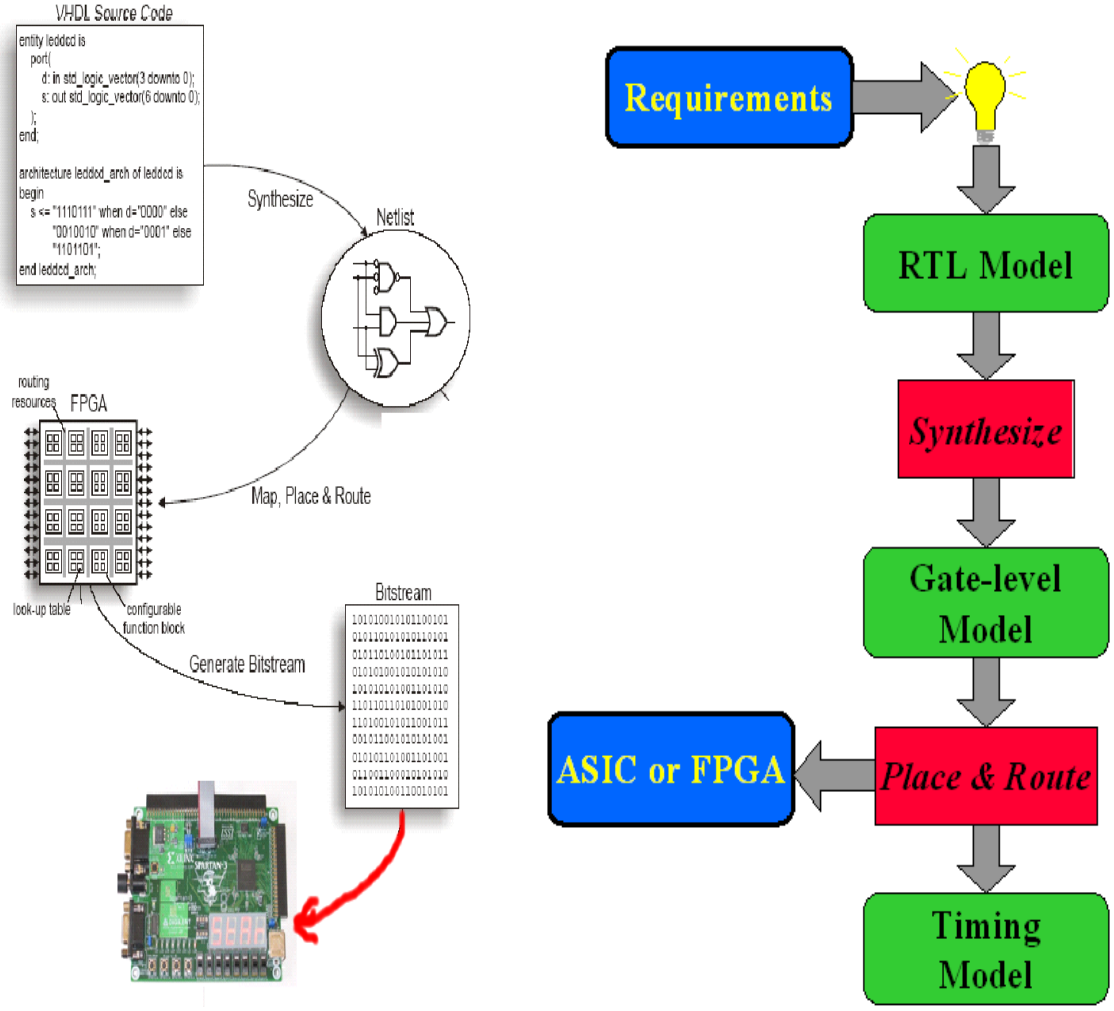


Fig. 3.5: View of FPGA Programming step

All the units of SCA system has been designed by FPGA using VHDL. These units were described in VHDL-modules and synthesized by Xilinx ISE Design suite 9.2. In VHDL designs for user

constraints have to mention the real location of the used hardware components. After simulation design summery and I/O Pin Planning RTL schematic design of the system is generated. Through ISE iMPACT process [Boundary Scan] is completed and finally the design has been implemented on Xilinx Spartan-3E Starter board.

3.4 Conclusion

From this chapter we have learnt about FPGA, VHDL programming and use of Xilinx ISE Design suite 9.2 for FPGA programming and we are going to apply this knowledge for developing VHDL code of FPGA based SCA.

4.1 Introduction

This chapter explains on the different section of FPGA based nuclear counting system, software development and schematic design after simulation.

4.2 Design Scenario for FPGA based Nuclear Counting System

Nuclear counting system is used to detect and monitor radiation level. This system includes detector, preamplifier, High Voltage Power System (HVPS) and FPGA based SCA section. A detector is Geiger Muller (GM) tube having a thin end window (e.g. made of mica), a high voltage supply for the tube, a preamplifier to amplify the electrical pulses which detected by the GM tube. Our work consists of designing and developing the part of the system enclosed by the inner rectangle in Fig. 4.1. In this design Gain amplifier and Analog to Digital Converter (ADC) again amplify those pulses which come from preamplifier and shape for discriminator. Discriminator has a function to discriminate the analog incoming pulses, which comes from the amplifier. The discriminator will also produce a TTL logic signal, when the incoming pulse fulfills the energy range criteria, which is defined by the user selectable lower and upper level. Counters are used for counting the logic signal from the discriminator for certain interval time (counting time). User sets the counting time through the timer in order of seconds, minutes or hours.

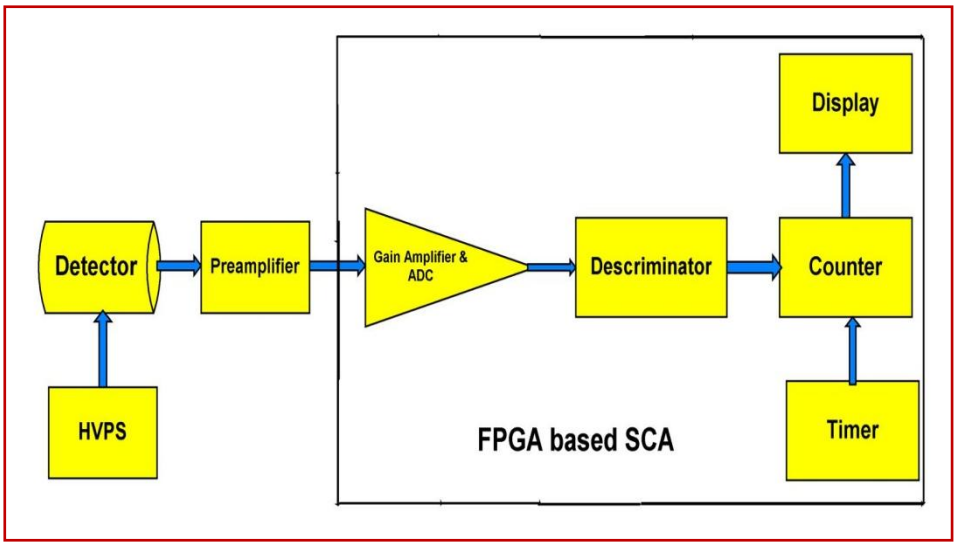


Fig. 4.1: Block diagram of the FPGA based Nuclear Counting System

Specifications of the developed FPGA based SCA systems are as follows.

Table 4.1 Specification of the developed FPGA based SCA system

Sl. No.	Components name	Quantity	Description
1.	Detector (GM Tube)	1	Halogen GAS Model 712
2.	HVPS	1	550 Volt
3.	Preamplifier	1	From Gm counter circuit
4.	Xilinx FPGA	1	XC3S500E FG-320Spartan-3E FPGA ♦ Up to 232 user-I/O pins ♦ 320-pin FPGA package ♦ Over 10,000 logic cells
5.	ADC	1	LT1407A
6.	Programmable-gain amplifier	1	LTC6912
7.	Clock oscillator	1	50 MHz Oscillator CLK_50MHz: (C9)
8.	LCD	1	Character LCD
9.	LED	8	Eight discrete LEDs

4.3 Setting of High Voltage of the Detector

High Voltage is a vital part of Nuclear Counting System. High Voltage Power Supply (HVPS) is used for the detector (Geiger Muller Tube) which is adjusted to get better detector performance. By plateau measurement the voltage is varied from 400 V to 600 V in step of 50 V with the source kept at a distance of 18 cm from detector. Three counts have been taken for different voltages which are shown next page in Table 4.2. Fig. 4.2 shows and plateau measurement curve drawn using the average count per minute (CPM) data for different voltages, taken from Table 4.2 in next page. From the table and Graph in the next page it is shown that from 500 to 600 Volts we get higher counts 56 and 57 CPM respectively. There is very little difference between two counts. As we can get the

better performance (counts) at 550 volts than 600 volts so it will be set as the detector voltage. Because at lower voltage and higher performance is better for SCA system and will also consume low power for the system.

Table: 4.2 Detector (GM Tube) Plateau measurement for different voltage

No. of obs.	Applied Voltage(Volt)	Count 1 (CPM)	Count 2 (CPM)	Count 3 (CPM)	Average Count per min (CPM)
1	400	42	47	51	46.67
2	450	46	50	59	51.67
3	500	54	58	50	54
4	550	53	59	56	56
5	600	52	56	63	57

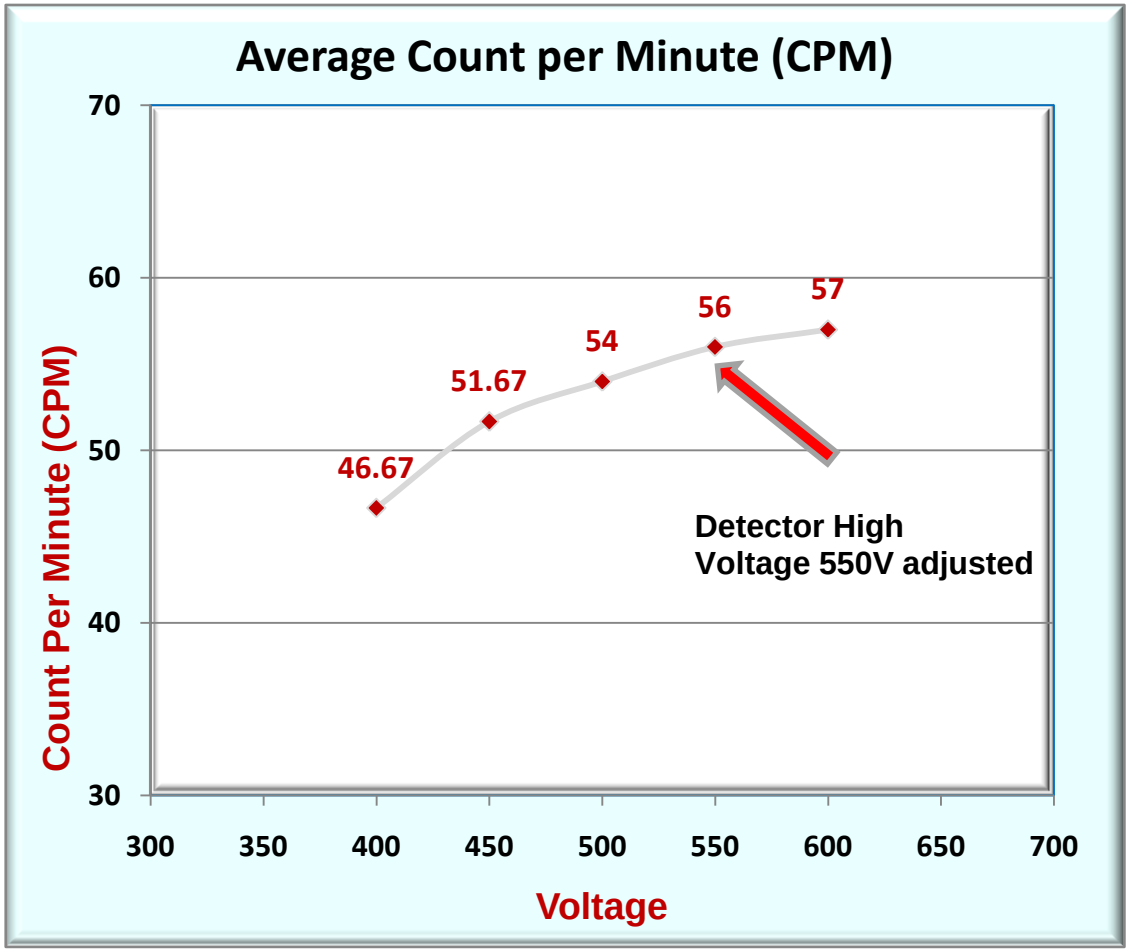
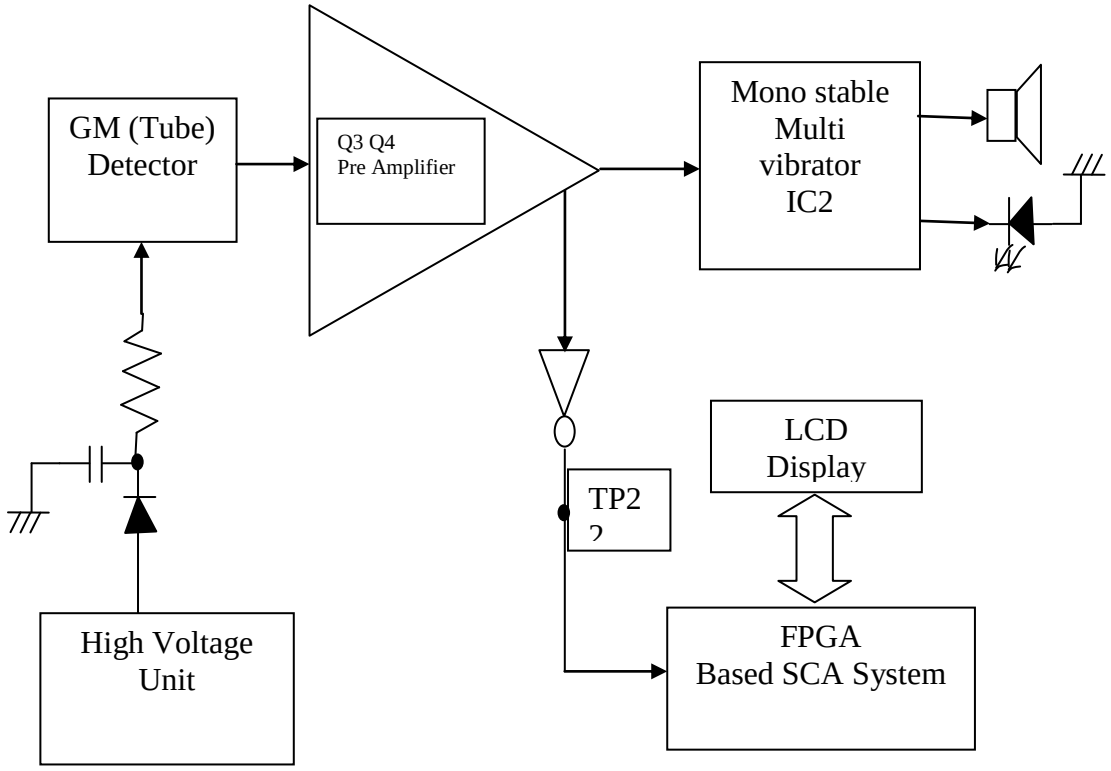


Fig. 4.2: GM Tube Plateau measurement curve for setting High Voltage of detector

As can be seen in the Fig. 4.2 at 550 volt we get best counts. Therefore in this design high voltage is adjusted at 550 for better performance of GM Tube (Halogen GAS Model 712).

Fig. 4.3 shows the block diagram of GM Counter. Preamplifier of Nuclear Counting system has two functions, shaping and amplifying the electric pulses from detector. The peak of exponential pulses from detector is too sharp to be measured or distinguished and the tail is too long. So, they have to be shaped as Gaussian pulses, which are more flat at the peak and do not have such a long tail. In our design the output of preamplifier at Test Point (TP2) which is 5 volt is collected from GM counter Fig.4.3 and this voltage is processed into 1.6 volt at processing circuit then fed to ADC of FPGA based SCA system.



Block diagram of GM Counter (Model – 924)

4.4 The proposed FPGA based SCA system

This section present a description of the various components of the proposed FPGA based SCA system as shown in Fig. 4.4. In this design gain amplifier and ADC are configured by FPGA. This communication is Serial Peripheral Interface (SPI) which connects the FPGA to

major external devices, gain amplifier and ADC. The other components in the developed system have been designed by FPGA using VHDL code.

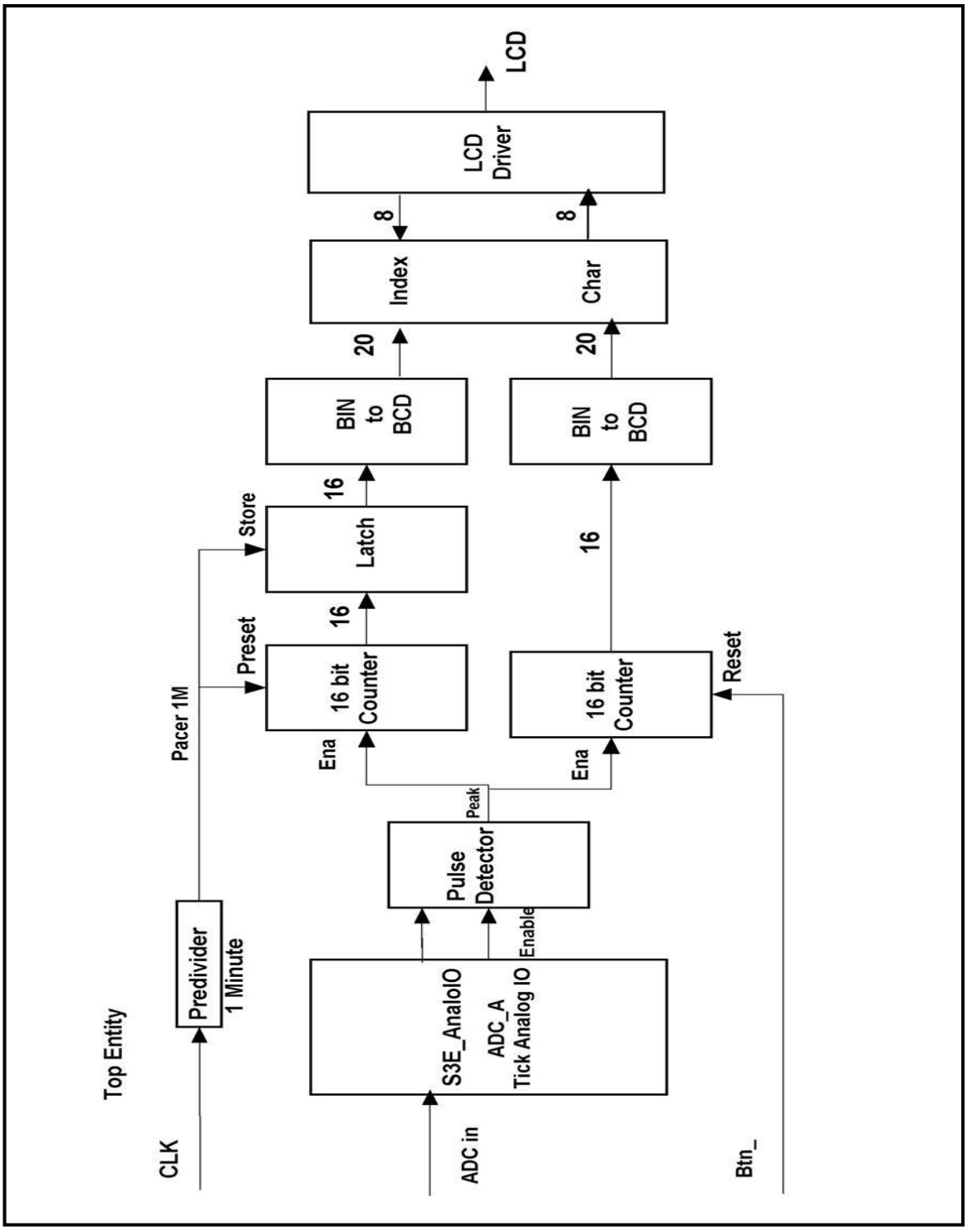


Fig. 4.4: Block diagram of FPGA based SCA system

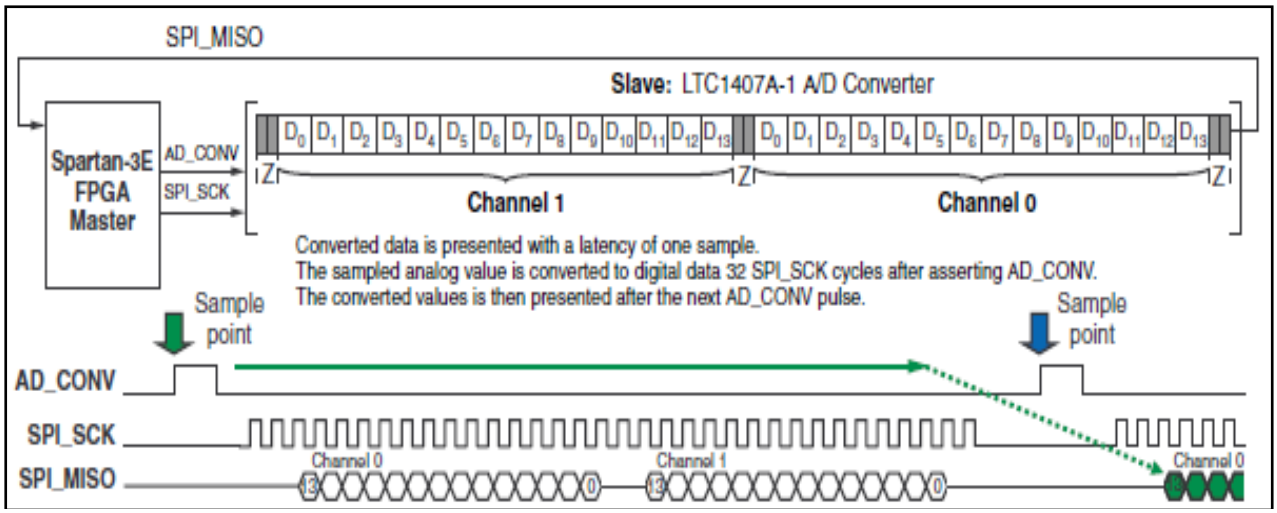


Fig. 4.5: Analog-to-Digital Conversion Interface [9]

4.4.1 Gain Amplifier and ADC

The AD_CONV signal is not a traditional SPI slave select enable. Enough SPI_SCK clock cycles has to be provided so that the ADC leaves the SPI_MISO signal in the high-impedance state. The ADC 3-states its data output for two clock cycles before and after each 14-bit data transfer. Table: 4.3 lists the interface signals between the FPGA and the amplifier. The SPI_MOSI and SPI_SCK signals are shared with other devices on the SPI bus. The AMP_CS signal is the active-Low slave select input to the amplifier [9]. Above Fig. 4.5 shows the details Analog-to-Digital Conversion Interface.

Table: 4.3 Amplifier interfacing signals [9]

Signal	FPGA Pin	Direction	Description
SPI_MOSI	T4	FPGA → AD	Serial data: Master Output, Slave Input. Presents 8-bit programmable gain settings.
AMP_CS	N7	FPGA → AMP	Active-Low chip-select. The amplifier gain is set. When signal returns High.
SPI_SCK	U16	FPGA → AMP	Clock
AMP_SHDN	P7	FPGA → AMP	Active-High shutdown, reset
AMP_DOUT	E18	FPGA → AMP	Serial data. Echoes previous amplifier gain settings. Can be ignored in most applications.

The AD_CONV, SPI_MISO, and SPI_SCK signals are the bus interface signals between the FPGA, ADC and the gain amplifier shown in Table: 4.4. When the AD_CONV signal goes high, the ADC simultaneously samples both analog channels. The results of this conversion are not presented until the next time AD_CONV is asserted, a latency of one sample. The maxim sample rate is approximately 1.5 MHz. The ADC presents the digital representation of the sampled analog values as a 14-bit, two's complement binary value [9].

Table: 4.4 ADC interfacing signals [9]

Signal	FPGA Pin	Direction	Description
SPI_SCK	U16	FPGA → AMP	Clock
AD_CONV	P11	FPGA → ADC	Active-High shutdown, reset
SPI_MISO	N10	FPGA → ADC	Serial data: Master Input, Serial Output. Presents the digital representation of the sample analog values as two 14-bit two's complement binary values.

4.4.2 Discriminator

When the ADC output value is between higher than lower threshold value Lower Level Detection (LLD) and lower than higher threshold value Upper Level Detection (ULD), then pulse detector gives the peak found signal to the counter to increase the count value. In this design ULD and LLD is set LLD = 800 mV and ULD = 1600 mV respectively.

4.4.3 Counter

When pulse detector finds peak, it provides a peak found signal to the counter and as a result, count value increases. Two 16 bit counters are used in counter circuit. One of the counters counts over a period of one minute and stores the counting value in register and another one is used for total count.

4.4.4 Timer

Spartan 3E, Starter board includes a 50MHz oscillator with a 40% to 60% output duty cycle. In this design 16 bit Counter is used for count pulse and data held in Latch [9].

4.4.5 Display

Finally the stored counting values are given to LCD through Latch and BIN to BCD counter. In addition, maximum peak value, total counts and counts per minute are also displayed to LCD through LCD driver circuit.

4.5 Software description

Associate firmware of the SCA system has been developed by Xilinx ISE Design suite 9.2 using VHDL code and tested on Xilinx Spartan 3E Starter board.

4.5.1 View of RTL Schematic design of FPGA based SCA system

The RTL schematic with all entities and components of SCA design is shown in the following Fig. 4.6 is generated after simulation in VHDL at Xilinx ISE Design suite 9.2.

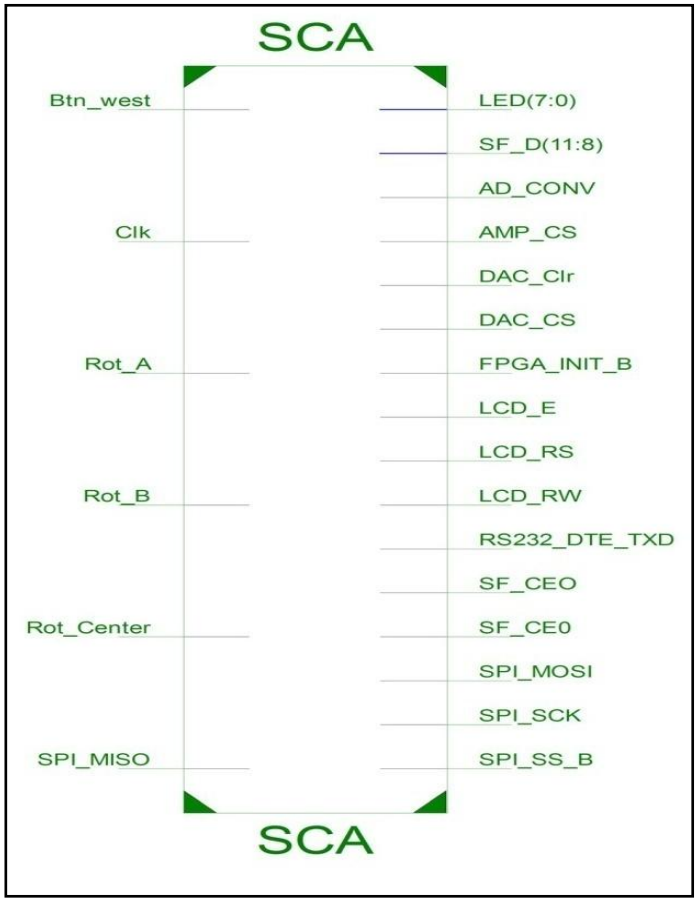


Fig. 4.6: RTL Schematic after simulation

4.5.2 Flow diagram of VHDL code of FPGA based SCA system

The next page in Fig. 4.7 shows the flow diagram of FPGA based SCA system. In VHDL programming, the first step is to declare library. In our design IEEE, Arithmetic & Un-sign libraries have been declared. In the Next step, entity for different ports as input, output, signal & its type of Analog IO, Pre divider and peak detector, Counter, Latch and Bin to BCD Counter are declared. Other devices connected to SPI bus should be disabled during SPI communication. Only communication is done between FPGA to ADC and gain amplifier. Then different processes for Analog IO, Pre divider, peak detector, Counter, Latch and Bin to BCD are called within the main program.

Clock process: 50MHz clock frequency is used in Xilinx Starter board which is very fast then it is divided into 25MHz for decreasing execution speed. 50MHz is pre divided into 1 sec for reset counter through Pre divider process. Analog IO process: when Amplifier chip select is low and on the clock rising edge amplifier capture data on SPI MOSI then 32 bit digital data is transfer at ADC output. In pPeak Process, when pulse detector finds peak, it provides a peak found signal to the counter. Counter counts the peak signals of one minute during pCount Process to get rate of counting (CPM) and this count is hold in latch during latch process. To get total number of pulse during on condition of system, Process Total Count is used. For binary to BCD representation pBinBCD, Tot Count Process and finally display on LCD pLCD Process is developed.

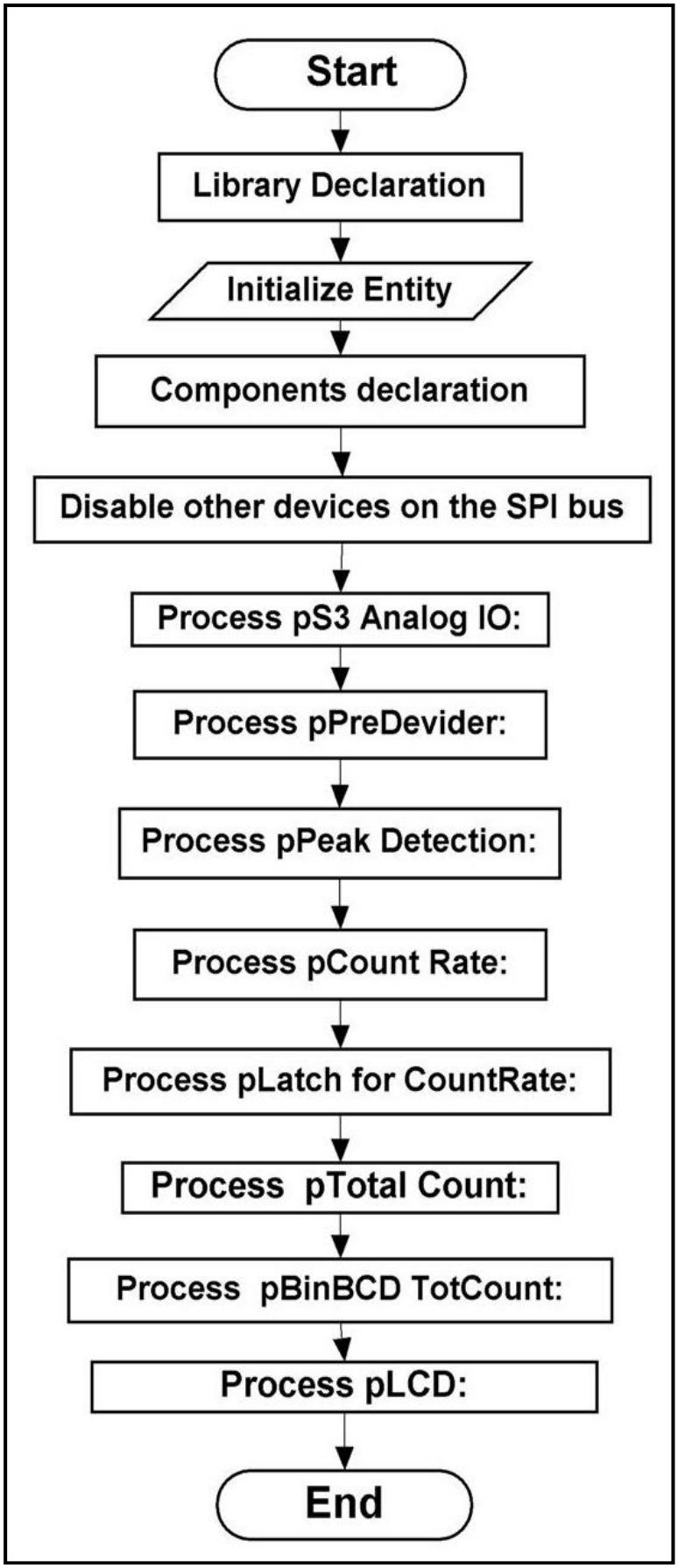


Fig. 4.7: Flow diagram of VHDL code of FPGA based SCA system

4.5.3 Schematic design of FPGA based SCA

The schematic designed of FPGA based SCA is automatically generated by VHDL at Xilinx ISE Design suite 9.2 has been shown in Fig. 4.8

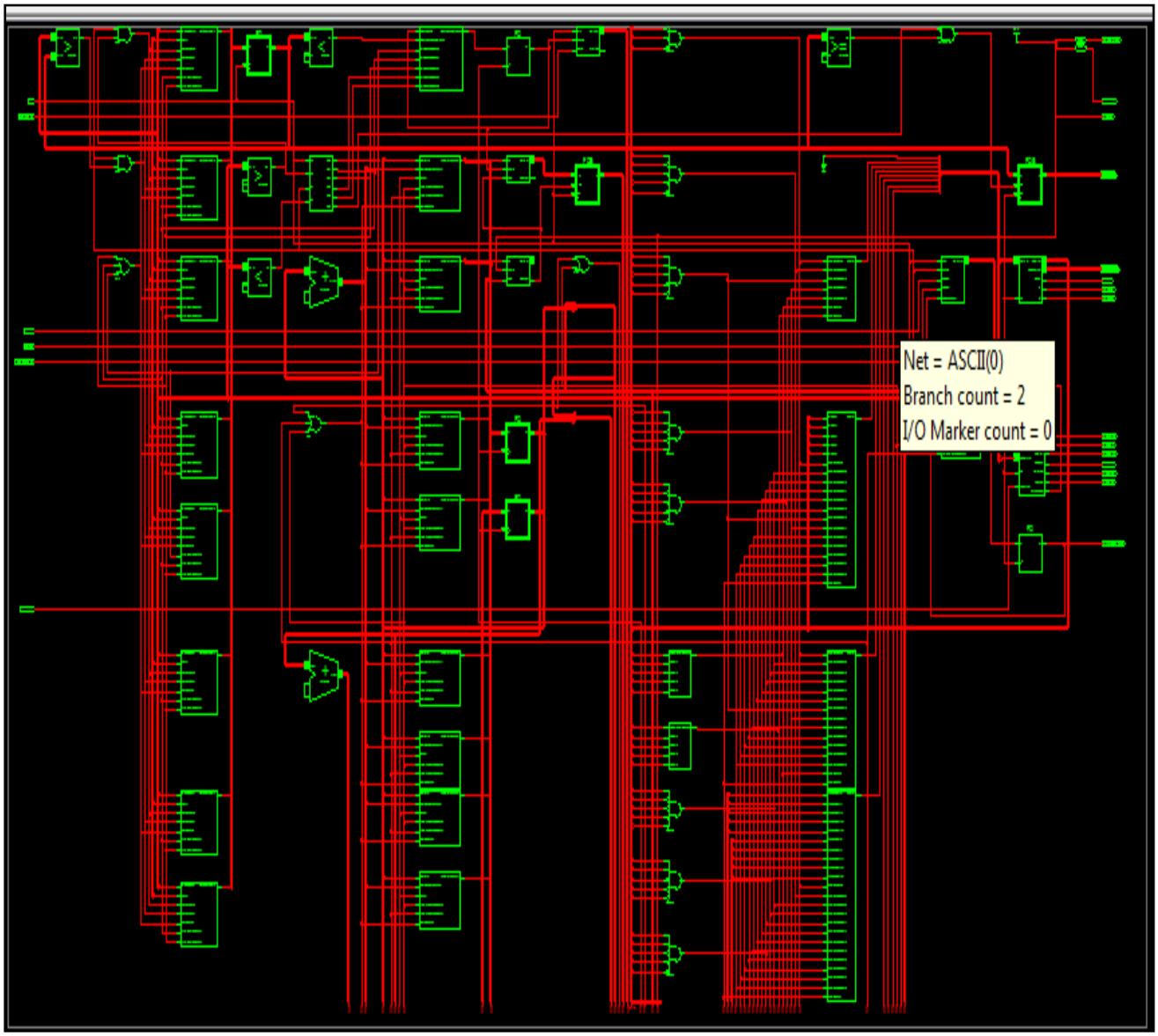


Fig. 4.8: Schematic designed of FPGA based SCA

4.5.4 Design Summery of SCA system

The design summery of FPGA based SCA has been automatically created after simulation**.** There are three parts in design summery, 1st part is details of SCA project status, 2nd is partition summery and last part is detailed description of device utilization where total number of flipflop, look up tables (LUT), slice and logic distribution are explained below.

SCA Project Status			
Project File:	SCA.ise	Current State:	Placed and Routed
Module Name:	SCA	• Errors:	No Errors
Target Device:	xc3s500e-4fg320	• Warnings:	No Warnings
Product Version:	ISE 9.2i	• Updated:	Thu Oct 16 15:02:17 2014

SCA Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	620	9,312	6%	
Number of 4 input LUTs	966	9,312	10%	
Logic Distribution				
Number of occupied Slices	773	4,656	16%	
Number of Slices containing only related logic	773	773	100%	
Number of Slices containing unrelated logic	0	773	0%	
Total Number of 4 input LUTs	1,316	9,312	14%	
Number used as logic	966			
Number used as a route-thru	350			
Number of bonded IOBs	31	232	13%	
IOB Flip Flops	8			
Number of GCLKs	4	24	16%	
Number of MULT18X18SIOs	1	20	5%	

Fig. 4.9: Design Summery of SCA system after simulation

4.6 Conclusion

In this chapter firstly all part of developed FPGA based SCA system has been described then simulated schematic design, flow diagram of the developed VHDL code and design summery has been presented.

5.1 Introduction

This chapter discusses the results obtained when the system was tested. It starts with the signal obtained from the detector, ADC then peak detector of FPGA based SCA system and finally displays radiation counts in LCD. The results of the developed system compared with commercial system and also it shows the full development system.

5.2 Performance Evaluation

After every system design performance evaluation is necessary. For performance assessment of the developed FPGA based SCA system has been compared with commercial Survey Meter (GAMMA-SCOUT). A radioactive point source ^{137}Cs is used for getting result.

All types of ionizing radiation are controlled by three ways: Time, Distance and Shielding. Distance is a prime concern when dealing with gamma rays, because they can travel long distances. The farther away people are from a radiation source, the less is their exposure. It depends on the activity of the source and dose rate. In this work, distance parameter has been considered for measurement.

As radiation is harmful, some care and precaution should be taken while carrying out the experiment. After experiment the source must be kept in a well shielded container and placed the container in a safe distance. Tongs must be used always for handling radioactive source. In the time of experiment had to use a digital pocket dosimeter for observe personal dose. Radiation source must be kept away from the human body as possible. For the use of radioactive source we should follow the ALARA (**A**s **L**ow **A**s **R**easonably **A**chievable) principal. Following the above consideration performance study has been completed.

5.3 Experimental Setup

Block diagram of hardware setup has been shown in the following Fig. 5.1. For this setup radioactive point source Cs-137 is placed in front of detector (GM Tube) of GM Counter. Preamplifier output 5V (at point TP2) from GM detector is processed into 1.6 V and then fed to the input of ADC of FPGA based SCA. When the ADC output value is between LLD and ULD then counter counts those values over a period of one minute and stores the counting

value in register. Finally the stored counting values are given to LCD through other necessary circuits.

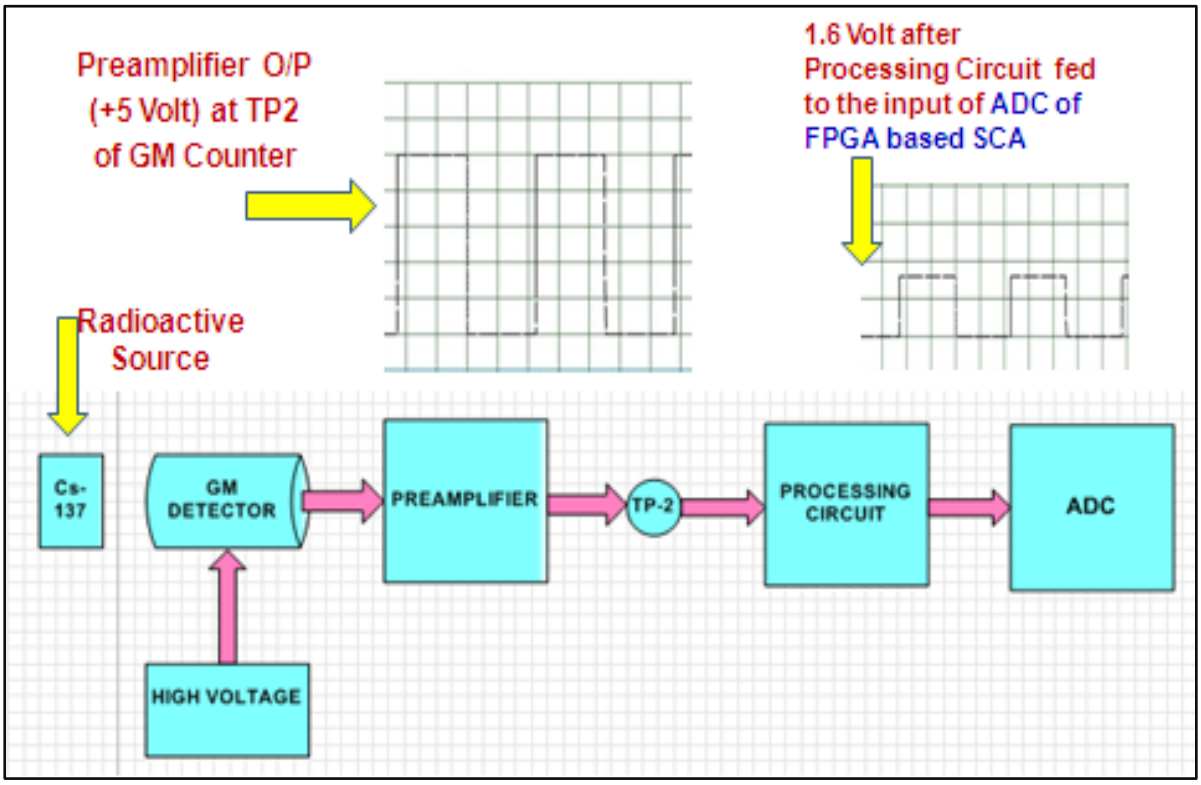


Fig. 5.1: Block diagram of Experiment Setup

Developed FPGA based SCA system is implemented in Xilinx Spartan 3E, Starter board has been shown in following Fig. 5.2. Radioactive point source ^{137}Cs (red box) is placed in front of detector at a distance of 18 cm from the detector and this distance is varied. Developed FPGA based SCA has been compare with commercial survey meter Gamma Spout (yellow colour). When radiation hits the glass window of detector then detector converts this radiation into electric pulses and gives the output of preamplifier then FPGA based SCA system and finally provides radiation counts at LCD display.

Preamplifier output at **TP2** point (5V) of GM Counter is shown in Fig. 5.2 which is fed to the input of ADC of FPGA based SCA through processing circuit. Output from the preamplifier is fed to ADC of FPGA based SCA through processing circuit. During the different stages of SCA and finally radiation counts has been displayed at LCD in CPM, Total count and Max value. Survey Meter (GAMMA-SCOUT) was placed at a fixed point and the distance of radioactive point source ^{137}Cs was varied in cm.

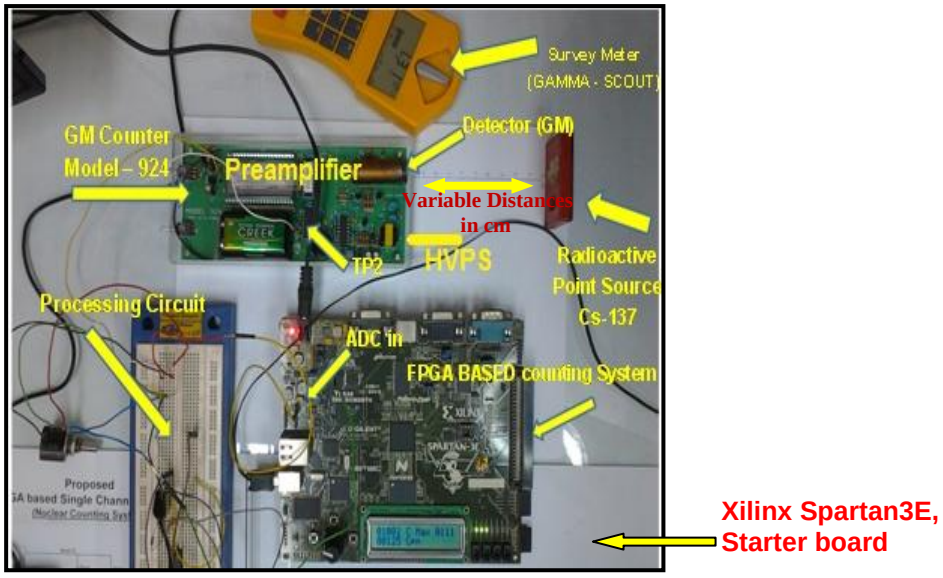


Fig. 5.2: Total System of Nuclear Counting System

5.4 Results

Table: 5.1 Comparison of developed FPGA based SCA system with commercial Survey meter

No. of Obs.	Distances in cm	Commercial Survey meter		Developed FPGA Based System CPM C_2	% Difference $\frac{C1 - C2}{C1} \times 100$	Standard Deviation $\sum(A_1^2 + A_2^2 + A_3^2 + \cdots)$
		μSv h^{-1}	in CPM (1 μS h^{-1} = 100CPM) C_1			
1.	18	0.56	56	57	-1.78571	3.248905
2.	16	0.61	61	63	-3.27868	
3.	14	0.93	93	93	0	
4.	10	1.28	128	125	2.34375	
5.	8	1.38	138	130	5.797101	
6.	6	1.68	168	165	1.785714	
7.	5	2.20	220	225	-2.27273	
8.	4	2.57	257	255	0.77821	
9.	3	2.90	290	300	-3.44828	
10.	2	3.49	349	349	0	

This system has been compared with other commercially system (Survey Meter GAMMA-SCOUT) considering distance in cm and uses gamma point source (^{137}Cs) as demonstrated in Fig. 5.2. The results are continuous changeable because the radiation intensity is random in time, following the Gaussian or normal distribution. So, if we carry out repetitive measurements of radiation intensity with the same condition, we will not get the same result. There will be a fluctuation between those values [13]

Table: 5.1 shows the radiation counts in one minute which are collected from the two systems, developed FPGA based SCA system and commercially available survey meter (Gamma Scout) for different distance of the source from the detector. From observation of table some radiation counts of commercial system is higher than developed FPGA based SCA system and some counts is lower. Finally Standard Deviation is 3.248905

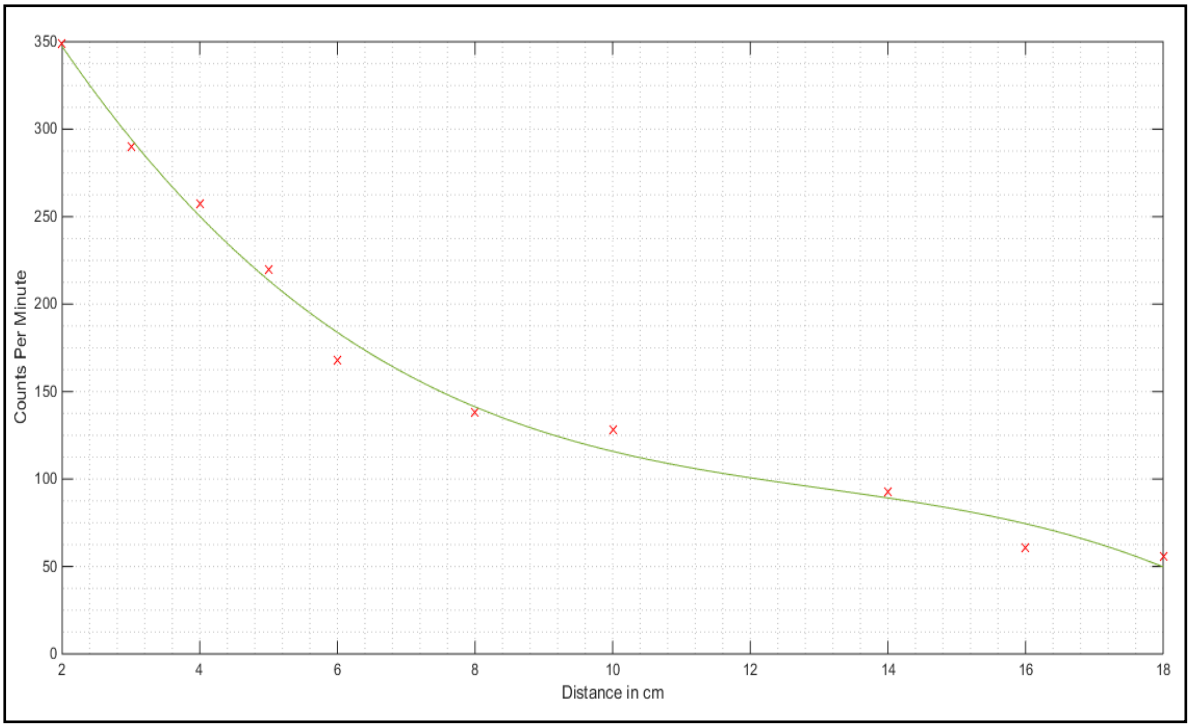


Fig. 5.3: Two results are compared and shown in chart

Fig. 5.3 shows the radiation counts obtained from two systems, the FPGA based Nuclear Counting System and Survey Meter (GAMMA-SCOUT), in cpm. The survey meter gives data in µSv h^{-1} which is converted in cpm for the convenience of comparison. FPGA system is showing almost similar result with commercial system. For fluctuating results, it is recommended that for low range activity, average of the maximum and minimum

radiation count is acceptable. I have used the cubic fitting equation for the two counts in Fig.5.3.

$$Y = P1 * X^3 + P2 * X^2 + P3 * X + P4 \quad \text{.....5.1}$$

The deviation is the measurement data as obtained from the two methods is also shown in Table 5.1 as % difference which is calculated as,

$$\%\ \text{Difference} = \frac{\text{Commercial System} - \text{Develope System}}{\text{Commercial System}} \times 100\% \text{.....} \quad 5.2$$

For most cases deviation is within 3% with a standard deviation of

$$\text{Standard Deviation} = \sqrt{\frac{1}{N} \sum (A_1^2 + A_2^2 + A_3^2 + \cdots)} \quad \text{.....} \quad 5.3$$

5.5 Discussion

In this work has given attention to the design, simulation and implementation of FPGA based Nuclear Counting System. To do this work, it is observed that radiation counts are always changeable. Another difficulty faced in this work, is the unavailability of detector also during measurement of radiation at have to care of experiment time since it is harmful for human and environment. Except for these difficulties, as our design is FPGA based so the system has flexibility to configure hardware and it can replace complex analog nuclear counting circuitry.

5.6 Conclusion

The thesis is an implication of modern radiation monitoring system which is necessary for environment and creatures.

An FPGA based Single Channel Analyze system has been developed and tested for nuclear radiation counting. The designed FPGA based system has flexibility to configure hardware. In traditional system, SCA design needs individual circuit for amplifier, discriminator, counter and timer but in FPGA based system it is possible to design all these circuits in a single system as an integrated device. This FPGA based system can replace complex analog SCA circuitry.

Results have been compared for several times. Results of FPGA based system has been compared with the commercial Survey Meter and their results are approximately same. A radiation count normally varies at low range of activity. These counts are continuous changeable because the radiation intensity is random in time, following the Gaussian or normal distribution. So, if we carry out repetitive measurements of radiation intensity with the same condition, we will not get the same result. So, it is recommended that at low range of radiation count, average of the minimum and maximum count be taken. This developed nuclear radiation (especially gamma radiation) counting system may be used for diagnostic purposes in medical and research purpose in laboratory.

Future objective of this work is to develop detector circuitry including PC based data acquisition system through USB port using LabVIEW. Because of radiation hazards so many diseases occur and in the long run death. For growing awareness in the people about radiation, it is needed to develop facility available for radiation detection and monitoring. In order to meet the above requirements a precision, portable and fast FPGA based nuclear counting system should be designed and developed.

1. Xilinx UG230 Spartan-3E Starter Kit Board User Guide.
2. A. Ezzatpanah latifi1, f. Abbasi davani1**, m. Ahriari1 and a. Sharghi ido2, design and construction of an accurate timing single channel analyzer* Iranian Journal of Science & Technology, Transaction A, Vol. 33, No. A3, Islamic Republic of Iran, 2009
3. Design and Simulation of FPGAs Based Digital Multi Channel Analyzer for Nuclear Spectroscopy Application, Amitkumar Singh* S. K. Dubey M. G. Bhatia, Department of Physics Department of Physics, India University of Mumbai, India Ameya Centre of Robotics, Andheri, Mumbai, Volume 4, Issue 8, August 2014 ISSN: 2277 128X
4. Wolfgang Hennig, Hui Tan, William K Warburton, and Justin I McIntyre, Single Channel Beta-Gamma Coincidence Detection of Radioactive Xenon Using Digital Pulse Shape Analysis of Phoswich Detector Signals, J. I. McIntyre is with Pacific Northwest National Laboratory, Richland, WA 99352, USA, November 15, 2005.
5. Dudi Hendriyanto Haditjahyono, Introduction to The Nuclear Counting Systems, Education and Training Center – BATAN, October 2004
6. Dr. Heinz Rongen, VHDL Quick Start, Forschungszentrum Jülich Zentrallabor für Elektronik, ZEL
7. Dr. Heinz Rongen, Introduction to FPGA, Forschungszentrum Jülich Zentrallabor für Elektronik, ZEL
8. ISE 8.2i VHDL Quick Start Tutorial
9. Spartan-3E completes datasheet.
10. Volnei A. Pedroni, Circuit Design with VHDL.
11. LTC2624 Quad DAC Data Sheet
12. PicoBlaze Based D/A Converter Control for the Spartan-3E Starter Kit (Reference Design)
13. Xilinx PicoBlaze Soft Processor
14. Digilent, Inc. Peripheral Modules
15. Amplifier and A/D Converter Control for the Spartan-3E Starter Kit (Reference Design)
16. For an in-depth explanation of the ISE design tools, see the ISE In-Depth Tutorial on The Xilinx® web site at: http://www.xilinx.com/support/techsup/tutorials/
17. For more information about installing Xilinx® software, see the ISE Release Notes

And Installation Guide at: http://www.xilinx.com/support/software_manuals.htm.

18. For an in-depth explanation of the ISE design tools, see the ISE In-Depth Tutorial on the Xilinx® web site at: http://www.xilinx.com/support/techsup/tutorials/
19. G.F. Knoll, Radiation Detection and Measurement, 3rd edition, 2000.
20. Selected topics in nuclear electronics, a technical document issued by the International Atomic Energy Agency, Vienna, 1986.
21. http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1155,C1005,C1156,P2_048,D2170
22. http://www.xilinx.com/s3estarter
23. http://www.xilinx.com/picoblaze
24. http://www.digilentinc.com/Products/Catalog.cfm?Nav1=Products&Nav2=Peripheral&C at=Peripheral
25. http://www.xilinx.com/s3estarter
26. http://www.linear.com/pc/downloadDocument.do?navId=H0,C1,C1154,C1009,C1121,P7 596, D5359
27. http://www.xilinx.com/support/techsup/tutorials/
28. http://www.xilinx.com/support/techsup/tutorials/

Appendix A

Programing in Software Xilinx ISE design suite 9.2

To make a program takes just a few steps

- "Getting Started"
- "Create a New Project"
- "Create an HDL Source"
- "Design Simulation"
- "Create Timing Constraints"
- "Implement Design and Verify Constraints"
- "Reimplement Design and Verify Pin Locations"
- "Download Design to the Spartan™-3 Demo Board"

Getting Started

Software Requirements

To use this tutorial, must have to install the following software:

- ISE 9.2i

Hardware Requirements

To use this tutorial, must have the following hardware:

- Spartan-3E Startup Kit, containing the Spartan-3E Startup Kit Demo Board

Starting the ISE Software

To start ISE, double-click the desktop icon, or start ISE from the Start menu by selecting and shows the following view.

Fig. 1: View of Xilinx ISE

Start → All Programs → Xilinx ISE 9.2i → Project Navigator

Accessing Help At any time during the tutorial, can access online help for additional information about the ISE software and related tools.

To open Help, do either of the following:

- Press F1 to view Help for the specific tool or function that you have selected or highlighted.
- Launch the ISE Help Contents from the Help menu. It contains information about creating and maintaining your complete design flow in ISE and shows the following window.

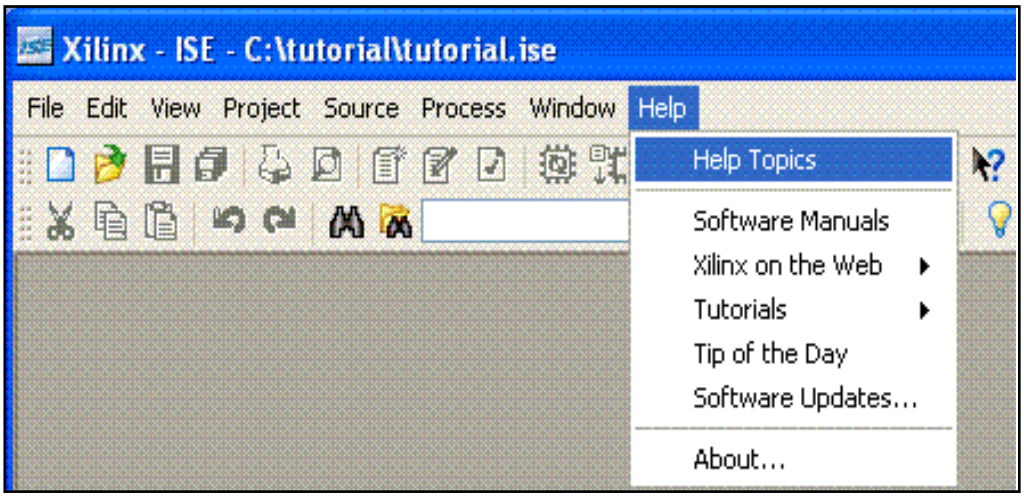


Fig. 2: View of ISE Help Contents Xilinx ISE

ISE Help Topics

Create a New Project

Create a new ISE project which will target the FPGA device on the Spartan-3 Startup Kit Demo board.

To create a new project:

1. Select **File** > **New Project...** The New Project Wizard appears.

2. Type **tutorial** in the Project Name field.

3. Enter or browse to a location (directory path) for the new project. A tutorial Subdirectory is created automatically.

4. Verify that **HDL** is selected from the Top-Level Source Type list.

5. Click **Next** to move to the device properties page.

6. Fill in the properties in the table as shown below:

- ♦ Product Category: **All**
- ♦ Family: **Spartan3E**

♦ Device: **XC3S500**

♦ Package: **FG320**

♦ Speed Grade: **-4**

♦ Top-Level Module Type: **HDL**

♦ Synthesis Tool: **XST (VHDL/Verilog)**

♦ Simulator: **ISE Simulator (VHDL/Verilog)**

♦ Verify that **Enable Enhanced Design Summary** is selected.

Leave the default values in the remaining fields.

When the table is complete, project properties will look like the following Fig. 3.

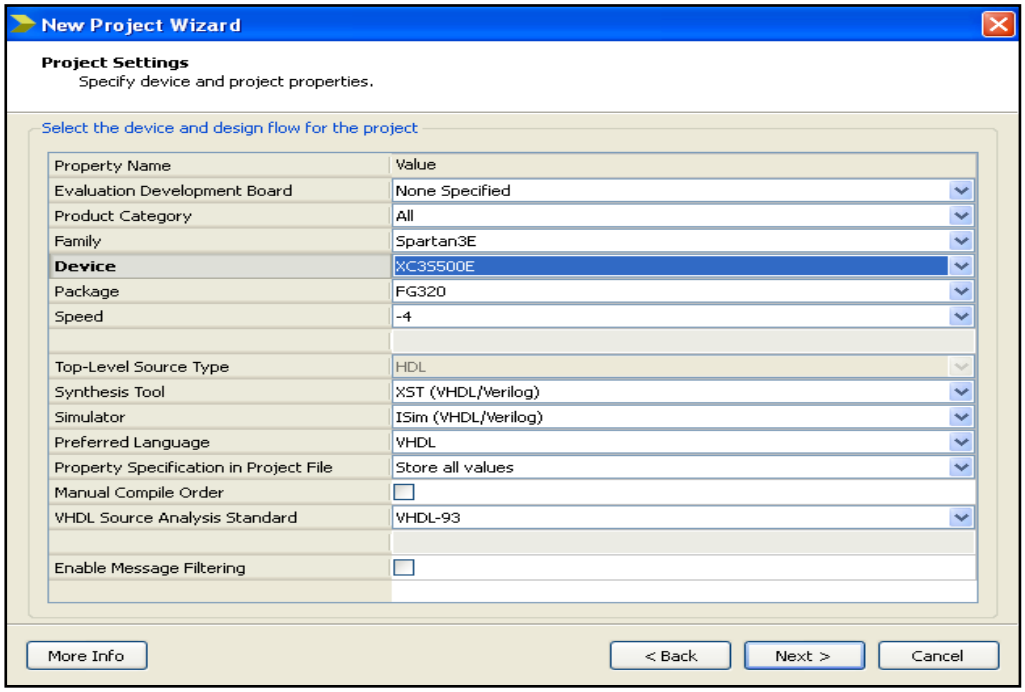


Fig. 3: View of Xilinx ISE Project Device Properties

7. Click **Next** to proceed to the Create New Source window in the New Project Wizard. At the end of the next section, new project will be complete.

Create an HDL Source

In this section, will create the top-level HDL file for design. Determine the language that wishes to use for the tutorial. Then, continue either to the "Creating a VHDL Source" section below, or skip to the "Creating a Verilog Source" section.

Creating a VHDL Source

Create a VHDL source file for the project as follows:

1. Click the **New Source** button in the New Project Wizard.

2. Select **VHDL Module** as the source type.

3. Type in the file name **AND_gate**.

4. Verify that the **Add to project** checkbox is selected.

5. Click **Next**.

6. Declare the ports for the counter design by filling in the port information as shown below:

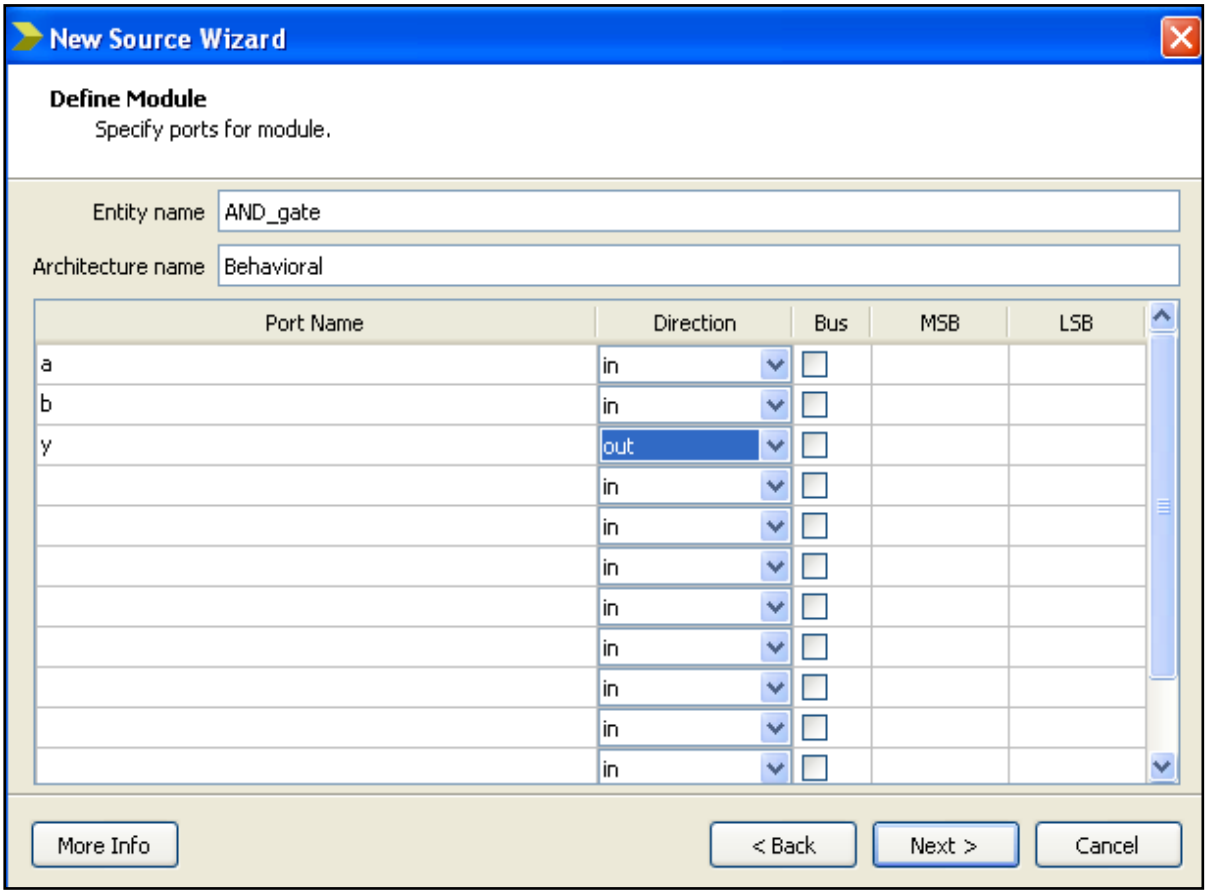


Fig. 4: View of port information of Xilinx ISE

Define Module

7. Click **Next**, then **Finish** in the New Source Information dialog box to complete the new source file template.

8. Click **Next**, then **Next**, then **Finish**.

The source file containing the entity/architecture pair displays in the Workspace, and the **AND_gate** displays in the Source tab, as shown below in Fig. 5.

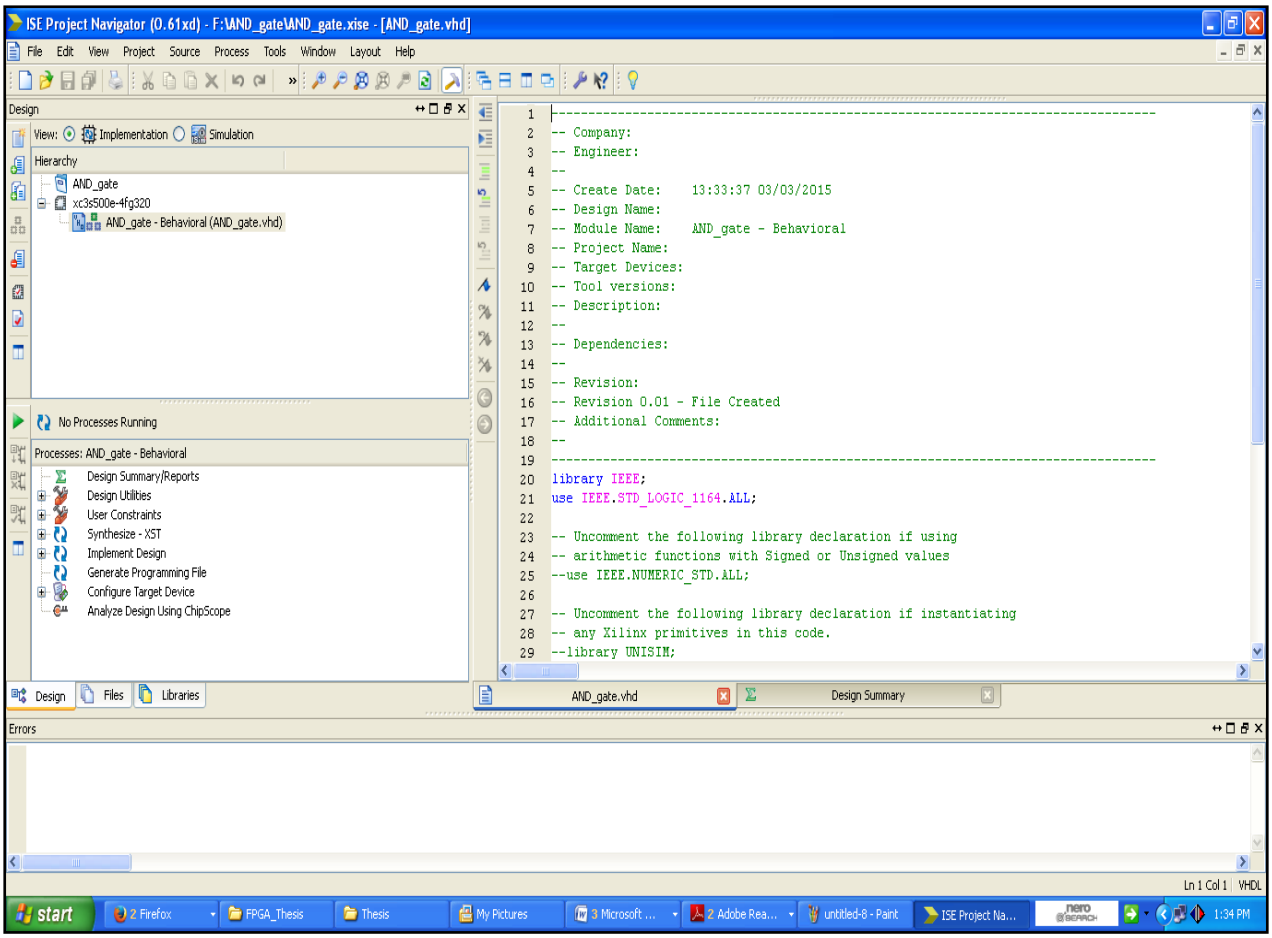
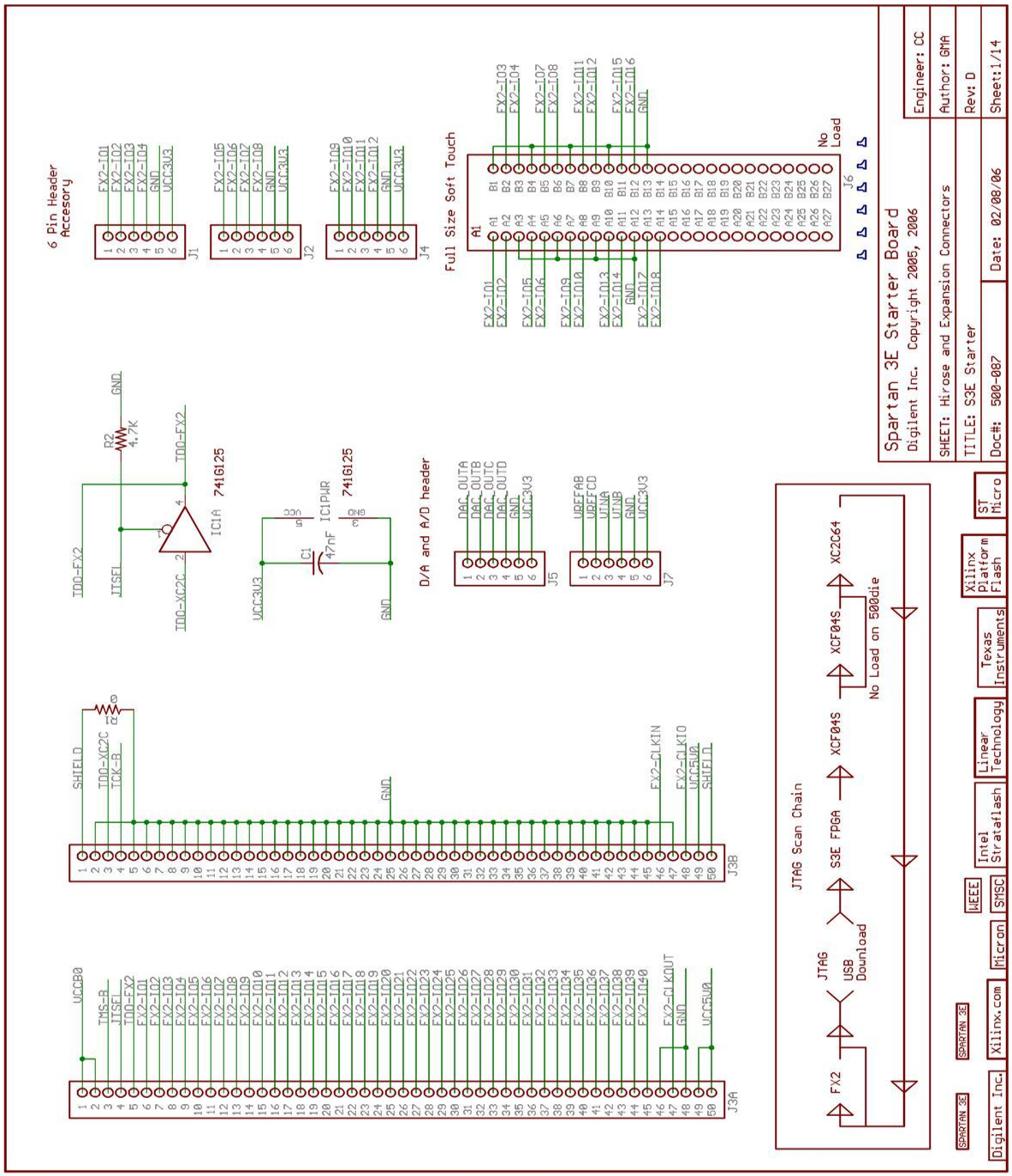


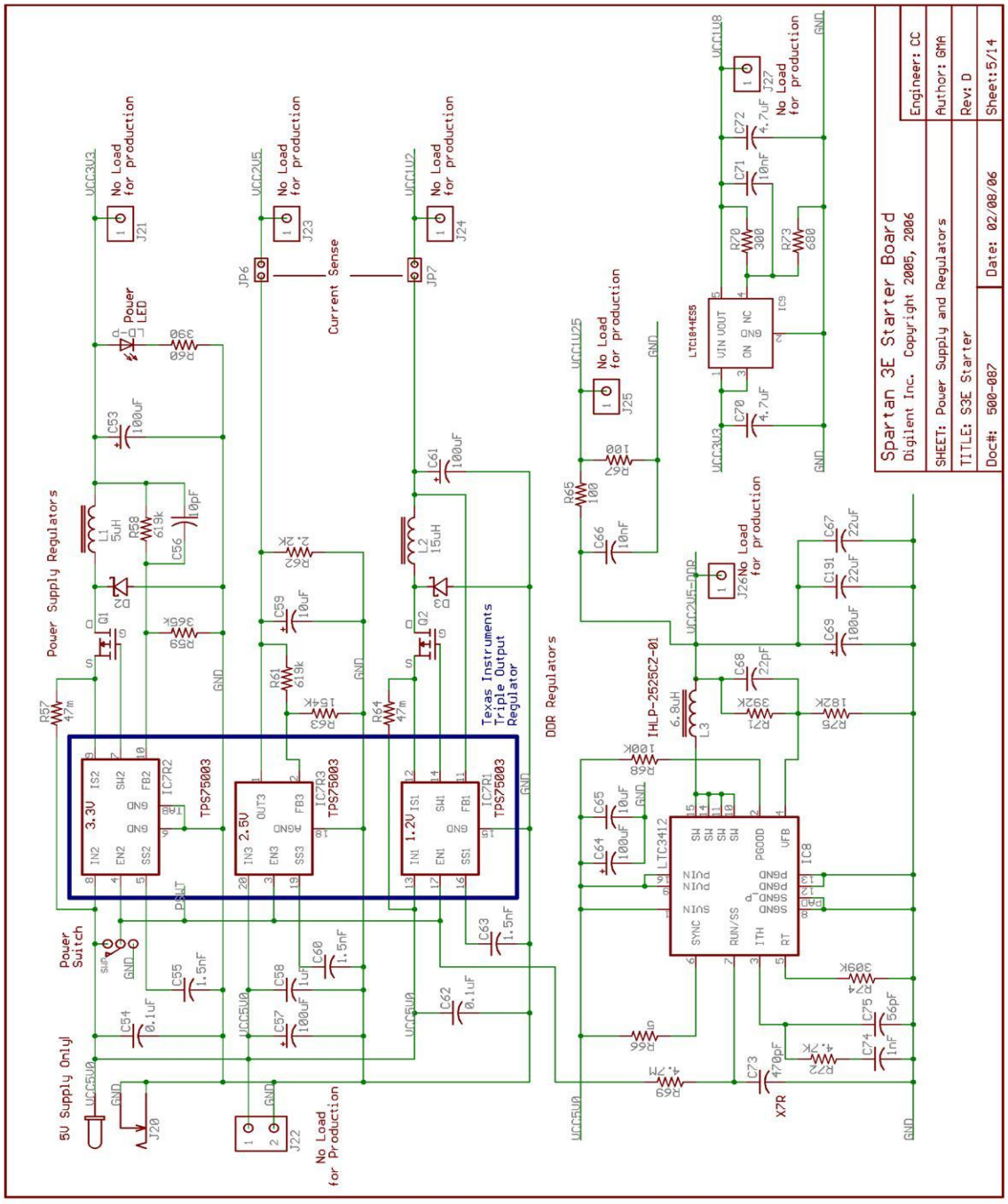
Fig. 5: View of New Project in ISE

Appendix B

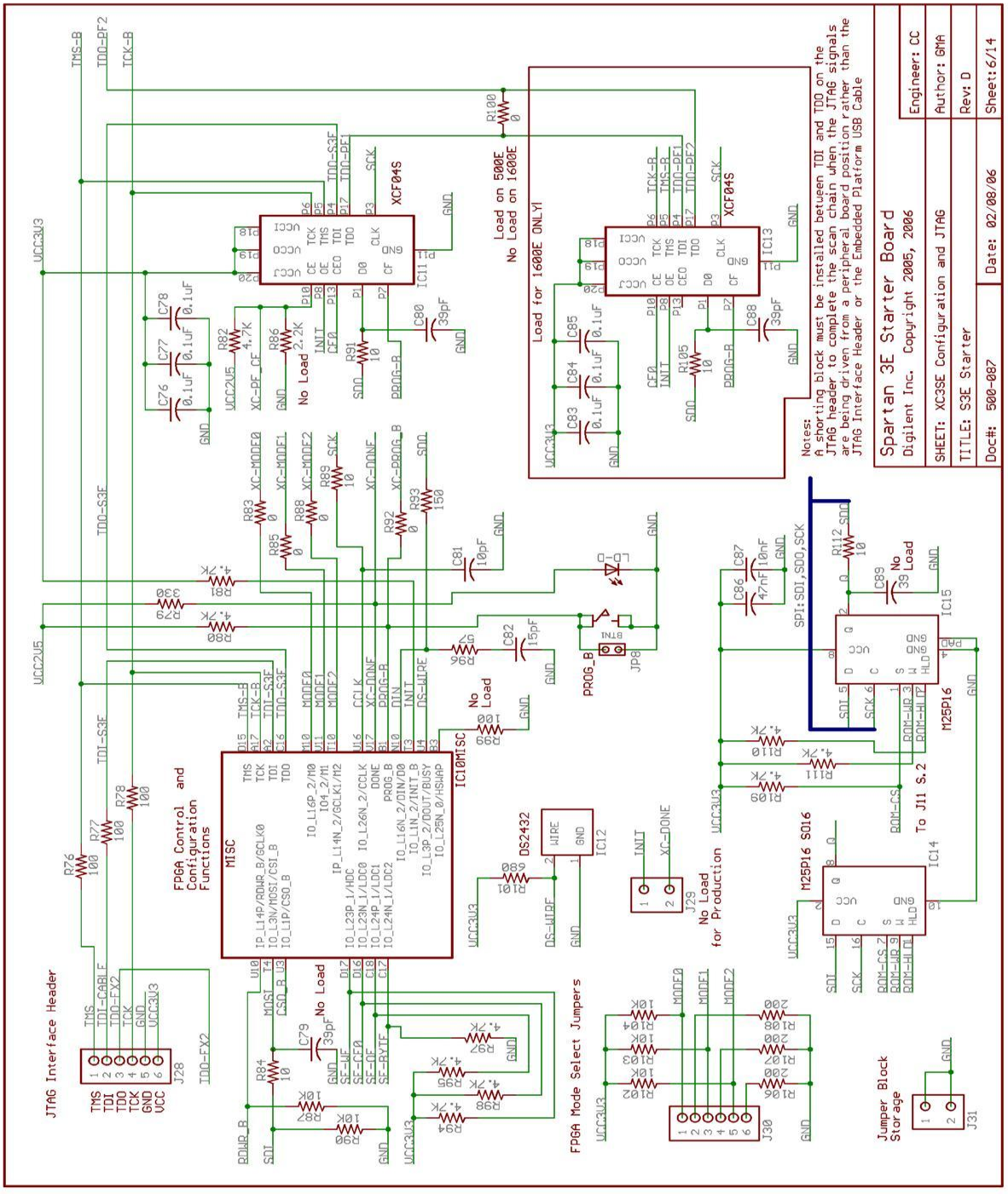
FX2 Expansion Header, 6-pin Headers, and Connector less Probe Header



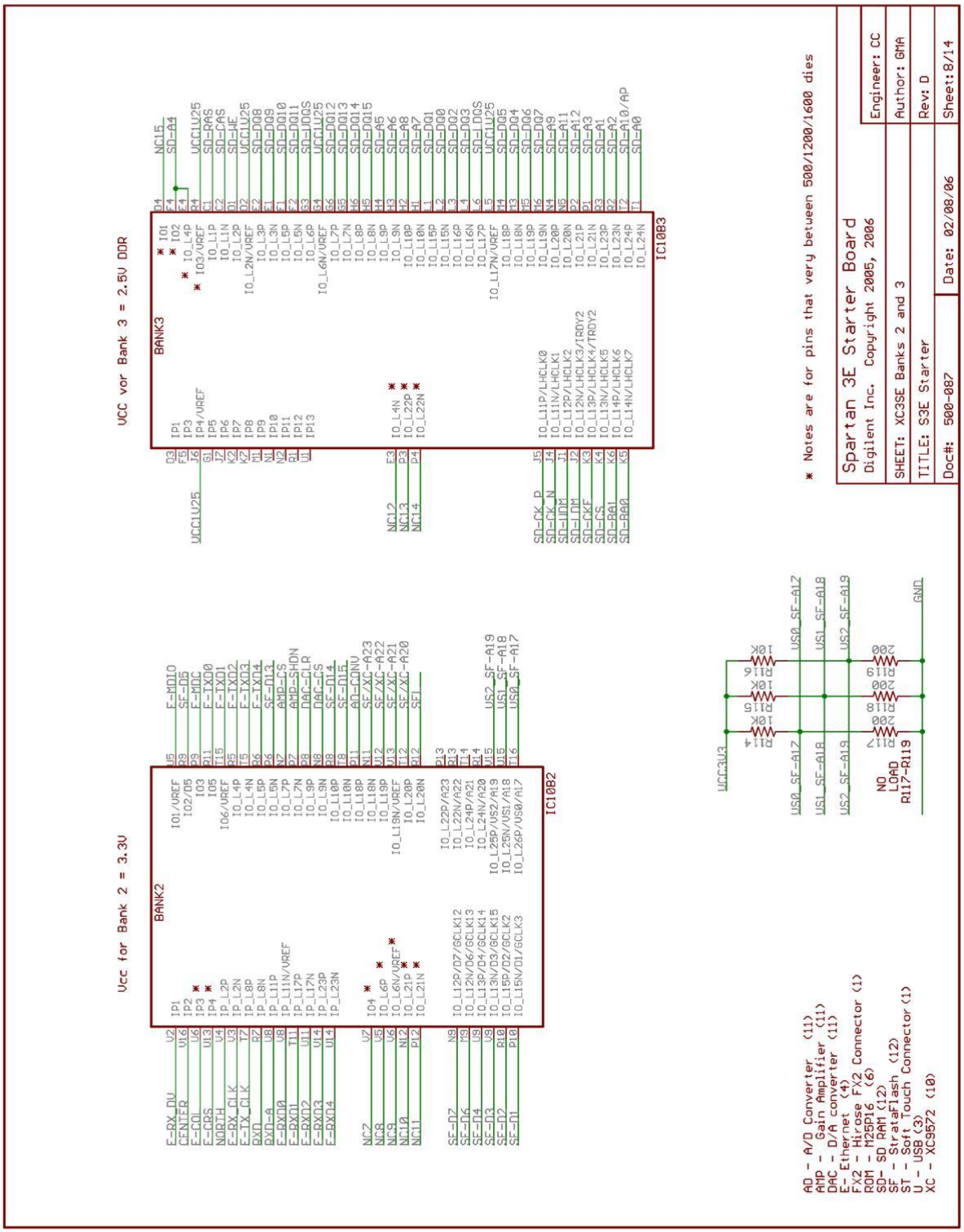
Voltage Regulators



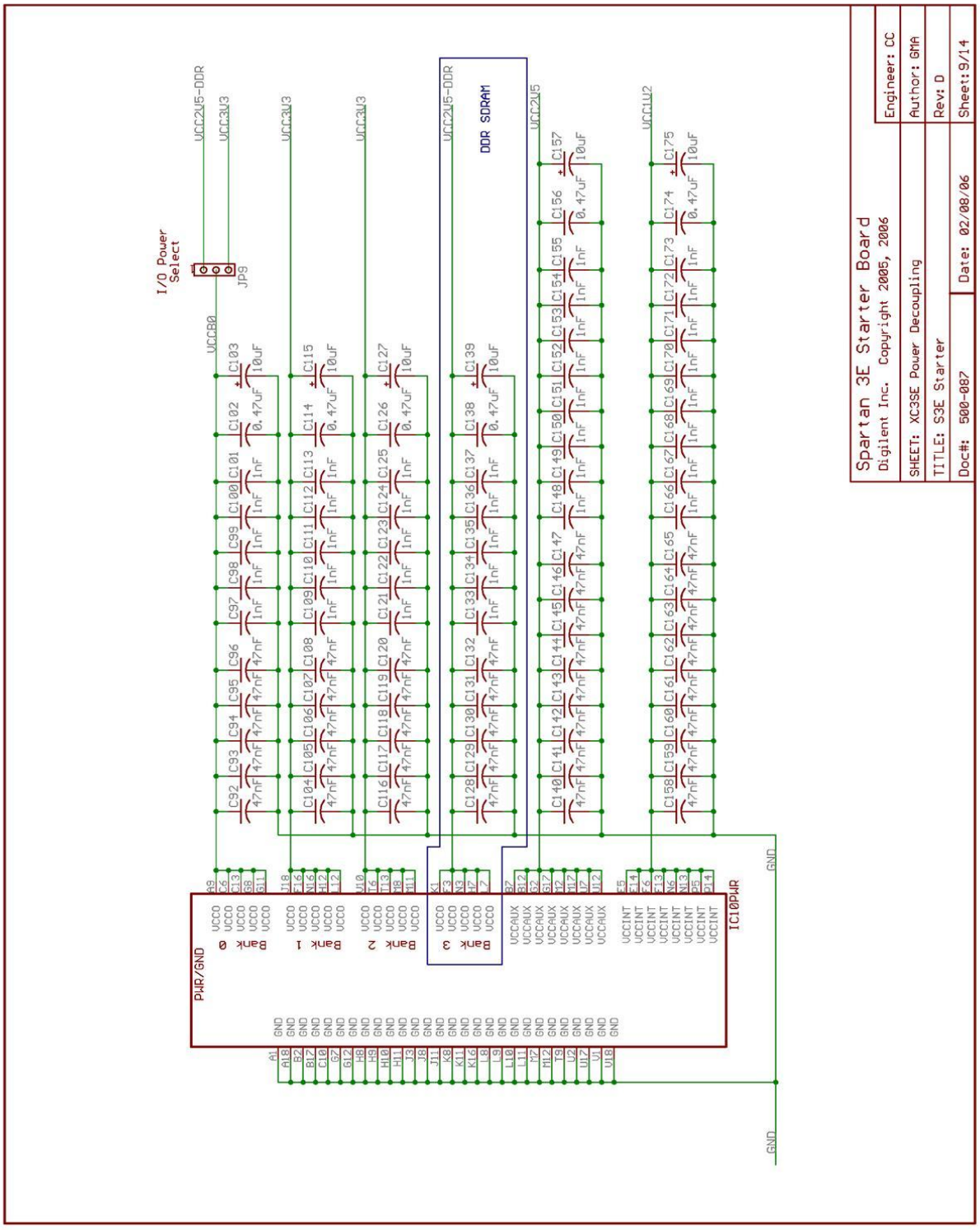
FPGA Configurations Settings, Platform Flash PROM, SPI Serial Flash, JTAG Connections



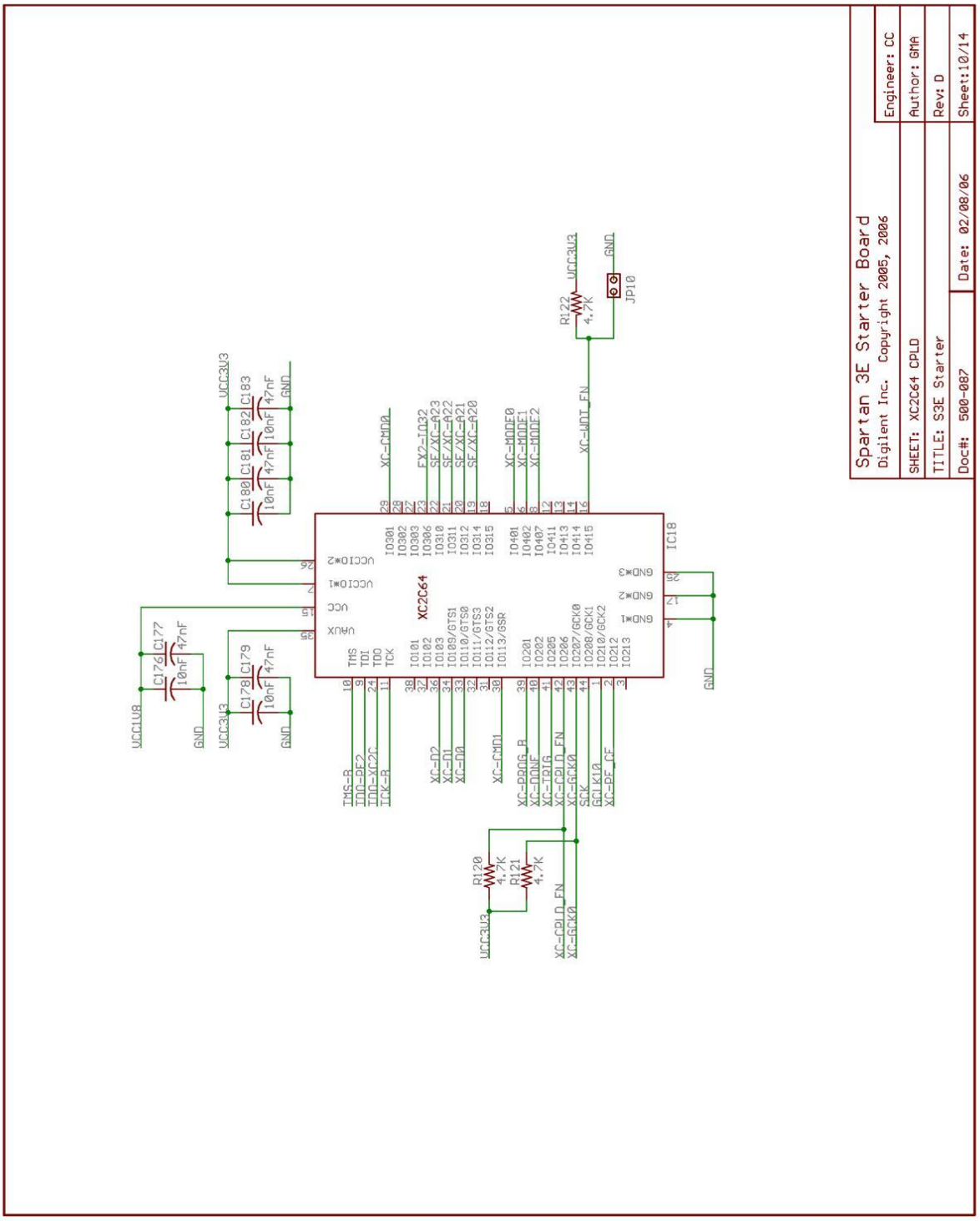
FPGA I/O Banks 2 and 3



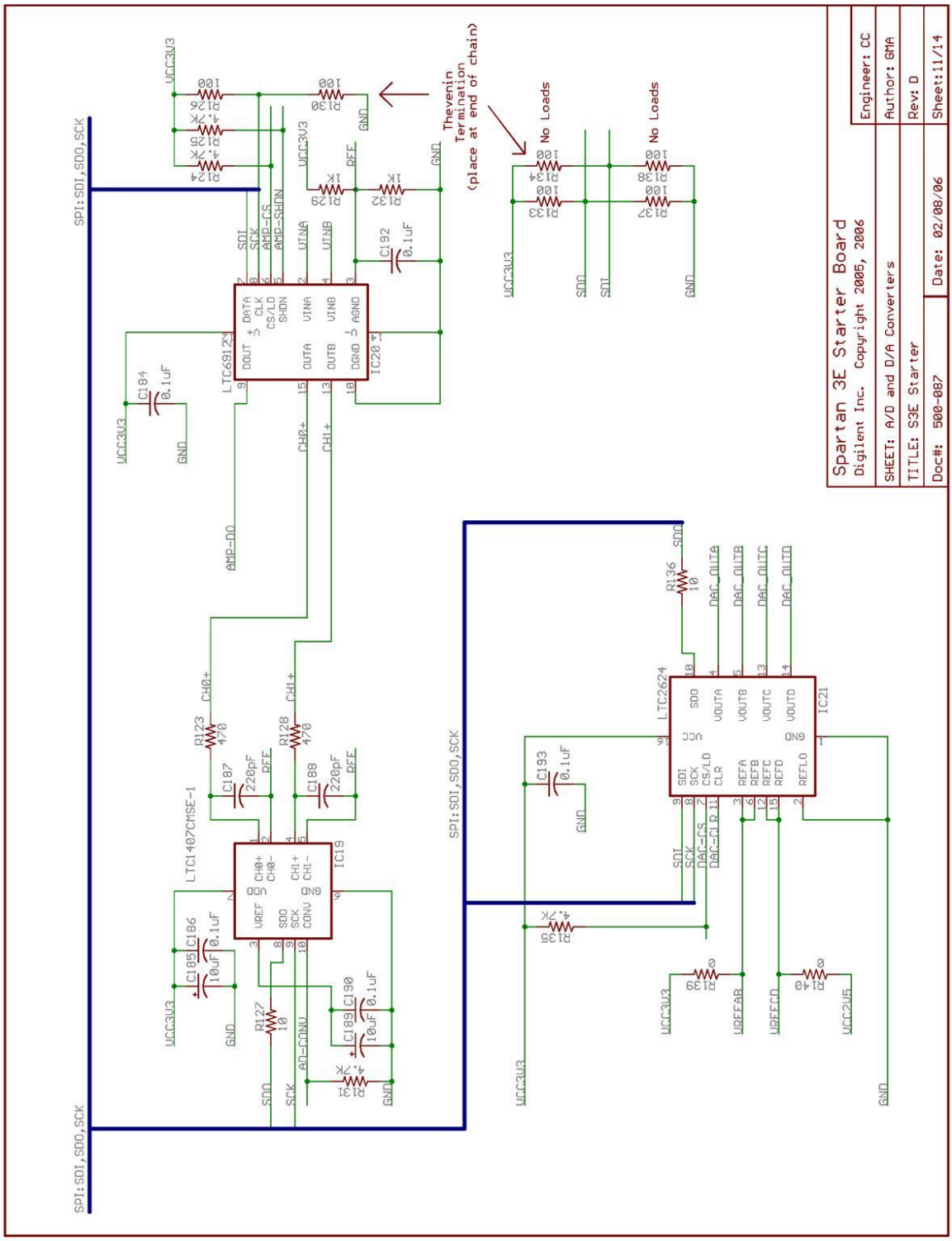
Power Supply Decoupling



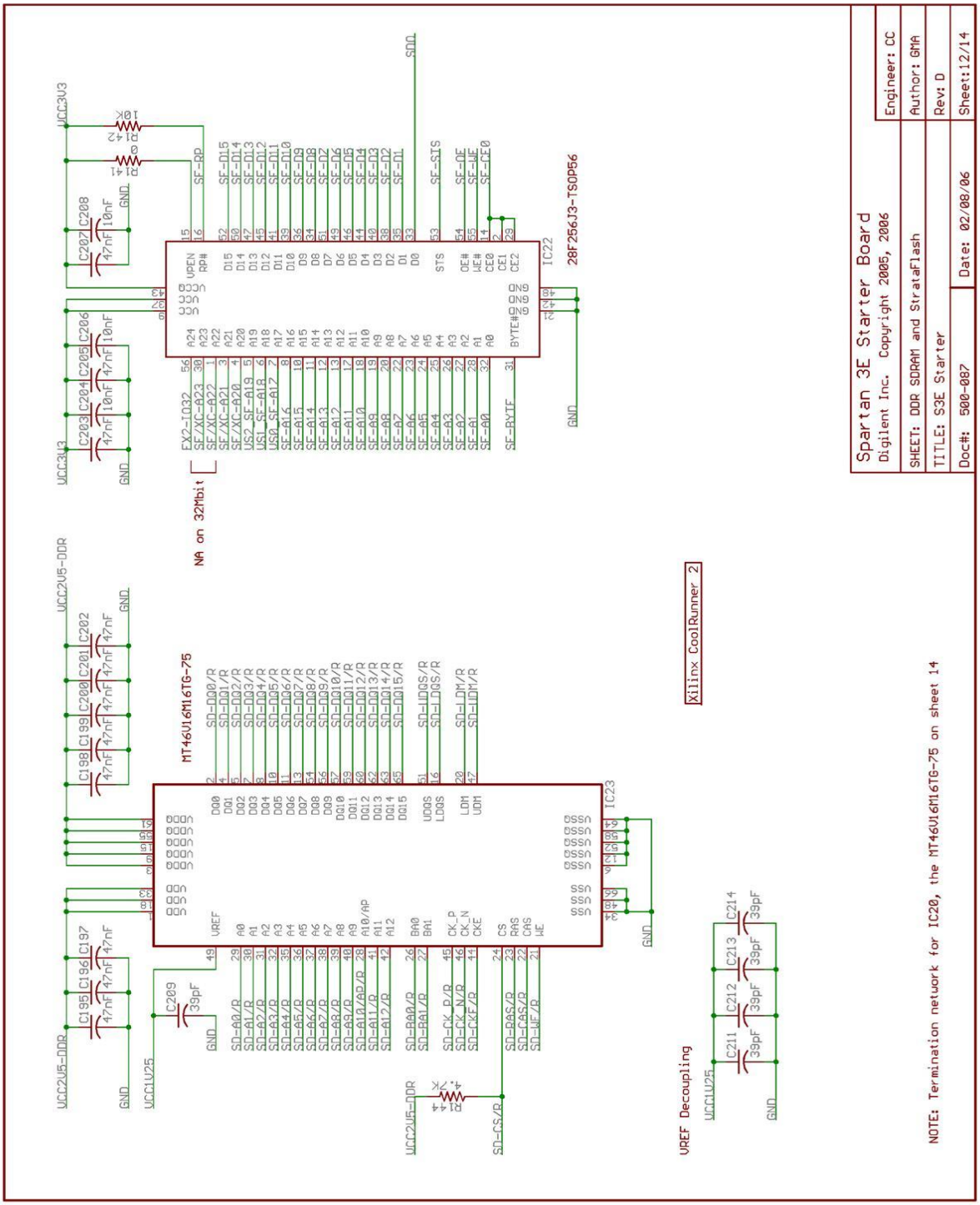
XC2C64A Cool Runner-II CPLD



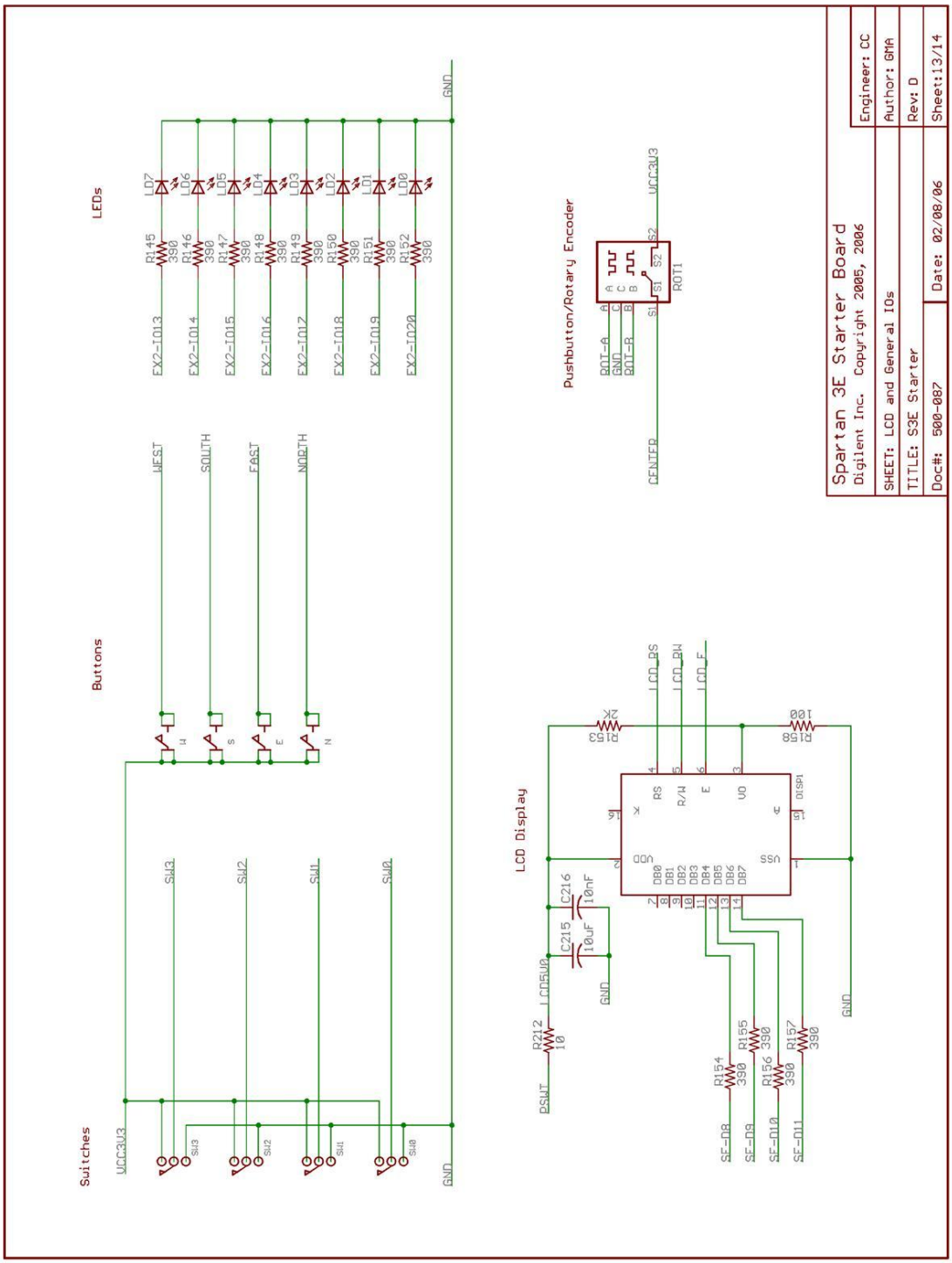
Linear Technology ADC and DAC



Intel Strata Flash Parallel NOR Flash Memory and Micron DDR SDRAM



Buttons, Switches, Rotary Encoder and Character LCD



Appendix C

VHDL code for Firmware of SCA

SCA.vhd Sun Mar 01 12:25:39 2015

```
 1
 2
 3   library IEEE;
 4   use IEEE.STD_LOGIC_1164.ALL;
 5   use IEEE.STD_LOGIC_ARITH.ALL;
 6   use IEEE.STD_LOGIC_UNSIGNED.ALL;
 7
 8
 9   entity SCA is
10       Port ( Clk : in  STD_LOGIC;
11              Btn_west: in STD_LOGIC;
12              LED: out STD_LOGIC_VECTOR (7 downto 0);
13
14              Rot_A: STD_LOGIC;               ---This pins for generate Gausian pulse
    with Rotary SW
15              Rot_B: STD_LOGIC;
16              Rot_Center: STD_LOGIC;
17
18              RS232_DTE_TXD : out STD_LOGIC;  -- For RS232 12 bit serial data out
19
20              SPI_MOSI  : OUT std_logic;       ---This pins for comunucate with ADC and
    DAC chip
21              SPI_MISO  : in  std_logic;
22              SPI_SCK   : OUT std_logic;
23              DAC_CS    : OUT std_logic;
24              DAC_Clr   : OUT std_logic;
25              AMP_CS    : out std_logic;
26              AD_CONV   : out std_logic;
27
28              SPI_SS_B  : out std_logic;      ---- This pins for Enable ADC and DAC
29              SF_CE0    : out std_logic;
30              FPGA_INIT_B   : out std_logic;
31
32              SF_D      : out STD_LOGIC_VECTOR (11 downto 8);  --for LCD data in
33              LCD_E     : out STD_LOGIC;                --for LCD Enable
34              LCD_RS    : out STD_LOGIC;                --for LCD Reset
35              LCD_RW    : out STD_LOGIC;                --for LCD Read/Write
36              SF_CEO    : out STD_LOGIC
37
38              );
39
40   end SCA;
41
42
43   architecture Behavioral of SCA is
44
45            Component Counter_Ena_Ovl is                              --- Declear componet
46                   Generic (bits : positive := 4; max : positive := 9);
47                   port (
48                     Clk      :  in std_logic;
49                     Reset    : in  std_logic := '0';
50                     Enable   :  in std_logic := '1';
51                     Overflow :  out std_logic;
52                     Cnt      : out std_logic_vector (bits-1 downto 0) );
53            end component;
54
55            component S3E_AnalogIO
```

```
56                  port (
57                        Clk             :  IN std_logic;
58                        SPI_MOSI     : OUT std_logic;
59                        SPI_MISO     : in  std_logic;
60                        SPI_SCK         : OUT std_logic;
61                        DAC_CS       : OUT std_logic;
62                        DAC_Clr         : OUT std_logic;
63                        AMP_CS       : out std_logic;
64                        AD_CONV         : out std_logic;
65
66                        TickAnalogIO    : out std_logic;   -- Give signal with New ADC
    value
67                        Dac_A           : IN  std_logic_vector (11 downto 0) := x"000";
68                        Dac_B           : IN  std_logic_vector (11 downto 0) := x"000";
69                        Dac_C           : IN  std_logic_vector (11 downto 0) := x"000";
70                        Dac_D           : IN  std_logic_vector (11 downto 0) := x"000";
71                        Adc_A           : out std_logic_vector (13 downto 0);
72                        Adc_B           : out std_logic_vector (13 downto 0) );
73           end component;
74
75
76           Component lcd_driver                                          ---LCD Driver
77              PORT( Clk         : IN    STD_LOGIC;
78                    rs          : OUT   STD_LOGIC;
79                    rw          : OUT   STD_LOGIC;
80                    enable      : OUT   STD_LOGIC;
81                    lcd_data    : OUT    STD_LOGIC_VECTOR(3 DOWNTO 0);
82
83                    index       : OUT    std_logic_vector (7 downto 0);
84                    char        : IN     std_logic_vector (7 downto 0)
85                    );
86           end component;
87
88
89           Component Bin16_Bcd5                                  -- Componet to
    convert 16 bit Binary to 5 digit BCD
90              PORT (   Clk         : IN  STD_LOGIC;
91                       BinIN       : IN  std_logic_vector (15 downto 0);
92                       BcdOut      : OUT std_logic_vector (19 downto 0) );
93           end component;
94
95
96           Component GaussianPulse is                            -- Component for
    Gaussian pulse generator
97              port (   CLK              : in  std_logic;
98                       ROT_A       : IN  STD_LOGIC;
99                       ROT_B       : IN  STD_LOGIC;
100                      ROT_CENTER  : IN  STD_LOGIC;
101                      Enable      : in  std_logic;
102                      Data        : out std_logic_vector (11 downto 0) );
103          end component;
104
105          Component RS232 is
106             Port (  CLK   : in  STD_LOGIC;
107                     DATA  : in  STD_LOGIC_VECTOR (7 downto 0);
108                     START : in  STD_LOGIC;
109                     BUSY  : out STD_LOGIC;
```

```
110                     TXD   : out STD_LOGIC);
111           end component;
112
113
114
115
116           Type EventType is (IDEL,S1,S2,S3,S4);
117           Signal sState : EventType;
118
119           Signal LLD: std_logic_vector (7 downto 0) := x"40";
120           Signal ULD: std_logic_vector (7 downto 0) := x"80";
121           Signal MAX: std_logic_vector (7 downto 0);
122           Signal ADCA: std_logic_vector (13 downto 0);
123           Signal sADC_A : std_logic_vector (7 downto 0);
124           Signal NewADC, sPeekFound : std_logic;
125
126           Signal Sec: std_logic;
127           Signal Min: std_logic;------------
128
129           Signal CountRate, CountRateout, totCount, MAX16: std_logic_vector (15 downto
      0);
130
131           Signal BCDCountRate, BCDTotCount, BCDMAX :std_logic_vector (19 downto 0);
132
133           Signal Index : Std_logic_vector (7 downto 0);   --- Signal slect LCD writing
      place
134           Signal ASCII : Std_logic_vector (7 downto 0);   --- Send ASCII letter to LCD
135
136           Signal Gausepulse : std_logic_vector (11 downto 0);
137
138
139   begin
140
141      SF_CEO <= '1';
142
143      SPI_SS_B  <= '1';
144      SF_CE0    <= '1';
145      FPGA_INIT_B   <= '1';
146
147
148
149      pS3AnalogIO:  S3E_AnalogIO                    ---Component for S3E Analog IO
      process
150               port map (
151                           Clk => Clk,
152                           SPI_MOSI => SPI_MOSI,
153                           SPI_MISO => SPI_MISO,
154                           SPI_SCK => SPI_SCK,
155                           DAC_CS => DAC_CS,
156                           DAC_Clr => DAC_Clr,
157                           AMP_CS =>AMP_CS,
158                           AD_CONV => AD_CONV,
159
160                           TickAnalogIO => NewADC,
161                           Dac_A  => Gausepulse,
162                           --Dac_B   =>
163                           --Dac_C   =>
```

```
164                         --Dac_D =>
165                         Adc_A =>  ADCA);
166                         --Adc_B    =>
167
168     sADC_A <= ADCA (13 downto 6);
169     LED <= Max;
170
171
172     pPreDevider: Counter_Ena_Ovl                    ---Component for Pre Divider to
    generate seconds
173              Generic map (bits =>32,  max => 50000000)  --- sClock for 1S
174               --Generic map (bits =>32,  max => 300000000)  --- sClock for 1M
175               port map    ( Clk  => Clk,
176                             Reset =>btn_west, --- '0',
177                             Enable => '1',
178                             Overflow => Sec);
179
180   ------------------------------------ own
181   pPreDeviderMin: Counter_Ena_Ovl               --- Component for Pre Divider to
    generate minutes
182              Generic map (bits =>8,  max => 120)  --- sClock for 1S
183               --Generic map (bits =>32,  max => 300000000)  --- sClock for 1M
184               port map    ( Clk  => Clk,
185                             Reset => '0',
186                             Enable => sec,
187                             Overflow => Min);
188
189
190
191   ----------------------------------------------------
192
193
194
195     pPeekDetector: process(Clk)                     --- Process for detect peak
196            begin
197              if rising_edge (Clk) then
198                 case sState is
199                    when IDEL => sPeekFound <= '0';
200                                 sState <= S1;
201
202                    when S1   => sPeekFound <= '0';   -- search for Event Start
203                                 if (sADC_A > LLD)and (NewADC = '1')then
204                                    max <= sADC_A;
205                                   sState <= S2;
206                                 end if;
207
208                    when S2   => sPeekFound <= '0';   -- Follow the signal curve
209                                 if (sADC_A > max) and (NewADC = '1') then
210                                    max <= sADC_A;    --Assign pulse ADC value to MAX
    until it reach Max
211                                 end if;
212
213                                 if (sADC_A < LLD) and (NewADC = '1') then
214                                   sState <= S3;   -- Retern to S3 when pulse faling
    down below LLD
215                                 end if;
216
```

```
217
218                   when s3   => if (MAX < ULD) then   -- After pulse falling down below
      LLD check that pulse below ULD
219                                   sPeekFound <= '1';
220                                 else
221                                   sPeekFound <= '0';
222                                 end if;
223                                 sState <= S4;
224
225                   when S4   => sPeekFound <= '0';
226                                 sState <= IDEL;
227
228                   when others => sState <= IDEL;
229                end case;
230             end if;
231           end process;
232
233        pCountRate: Counter_Ena_Ovl                    ---Component for count rate
234             Generic map (bits =>16,  max => 65536)
235             port map    ( Clk  => Clk,
236                           Reset => Min,  ----------------------------
237                           Enable => sPeekFound,
238                           --Overflow => ,
239                           Cnt => CountRate);     --- This will gives  # 0f sPeekFound
      in one Sec
240
241
242
243        pLatchforCountRate: process (Clk)
244             begin
245                If rising_edge(Clk) then
246                   If (Min='1') then
247                     CountRateout <= CountRate;
248                   End if;
249                End if;
250              end process;
251
252        pTotalCount: Counter_Ena_Ovl                    ---Component for total count
253             Generic map (bits =>16,  max => 65536)  --- sClock for 1S
254             port map    ( Clk  => Clk,
255                           Reset => Btn_west,
256                           Enable => sPeekFound,
257                           --Overflow => ,
258                           Cnt => TotCount);
259
260
261
262        pLcd: lcd_driver                                 ---Component for LCD Driver
263              port map (  Clk     =>  Clk,
264                          rs      =>  LCD_RS,
265                          rw      =>  LCD_RW,
266                          enable  =>  LCD_E,
267                          lcd_data=>  SF_D,
268
269                          index    =>  Index,
270                          char     =>  ASCII);
271
```

```
272
273         pBinBcdCountRate: Bin16_Bcd5              ---Component for BIN to BCD for Count
    Rate
274             port map (Clk    => CLK,
275                       BinIN  => CountRateout,
276                       BcdOut => BCDCountRate );
277
278         pBinBcdTotCount: Bin16_Bcd5             ---Component for BIN to BCD for Total
    Count
279             port map (Clk    => CLK,
280                       BinIN  => TotCount,
281                       BcdOut => BCDtotCount );
282
283         pBinBcdTotMAX: Bin16_Bcd5             ---Component for BIN to BCD for pulse
    MAX
284             port map (Clk    => CLK,
285                       BinIN  => MAX16,
286                       BcdOut => BCDMAX );
287
288         MAX16 <=  x"00" & MAX;        -- 8 bit Pulse MAX value assignto 16 bit MAX value
289
290          with Index select
291            ASCII <=    conv_std_logic_vector (67,8)  when x"06",  --1. Char 'C'
292                        conv_std_logic_vector (67,8) when x"46",  --1. Char 'C'
293                        conv_std_logic_vector (112,8) when x"47",  --1. Char 'p'
294                        conv_std_logic_vector (109,8) when x"48",  --1. Char 'm'
295                        conv_std_logic_vector (77,8) when x"08",  --1. Char 'M'
296                        conv_std_logic_vector (97,8) when x"09",  --1. Char 'a'
297                        conv_std_logic_vector (120,8) when x"0a",  --1. Char 'x'
298
299                        "0011" & BCDtotCount( 19 downto 16)   when x"00",
300                        "0011" & BCDtotCount( 15 downto 12)   when x"01",
301                        "0011" & BCDtotCount( 11 downto 8)    when x"02",
302                        "0011" & BCDtotCount( 7 downto 4)     when x"03",
303                        "0011" & BCDtotCount( 3 downto 0)     when x"04",
304
305                        --"0011" & BCDMAX( 19 downto 16)     when x"0b",
306                        "0011" & BCDMAX( 15 downto 12)     when x"0c",
307                        "0011" & BCDMAX( 11 downto 8)     when x"0d",
308                        "0011" & BCDMAX( 7 downto 4)     when x"0e",
309                        "0011" & BCDMAX( 3 downto 0)     when x"0f",
310
311                        "0011" & BCDCountRate( 19 downto 16)  when x"40",
312                        "0011" & BCDCountRate( 15 downto 12)  when x"41",
313                        "0011" & BCDCountRate( 11 downto 8)  when x"42",
314                        "0011" & BCDCountRate( 7 downto 4)  when x"43",
315                        "0011" & BCDCountRate( 3 downto 0)  when x"44",
316
317                        "00000000"           when OTHERS;
318
319           pGauss:      GaussianPulse
320                  port map (  Clk            => Clk,
321                              ROT_A       => ROT_A,
322                              ROT_B       => ROT_B,
323                              ROT_CENTER  => ROT_Center,
324                              Enable         => NewADC,
325                              Data           => Gausepulse);
```

```
326
327
328
329           pRs232:    RS232
330                  Port map (  CLK   => Clk,
331                              DATA  => CountRateout(10 downto 3),
332                              START => Sec,
333                              --BUSY  => ,
334                              TXD   => RS232_DTE_TXD);
335
336
337
338   end Behavioral;
339
340
```

VHDL Code of ADC and DAC

S3E_AnalogIO.vhd Sun Mar 01 12:51:55 2015

```
1   ----------------------------------------------------------------
2   --   Spartan-3E Kit: Analog IO Component
3   --       DAC component: LTC2624  4 channel, 12 bit DAC
4   --       ADC component: LTC1407  2 channel, 14 bit ADC
5   --
6   ----------------------------------------------------------------
7
8   library IEEE;
9   use IEEE.STD_LOGIC_1164.ALL;
10  use IEEE.STD_LOGIC_ARITH.ALL;
11  use IEEE.STD_LOGIC_UNSIGNED.ALL;
12
13
14  -- component S3E_AnalogIO  port (
15  --       Clk                :  IN std_logic;
16  --       SPI_MOSI        : OUT std_logic;
17  --       SPI_MISO    : in  std_logic;
18  --       SPI_SCK         : OUT std_logic;
19  --       DAC_CS          : OUT std_logic;
20  --       DAC_Clr         : OUT std_logic;
21  --       AMP_CS          : out std_logic;
22  --       AD_CONV         : out std_logic;
23  --       TickAnalogIO: out std_logic;
24  --       Dac_A           :  IN std_logic_vector (11 downto 0) := x"000";
25  --       Dac_B           :  IN std_logic_vector (11 downto 0) := x"000";
26  --       Dac_C           :  IN std_logic_vector (11 downto 0) := x"000";
27  --       Dac_D           :  IN std_logic_vector (11 downto 0) := x"000";
28  --       Adc_A           : out std_logic_vector (13 downto 0);
29  --       Adc_B           : out std_logic_vector (13 downto 0) );
30  --end component;
31
32  --Please also connect:
33  -- SPI_SS_B       <= '1';
34  -- SF_CE0            <= '1';
35  -- FPGA_INIT_B   <= '1';
36  ----------------------------------------------------------------
37
38  entity S3E_AnalogIO is
39  port (
40          Clk                :  IN std_logic;
41          SPI_MOSI        : OUT std_logic;
42          SPI_MISO    : in  std_logic;
43          SPI_SCK         : OUT std_logic;
44          DAC_CS          : OUT std_logic;
45          DAC_Clr         : OUT std_logic;
46          AMP_CS          : out std_logic;
47          AD_CONV         : out std_logic;
48          TickAnalogIO: out std_logic;
49          Dac_A           :  IN std_logic_vector (11 downto 0) := x"000";
50          Dac_B           :  IN std_logic_vector (11 downto 0) := x"000";
51          Dac_C           :  IN std_logic_vector (11 downto 0) := x"000";
52          Dac_D           :  IN std_logic_vector (11 downto 0) := x"000";
53          Adc_A           : out std_logic_vector (13 downto 0);
54          Adc_B           : out std_logic_vector (13 downto 0)
55        );
56  end S3E_AnalogIO;
57
```

```
58  ----------------------------------
59
60  architecture Behav of S3E_AnalogIO is
61
62     type     EVENT_TYPE  is    (S0,  S1,  S2,  S3,  S4,  S5,  S6,  S7,  S8,  S9,  S10,
    S11, S12, S13, S14, S15,
63                                 S16, S17, S18, S19, S20, S21, S22, S23, S24, S25, S26,
    S27, S28, S29, S30, S31 );
64
65        signal   sDacState   : EVENT_TYPE;
66
67        signal  Clk2 : std_logic;
68
69        signal   sData : std_logic_vector (31 downto 0);
70
71  begin
72
73  pClk2:   process (Clk)
74              begin
75                 if rising_edge(Clk) then
76                    Clk2 <= not Clk2;
77                 end if;
78              end process;
79
80
81     DAC_Clr     <= '1';
82
83
84  pDAC: process (Clk2, sDacState)
85
86           constant kDacA : std_logic_vector(7 downto 0)  := "00110000";     -- update
    immediately
87           constant kDacB : std_logic_vector(7 downto 0)  := "00110001";
88           constant kDacC : std_logic_vector(7 downto 0)  := "00110010";
89           constant kDacD : std_logic_vector(7 downto 0)  := "00110011";
90
91           constant kAmpD : std_logic_vector(7 downto 0)  := "00010001";
92
93
94           variable bitnr    : integer;
95           variable i        : integer;
96
97  begin
98              if rising_edge (Clk2)
99              then
100                   case sDacState is
101                      when S0  => Dac_CS      <= '1';
102                                  AMP_CS      <= '1';
103                                  AD_CONV     <= '0';
104                                  SPI_SCK  <= '0';
105                                  SPI_MOSI <= '0';
106                                  sDacState <= S1;
107
108 -- DAC_A
109                      when S1 =>  Dac_CS    <= '0';             -- Prepare
110                                  bitnr       := 0;
111                                  sData      <= "00000000" & kDacA & Dac_A &
```

```
      "0000";
112                                     sDacState <= S2;
113
114                     when S2 =>  Dac_CS      <= '0';             -- LOOP: set Data
115                                 SPI_SCK     <= '0';
116                                 SPI_MOSI <= sData(31);
117                                 sData       <= sData (30 downto 0) & '0';
118                                 sDacState <=   S3;
119
120                     when S3 =>  SPI_SCK  <= '1';
121                                 bitnr := bitnr +1;
122                                 if (bitnr < 32) then          -- Set Clock
123                                    sDacState <= S2;
124                                 else
125                                    sDacState <= S4;
126                                 end if;
127
128                     when S4 =>  Dac_CS      <= '1';             -- OK
129                                 SPI_SCK  <= '0';

130                                 sDacState <= S5;
131
132     -- DAC_B ------------------------------------------------
133                     when S5 =>  Dac_CS    <= '0';             -- Prepare
134                                 bitnr       := 0;
135                                 sData      <= "00000000" & kDacB & Dac_B &
      "0000";
136                                 sDacState <= S6;
137
138                     when S6 =>  Dac_CS      <= '0';             -- LOOP: set Data
139                                 SPI_SCK     <= '0';
140                                 SPI_MOSI <= sData(31);
141                                 sData       <= sData (30 downto 0) & '0';
142                                 sDacState <=   S7;
143
144                     when S7 =>  SPI_SCK  <= '1';
145                                 bitnr := bitnr +1;
146                                 if (bitnr < 32) then          -- Set Clock
147                                    sDacState <= S6;
148                                 else
149                                    sDacState <= S8;
150                                 end if;
151
152                     when S8 =>  Dac_CS      <= '1';             -- OK
153                                 SPI_SCK  <= '0';

154                                 sDacState <= S9;
155     -- DAC_C ------------------------------------------------
156                     when S9 =>  Dac_CS    <= '0';             -- Prepare
157                                 bitnr       := 0;
158                                 sData      <= "00000000" & kDacC & Dac_C &
      "0000";
159                                 sDacState <= S10;
160
161                     when S10 => Dac_CS      <= '0';             -- LOOP: set Data
162                                 SPI_SCK     <= '0';
163                                 SPI_MOSI    <= sData(31);
```

```
164                                     sData          <= sData (30 downto 0) & '0';
165                                     sDacState   <= S11;
166
167                         when S11 => SPI_SCK  <= '1';
168                                     bitnr := bitnr +1;
169                                     if (bitnr < 32) then          -- Set Clock
170                                           sDacState <= S10;
171                                     else
172                                           sDacState <= S12;
173                                     end if;
174
175                         when S12 => Dac_CS      <= '1';             -- OK
176                                     SPI_SCK  <= '0';

177                                     sDacState <= S13;
178     -- DAC_D ------------------------------------------------------
179                         when S13 => Dac_CS    <= '0';             -- Prepare
180                                     bitnr       := 0;
181                                     sData      <= "00000000" & kDacD & Dac_D &
        "0000";
182                                     sDacState <= S14;
183
184                         when S14 => Dac_CS      <= '0';             -- LOOP: set Data
185                                     SPI_SCK     <= '0';
186                                     SPI_MOSI <= sData(31);
187                                     sData       <= sData (30 downto 0) & '0';
188                                     sDacState <=   S15;
189
190                         when S15 => SPI_SCK  <= '1';
191                                     bitnr := bitnr +1;
192                                     if (bitnr < 32) then          -- Set Clock
193                                        sDacState <= S14;
194                                     else
195                                        sDacState <= S16;
196                                     end if;
197
198                         when S16 => Dac_CS      <= '1';             -- OK
199                                     SPI_SCK     <= '0';

200                                     sDacState <= S17;
201     --- Prog. Gain Amplifier ---------------------------------------
202
203                         when S17 => AMP_CS    <= '0';             -- Prepare
204                                     bitnr       := 0;
205                                     sData      <= kAmpD & "00000000" & "00000000" &
         "00000000";
206                                     i := 0;
207                                     sDacState <= S18;
208
209                         when S18 => Dac_CS      <= '0';             -- LOOP: set Data
210                                     SPI_SCK     <= '0';
211                                     SPI_MOSI <= sData(31);
212                                     i := i + 1;
213                                     if (i>1) then
214                                        bitnr := bitnr +1;
215                                        sData <= sData (30 downto 0) & '0';
216                                        i := 0;
```

```
217                                         sDacState   <= S19;
218                                      end if;
219
220                        when S19 => SPI_SCK  <= '1';
221                                      i := i + 1;
222                                      if (i>1) then
223                                         if (bitnr < 8) then        -- Set Clock
224                                            i := 0;
225                                            sDacState <= S18;
226                                         else
227                                            i := 0;
228                                            sDacState <= S20;
229                                         end if;
230                                      end if;
231
232                        when S20 => AMP_CS      <= '1';              -- OK
233                                      SPI_SCK     <= '0';
234                                      i := i + 1;
235                                      if (i>1) then
236                                            i := 0;
237                                            sDacState <= S21;
238                                      end if;
239
240   --- ADC ------------------------------------------------------------
241
242                        when S21 => AD_CONV   <= '1';              -- Prepare: ADC Convert
243                                      SPI_SCK     <= '0';
244                                      sDacState <= S22;
245
246                        when S22 => AD_CONV   <= '1';              -- 1. Clk
247                                      SPI_SCK     <= '1';
248                                      sDacState <= S23;
249
250                        when S23 => AD_CONV   <= '0';              --
251                                      SPI_SCK     <= '0';
252                                      bitnr       := 0;
253                                      sDacState <= S24;
254
255                        when S24 => AD_CONV   <= '0';
256                                      SPI_SCK     <= '1';
257                                      sDacState <= S25;
258
259                        when S25 => AD_CONV   <= '0';
260                                      SPI_SCK     <= '0';
261                                      sData       <= sData (30 downto 0) & SPI_MISO;
262                                      bitnr := bitnr + 1;
263                                      if (bitnr < 33) then
264                                         sDacState <= S24;
265                                      else
266                                         sDacState <= S26;
267                                      end if;
268
269                        when S26 => AD_CONV   <= '0';
270                                      SPI_SCK     <= '0';
271                                      Adc_A <= sData(31) & not sData (30 downto  18);
272                                      Adc_B <= sdata(15) & not sData (14 downto   2);
273                                      sDacState <= S27;
```

```
274
275   ------------------------------------------------------

276                       when S27 => TickAnalogIO <= '1';
277                                   sDacState <= S28;
278
279                       when S28 => TickAnalogIO <= '0';
280                                   sDacState <= S0;
281
282
283

284                       when others => sDacState <= S0;
285
286                   end case;
287               end if;
288           end process;
289
290
291   end Behav;
292
293
294
295
```

VHDL Code of Counter Enable Overflow

Counter_Ena_Ovl.vhd — Sun Mar 01 13:00:52 2015

```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.STD_LOGIC_UNSIGNED.all;
4
5   --component Counter_Ena_Ovl is
6   -- Generic (bits: positive:=3; max: positive:= 9);
7   -- port (
8   --       Clk               :  IN std_logic;
9   --       Reset          : in  std_logic := '0';
10  --       Enable         :  IN std_logic := '1';
11  --       Overflow    :  out std_logic;
12  --       Cnt               :  OUT std_logic_vector (bits-1 downto 0) );
13  --end component;
14  --
15  -- pxx: Counter_Ena_Ovl Generic map (bits=>32, max=>50000)
16  --                              port map (Clk=>CLk, Cnt=>c32 );
17
18
19
20  entity Counter_Ena_Ovl is
21     Generic (bits : positive := 4; max : positive := 9);
22     port (
23           Clk               :  IN std_logic;
24           Reset          : in  std_logic := '0';
25           Enable         :  IN std_logic := '1';
26           Overflow    :  out std_logic;
27           Cnt               : OUT std_logic_vector (bits-1 downto 0) );
28  end Counter_Ena_Ovl;
29
30
31  architecture behav of Counter_Ena_Ovl is
32
33     signal  sCnt      : std_logic_vector (bits-1 downto 0);
34     signal   sOvl     : std_logic;
35
36  begin
37
38     Cnt         <= sCnt;
39     Overflow    <= sOvl;
40
41     process (Clk)
42     begin
43        if rising_edge (Clk) then
44           sOvl <= '0';
45           if Reset = '1' then
46                 sCnt <= (others=>'0');
47           elsif (Enable = '1') then
48              if (sCnt < max) then
49                 sCnt <= sCnt + 1;
50              Else
51                 sCnt <= (others=>'0');
52                 sOvl <= '1';
53              End if;
54           End if;
55        end if;
56     end process;
57
```

VHDL Code of LCD Driver

lcd_driver.vhd Sun Mar 01 13:13:07 2015

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7   --Component lcd_driver
8   --    PORT( Clk         : IN    STD_LOGIC;
9   --             rs          : OUT   STD_LOGIC;
10  --          rw          : OUT   STD_LOGIC;
11  --          enable      : OUT   STD_LOGIC;
12  --          lcd_data    : OUT    STD_LOGIC_VECTOR(3 DOWNTO 0);
13  --
14  --             index       :  OUT      std_logic_vector (7 downto 0);
15  --             char        : IN     std_logic_vector (7 downto 0)
16  --          );
17  --end component;
18  --
19  --Please also connect:
20  -- SF_CE0            <= '1';
21  --
22  ---- Instantiation of the LCD Display driver --------------
23  --pLcd:        lcd_driver port map (   Clk         =>    CLK,
24  --                                                 rs      =>  LCD_RS,
25  --                                                 rw      =>  LCD_RW,
26  --                                                 enable  =>  LCD_E,
27  --                                                 lcd_data=>  SF_D,
28  --
29  --                                                 index    =>  LcdIndex,
30  --                                                 char     =>  LcdChar);
31  --
32  -- with LcdIncex select
33  --    LcdChar  <= conv_std_logic_vector (68,8)  when x"00",
34  --                          conv_std_logic_vector (68,8)  when x"01",
35  --                          "0011" & sBcd1 (3 downto 0)   when x"02",
36  --                          "0011" & sBcd2 (3 downto 0)   when
    x"40",
37  --                          "00000000"                              when OTHERS;
38  --
39
40
41
42  ENTITY lcd_driver IS
43      PORT(   Clk         : IN    STD_LOGIC;
44              rs          : OUT   STD_LOGIC;
45          rw          : OUT   STD_LOGIC;
46          enable      : OUT   STD_LOGIC;
47          lcd_data    : OUT    STD_LOGIC_VECTOR(3 DOWNTO 0);
48
49              index       :  OUT      std_logic_vector (7 downto 0);
50              char        : IN     std_logic_vector (7 downto 0)
51          );
52  END lcd_driver;
53
54
55
56  ----------------------------------------------------------------
```

```
57  ARCHITECTURE behavioral OF lcd_driver IS
58
59     type     charSTATE_TYPE is (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11 );
60     signal   charState : charSTATE_TYPE;
61
62  constant t_InstrWait : integer := 100000; -- 2 ms
63  constant    t_WRPulse      : integer := 10;        -- 0,2 us (200 ns)
64  constant    t_SetupHold : integer := 10;
65  constant    t_DatWait      : integer := 2500;     -- 50 us
66
67
68  TYPE STATE_TYPE IS (init,
69                              wait_for_data,
70
71                              write_addrH1, write_addrH2, write_addrH3,
72                              write_addrL1, write_addrL2, write_addrL3,
73                              chk_busyI1,
74                              chk_busyI2,
75                              write_dataH1, write_dataH2, write_dataH3,
76                              write_dataL1, write_dataL2, write_dataL3,
77                              chk_busyD1
78                              );
79
80  SIGNAL state : STATE_TYPE := init;
81
82  signal   sLcdAdr : STD_LOGIC_VECTOR(7 DOWNTO 0);
83  signal   sLcdDat : STD_LOGIC_VECTOR(7 DOWNTO 0);
84  signal  sLcdWR  : STD_LOGIC;
85  signal  sLcdRDY : STD_LOGIC;
86
87
88  SIGNAL int_addr : STD_LOGIC_VECTOR( 7 DOWNTO 0 );
89  SIGNAL int_data : STD_LOGIC_VECTOR( 7 DOWNTO 0 );
90  SIGNAL enrwrs  : STD_LOGIC_VECTOR( 2 DOWNTO 0 );
91
92  ---------------------------------------------------------------------------
93  BEGIN
94
95
96  pTextOut:   process (Clk)
97                 variable Cnt : std_logic_vector (31 downto 0);
98                 variable i   : std_logic_vector ( 7 downto 0);  --integer range 0 TO 40;
99                 begin
100
101 --                 index <= conv_std_logic_vector (i,8);
102                    index <= i;
103
104                    if rising_edge (CLK) then
105                       CASE charState IS
106 ---- CLR
107                             when S0  =>    sLcdWR <= '0';
108                                            sLcdAdr <= x"01";  -- Clear Display
109                                            sLcdDat <= x"20";  -- Space
110                                            if (sLcdRDY = '1')   then
111                                                  sLcdWR  <= '1';
112                                                  cnt := (others => '0');
113                                                  charState   <= S1;
```

```
114                                                 end if;
115
116                             when S1  => sLcdWR   <= '0';
117                                                 cnt := cnt + 1 ;
118                                                 if (cnt > conv_std_logic_vector (
    100000, 16)) then    -- 100.000 = 2ms
119                                                    charState   <= S2;
120                                                    i := (others=>'0');
121                                                 end if;
122
123   ----1
124                             when S2  =>    if (char = "00000000")
125                                                 then        -- NULL CHARACTER
126                                                    if (i >= x"79") then
127                                                       cnt := (others => '0');
128                                                       charState <= S11;
129                                                    else
130                                                       i := i + 1;
131                                                       charState <= S2;
132                                                    end if;
133                                                 else
134                                                    sLcdWR  <= '0';
135                                                    sLcdAdr <= '1' &  i(6 downto 0);
        --x"81";       -- the Character Location
136                                                    sLcdDat <= char;
                                                     --Memory(i);  --
    conv_std_logic_vector (66,8);
137                                                    if (sLcdRDY = '1')   then
138                                                          sLcdWR  <= '1';
139                                                          cnt := (others => '0');
140                                                          charState   <= S3;
141                                                    end if;
142
143                                                 end if;
144
145                             when S3  => sLcdWR   <= '0';
146                                                 charState   <= S4;
147
148                             when S4  => sLcdWR   <= '0';
149                                                 if (sLcdRDY = '1') then
150                                                    charState   <= S2;
151                                                    i := i + 1;
152                                                 end if;
153
154   --- Wait Loop
155
156   --                        when S10    =>  if (i < 20) then
157   --                                           i := x"40";
158   --                                           charState   <= S2;
159   --                                        else
160   --                                           cnt := (others => '0');
161   --                                           charState   <= S11;
162   --                                        end if;
163   --
164                             when S11    => cnt := cnt + 1;
165                                                 if (cnt > 10000000) then -- wait
    200 ms : 5 updates /Sec.
```

```
166                                                     charState <= S0;
167                                                 end if;
168
169                             when others => charState <= S0;
170                         end case;
171                     end if;
172                 end process;
173
174
175
176
177
178 ---------------------------------------------------------------------------
179     enable   <= enrwrs(2);
180     rw       <= enrwrs(1);
181     rs       <= enrwrs(0);
182
183 pLcdInstr:
184     process (Clk)
185     variable counter : INTEGER RANGE 0 TO 50000000 := 0;
186     begin
187        if rising_edge (Clk) THEN
188        CASE state IS
189
190           WHEN init =>
191                 -- state <= wait_for_data;
192
193                 counter := counter + 1;
194                 if ( counter >= 500000 ) then       -- 10 ms
195                    counter := 0;
196                    state <= wait_for_data;
197                 end if;
198
199
200 ---- WAIT for new Data HERE
201
202           WHEN wait_for_data =>
203              counter := 0;
204              if (sLcdWR = '1') then
205                 int_addr <= sLcdAdr;
206                 int_data <= sLcdDat;
207                 state <= write_addrH1; -- chk_busy1;
208              end if;
209
210
211 -- Address:  High NIBBLE
212           WHEN write_addrH1 =>
213              counter := counter + 1;
214              IF ( counter >= t_WRPulse ) THEN    -- 2 us WR Time
215                 counter := 0;
216                 state <= write_addrH2;
217              END IF;
218
219           WHEN write_addrH2 =>
220              counter := counter + 1;
221              IF (counter >= t_SetupHold) THEN
222                 counter := 0;
```

```
223                  state <= write_addrL1;
224               END IF;
225
226   -- Address LOW NIBBLE
227            WHEN write_addrL1 =>
228               counter := counter + 1;
229               IF ( counter >= t_WRPulse ) THEN
230                  counter := 0;
231                  state <= write_addrL2;
232               END IF;
233
234            WHEN write_addrL2 =>
235               counter := counter + 1;
236               IF (counter >= t_SetupHold) THEN
237                  counter := 0;
238                  state <= chk_busyI1;
239               END IF;
240
241
242   --- WAIT
243            WHEN chk_busyI1 =>
244               counter := counter + 1;
245               if ( counter >= t_DatWait ) then
246                  counter := 0;
247                  if (int_addr(7) = '1') then
248                     state <= write_dataH2;
249                  else
250                     state <= chk_busyI2;
251                  end if;
252               END IF;
253
254   --- WAIT
255            WHEN chk_busyI2 =>
256               counter := counter + 1;
257               if ( counter >= t_InstrWait ) then
258                  counter := 0;
259                  state <= write_dataH2;
260               END IF;
261
262
263   ---------- NOW the DATA -------------------------------------------------
264
265   -- DATA High NIBBLE
266   --         WHEN write_dataH1 =>
267   --            counter := counter + 1;
268   --            IF ( counter >= t_WRPulse ) THEN
269   --               counter := 0;
270   --               state <= write_dataH2;
271   --            END IF;
272
273            WHEN write_dataH2 =>
274               counter := counter + 1;
275               IF ( counter >= t_WRPulse ) THEN
276                  counter := 0;
277                  state <= write_dataH3;
278               END IF;
279
```

```
280            WHEN write_dataH3 =>
281               counter := counter + 1;
282               IF (counter >= t_SetupHold) THEN
283                  counter := 0;
284                  state <= write_dataL2;
285               END IF;
286
287   -- DATA Low NIBBLE
288   --          WHEN write_dataL1 =>
289   --             counter := counter + 1;
290   --             IF ( counter >= t_WRPulse ) THEN
291   --                counter := 0;
292   --                state <= write_dataL2;
293   --             END IF;
294
295            WHEN write_dataL2 =>
296               counter := counter + 1;
297               IF ( counter >= t_WRPulse ) THEN
298                  counter := 0;
299                  state <= write_dataL3;
300               END IF;
301
302            WHEN write_dataL3 =>
303               counter := counter + 1;
304               IF (counter >= t_SetupHold) THEN
305                  counter := 0;
306                  state <= chk_busyD1;
307               END IF;
308
309   --- WAIT
310            WHEN chk_busyD1 =>
311               counter := counter + 1;
312               if ( counter >= t_DatWait ) then    -- 50 us
313                  counter := 0;
314                  state <= wait_for_data;   --write_dataH1;
315               END IF;
316
317
318            WHEN OTHERS => state <= init;
319         END CASE;
320      END IF;
321   END PROCESS;
322
323
324
325      sLcdRDY <= '1' WHEN (state = wait_for_data) ELSE '0';
326
327
328
329      with state select
330      lcd_data <=
331                     "ZZZZ"      WHEN init,
332
333                     int_addr(7 downto 4) WHEN write_addrH1,
334                     int_addr(7 downto 4) WHEN write_addrH2,
335                     int_addr(3 downto 0) WHEN write_addrL1,
336                     int_addr(3 downto 0) WHEN write_addrL2,
```

```
337
338                     int_data(7 downto 4) WHEN write_dataH1,
339                     int_data(7 downto 4) WHEN write_dataH2,
340                     int_data(7 downto 4) WHEN write_dataH3,
341
342                     int_data(3 downto 0) WHEN write_dataL1,
343                     int_data(3 downto 0) WHEN write_dataL2,
344                     int_data(3 downto 0) WHEN write_dataL3,
345
346                     "ZZZZ"                       WHEN wait_for_data,
347                     "ZZZZ"                       WHEN OTHERS;
348
349   --  ENABLE RW RS
350
351      WITH state SELECT
352         enrwrs <=   "000"    WHEN init,
353                        "000"    WHEN wait_for_data,
354
355                        "100"    WHEN write_addrH1,
356                        "000"    WHEN write_addrH2,
357                        "100"    WHEN write_addrL1,
358                        "000"    WHEN write_addrL2,
359
360                        "001"    WHEN write_dataH1,          -- LcdData:
361                        "101"    WHEN write_dataH2,    -- output LCD Data: ENABLE = 1;
362                        "001"    WHEN write_dataH3,
363
364                        "001"    WHEN write_dataL1,          -- LcdData:
365                        "101"    WHEN write_dataL2,    -- output LCD Data: ENABLE = 1;
366                        "001"    WHEN write_dataL3,
367
368                        "000"    WHEN OTHERS;
369
370   END behavioral;
371
```

VHDL Code of Bin 16 to BCD

Bin16_BCD5.vhd **Sun Mar 01 13:20:39 2015**

```
1   LIBRARY IEEE;
2   USE  IEEE.STD_LOGIC_1164.all;
3   USE  IEEE.STD_LOGIC_ARITH.all;
4   USE  IEEE.STD_LOGIC_UNSIGNED.all;
5
6
7   --Component Bin16_Bcd5
8   -- PORT (   Clk           : IN  STD_LOGIC;
9   --             BinIN       : IN  std_logic_vector (15 downto 0);
10  --             BcdOut      : OUT std_logic_vector (19 downto 0) );
11  --end component;
12  --
13
14  --pBinBcd: Bin16_Bcd5 port map (Clk    => CLK,
15  --                                    BinIN  => sBinIN,
16  --                                    BcdOut => sBcdOut );
17
18  ----------------------------------------------------------------------
19  ENTITY Bin16_Bcd5 IS
20     PORT
21     (
22        Clk            : IN  STD_LOGIC;
23        BinIN       : IN  std_logic_vector (15 downto 0);
24        BcdOut      : OUT std_logic_vector (19 downto 0)
25     );
26  END Bin16_Bcd5;
27
28
29  ARCHITECTURE a OF Bin16_Bcd5 is
30
31      type    TStates is (S0, S1, S2);
32
33     subtype Nibble is std_logic_vector (3 downto 0);
34     type     TBcd is array (0 to 4) of Nibble;
35     signal   sBCD     : TBcd;
36
37  begin
38
39  pBin16BCD:  process (Clk)
40                    variable State : TStates;
41                    variable cnt     : std_logic_vector (15 downto 0);
42                    begin
43                       if rising_edge (Clk) then
44                          case State is
45                             when S0 =>  cnt := (others => '0');
46                                         for i in 0 to 4 loop
47                                            sBcd(i) <= x"0";
48                                         end loop;
49                                         State := S1;
50
51                             when S1 =>  cnt := cnt + 1;
52                                         if (cnt < BinIN) then
53
54                                               if (sBcd(0) < 9) then
55                                                  sBcd(0) <= sBcd(0) + 1;
56                                               else
57                                                  sBcd(0) <= x"0";
```

```
58
59                                                      if (sBcd(1) < 9) then
60                                                         sBcd(1) <= sBcd(1) + 1;
61                                                      else
62                                                         sBcd(1) <= x"0";
63
64                                                         if (sBcd(2) < 9) then
65                                                            sBcd(2) <= sBcd(2) + 1;
66                                                         else
67                                                            sBcd(2) <= x"0";
68
69                                                            if (sBcd(3) < 9) then
70                                                               sBcd(3) <= sBcd(3) + 1;
71                                                            else
72                                                               sBcd(3) <= x"0";
73
74                                                               if (sBcd(4) < 9) then
75                                                                  sBcd(4) <= sBcd(4) +
       1;
76                                                               else
77                                                                  sBcd(4) <= x"0";
78                                                               end if;
79                                                            end if;
80                                                         end if;
81                                                      end if;
82                                                   end if;
83                                                else
84                                                   State := S2;
85                                                end if;
86
87                                 when S2 =>  for i in 0 to 4 loop
88                                                   BcdOut(i*4+3 downto i*4) <= sBcd(i);
89                                                end loop;
90                                                State := S0;
91
92                                 when others => State := S0;
93                              end case;
94                           end if;
95                        end process;
96
97    END a;
98
99
```

Code of Rotary Counter

Rotary_Counter.vhd — Sun Mar 01 13:27:50 2015

```
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3  use IEEE.STD_LOGIC_ARITH.ALL;
 4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
 5
 6
 7  entity Rotary_Counter is
 8        Generic (   bits : Integer := 8);
 9      Port       ( CLK : in  STD_LOGIC;
10                   ROT_A : in  STD_LOGIC;
11                   ROT_B : in  STD_LOGIC;
12                   ROT_CENTER : in STD_LOGIC;
13                   COUNTER : out  STD_LOGIC_VECTOR (bits -1 downto 0));
14  end Rotary_Counter;
15
16
17
18
19  architecture Behavioral of Rotary_Counter is
20
21     component Monoflop   port (
22           Clk          : IN  std_logic;      -- System Clock
23           Trigger      : IN  std_logic;
24           PULSOUT      : OUT std_logic );
25     end component;
26
27
28  signal rst    : STD_LOGIC;
29  signal sROT_A : STD_LOGIC;
30  signal sROT_B : STD_LOGIC;
31
32  signal sCount : STD_LOGIC_VECTOR (bits -1 downto 0);
33  signal sROT   : STD_LOGIC_VECTOR (1 downto 0);
34
35  type     tstates is (state0, state1, state2, state3, state4, state5, state6);
36  signal   State:   tstates;
37
38
39  begin ------------------------
40
41  rst <= ROT_CENTER;
42
43
44  MF1:  Monoflop port map (Clk => CLK,    Trigger => ROT_A, PULSOUT => sROT_A);
45  MF2:  Monoflop port map (Clk => CLK,    Trigger => ROT_B, PULSOUT => sROT_B);
46
47  sROT(0) <= ROT_A or sROT_A;
48  sROT(1) <= ROT_B or sROT_B;
49
50  process (clk, rst)
51  begin
52     if rst='1' then
53        state <= state0;
54        sCount<= (others => '0');
55     elsif rising_edge(Clk) then
56        case state is
57
```

```
58              when state0 =>
59                 if sROt = "11" then
60                    state <= state0;
61                 elsif sROt = "01" then
62                    state <= state1;
63                 else
64                    state <= state4;
65                 end if;
66
67              when state1 =>
68                 if sROt = "01" then
69                    state <= state1;
70                 elsif sROt = "00" then
71                    state <= state2;
72                 else
73                    state <= state0;
74                 end if;
75
76              when state2 =>
77                 if sROt = "00" then
78                    state <= state2;
79                 elsif sROt = "10" then
80                    state <= state3;
81                 else
82                    state <= state1;
83                 end if;
84
85              when state3 =>
86                 if sROt = "10" then
87                    state <= state3;
88                 elsif sROt = "11" then
89                    state <= state0;
90                    sCount <= sCount + 1;
91                 else
92                    state <= state2;
93                 end if;
94
95              when state4 =>
96                 if sROt = "10" then
97                    state <= state4;
98                 elsif sROt = "00" then
99                    state <= state5;
100                else
101                   state <= state0;
102                end if;
103
104             when state5 =>
105                if sROt = "00" then
106                   state <= state5;
107                elsif sROt = "01" then
108                   state <= state6;
109                else
110                   state <= state4;
111                end if;
112
113             when state6 =>
114                if sROt = "01" then
```

```
115                  state <= state6;
116               elsif sROt = "11" then
117                  state <= state0;
118                  sCount <= sCount - 1;
119               else
120                  state <= state5;
121               end if;
122
123            when others =>    state <= state0;
124
125
126         end case;
127      end if;
128   end process;
129
130   COUNTER <= sCount;
131
132
133   end Behavioral;
134
135
```

```
1  ----------------------------------------------------------------
2  --   Spartan-3E Kit: Analog IO Component
3  --       DAC component: LTC2624  4 channel, 12 bit DAC
4  --       ADC component: LTC1407  2 channel, 14 bit ADC
5  ----------------------------------------------------------------
6
7  library IEEE;
8  use IEEE.std_logic_1164.all;
9  use IEEE.STD_LOGIC_UNSIGNED.all;
10
11
12 entity Monoflop is
13 port (
14       Clk            :  IN std_logic;
15       Trigger        :  IN std_logic;
16       PULSOUT        :  OUT   std_logic
17    );
18 end Monoflop;
19
20
21 architecture behav of Monoflop is
22    signal      TriggerF    : std_logic;
23    signal      SReset      : std_logic;
24    signal      PORes       : std_logic;
25    signal      nPO         : std_logic;
26    signal      IQ          : std_logic_vector(16 downto 0);
27    signal      Cnt         : std_logic_vector(16 downto 0);
28 begin
29
30    PULSOUT     <= TriggerF;
31    nPO         <= not TriggerF;
32    IQ          <= cnt;
33
34 COUNTERP:   process(Clk, nPO)
35             begin
36                if nPO='1' then
37                   cnt <= "00000000000000000";
38                elsif rising_edge(Clk) then
39                   cnt <= cnt + 1;
40                end if;
41             end process;
42
43 -- Einfangen des Triggers
44 CATCHTRIGP: process(SReset,Trigger)
45             begin
46                if SReset='1' then
47                   TriggerF <= '0';
48                elsif rising_edge(Trigger) then
49                   TriggerF <= '1';
50                end if;
51             end process;
52
53
54 -- Counter Ausgang Clk synchronisieren
55 SYNCCOUNTP: process(Clk,SReset)
56             begin
57                if SReset='1' then
```

```
58                      PORes <= '0';
59                   elsif rising_edge(Clk) then
60                      PORes <= IQ(16);
61                   end if;
62                end process;
63
64
65    -- Mono Reset zurücksetzen
66    RESMONOP:   process(Clk)
67                begin
68                   if rising_edge(Clk) then
69                      SReset <= PORes;
70                   end if;
71                end process;
72
73
74    end behav;
75
```

Pining User Constrains File

S3E_Pining.ucf Sun Mar 01 13:34:31 2015

```
1  ##############################################################
2  ### SPARTAN-3E STARTER KIT BOARD CONSTRAINTS FILE
3  ##############################################################
4  #
5  # in PROCESSES  / Implement Design  (Right CLick) / Properties:  Allow unmatched LOC
   constraints
6  #
7
8  # ==== Clock inputs (CLK) ====
9  NET "CLK" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
10 #NET "CLK" PERIOD = 20.0ns HIGH 40%;
11
12
13 NET "CLK_AUX" LOC = "B8" | IOSTANDARD = LVCMOS33 ;
14 NET "CLK_SMA" LOC = "A10" | IOSTANDARD = LVCMOS33 ;
15
16
17
18 # ==== Discrete LEDs (LED) ====
19 # These are shared connections with the FX2 connector
20 NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
21 NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
22 NET "LED<2>" LOC = "E11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
23 NET "LED<3>" LOC = "F11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
24 NET "LED<4>" LOC = "C11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
25 NET "LED<5>" LOC = "D11" | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
26 NET "LED<6>" LOC = "E9"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
27 NET "LED<7>" LOC = "F9"  | IOSTANDARD = LVTTL | SLEW = SLOW | DRIVE = 8 ;
28
29 # ==== Digital-to-Analog Converter (DAC) ====
30 # some connections shared with SPI Flash, DAC, ADC, and AMP
31 NET "DAC_CLR" LOC = "P8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
32 NET "DAC_CS" LOC = "N8" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 8 ;
33
34 # ==== Analog-to-Digital Converter (ADC) ====
35 # some connections shared with SPI Flash, DAC, ADC, and AMP
36 NET "AD_CONV" LOC = "P11" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
37
38 # ==== Programmable Gain Amplifier (AMP) ====
39 # some connections shared with SPI Flash, DAC, ADC, and AMP
40 NET "AMP_CS" LOC = "N7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
41 NET "AMP_DOUT" LOC = "E18" | IOSTANDARD = LVCMOS33 ;
42 NET "AMP_SHDN" LOC = "P7" | IOSTANDARD = LVCMOS33 | SLEW = SLOW | DRIVE = 6 ;
43
44 # ==== Pushbuttons (BTN) ====
45 #NET "BTN_EAST" LOC = "H13" | IOSTANDARD = LVTTL | PULLDOWN ;
46 #NET "BTN_NORTH" LOC = "V4" | IOSTANDARD = LVTTL | PULLDOWN ;
47 #NET "BTN_SOUTH" LOC = "K17" | IOSTANDARD = LVTTL | PULLDOWN ;
48 NET "BTN_WEST" LOC = "D18" | IOSTANDARD = LVTTL | PULLDOWN ;
49
```

```
285  NET "SD_LDM" LOC = "J2" | IOSTANDARD = SSTL2_I ;
286  NET "SD_LDQS" LOC = "L6" | IOSTANDARD = SSTL2_I ;
287  NET "SD_RAS" LOC = "C1" | IOSTANDARD = SSTL2_I ;
288  NET "SD_UDM" LOC = "J1" | IOSTANDARD = SSTL2_I ;
289  NET "SD_UDQS" LOC = "G3" | IOSTANDARD = SSTL2_I ;
290  NET "SD_WE" LOC = "D1" | IOSTANDARD = SSTL2_I ;
291
292
293  # Path to allow connection to top DCM connection
294  NET "SD_CK_FB" LOC = "B9" | IOSTANDARD = LVCMOS33 ;
295
296
297  # Prohibit VREF pins
298  CONFIG PROHIBIT = D2;
299  CONFIG PROHIBIT = G4;
300  CONFIG PROHIBIT = J6;
301  CONFIG PROHIBIT = L5;
302  CONFIG PROHIBIT = R4;
303
304
305  # ==== Intel StrataFlash Parallel NOR Flash (SF) ====
306  NET "SF_A<0>" LOC = "H17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
307  NET "SF_A<1>" LOC = "J13" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
308  NET "SF_A<2>" LOC = "J12" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
309  NET "SF_A<3>" LOC = "J14" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
310  NET "SF_A<4>" LOC = "J15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
311  NET "SF_A<5>" LOC = "J16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
312  NET "SF_A<6>" LOC = "J17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
313  NET "SF_A<7>" LOC = "K14" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
314  NET "SF_A<8>" LOC = "K15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
315  NET "SF_A<9>" LOC = "K12" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
316  NET "SF_A<10>" LOC = "K13" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
317  NET "SF_A<11>" LOC = "L15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
318  NET "SF_A<12>" LOC = "L16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
319  NET "SF_A<13>" LOC = "T18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
320  NET "SF_A<14>" LOC = "R18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
321  NET "SF_A<15>" LOC = "T17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
322  NET "SF_A<16>" LOC = "U18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
323  NET "SF_A<17>" LOC = "T16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
324  NET "SF_A<18>" LOC = "U15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
325  NET "SF_A<19>" LOC = "V15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
326  NET "SF_A<20>" LOC = "T12" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
327  NET "SF_A<21>" LOC = "V13" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
328  NET "SF_A<22>" LOC = "V12" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
329  NET "SF_A<23>" LOC = "N11" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
330  NET "SF_A<24>" LOC = "A11" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
331  NET "SF_BYTE" LOC = "C17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
332  NET "SF_CE0" LOC = "D16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
333  NET "SF_D<1>" LOC = "P10" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
334  NET "SF_D<2>" LOC = "R10" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
335  NET "SF_D<3>" LOC = "V9" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
336  NET "SF_D<4>" LOC = "U9" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
337  NET "SF_D<5>" LOC = "R9" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
338  NET "SF_D<6>" LOC = "M9" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
339  NET "SF_D<7>" LOC = "N9" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
340  NET "SF_D<8>" LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
341  NET "SF_D<9>" LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
```

```
342   NET "SF_D<10>" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
343   NET "SF_D<11>" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
344   NET "SF_D<12>" LOC = "M16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
345   NET "SF_D<13>" LOC = "P6" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
346   NET "SF_D<14>" LOC = "R8" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
347   NET "SF_D<15>" LOC = "T8" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
348   NET "SF_OE" LOC = "C18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
349   NET "SF_STS" LOC = "B18" | IOSTANDARD = LVCMOS33 ;
350   NET "SF_WE" LOC = "D17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
351
352
353   # ==== Xilinx CPLD (XC) ====
354   NET "XC_CMD<0>" LOC = "P18" | IOSTANDARD = LVTTL | DRIVE = 4 | SLEW = SLOW ;
355   NET "XC_CMD<1>" LOC = "N18" | IOSTANDARD = LVTTL | DRIVE = 4 | SLEW = SLOW ;
356   NET "XC_CPLD_EN" LOC = "B10" | IOSTANDARD = LVTTL ;
357   NET "XC_D<0>" LOC = "G16" | IOSTANDARD = LVTTL | DRIVE = 4 | SLEW = SLOW ;
358   NET "XC_D<1>" LOC = "F18" | IOSTANDARD = LVTTL | DRIVE = 4 | SLEW = SLOW ;
359   NET "XC_D<2>" LOC = "F17" | IOSTANDARD = LVTTL | DRIVE = 4 | SLEW = SLOW ;
360   NET "XC_TRIG" LOC = "R17" | IOSTANDARD = LVCMOS33 ;
361   NET "XC_GCK0" LOC = "H16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
362   NET "GCLK10" LOC = "C9" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
363
```