Graduate Theses, Dissertations, and Problem Reports

2006

Development of FPGA based low-power digital pulse height fitting

Santosh Hooli
West Virginia University

Follow this and additional works at: <https://researchrepository.wvu.edu/etd>

Recommended Citation

Hooli, Santosh, "Development of FPGA based low-power digital pulse height fitting" (2006). *Graduate Theses, Dissertations, and Problem Reports*. 2389.
<https://researchrepository.wvu.edu/etd/2389>

This Thesis is protected by copyright and/or related rights. It has been brought to you by the The Research Repository @ WVU with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you must obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/ or on the work itself. This Thesis has been accepted for inclusion in WVU Graduate Theses, Dissertations, and Problem Reports collection by an authorized administrator of The Research Repository @ WVU. For more information, please contact researchrepository@mail.wvu.edu.

Development of FPGA Based Low-Power Digital Pulse Height Fitting

Santosh Hooli

Thesis submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science
In
Electrical Engineering

Powsiri Klinkhachorn, Ph.D., Chair
Manfred Boehm, Ph.D.
Roy S. Nutter, Jr., Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2006

Keywords: Pulse height fitting, Pulse processing, Particle detectors

Abstract

Development of FPGA-Based Low-Power Digital Pulse Height Fitting

Santosh Hooli

Energetic ion analyzers and mass spectrometers commonly produce electronic pulses whose amplitude and time depend on the energy, mass, and direction of an incoming ion. Analysis of the nanosecond scale timing and amplitude of these approximately 10^{-12}C charge pulses has required complex, fast analog electronics. Given recent advances in digital electronics, it seems likely that converting much of the analysis to digital form will result in a simpler, lower cost, and lower power electron package. The aim of the thesis is to develop an algorithm to fit a 500 ns long noisy voltage pulse obtained from an ion analyzer to a known pulse shape, and implement it in digital electronics. The measured pulse will have an unknown amplitude, unknown starting time, and also, added random noise. The thesis includes derivation of the algorithm to least-square fit a noisy pulse to determine its height and starting time. A ratio of two pulse heights is used to identify the ion. The fitting process has been implemented using FIR filters.

The algorithm has been implemented in an FPGA which fits pulses sampled by a high-speed 8-bit ADC at 50ns sampling time. Simulation tests have been carried out using Matlab to test the accuracy of the algorithm in finding its height and starting time by varying the noise level in the signal. Results show that the algorithm can detect the pulse even when the SNR in the signal is 3dB or better, and it can detect the starting time of the pulse to within 6ns when the SNR in the signal was approximately 21dB. The pulse height found varies by just 20% when the SNR in the signal is varied from 25 to 7dB. The present algorithm implementation in hardware determines only the amplitude of the pulses. A position sensitive anode board was used to generate the pulses. The results show that the implementation can identify the ion position based on the amplitude of the pulses generated, to typical accuracy of 1%. Techniques for future improvements are also discussed.

Acknowledgements

I would like to thank Dr. Powsiri Klinkhachorn for his constant support and encouragement throughout my thesis. I would like to thank Dr. Manfred Boehm for giving me an opportunity to work under him. His guidance and advice throughout this project have helped me to complete my thesis. I would like to thank Dr. Roy S. Nutter for his invaluable suggestions. I would also like to thank all my friends for their invaluable help and support. Finally I like to thank my committee members for their time and suggestions which helped to improve my thesis.

Dedication

I would like to dedicate this thesis to my parents who have always encouraged and stood by me throughout my life. I would like to express my gratitude to my parents for their hard work and continuous support, which helped me in pursuing higher studies.

Abbreviations

FIR – Finite Impulse Response

IIR – Infinite Impulse Response

SNR – Signal-to-Noise Ratio

FPGA – Filed Programmable Gate Array

DCM – Digital Clock Manager

CLB – Configurable Logic Block

LUT – Look Up Table

RAM – Random Access Memory

ROM – Read Only Memory

PCB – Printed Circuit Board

USB – Universal Serial Bus

CAD – Computer Aided Design

Table of Contents

Abstract	ii
Acknowledgements	iii
Dedication	iv
Dedication	iv
Abbreviations	v
Table of Contents	vi
List of Tables	viii
List of Figures and Illustrations	ix
 CHAPTER ONE	 1
OVERVIEW	1
1.1 Background	1
1.2 Overview of Ion Analyzer	2
1.2.1 Microchannel plate and anode board	6
1.3 Pulse shaping and analysis	11
1.4 Description of the problem	13
1.5 Algorithm to find the pulse height	13
1.6 Research Objectives	13
1.7 Organization of thesis	15
 CHAPTER TWO	 16
LITERATURE REVIEW	16
2.1 Analog pulse height analyzers vs. Digital pulse height analyzers	16
2.2 Optimizing signal-to-noise ratio	16
2.3 Previous pulse height fitting algorithms: overview	17
2.3.1 Offline analysis methods	18
2.3.2 Online analysis methods	18
2.4 Field programmable gate Arrays (FPGA)	20
2.5 Overview of FIR filter	21
 CHAPTER THREE	 23
ALGORITHM DEVELOPMENT	23
3.1 Algorithm Description:	23
3.1.1 Fit curves	23
3.2 Derivation of the algorithm	26
3.2.1 Fitting a time-delayed pulse with respect to first fit curve	26
3.2.2 Least-square fitting the noisy pulse	28
3.2.3 Fitting a pulse with DC offset	34
3.3 Simulation of the algorithm using Matlab	35
3.3.1 Best-fit result	39
3.3.2 Result of fitting a pulse with DC offset	40

3.3.3 Detecting the presence of pulse using both the FIR filter outputs	41
3.3.4 Accuracy of the pulse height fit in simulation	50
3.4 Calculating the exact arrival time of pulse from C_1 and C_2 amplitudes	53
3.4.1 Calculating starting time for the pulse with no noise	53
3.4.2 Calculating starting time for the pulse with noise	55
CHAPTER FOUR.....	58
HARDWARE IMPLEMENTATION.....	58
4.1 Components overview	58
4.2 PCBs used for hardware implementation	59
4.3 Anode board.....	60
4.4 Main board.....	61
4.4.1 Pulse-shaping circuit	64
4.4.2 Pulse amplifier.....	67
4.4.3 Analog to Digital Converter (ADC).....	68
4.4.4 Field Programmable Gate Array (FPGA)	68
4.4.5 Algorithm implementation in FPGA.....	69
4.4.6 Maximum detection logic.....	71
4.5 PC interface board	74
4.5.1 Interface between PC interface board and FPGA	75
4.5.2 FPGA design and verification tools	76
CHAPTER FIVE	79
HARDWARE TESTING AND RESULTS.....	79
5.1 Generating pulses from anode board	79
5.2 Testing main board	80
5.2.1 Testing ADC (ADC08L060) [30]	81
5.3 Testing the algorithm in hardware	82
5.3.1 Testing of PC interface board.....	85
CHAPTER SIX.....	86
CONCLUSIONS AND FUTURE WORK.....	86
6.1 Conclusion	86
6.2 Future Work.....	88
References.....	90
APPENDIX A: VHDL PROGRAMS OF THE MAIN BOARD	94
APPENDIX B: MATLAB PROGRAMS	163
APPENDIX C: C++ AND VHDL PROGRAMS OF THE PC INTERFACE BOARD	213

List of Tables

Table 3.1: Table showing actual delay, obtained delay, and their difference.....	54
Table 3.2: Table showing actual delay, obtained delay, and the percentage difference...	56
Table 5.1: Some results of the five test regions	83
Table 5.2: Ratio values for corresponding ions	85

List of Figures and Illustrations

Figure 1.1: Cross-section of the ion analyzer and its components [24].....	2
Figure 1.2: Top view of ion analyzer showing deflection and separation of ions [24]	3
Figure 1.3: Side view of ion analyzer showing electromagnetic yoke and channel plate... [24].....	3
Figure 1.4: Ion analyzer components.....	4
Figure 1.5: Change in ion trajectory due to the magnetic field	5
Figure 1.6: Microchannel plate and anode board [13].....	7
Figure 1.7: Stacked Microchannel plates.....	8
Figure 1.8: Anode board	10
Figure 1.9: Anode board with ion regions	10
Figure 1.10: Shaped pulse without any noise	12
Figure 2.1: Conventional FIR filter implementation	22
Figure 3.1: First fit curve $F_1(n)$ and second fit curve $F_2(n)$	24
Figure 3.2: Noisy pulse	25
Figure 3.3: Fit curves moved left.....	28
Figure 3.4: FIR filter coefficients of the first fit curve	31
Figure 3.5: FIR filter coefficients of the second fit curve.....	33
Figure 3.6: Integer FIR filter coefficients of the first fit curve.....	36
Figure 3.7: Integer FIR filter coefficients of the second fit curve	36
Figure 3.8: Pulse to be detected	37
Figure 3.9: Random noise signal	38
Figure 3.10: Random noise with added pulse (to be detected).....	39

Figure 3.11: The fitted pulse with the background noisy pulse	40
Figure 3.12: Fitting a pulse with offset 0.2	41
Figure 3.13: Presence of the maximum indicates the presence of the pulse.....	42
Figure 3.14: Random noise with pulse added	44
Figure 3.15: C_1 , C_2 and C outputs.....	45
Figure 3.16: Random noise with pulse added at time $t=10000\text{ns}$	46
Figure 3.17: C_1 , C_2 and C outputs.....	47
Figure 3.18: Random noise with pulse added at time $t=10000\text{ns}$	48
Figure 3.19: Fit operation result.....	49
Figure 3.20: Variation in pulse height as the noise amplitude is increased.....	51
Figure 3.21: Variation in pulse height as the noise amplitude is increased..... (for $\text{SNR} > 6.7$)	52
Figure 3.22: Time delay δ between the pulses generated and the sampling time.....	55
Figure 3.23: Time delay δ between the pulses generated and the sampling time..... (with noise)	57
Figure 4.1: Pulse generation and processing stages.....	59
Figure 4.2: Boards used to implement the algorithm.....	59
Figure 4.3: Anode board with four wedge patterns on top of the ground plain.....	60
Figure 4.4: Pulse input	61
Figure 4.5: Schematic diagram of the pulse shaping and amplifying circuits	62
Figure 4.6: Schematic diagram of FPGA and ADCs	63
Figure 4.7: Wedge anode board with other components	64
Figure 4.7: Pulse-shaping circuit	66

Figure 4.8: Pulse amplification circuit.....	67
Figure 4.10: Algorithm implementation in FPGA.....	70
Figure 4.11: Maximum detection logic.....	72
Figure 4.12: PC interface board.....	74
Figure 4.13: Interfacing between main board and PC interface board	75
Figure 4.13: Gated clock.....	75
Figure 5.1: Generating pulses from anode board using pulse generator.....	80
Figure 5.2: ADC output and the ideal pulse comparison.....	81
Figure 5.3: Test regions on the wedge pattern of the anode board	82
Figure 5.4: Plot summarizing the results obtained from the five tests.....	84

Chapter One

OVERVIEW

1.1 Background

A mass spectrometer is a device that makes use of the difference in mass-to-charge ratio of ions to separate and measure them. Mass spectrometers are used to identify unknown ions, to find the isotopic composition of an element, and to determine the chemical and structural information of molecules [1]. Mass spectrometers ionize the sample to be analyzed and separate the individual components by applying electric and magnetic fields. A mass-spectrum of the sample is then generated, which shows all the individual components of the sample [1].

One kind of mass spectrometer is the ion analyzer which is used for the measurement of ion energy or mass entering it. The input to the ion analyzer must be ions. Energetic particles like electrons excite atoms and ions creating the phenomenon called an aurora in which different colored lights appear in the earth's upper atmosphere [2]. The present ion analyzer being built will be used for auroral research and is designed to mainly detect and identify H^+ , He^{++} , He^+ , and O^+ ions in an aurora and to determine if these particles originated from earth's atmosphere or from solar wind.

1.2 Overview of Ion Analyzer

Ion analyzers for Kilo electron Volt (KeV) ions usually measure the energy to charge and the mass to charge ratio of the ions (Solid state detectors, which use semiconductors to detect ions, can also measure the total energy for energies $>50\text{KeV}$ [12]). Figures 1.1, 1.2 and 1.3 depict the ion analyzer being built for this experiment.

Figure 1.1: Cross-section of the ion analyzer and its components [24]

Figure 1.1 shows the cross-section of the ion analyzer including the trajectory of an ion entering from the top, passing through the electric field set-up by the two spheres, passing through the magnetic field of the electromagnetic yoke, and finally hitting the channel plates (Micro channel plates).

Figure 1.2: Top view of ion analyzer showing deflection and separation of ions [24]

Figure 1.3: Side view of ion analyzer showing electromagnetic yoke and channel plate [24]

Figure 1.2 shows the deflection of the H^+ , He^{++} , He^+ , and O^+ ions by different angles due to the magnetic field of the electromagnetic yoke. Figure 1.3 shows the side view of the ion analyzer showing the ion deflecting due to the electromagnetic yoke and subsequently hitting channel plate.

Different components of the ion analyzer are shown in Figure 1.4 and are explained below. The electric field of the ion analyzer is used to select particles with certain energy-to-charge ratio. The magnetic field of the ion analyzer is used to separate the mass-to-charge ratio of ions for a given energy per charge. Whenever an ion of the right energy per charge enters the ion analyzer, it is maintained in a circular orbit between the two spheres by an electric field and then deflected by the magnetic field of the electromagnetic yoke. The amount of deflection of an ion depends on mass per charge and energy.

Figure 1.4: Ion analyzer components

Force ($\vec{F}$) experienced by an ion in an electric field ($\vec{E}$) is given by:

$$\vec{F} = q \vec{E}$$

where q is the charge on the ion. The force experienced by an ion in a magnetic field ($\vec{B}$) is given by:

$$\vec{F} = q \left(\vec{v} \times \vec{B} \right),$$

a vector operation where $\vec{v}$ is the velocity of the ion.

Ions under the influence of the magnetic field move in a circular path. Therefore, by equating the centripetal force experienced by an ion in a circular motion and the force due to the magnetic field:

$$\frac{mv^2}{r_g} = qvB \Rightarrow r_g = \frac{mv}{qB}$$

where $\frac{mv^2}{r_g}$ is the centripetal force experienced by an ion and r_g (gyroradius) is the radius of the circular motion of the ion:

$$r_g = \frac{mv}{qB} \quad (1.1)$$

The change in ion trajectory due to the magnetic field is shown in Figure 1.5. For small deflections, the angle of deflection is approximately given by $\Theta \approx 2r_m / r_g$, where r_m is the radius of the magnet.

Figure 1.5: Change in ion trajectory due to the magnetic field

Substituting the value of r_g from Equation 1.1, we get

$$\Theta = \frac{2r_m qB}{mv} = r_m \sqrt{2B} \sqrt{\frac{q}{m}} \sqrt{\frac{q}{\varepsilon}} \quad (1.2)$$

where $\varepsilon = \frac{mv^2}{2}$ is the kinetic energy of the particle and $\frac{q}{\varepsilon}$ is fixed by the preceding

electrostatic analyzer (electric field with the two spheres) [14].

From Equation 1.2, the change in angle in the trajectory of an ion per unit distance for a fixed $\sqrt{\frac{q}{\varepsilon}}$ is directly proportional to the square root of charge of the ion and the magnetic field and is inversely proportional to the square root of mass of the ion. The amount of deflection is highest for a H^+ ion followed by He^{++} , He^+ , and O^+ ions. In the planned instrument, the magnetic field will be set such that the expected angles of deflection for H^+ , He^{++} , He^+ and O^+ ions are 20° , 14° , 10° , and 5° respectively.

1.2.1 Microchannel plate and anode board

At the end of the trajectory described above, an ion impacts a microchannel plate, thereby releasing 10^7 electrons. These electrons are gathered by an anode board. Both the micro channel plate and the anode board are shown in Figure 1.6.

Figure 1.6: Microchannel plate and anode board [13]¹

A microchannel plate (MCP) is a two-dimensional periodic array of very small diameter glass tube channels fused together and sliced in a thin plate. Each of the channels is coated on the inside with an electron-emissive material, which amplifies electron signals similar to a Secondary Electron Multiplier (SEM). Some advantages of the MCP are their high gain and high spatial resolution [10].

The two ends of the channel plate are connected to a high voltage source with the top side connected to the negative voltage to attract the positively charged ions and with the bottom of the plate connected to the positive of the voltage source. A single incident radiation (ion, electron or photon) enters a channel and emits an electron from the channel wall. Secondary electrons are emitted when this electron strikes the surface and

¹ with permission from Dr. Juergen Barnstedt

are accelerated by an electric field developed by the voltage applied across both ends of the microchannel plate. They travel along the channel and, in turn, strike the channel surface, thus, producing more secondary electrons. This process is repeated many times along the channel. As a result, this cascade process yields a cloud of several thousand electrons which emerges from the rear of the plate.

If two or more MCPs are stacked in series (chevron pair of channel plates), a single input event will generate 10^7 or more electrons [12, 22]. Figure 1.7 shows the stacked MCPs or chevron pair of channel plates. The transit time of the emitted electrons through the MCP glass tube channel is of the order of 1ns [10].

Figure 1.7: Stacked Microchannel plates

1.2.1.1 Anode Board

The anode board and the MCP are separated by only a few millimeters. The electron cloud leaving MCP disperses due to repulsion between the electrons and due to the spread of $\pm 20^\circ$ in the direction of their velocity when exiting the plate. When the electron cloud reaches the anode board, current starts to flow through it, generating an electrical pulse.

There are several types of ion analyzers. Solid state detectors generate electrical pulses, whose height is proportional to the ion energy [12]. Electrostatic and magnetostatic analyzers often require determination of the final ion position. They utilize a position-sensitive anode board, whose output is proportional to the position of the ion impact on a channel plate. Determining the final position can be used to identify the ion. In this research, a second type of ion analyzer has been used to determine the ion position, which uses position sensitive anode board called wedge anode board.

Figure 1.8 shows the wedge anode board with two interleaved anodes. It has two outputs that are used to determine the position of the electron cloud on the wedge anode board. The side with more electrons generates the stronger signal, and the side with fewer electrons generates a weaker output [15]. So the location of the electron cloud on the anode board can be properly determined, the electron cloud should cover at least 2 patterns on the wedge pattern [11]. This spreading is obtained due to the spacing between the MCP and the anode board.

Figure 1.8: Anode board

Because they end at different points on the MCP, each of the ions (H^+ , He^{++} , He^+ and O^+) generates an electron cloud on a different region (approximated) of the wedge pattern which is shown in Figure 1.9.

Figure 1.9: Anode board with ion regions

If an O^+ or He^+ ion enters the ion analyzer, then a stronger signal will result at the left end and a weaker signal at the right end. Conversely, the He^{++} and H^+ ions produce a stronger output at the right end and a weaker output at the left end. The pulse height (amplitude) from each anode can be determined, and the ratio of pulse heights from these two anodes gives the 1-dimensional position of the electron cloud on the anode board. Hence, this position can be used to identify the ion.

1.3 Pulse shaping and analysis

The electron cloud of charge pulse of approximately $10^{-12}C$ generates millivolt pulses at two outputs of the anode board given that the anode capacitances are approximately 15pF. These electric pulses are amplified and shaped before they can be analyzed digitally. A pulse formed from an edge by a single pole high pass filter and a single pole low pass filter gives the better trade-off between the optimal signal-to-noise ratio and practical analog circuitry design [19]. For this experiment, both the outputs from the anode board are shaped to have an exponential rise time of 75ns and exponential fall time of 300ns. After sampling at 50ns intervals, the resulting waveform will have an approximate width of 500ns as shown in Figure 1.10.

Figure 1.10: Shaped pulse without any noise

The above plot can be expressed by Equation 1.3 which is shown below [3].

$$(1 - e^{-t_n/t_1})e^{-t_n/t_2} \quad (1.3)$$

where $t_1=75$ ns, $t_2= 300$ ns, $t_n= (n-1)*50$, and $n=1$ to 16.

Thus, the problem of ion identification reduces to detecting the pulse amplitudes coming out of the two outputs of the anode board. The process of finding the pulse height is called “pulse height fitting.”

1.4 Description of the problem

The generated pulse from the ion analyzer is usually distorted by the noise. Thus, noise should be minimized before finding the accurate pulse height. Noise gets into the system from electronic amplifiers and from the ion detector [9,19]. Due to the noise there is always some offset added to the baseline, which is the output of the detector with no ion events. When it is added to the pulse, the offset distorts the pulse shape as well as its peak.

1.5 Algorithm to find the pulse height

One method of calculating pulse height is to make use of all the pulse samples to determine its height. The advantage of this approach is that it reduces the noise in the signal by averaging [21]. An algorithm based on this approach will be used to detect the pulse and to find its height.

1.6 Research Objectives

The problem to be addressed is the development of an algorithm to detect the presence, amplitude, and timing of electronic pulses within an electronic signal coming from an ion analyzer, followed by its implementation in hardware. The filtered pulses obtained from the ion analyzer are typically 500 ns in width. They have added random noise, unknown starting times, and unknown amplitudes. The algorithm implementation will find the amplitudes of these pulses. The present algorithm should be easy to modify, and to determine the pulse starting time to within a few nano seconds.

The amplitudes of these pulses are used to identify the ion position on a position sensitive (wedge) board. The algorithm implementation will be used primarily to detect and to register the presence of H^+ , He^{++} , He^+ , and O^+ ions.

To accomplish these objectives, the following tasks will be performed:

- Investigate the algorithm to determine the pulse height in simulation. A pulse of the shape expected from an analog filter with added random noise is to be processed by the exact algorithm which will later be implemented in hardware.
 - Simulation of the algorithm will be carried out on a very long sequence of random noise signal with a voltage pulse added.
 - The algorithm will be tested for efficiency in detecting the presence of the voltage pulse by varying noise levels.
- Design and implementation of PCB with an FPGA.
 - The algorithm will be implemented in an FPGA.
- Minimize the power required to implement the algorithm in digital electronics.
- Verify data input from analog amplifiers to the digital hardware using logic analyzers and oscilloscopes.
- Verify the PCB in real-time set-up with output data transferred to PC for further analysis.
 - The algorithm implementation in hardware will be verified by analyzing the data stored in PC.

1.7 Organization of thesis

Chapter 1: This chapter discusses operation of the ion analyzer and its different components. It also explains the wedge anode board and its output and characteristics of the pulse generated (e.g. shape and duration). Pulse distortion due to noise and the research objectives are also explained.

Chapter 2: This is a literature review chapter that describes previous work on noise optimization and the different algorithms used to find the pulse height. It also explains the FPGAs and FIR filters which are parts of the algorithm implementation.

Chapter 3: Here, the algorithm of pulse height fitting using two fit curves is explained, and equations for least-squares fitting are derived. Procedures used to simulate the working of the algorithm in Matlab are explained.

Chapter 4: Each of the components used in the current implementation are explained in this chapter along with the algorithm implementation in FPGA.

Chapter 5: This chapter explains testing of the hardware boards using logic analyzers, oscilloscopes, and the Xilinx software tools. It also explains algorithm testing in hardware and the results obtained.

Chapter 6: This chapter discusses the conclusion and suggestions for future work.

Chapter Two

LITERATURE REVIEW

2.1 Analog pulse height analyzers vs. Digital pulse height analyzers

Using an analog pulse height analyzer to analyze closely lying pulses and variations of the baseline is difficult. Digital systems, on the other hand, can be configured for these varying factors relatively easily. Also, analog analysis requires complex precision circuits compared to the digital analysis systems. Due to these advantages, digital systems have been replacing the analog systems in pulse processing.

Due to the high computation time required to process the pulse height, digital systems have previously been limited to applications where pulses were detected by analog means, and only the few digital samples corresponding to a pulse were processed. With the availability of high speed processors, i.e. DSP processors, microcontrollers, and FPGAs, digital systems are used in increasingly high ion count rate applications [5, 6, 9].

2.2 Optimizing signal-to-noise ratio

Noise mainly gets into the system from the detector and the electronic amplifiers and distorts the pulse by changing its shape or its amplitude. These distortions may lead to errors in the measurement of the pulse height. Noise needs to be minimized before the pulse height can be estimated. In analog pulse detection, signal-to-noise ratio optimization can be carried out by using a low pass filter and a baseline-restoring circuit

[9]. The low pass filter filters the high frequency noise, and the baseline-restoring circuit filters the low frequency noise. A digital IIR (Infinite Impulse Response) filter can also be used to smooth the high frequency content in the signal. Though it reduces the high frequency content and helps in better height detection, like our fully digital approach, it requires multipliers and other high gate count digital logic to implement [4].

2.3 Previous pulse height fitting algorithms: overview

Techniques which analyze the stored data in a computer to find the pulse height are called “offline analysis methods,” and the techniques which use the specialized processors to do pulse height fitting in real time are called “online analysis methods.” Even though offline analysis methods are not appropriate for our application due to telemetry limitations, the offline precedents provide possible methods for our online analysis.

Some of the techniques for finding the pulse height are based on maximum search after analog noise optimization while some are based on threshold crossing or on polynomial fitting of the pulse. The factors that are considered to evaluate the pulse height fitting algorithm are the maximum ion count rate it can process in a give time, minimum pulse width it can process, and simplicity of its implementation [7, 9].

2.3.1 Offline analysis methods

2.3.1.1 Maximum search after baseline restoration

A simple maximum search after baseline restoration mechanism is used to find the pulse height [9]. Though this method provides better results compared to the simple maximum search method, the presence of closely spaced pulses introduces errors in the baseline correction mechanism.

2.3.1.2 Pulse shape factor calculation

To better detect the closely lying or overlapped pulses, the shape factor calculation method was used [8]. This method provides better results in finding and rejecting the closely spaced or piled-up pulses compared to the pulse width calculation by threshold crossing method. Pulse presence is detected by the threshold crossing method, and then the shape factor is calculated using a reference pulse. Shape factor gives the degree of closeness of the pulse to the reference pulse. If the pulse is of good shape, then a baseline restoration mechanism is used to filter the noise before calculating its height by simple peak search.

2.3.2 Online analysis methods

2.3.2.1 Moving average technique

The moving average technique integrates the pulse sample values to find the pulse height [7]. It is a threshold-free algorithm because it detects the presence of pulse height based on the pulse sample values rather than by detecting the crossing of a threshold. This

method can be used to detect the pulses when the ion count rate is high. The pulse is integrated for a fixed interval of time with the integration time equal to the width of input pulse. A pulse height is detected if the current moving average output minus the previous moving average output is greater than zero and if the subtraction of the next moving average output and the current moving average output is less than zero. The delayed moving average output is the output of the moving average technique delayed by the time equal to the samples in the integration time window.

2.3.2.2 Pulse fitting using 8th degree polynomial

A technique based on the polynomial fitting to do pulse height fitting, which uses the eighth degree polynomial, was implemented using a DSP processor [5]. This fitting procedure estimates the pulse height by taking the pulse sample values around the pulse height and fitting them with the eighth degree polynomial using an iterative equation. Because this is an iterations-based method, it requires some time before finding the pulse height. Due to this process being implemented in a DSP processor which executes the tasks serially, it takes more time compared to an FPGA-based implementation which executes the tasks in a parallel fashion. This method can process pulses in μs range only.

2.3.2.3 Pulse fitting using a fixed-point second degree polynomial

One more polynomial fitting method, which uses the function of second degree to find the pulse height, is implemented using FPGA (see Section 2.4) [4]. This is a threshold-based algorithm, and thus, it detects the presence of the pulse by detecting the crossing of

the threshold by the pulse. End of the pulse is calculated by detecting the crossing of the threshold by the pulse in the opposite direction. Each time when the pulse crosses the threshold time is recorded. The duration of the pulse is calculated by subtracting the starting time and the ending time of the pulse. Height calculations are started when the presence of the pulse is detected by threshold crossing. Height estimation calculations are started each time the pulse sample value is greater than its previous value. New height estimation values are calculated using the previous height calculations and the current sample value. This is an iterative process. This method can also process pulses in μs range.

2.4 Field programmable gate Arrays (FPGA)

Field programmable gate Array is a semiconductor device with a high number of programmable components and programmable interconnects. These programmable components can be programmed to work as basic logic gates or counters to complex math functions. FPGAs also contain other built-in blocks such as RAMs and multipliers which can be used to implement RAMs, ROMs, FIFOs or multipliers.

FPGAs can process the inputs in parallel unlike DSP processors or microcontrollers. Therefore FPGAs have been used to implement the algorithm in hardware. FPGAs are a class of ICs that can be programmed multiple times by the user. FPGAs consist of a large number of Configurable Logic Blocks (CLBs) arranged in a regular rectangular pattern to store the user's logic. CLBs consist of few of static RAM-based look up tables (LUTs)

and flip-flops to implement synchronous or combinational logic. Static RAM blocks in these LUTs store the user's logic. Because static RAM does not hold the data without power, FPGA needs to be reprogrammed every time it is powered on. For this purpose, an external EEPROM chip is used which stores the user's logic and also programs the FPGA every time it is powered on. Apart from a large number of CLBs, other built-in resources found in FPGAs are hardware multipliers, RAM blocks, clock conditioning and routing circuits, and I/O buffers. In addition, FPGAs have the interfacing logic to move the configuration bits from external sources to the SRAM LUTs. Other types of FPGAs store the user's logic in flash or in an antifuse-based cell that does not need any EEPROM chip to program it [40].

2.5 Overview of FIR filter

Finite impulse response (FIR) filters have been used to implement the algorithm in this research. A FIR filter is a digital filter used to filter a noisy signal. As the name signifies, the response of the filter is finite (i.e. the response of the filter settles down to zero after some finite time) when there is no input [17]. Conventional FIR filter implementation is shown in Figure 2.1.

The FIR filter shown in Figure 2.1 is called an 'n' tap filter, and it has 'n' filter coefficients ' a_0 ' and ' a_1 ' through ' a_n '. The implementation also requires 'n-1' delay components and 'n' multipliers and 'n-1' adders.

Figure 2.1: Conventional FIR filter implementation

2.5.1.1 Filter operation

Whenever new data comes into the filter, it is delayed through ‘n’ delays, and at each step, it is multiplied with the corresponding FIR filter coefficient. The output of the multiplier is added to the other products. The data exits the filter after going through n-1 delays.

The output of the FIR filter is given by Equation 2.1.

$$\text{Output} = D(n) * a_0 + D(n-1) * a_1 + D(n-2) * a_2 + \dots + D(0) * a_n \quad (2.1)$$

where $D(n-1)$, $D(n-2)$, ..., $D(0)$ are the previous inputs to the filter.

Some of the advantages of the FIR filter are that they are linear-phase filters and are also easy to implement. FIR filters can be designed to function as low-pass, high-pass, band-pass, or stop-band types of filter [17].

Chapter Three

ALGORITHM DEVELOPMENT

3.1 Algorithm Description:

The algorithm¹ used in this research to determine the pulse height is called “Pulse height fitting with two fit curves.” This algorithm uses two fit curves to fit the pulse using least-square fit techniques. Two fit curves are required to determine both the pulse height and its starting time. The two fit curves used are of the same shape and duration as the pulse to be detected but have different timings. The concept of pulse height fitting using two fit curves is explained in the following sections.

3.1.1 Fit curves

The ideal pulse without any noise generated by the hardware is shown in Figure 1.10. Equation 1.4 of chapter 1 can be used to represent that pulse.

$$\text{i.e.} \quad (1 - e^{-t_n/t_1})e^{-t_n/t_2} \quad (3.1)$$

where $n=1$ to 16 , $t_n = (n-1)*50\text{ns}$, $t_1=100\text{ns}$, and $t_2=300\text{ns}$. 50ns in t_n is the sampling time of the ADC. The fit curves are of the same shape and duration as the ideal pulse (without any noise distortions) to be fit. The equation for the first fit curve (F_1) is the same as Equation 3.1.

¹Algorithm suggested by Professor Manfred Boehm, Asst. Prof, Dept of Physics, WVU

i.e.
$$F_1(n) = (1 - e^{-t_n/t_1})e^{-t_n/t_2} \quad (3.2)$$

The expression for the second fit curve is given by

$$F_2(n) = (1 - e^{-t'_n/t_1})e^{-t'_n/t_2} \quad (3.3)$$

where $t'_n = (n-2)*50\text{ns}$. Due to the term t'_n , the second fit curve (F_2) lags the fit curve F_1 by 50 ns. Both the fit curves are plotted using Matlab [36] and are shown in Figure 3.1.

Figure 3.1: First fit curve $F_1(n)$ and second fit curve $F_2(n)$

Because the pulse obtained from the pulse shaping circuit is sampled by ADC at 50ns sampling time, some time difference may occur between the pulse and the first fit curve. The sampled pulse will also have some added noise. Thus, the noisy pulse with added

random noise and with some time difference with respect to the first fit curve can be represented by

$$D(n) = (1 - e^{-(t_n + \Delta t)/t_1})e^{-(t_n + \Delta t)/t_2} + \text{noise}(t) \quad (3.4)$$

where $\text{noise}(t)$ is the broadband noise and “ Δt ” is the time difference between the noisy pulse $[D(n)]$ and the first fit curve $[F_1(n)]$. All the other parameters of Equation 3.4 are the same as the first fit curve Equation 3.2. A typical noisy pulse (simulated in Matlab [36]) is shown in Figure 3.2.

Figure 3.2: Noisy pulse

3.2 Derivation of the algorithm

Consider a measured pulse $[D(n)]$ with no noise but with some time difference between it and the first fit curve $[F_1(n)]$. It can be shown that it is possible to get the exact fit for such a pulse using both the fit curves $F_1(n)$ and $F_2(n)$, provided that no leading 0 points are included in F_1 and F_2 .

3.2.1 Fitting a time-delayed pulse with respect to first fit curve

The measured pulse $[D(n)]$ is to be fitted with $C_1 * F_1(n) + C_2 * F_2(n)$, where only C_1 and C_2 are adjustable.

$$\text{i.e., } D(n) = C(1 - e^{-(t_n - \delta)/t_1})e^{-(t_n - \delta)/t_2} \quad (3.5)$$

where C is a scaling factor

$$F_1(n) = (1 - e^{-t_n/t_1})e^{-t_n/t_2} \quad (3.6)$$

$$F_2(n) = (1 - e^{-(t_n - \Delta)/t_1})e^{-(t_n - \Delta)/t_2} \quad (3.7)$$

where δ is the time difference between the first fit curve $[F_1(n)]$ and the start of the $D(n)$ pulse and Δ is the time difference between $F_1(n)$ and $F_2(n)$ which is 50 ns.

To get an exact fit, $D(n) = C_1 * F_1(n) + C_2 * F_2(n)$

$$C(1 - e^{-(t_n - \delta)/t_1})e^{-(t_n - \delta)/t_2} = C_1(1 - e^{-t_n/t_1})e^{-t_n/t_2} + C_2(1 - e^{-(t_n - \Delta)/t_1})e^{-(t_n - \Delta)/t_2}$$

$$Ce^{-(t_n - \delta)/t_2} - Ce^{-(t_n - \delta)(1/t_1 + 1/t_2)} = C_1e^{-t_n/t_2} - C_1e^{-t_n(1/t_1 + 1/t_2)} + C_2e^{-(t_n - \Delta)/t_2} - C_2e^{-(t_n - \Delta)(1/t_1 + 1/t_2)}$$

$$Ce^{-t_n/t_2}e^{\delta/t_2} - Ce^{-(t_n - \delta)(1/T)} = C_1e^{-t_n/t_2} - C_1e^{-t_n/T} + C_2e^{-t_n/t_2}e^{\Delta/t_2} - C_2e^{-t_n/T}e^{\Delta/T} \quad (3.8)$$

$$\text{where } \frac{1}{T} = \frac{1}{t_1} + \frac{1}{t_2} \Rightarrow \frac{1}{T} = \frac{1}{100\text{ns}} + \frac{1}{300\text{ns}} \Rightarrow T = 75\text{ns} \quad (3.9)$$

Equation 3.8 is satisfied if

$$Ce^{\delta/t_2} = C_1 + C_2 e^{\Delta/t_2} \quad (3.10)$$

and

$$Ce^{\delta/T} = C_1 + C_2 e^{\Delta/T} \quad (3.11)$$

Solving for C_1 and C_2 using above two Equations 3.10 and 3.11:

$$C_1 = \left[e^{\delta/t_2} - \left(\frac{e^{\delta/t_2} - e^{\delta/T}}{e^{\Delta/t_2} - e^{\Delta/T}} \right) e^{\Delta/t_2} \right] C, \text{ and}$$

$$C_2 = \left(\frac{e^{\delta/t_2} - e^{\delta/T}}{e^{\Delta/t_2} - e^{\Delta/T}} \right) C$$

From the above equations, C_1 and C_2 depend only on δ and C , and everything else is constant. By using $\delta=20\text{ns}$ (arbitrary value) and $\Delta=50\text{ns}$, $t_2=300\text{ns}$, and $t_1=100\text{ns}$, we get $C_1=0.3088C$ and $C_2=0.704C$. Thus, for an input pulse of any amplitude and delay, it is possible to get the exact fit. Two fit curves are required to find both the amplitude and starting time of the pulse.

In the above plots, the fit curves start with zero, i.e., F_2 is not equal to $(1 - e^{-(t-\Delta)/\tau_a})e^{-(t-\Delta)/\tau_b}$ at the first point. Due to this, the presence of the pulse was detected at a location '1' less than the location where it was added. If the fit curves equal Equations 3.6 and 3.7, then the above fit procedure works exactly. Fit curves that match

this description are shown below in Figure 3.3. Because most of the work was done using the old fit curves, new fit curves were not used in this research.

Figure 3.3: Fit curves moved left

3.2.2 Least-square fitting the noisy pulse

The correct fit of the noisy input pulse $[D(n)]$ with the fit curves $F_1(n)$ and $F_2(n)$ is obtained by minimizing the mean square error of the fit.

$$\sum_{n=1}^{16} (D(n) - C_1 F_1(n) - C_2 F_2(n))^2 \quad (3.12)$$

for $F_1(n) = (1 - e^{-t_n/t_1})e^{-t_n/t_2}$ and $F_2(n) = (1 - e^{-(t_n-\Delta)/t_1})e^{-(t_n-\Delta)/t_2}$

where $n=1$ to 16, $t_1=100\text{ns}$, $t_2=300\text{ns}$, $t_n = (n-1)*50\text{ns}$, and $t_n' = (n-2)*50\text{ns}$, while varying C_1 and C_2 .

At the minimum:

$$\frac{d}{dC_1} \left(\sum_{n=1}^{16} (D(n) - C_1 F_1(n) - C_2 F_2(n))^2 \right) = 0 \quad (3.13)$$

and

$$\frac{d}{dC_2} \left(\sum_{n=1}^{16} (D(n) - C_1 F_1(n) - C_2 F_2(n))^2 \right) = 0 \quad (3.14)$$

Expanding the above Equation 3.13 gives

$$\frac{d}{dC_1} \left(\sum_{n=1}^{16} (D(n)^2 + C_1^2 F_1(n)^2 + C_2^2 F_2(n)^2 - 2 D(n) C_1 F_1(n) - 2 D(n) C_2 F_2(n) + 2 C_1 C_2 F_1(n) F_2(n)) \right) = 0$$

$$0 + 2 C_1 \sum_{n=1}^{16} F_1(n)^2 + 0 - 2 \sum_{n=1}^{16} D(n) F_1(n) - 0 + 2 C_2 \sum_{n=1}^{16} F_1(n) F_2(n) = 0$$

$$2 C_1 \sum_{n=1}^{16} F_1(n)^2 - 2 \sum_{n=1}^{16} D(n) F_1(n) + 2 C_2 \sum_{n=1}^{16} F_1(n) F_2(n) = 0$$

$$2 C_1 \sum_{n=1}^{16} F_1(n)^2 + 2 C_2 \sum_{n=1}^{16} F_1(n) F_2(n) = 2 \sum_{n=1}^{16} D(n) F_1(n)$$

Rewriting the above Equation

$$\frac{C_1 \sum_{n=1}^{16} F_1(n)^2}{\sum_{n=1}^{16} F_1(n) F_2(n)} + C_2 = \frac{\sum_{n=1}^{16} D(n) F_1(n)}{\sum_{n=1}^{16} F_1(n) F_2(n)} \quad (3.15)$$

Similarly, for the second fit curve

$$C_1 \frac{\sum_{n=1}^{16} F_1(n) F_2(n)}{\sum_{n=1}^{16} F_2(n)^2} + C_2 = \frac{\sum_{n=1}^{16} D(n) F_2(n)}{\sum_{n=1}^{16} F_2(n)^2} \quad (3.16)$$

Using Equations 3.15 and 3.16 and solving for C_1 gives

$$C_1 = \frac{\sum_{n=1}^{16} D(n) F_1(n) * \sum_{n=1}^{16} F_2(n)^2 - \sum_{n=1}^{16} D(n) F_2(n) * \sum_{n=1}^{16} F_1(n) F_2(n)}{\sum_{n=1}^{16} F_1(n)^2 * \sum_{n=1}^{16} F_2(n)^2 - \left(\sum_{n=1}^{16} F_1(n) F_2(n) \right)^2} \quad (3.17)$$

After replacing constants

$$A = \sum_{n=1}^{16} F_1(n)^2 \quad \text{and} \quad B = \sum_{n=1}^{16} F_2(n)^2 \quad \text{and} \quad T = \left(\sum_{n=1}^{16} F_1(n) F_2(n) \right) \quad (3.18)$$

$$C_1 = \frac{\sum_{n=1}^{16} D(n) F_1(n) * B - \sum_{n=1}^{16} D(n) F_2(n) * T}{A * B - T^2}$$

The above equation can be reduced to

$$C_1 = \frac{\sum_{n=1}^{16} D(n) (F_1(n) * B - F_2(n) * T)}{A * B - T^2} \quad (3.19)$$

$$C_1 = \sum_{n=1}^{16} D(n) a(n) \quad (3.20)$$

From the above equation we can see that C_1 is the FIR operation between $D(n)$ and $a(n)$.

$$\text{where } a(n) = \frac{(F_1(n) * B - F_2(n) * T)}{A * B - T^2} \text{ is not dependent on data.} \quad (3.21)$$

Equation 3.21 gives the optimized coefficients for the first fit curve. From Equations 3.2 and 3.3 $F_1(n)$ and $F_2(n)$ are given by

$$F_1(n) = [0 \quad 0.3331 \quad 0.4529 \quad 0.4712 \quad 0.4439 \quad 0.3989 \quad 0.3496 \quad 0.3020 \quad 0.2588 \\ 0.2207 \quad 0.1876 \quad 0.1592 \quad 0.1350 \quad 0.1144 \quad 0.0969 \quad 0.0820] \quad (3.22)$$

$$F_2(n) = [0 \quad 0 \quad 0.3331 \quad 0.4529 \quad 0.4712 \quad 0.4439 \quad 0.3989 \quad 0.3496 \quad 0.3020 \\ 0.2588 \quad 0.2207 \quad 0.1876 \quad 0.1592 \quad 0.1350 \quad 0.1144 \quad 0.0969] \quad (3.23)$$

Using these values of $F_1(n)$ and $F_2(n)$ in Equation 3.21

$$a(n) = [0 \quad 2.4398 \quad 1.0007 \quad 0.3005 \quad -0.0262 \quad -0.1663 \quad -0.2147 \quad -0.2197 \quad -0.2055 \\ -0.1840 \quad -0.1609 \quad -0.1388 \quad -0.1189 \quad -0.1013 \quad -0.0861 \quad -0.0731] \quad (3.24)$$

Coefficients of the first FIR filter $[a(n)]$ are shown (plotted using Matlab [36]) in Figure 3.4.

Figure 3.4: FIR filter coefficients of the first fit curve

Similarly, solving for C_2 using Equations (3.15) and (3.16):

$$C_2 = \frac{\sum_{n=1}^{16} D(n)F_2(n) * \sum_{n=1}^{16} F_1(n)^2 - \sum_{n=1}^{16} D(n)F_1(n) * \sum_{n=1}^{16} F_1(n)F_2(n)}{\sum_{n=1}^{16} F_1(n)^2 * \sum_{n=1}^{16} F_2(n)^2 - \left(\sum_{n=1}^{16} F_1(n)F_2(n) \right)^2} \quad (3.25)$$

$$C_2 = \frac{\sum_{n=1}^{16} D(n)F_2(n) * A - \sum_{n=1}^{16} D(n)F_1(n) * T}{A * B - T^2}$$

where A, B, and T are given in Equation 3.18.

The above equation can be reduced to

$$C_2 = \frac{\sum_{n=1}^{16} D(n)(F_2(n) * A - F_1(n) * T)}{A * B - T^2} \quad (3.26)$$

$$C_2 = \sum_{n=1}^{16} D(n)b(n) \quad (3.27)$$

$$\text{where } b(n) = \frac{(F_2(n) * B - F_1(n) * T)}{A * B - T^2} \text{ is also not dependent on data.} \quad (3.28)$$

Thus, C_2 is the FIR operation between $D(n)$ and $b(n)$ (from Equation 3.27). Using the values of $F_1(n)$ and $F_2(n)$ given in Equation 3.22 and 3.23 in Equation 3.28

$$\begin{aligned} b(n) = & [0 \quad -2.3172 \quad -0.6989 \quad 0.0565 \quad 0.3807 \quad 0.4931 \quad 0.5051 \quad 0.4726 \quad 0.4232 \\ & 0.3701 \quad 0.3194 \quad 0.2735 \quad 0.2331 \quad 0.1981 \quad 0.1681 \quad 0.1425] \end{aligned} \quad (3.29)$$

The above equation gives the optimized FIR filter coefficients for the second fit curve, which are shown in Figure 3.5.

Figure 3.5: FIR filter coefficients of the second fit curve

From Equations 3.20 and 3.27, values of C_1 and C_2 are in standard FIR form, where $D(n)$ is the incoming data and $a(n)$ and $b(n)$ are the filter coefficients. Thus, the problem of fitting is reduced to implementing two FIR filters, which serve the purpose of averaging the incoming data as well as fitting. This is one of the main advantages of the algorithm implemented in this research.

3.2.3 Fitting a pulse with DC offset

In the presence of the DC offset in the input signal, Equation 3.17 changes to the following equation

$$C_1 = \frac{\sum_{n=1}^{16} (D(n)F_1(n) - GF_1(n)) * \sum_{n=1}^{16} F_2(n)^2 - \sum_{n=1}^{16} (D(n)F_2(n) - GF_2(n)) * \sum_{n=1}^{16} F_1(n)F_2(n)}{\sum_{n=1}^{16} F_1(n)^2 * \sum_{n=1}^{16} F_2(n)^2 - \left(\sum_{n=1}^{16} F_1(n)F_2(n) \right)^2}$$

where G is a constant offset.

$$C_1 = \frac{\sum_{n=1}^{16} (D(n) - G)(F_1(n) * B - F_2(n) * T)}{A * B - T^2} \quad (3.30)$$

where A, B, and T are given in Equation 3.18.

Similarly, Equation 3.25 changes to

$$C_2 = \frac{\sum_{n=1}^{16} (D(n)F_2(n) - GF_2(n)) * \sum_{n=1}^{16} F_1(n)^2 - \sum_{n=1}^{16} (D(n)F_1(n) - GF_1(n)) * \sum_{n=1}^{16} F_1(n)F_2(n)}{\sum_{n=1}^{16} F_1(n)^2 * \sum_{n=1}^{16} F_2(n)^2 - \left(\sum_{n=1}^{16} F_1(n)F_2(n) \right)^2}$$

$$C_2 = \frac{\sum_{n=1}^{16} (D(n) - G)(F_2(n) * A - F_1(n) * T)}{A * B - T^2} \quad (3.31)$$

where A, B, and T are given in Equation 3.18.

Thus, Equations 3.30 and 3.31 are the same as Equations 3.19 and 3.25 which are used to fit a pulse without any offset except for the presence of the offset term 'G'. If the offset in the signal is known, then the pulse with offset is fit using Equations 3.30 and 3.31 using the FIR filter coefficients a(n) and b(n) given in Equation 3.24 and 3.29 respectively.

3.3 Simulation of the algorithm using Matlab

Algorithm testing in Matlab was carried out by generating 20,000 samples of random noise signal using Matlab's "randn" function. This generated white noise up to the Nyquist frequency. To this random noise, a pulse shown in Figure 1.10 in chapter 1 was added at an arbitrary location. This signal represents a long sequence of random noise with the pulse added whose presence should be detected. The algorithm for pulse height fitting with two fit curves was applied to the noise signal. An attempt was made to emulate the hardware implementation (all integer operations) so that the results are similar to the results obtained from the hardware implementation.

Following are the steps involved in the simulation of the algorithm:

Step 1: Values of coefficients of first and second FIR filter $a(n)$ and $b(n)$ given in Equations 3.24 and 3.29 are implemented in Matlab. All the individual values are multiplied by 100 and rounded to the nearest integer. This is done to emulate the actual hardware implementation. FIR filter coefficients $a(n)$ and $b(n)$ are shown in Figures 3.6 and 3.7 respectively.

Figure 3.6: Integer FIR filter coefficients of the first fit curve

Figure 3.7: Integer FIR filter coefficients of the second fit curve

Step 2: The pulse to be detected is generated using Equation 3.2 of this chapter. The resulting pulse values were normalized (all pulse values were divided by the maximum value in the pulse) and the resulting pulse is shown in Figure 3.8.

Figure 3.8: Pulse to be detected

Step 3: A random noise signal was generated using the “randn” function available in Matlab. The randn(size(t)) function generates an array of random numbers with a Gaussian distribution and with a mean zero and variance 1. The total number of random numbers generated is specified by size(t) function. Size of “t” was set to 20,000 points. To set the noise amplitude level to some expected range, the noise signal was divided by a positive number. Figure 3.9 shows a typical noise signal generated.

Figure 3.9: Random noise signal

Step 4: The pulse shown in Figure 3.8 was added to the random noise. Figure 3.10 shows the resulting random noise with the pulse added (only a few points around the pulse location are shown).

Figure 3.10: Random noise with added pulse (to be detected)

Step 5: Now, the pulse height fitting algorithm is implemented on the noisy random signal. FIR operation of this noisy signal with the first fit curve (F_1) results in C_1 , and FIR operation of this noisy signal with the second fit curve (F_2) results in C_2 .

3.3.1 Best-fit result

To find the best fit pulse height, the outputs C_1 and C_2 are multiplied with their corresponding fit curves F_1 and F_2 . The result of $C_1 * F_1 + C_2 * F_2$ operation is shown in Figure 3.11 along with the noisy pulse. From the figure, we can see that the best-fit result follows the noisy pulse closely. $C_1 * F_1$ and $C_2 * F_2$ are also shown in Figure 3.11.

Figure 3.11: The fitted pulse with the background noisy pulse

3.3.2 Result of fitting a pulse with DC offset

Pulse height fitting a pulse with DC offset is obtained using Equations 3.29 and 3.30. DC offset in the signal is found by first detecting the pulse location. Averaging a selected number of samples before and after the pulse location gives an approximate offset. Subtracting the offset found from the pulse and applying the pulse height fitting algorithm gives the best-fit result for a pulse with DC offset which is shown in Figure 3.12.

Figure 3.12: Fitting a pulse with offset 0.2

3.3.3 Detecting the presence of pulse using both the FIR filter outputs

The results from the two FIR filters can be used to detect the presence of the pulse. This is done by adding the C_1 and C_2 outputs of the first and the second FIR filter respectively and then checking for the presence of the maximum.

To test this concept, a noisy pulse shown in Figure 3.10 was input to both the FIR filters. The results C_1 and C_2 of the first and the second FIR filter respectively are added which gives the fitted height C . The presence of maximum in C is detected by comparing the C output with its previous four and successive four outputs and also with a positive constant

number. Figure 3.13 shows the maximum obtained using the above method for pulse input shown in Figure 3.10.

Figure 3.13: Presence of the maximum indicates the presence of the pulse

Hence, this method can be used to detect the presence of the pulse, and the fitted amplitude C gives an approximate pulse height. Three test cases were used to test this concept. In the first test case, the random noise amplitude level was kept very small compared to the pulse amplitude to which the pulse to be detected shown in Figure 3.8 was added. Pulse height fitting algorithm was applied on that signal, and the presence of maximum was checked in the resulting fitted amplitude C_1+C_2 . In the second test, the amplitude of the random noise signal was increased so that it is comparable to the pulse amplitude. Again the pulse detection is carried out by checking for the presence of a maximum in fitted amplitude C_1+C_2 . In the third test case, the amplitude of noise was

increased further so that it is more than the pulse amplitude. Again the presence of maximum was checked in the fitted amplitude C_1+C_2 . SNR was calculated for each of the test cases. SNR in a signal is given by [27]

$$SNR (dB) = 20 * \log_{10} \left(\frac{RMS \text{ of pulse}}{RMS \text{ of noise}} \right) \quad (3.31)$$

Thus the SNR in the signal was calculated by calculating the RMS of the pulse (all pulse samples) and the RMS of noise (all the samples) and then using Equation 3.31.

3.3.3.1 Test case 1: Small noise level compared to the pulse amplitude

A random noise signal with small noise amplitude values compared to the pulse amplitude was generated so that the SNR of the resulting noisy pulse is high. The resulting noise signal with the pulse added is shown in Figure 3.14 (only a few points where the pulse is added are shown). The SNR of the noise signal with pulse added was approximately 28dB.

Figure 3.14: Random noise with pulse added

Then, the algorithm of pulse height fitting with two fit curves was applied on this noisy pulse. The fit operation results obtained C_1 , C_2 , and C (C_1+C_2) and are shown in Figure 3.15. From Figure 3.15, we can see that a maximum (in the C plot) is at the exact location or the precise time of the input pulse.

Figure 3.15: C_1 , C_2 and C outputs

3.3.3.2 Case 2: Noise level comparable to the amplitude of pulse

The amplitude of the noise was increased further so that the noise amplitude is comparable to the amplitude of the pulse. Figure 3.16 shows the noise signal with the pulse added (only a few points around the pulse location are shown). The SNR of the noise signal with pulse added was approximately 2.5dB. The fit operation results obtained C_1 , C_2 , and C (C_1+C_2) and are shown in Figure 3.17.

Figure 3.16: Random noise with pulse added at time $t=10000\text{ns}$

Figure 3.17: C_1 , C_2 and C outputs

From Figure 3.17, we can see that a maximum (in the C plot) is at the exact location or the precise time of the input pulse.

3.3.3.3 Case 3: Noise level more than the amplitude of pulse

The amplitude of the noise was increased further so that the noise value is more than the amplitude of the pulse. The resulting signal with the pulse added is shown in Figure 3.18 (only a few points are shown around the pulse location). The SNR of the noise signal with pulse added was -3dB.

Figure 3.18: Random noise with pulse added at time $t=10000\text{ns}$

Figure 3.19: Fit operation result

The fit operation results (shown in Figure 3.19) revealed many peaks around the expected pulse location. The maximum was not obtained at the expected location but before and after it. Therefore, the SNR of the noisy pulse should be roughly equal to 3dB or more than that, in order to properly detect the pulse. Thus, the algorithm can be verified using Matlab.

3.3.4 Accuracy of the pulse height fit in simulation

To test the accuracy of the pulse height fit, the fitting simulation was run for data consisting of a fixed pulse amplitude with variable noise and delayed by 20ns with respect to the first fit curve. The noise amplitude was increased in steps, and a new random noise pattern was used at each step.

C_1 and C_2 results obtained from the fitting operation were added to get the fitted pulse height C . The fitted height was normalized to 1. This is done to scale the pulse height to be around the pulse amplitude of 1.

In some of the iterations, the algorithm failed to detect the pulse location properly, hence, returning the wrong pulse height. This is due to the presence of a maximum, one location before the expected maximum location. To avoid finding the wrong pulse location, a modified method was used to detect the exact pulse location.

The ‘greater than’ condition was changed to ‘greater than or equal to’ when comparing the present C output with its previous four and successive four outputs. This was done to avoid the rejection of the proper maximum when the neighboring location also has the same amplitude level. Using this method presence of maximum was found. The maximum location found is usually the approximate pulse location (one location before the exact pulse location) or the exact pulse location. To determine whether this is the exact or the approximate pulse location, the minimum in $|C_1 - C_2|$ was also checked. $|C_1 -$

$C_2|$ obtained at the maximum location found is compared with $|C_1-C_2|$ value at the next location. The location with the minimum $|C_1-C_2|$ is the exact pulse location.

As the noise amplitude is increased, the variation in the pulse height found is plotted in Figure 3.20 for 950 iterations using the modified logic for finding the exact pulse location. Pulse amplitude found was normalized to 1.

Figure 3.20: Variation in pulse height as the noise amplitude is increased

Algorithm failed to detect the presence of the pulse location properly when the SNR was less than 6.7. Figure 3.21 shows the variation in pulse height found for SNR in the signal greater than 6.7.

Figure 3.21: Variation in pulse height as the noise amplitude is increased (for SNR > 6.7)

The maximum and the minimum pulse height that were found are 1.2 and 0.8, respectively. In Figure 3.21, the new modified logic to find the exact pulse location also gave 42 (i.e. 5% out of 850 locations) wrong locations (all of which were one location before the exact pulse location).

3.4 Calculating the exact arrival time of pulse from C_1 and C_2 amplitudes

Dividing the Equations 3.10 and 3.11 (chapter 3), we have

$$\frac{e^{\delta/t_2}}{e^{\delta/T}} = \frac{C_1 + C_2 e^{\Delta/t_2}}{C_1 + C_2 e^{\Delta/T}}$$

By substituting t_2 from Equation 3.1, Δ from Equation 3.7, and T from Equation 3.9,

$$e^{\delta/300 - \delta/75} = \frac{C_1 + C_2 * 1.1814}{C_1 + C_2 * 1.9477}$$

Applying log on both the sides

$$\delta/300 - \delta/75 = \ln(C_1 + C_2 * 1.1814) - \ln(C_1 + C_2 * 1.9477)$$

$$\delta = 100[\ln(C_1 + C_2 * 1.9477) - \ln(C_1 + C_2 * 1.1814)] \quad (3.32)$$

Equation 3.32 gives the time at which the pulse arrives with respect to the first fit curve (F_1). To test the accuracy of the algorithm in finding the exact arrival time of the pulse, the following procedure was used. Pulses with a time difference of 1ns to 49ns with respect to the sampling time were generated. Two tests were carried out. In the first test, no noise was added to the pulses, and in the second test, a small noise amplitude level compared to the pulse amplitude was added (SNR in the signal was approximately 22). The pulses generated with 1-49ns delays with respect to sampling time were fit using both the fit curves, and the exact location of the pulse was found using the logic discussed in the above Section 3.3.4.

3.4.1 Calculating starting time for the pulse with no noise

In this case, the modified logic was able to detect the exact pulse location for all the delays. Table 3.1 shows the obtained delays.

Actual delay in ns	Obtained delay in ns	Difference between actual and obtained delay
1	0.554	0.446
2	1.442	0.558
3	1.965	1.035
4	3.894	0.106
5	3.894	1.106
6	5.556	0.444
7	5.556	1.444
8	7.182	0.818
9	7.723	1.277
10	9.878	0.122
11	10.32	0.680
12	12.106	-0.106
13	12.321	0.679
14	14.013	-0.013
15	14.12	0.880
16	15.998	0.002

Actual delay in ns	Obtained delay in ns	Difference between actual and obtained delay
17	15.998	1.002
18	17.385	0.615
19¹	17.385	1.615
20	19.255	0.745
21	21.233	-0.233
22	21.234	0.766
23	22.716	0.284
24	24.092	-0.092
25	24.093	0.907
26	25.899	0.101
27	26.095	0.905
28	27.337	0.663
29	28.502	0.498
30	30.212	-0.212
31	30.381	0.619
32	31.569	0.431

Actual delay in ns	Obtained delay in ns	Difference between actual and obtained delay
33	32.659	0.341
34	33.188	0.812
35	34.35	0.650
36	35.393	0.607
37	36.917	0.083
38	37.993	0.007
39	38.01	0.99
40	39.538	0.462
41	40.723	0.277
42	41.662	0.338
43	43.032	-0.032
44	43.944	0.056
45	44.843	0.157
46	46.171	-0.171
47	47.034	-0.034
48	47.962	0.038
49	49.137	-0.137

Table 3.1: Table showing actual delay, obtained delay, and their difference

¹The maximum difference between the actual and the obtained delay is shown in bold.

Figure 3.22 shows the plot of delays shown in Table 3.1. From Figure 3.22 we can observe that the obtained delay δ shows an almost linear relationship with the actual delay. The maximum difference between the obtained delay δ and the actual delay was 1.6153ns.

Figure 3.22: Time delay δ between the pulses generated and the sampling time

3.4.2 Calculating starting time for the pulse with noise

At this point, the noise was added to the pulses generated, and the starting time of the pulse was calculated using the method explained above. Each time, a new random noise pattern with RMS amplitude of 5% of the pulse amplitude was added to the pulse (SNR of the resulting noisy signal with pulse added was approximately 21dB). Table 3.2 shows

the obtained and actual delays along with the percentage error [23] for each delay calculated.

Actual delay(ns)	Obtained delay(ns)	Difference between actual and obtained delay
1	-0.524*	1.524
2	0.821	1.179
3	4.397	-1.397
4	7.921	-3.921
5	3.223	1.777
6	8.136	-2.136
7	3.796	3.204
8	5.511	2.489
9	4.555	4.445
10	12.379	-2.379
11	9.467	1.536
12	9.054	2.946
13	15.265	-2.265
14	18.352	-4.352
15	14.367	0.633
16	18.852	-2.852

Actual delay(ns)	Obtained delay(ns)	Difference between actual and obtained delay
17	16.285	0.715
18	17.007	0.993
19	21.215	-2.215
20	18.76	1.240
21	22.455	-1.455
22	16.08	5.920
23	22.236	0.764
24	23.609	0.391
25	23.846	1.154
26	23.765	2.235
27	24.15	2.85
28	22.497	5.503
29	25.918	3.082
30	29.063	0.937
31	32.713	-1.713
32	31.353	0.647

Actual delay(ns)	Obtained delay(ns)	Difference between actual and obtained delay
33	32.744	0.256
34	27.207	6.793
35	33.618	1.382
36	32.707	3.293
37	39.046	-2.046
38	38.283	-0.282
39	38.975	0.025
40	38.255	1.745
41	41.128	-0.127
42	40.45	1.55
43	44.197	-1.197
44	42.324	1.676
45	46.599	-1.599
46	46.062	-0.062
47	47.19	-0.19
48	48.716	-0.714
49	48.343	0.657

Table 3.2: Table showing actual delay, obtained delay, and the percentage difference

¹The starred and bold delay entry in the table signifies wrong detection of pulse location by the modified algorithm, value of which was calculated using expected maximum location values

Figure 3.23 shows the plot of delays shown in Table 3.2. The plot shows the expected linear relationship between the obtained delay (δ) and the delay between the pulse and the first fit curve. The maximum difference between the obtained and the actual delay was 5.92ns. Standard deviation of the δ -actual delay line is 2.4736. Figure 3.23 also shows the zone where the starting time of a pulse will be detected.

Figure 3.23: Time delay δ between the pulses generated and the sampling time (with noise)

Chapter Four

HARDWARE IMPLEMENTATION

4.1 Components overview

Hardware implementation processes the pulses generated by the ion analyzer in real-time and sends the pulse heights determined to PC. Each of the pulses are shaped and amplified to give a pulse shape as shown in Figure 1.10 of chapter 1. A field programmable gate array (FPGA) has been used in this research to implement the algorithm in hardware. FPGA detects the presence of the pulses in the incoming signal and determines their height. The sum of two FIR filters is used to detect the presence of the pulse in FPGA (this concept has been explained in Section 3.3.3 of chapter 3). Figure 4.1 shows all the components that are required to generate and process the pulses and send the pulse heights found to PC.

Figure 4.1: Pulse generation and processing stages

4.2 PCBs used for hardware implementation

Three PCB boards that were used to accommodate all the components shown in Figure 4.1 (except ion analyzer) are shown below in Figure 4.2. The anode board generates the pulses, the main board is used for processing the pulses (FPGA board), and the third board is used to temporarily store, format, and transfer the results to PC through the USB port. Data is stored in PC for later analysis.

Figure 4.2: Boards used to implement the algorithm

4.3 Anode board

In this research, an anode board with four wedge patterns designed using Eagle PCB CAD software is shown in Figure 4.3. Four wedge patterns are used to gather the electron cloud at four different parts of the ion analyzer. The four wedge patterns will be located on top of a ground plain.

Figure 4.3: Anode board with four wedge patterns on top of the ground plain

Due to the presence of the wedge pattern on top of the ground plain, a virtual capacitor is formed between the wedge plain and the ground plain. Due to this capacitor, the electron cloud from the anode board discharges slowly, generating a ramp at its output. For this research, the output of the anode board can be approximated to be a pulse as shown in Figure 4.4.

Figure 4.4: Pulse input

4.4 Main board

The schematics of the main board are shown in Figures 4.5 and 4.6 (the main schematics are divided into two with the first schematic showing the pulse shaping circuits and voltage generating circuits and the second schematic showing the FPGA and ADCs). The schematic and the layout are done using Eagle CAD software [33]. The following circuit components were designed on the Main board.

- 1) The pulse-shaping and pulse-amplifying circuits
- 2) The 8 bit ADC (analog-to-digital converters) for converting the shaped pulses into digital data
- 3) FPGA to implement the two fit curve algorithm on a total of 4 digital inputs coming from 2 separate wedge anodes
- 4) Because FPGA requires three different voltage supplies, the main board also has circuits to generate these different voltages
- 5) Timing counters, whose values are used to time stamp the events
- 6) On board EEPROM is used to store the FPGA configuration data
- 7) A DC-DC converter which converts 28V input into 5V

Figure 4.5: Schematic diagram of the pulse shaping and amplifying circuits

Figure 4.6: Schematic diagram of FPGA and ADCs

Figure 4.7 shows a wedge pattern of the anode board with two outputs, two pulse shaping circuits, and two analog-to-digital converters (ADCs). Outputs of the wedge anode board are shaped and amplified by the pulse shaping and amplifying circuits. Resulting pulse is sampled by ADC at every 50ns time interval. Each of the components shown in Figure 4.7 are explained in the following sections.

Figure 4.7: Wedge anode board with other components

4.4.1 Pulse-shaping circuit

The pulse-shaping circuit converts the pulse input (shown in Figure 4.4) obtained from the anode board into a voltage pulse as shown in Figure 1.10 of Chapter 1. Because the input and the desired output are known, we can write the transfer function for the circuit,

$$\frac{V_o}{V_{in}} = (1 - e^{-t_n/t_1})e^{-t_n/t_2}$$

where $t_n = (n-1)*50\text{ns}$, $t_1=100\text{ns}$, and $t_2=300\text{ns}$ (Equation 3.1).

This can be rewritten as

$$\frac{V_o}{V_{in}} = (1 - e^{-t_n \left(\frac{1}{t_f} - \frac{1}{t_2} \right)}) e^{-t_n / t_2}$$

where $\frac{1}{t_f} - \frac{1}{t_2} = \frac{1}{t_1}$ (4.1)

$$\frac{V_o}{V_{in}} = (1 - e^{-t_n \left(\frac{1}{t_f} - \frac{1}{t_2} \right)}) e^{-t_n / t_2}$$

$$\frac{V_o}{V_{in}} = (e^{-t_n / t_2} - e^{-t_n / t_f})$$

Applying Laplace transforms [26] we have

$$\frac{V_o(s)}{V_{in}(s)} = \frac{\left(\frac{1}{s + \frac{1}{t_2}} - \frac{1}{s + \frac{1}{t_f}} \right)}{1/s}$$

$$\frac{V_o(s)}{V_{in}(s)} = s \left(\frac{t_2}{st_2 + 1} - \frac{t_f}{st_f + 1} \right)$$

$$\frac{V_o(s)}{V_{in}(s)} = s \left(\frac{st_f t_2 + t_2 - st_f t_2 - t_f}{s^2 t_2 t_f + st_f + st_2 + 1} \right)$$

Substituting $s = j\omega$

$$\frac{V_o}{V_{in}} = j\omega \left(\frac{t_2 - t_f}{1 - (\omega t_2)(\omega t_f) + j(\omega t_f + \omega t_2)} \right)$$

Substituting $t_2 = R_2 C_2$ and $t_f = R_f C_f$

$$\frac{V_o}{V_{in}} = -j\omega \left(\frac{R_f C_f - R_2 C_2}{1 - (\omega R_f C_f)(\omega R_2 C_2) + j(\omega R_2 C_2 + \omega R_f C_f)} \right)$$

The transfer function obtained is the same as the transfer function of the standard practical differentiator op-amp circuit [18] shown in Figure 4.7.

Figure 4.7: Pulse-shaping circuit

From the pulse Equation 3.1, we have

$$t_2 = R_2 C_2 = 300\text{ns} \text{ and } t_1 = R_1 C_1 = 100\text{ns}$$

From Equation 4.1

$$\frac{1}{t_f} = \frac{1}{t_1} + \frac{1}{t_2} = \frac{1}{100} + \frac{1}{300}$$

$$\frac{1}{t_f} = \frac{4}{300} \Rightarrow t_f = 75\text{ns} = R_f C_f$$

Selecting $C_1 = 10\text{pF}$ then $R_1 = 30\text{k}\Omega$

Taking $R_f > R_1$ so that the amplification factor R_f / R_1 is greater than 1, and

selecting $R_f = 37.5\text{k}\Omega$ which gives $C_f = \frac{75 * 10^{-9}}{37.5\text{k}} = 2\text{pF}$

LMH 6646[29], a dual op-amp IC national semiconductor , is used for pulse shaping. It has a -3db bandwidth of 55MHz, slew rate specification of 22V/ μs , and supply voltage range from 2.5v to 12v.

4.4.2 Pulse amplifier

The pulse shaping circuit output is followed by a simple amplifier stage as shown in Figure 4.8 [18]. Output of the pulse amplifier shown in Figure 4.9 is given

$$\text{by } V_{out} = \left(1 + \frac{R_2}{R_1}\right) V_{in}.$$

Figure 4.8: Pulse amplification circuit

Because the maximum analog input to ADC was set to 2V and the output of the pulse shaping circuit obtained was approximately 0.5V, the amplifier was designed with $R_2=3k$ ohm and $R_1=1k$ ohm which gave the maximum output of 2V. LMH 6646 IC (a dual op-amp IC) is used to implement this circuit also [29].

4.4.3 Analog to Digital Converter (ADC)

Amplified pulses are converted into digital data using ADC08L060 ADC [30]. This is a flash type of ADC with a chip track-and-hold circuit. ADC08L060 is a single channel ADC with 8-bit parallel output and with a maximum sampling frequency of 60MHz. ADC08L060 gives out new digital outputs for every new positive clock transition. FPGA generates the ADC clock and is set at 20MHz; therefore, new data is available after every 50ns ($1/20\text{MHz} = 50\text{ns}$).

4.4.4 Field Programmable Gate Array (FPGA)

A general introduction about the FPGAs has been given in Section 2.4 of chapter 2. Spartan-3 XC3S400 FPGA from Xilinx [32] is used in this research.

- This FPGA consists of 896 CLBs (configurable logic blocks), and each CLB consists of 8 LUTs (look up tables). LUTs are 16×1 static RAM blocks which store the logic for any 4-bit variable function. They are followed by flip-flops to store the output state of the logic.
- 16 RAM blocks are built in this FPGA and each can be independently configured to work as RAM, ROM, or FIFO. Each memory block is made up of 18k bits each.

- Out of 208 I/O pins of this FPGA, 141 pins can be used by the user. Each I/O pin can be configured to act as an input, an output, or a bidirectional. Sets of I/O pins can be programmed to work at different voltage standards.
- The clock events are synchronized at different points of the FPGA by using DCM (digital clock manager) and dedicated clock routing lines provided for routing clocks in the FPGA. DCM also generates multiples of or fractions of the input clock frequency.

4.4.5 Algorithm implementation in FPGA

The simplified block diagram to implement the “Pulse height fitting with two fit curves” algorithm in hardware is shown in Figures 4.10. Digital output obtained by each ADC is input to two FIR filters, and both the FIR filters together implement the fitting algorithm. FIR filter 1 with coefficients $a(n)$ given by Equation 3.24 calculates the coefficient C_1 [given by Equation 3.20], and the FIR filter 2 with coefficients $b(n)$ given by Equation 3.29 calculates the coefficient C_2 [given by Equation 3.27]. The presence of maximum in $C_1 + C_2$ with C_1 and C_2 being positive, indicates the presence of the pulse and also gives the approximate amplitude of the pulse.

The other two FIR filters take inputs from the other ADC and implement the fitting algorithm on the other output of the wedge anode board. Presence of a maximum from any output end of the wedge anode board is used to generate a FIFO (first in first out) enable signal. This signal stores 40 bits into FIFO of which 26 bits are the two pulse heights found from two wedge anode outputs, 10 bits are the timing counter (External to FPGA) values which are used to time stamp the pulse events, 2 bits indicate the wedge

anode from which the maximum was detected (4 wedge patterns are used in this research) and 2 bits to represent any overflow from two of the ADCs. Overflow of an ADC is detected by comparing its output with 256. At regular intervals, the FIFO data are sent to PC interface board (PC interface board is shown in Figure 4.2).

Figure 4.10: Algorithm implementation in FPGA

FIR filters are the main parts for detecting the pulse. A general introduction about the FIR filters is given in Section 2.5 of chapter 2. A FIR filter macro was generated using the core generator tool available from the Xilinx software. The architecture of the core generated, is suitable to the FPGA architecture. Therefore, it requires fewer gates to implement the FIR filter in an FPGA compared to the conventional implementation [26]. The first FIR filter macro was generated with 16 taps (corresponding to the 16 points used to represent the data pulse) and with coefficients given in Equation 3.23. The second FIR filter macro was also generated with the same parameters but with the coefficients given in Equation 3.29.

4.4.6 Maximum detection logic

The presence of maximum in the fitted amplitude (A_1 or B_1 shown in Figure 4.10) indicates the presence of the pulse in the signal (this concept has been explained in Section 3.3.3). Logic used to detect the presence of a maximum is shown in Figure 4.11. From the Figure 4.11 we can see that the presence of a maximum is found by comparing the output of the adder (A_1 or B_1 shown in Figure 4.10) with its previous four and successive four outputs and a positive constant number. Along with this, C_1 and C_2 values (shown as C_{1A} and C_{2A} in Figure 4.10) are also checked to see if they are positive. If they are not positive, then they will not provide the proper arrival time of the pulse. This logic is implemented for both the wedge anode outputs to detect the presence of pulse coming from any of the wedge anode outputs. When the maximum is found, the value in shift register entry 5 (shown in Figure 4.11) gives the approximate amplitude of the pulse.

FIFO stores the shift register entry 5 from both the anode outputs when the FIFO enable signal is high along with other data as explained in Section 4.4.5 of this chapter.

Figure 4.11: Maximum detection logic

4.4.6.1 Additional support components used in the main board

1) EEPROM: Because the Spartan-3 FPGA used in this research is a SRAM based FPGA, it needs to be configured every time it is powered up. EEPROM is used to store the configuration file for the FPGA.

2) DC-DC converter: ASD10 DC-DC converter, convert 28V DC supply input into 5V [37].

3) Power sources generator for FPGAs: A Texas Instruments PCB TI TPS75003 [41] board was used to convert the 5V input into 3.3V, 2.5V, and 1.2V voltages. This board is designed to ramp up the power supplies properly for the FPGA so that FPGA will be configured from the configuration file stored in EEPROM.

4) Timing counters: Timing counters are used to determine when a particular ion was detected. Timing counters values are also used to reconfigure the FPGAs because energetic ions can cause SEU (single event upsets in the FPGAs) logic errors in the FPGA. In the present implementation timing counter values are used to configure FPGA for every 3 seconds to overcome such errors. A NAND IC is used to generate the reconfiguration signal using the timing counter values.

4.5 PC interface board

PC interface board was used to take the results from the main board and store them in its internal memory blocks before sending them to PC at high speed. The XEM3001 board from Opal Kelly [31] shown below in Figure 4.12 is used for this purpose. It sends the data to PC through the USB at a speed of 12MB/sec. The main components of the XEM 3001 board are a spartan-3 XC3S400 FPGA, a USB (universal serial bus) port and a USB microcontroller.

Figure 4.12: PC interface board

4.5.1 Interface between PC interface board and FPGA

Figure 4.13: Interfacing between main board and PC interface board

PC interface board generates two signals (data read clock and frame sync) and also reads the data line signal through which the main board sends the results. Upon receiving the data read clock, the main board updates the results on the data line. The data read clock is a 125 ns period clock. It is a continuous ten-clock sequence with a time gap of 50 μ s between each sequence as shown in Figure 4.13.

Figure 4.13: Gated clock

Frame sync is a clock of 200 μ s period which increments a hardware counter (external to FPGA) called “timing counter” on the main board. The data line is read by the Spartan-3 FPGA of the XEM 3001 board, and the data are stored in its internal RAM blocks. Two RAM blocks, which are instantiated in the Spartan-3 FPGA of the XEM 3001 board, are used to store the results. When the first RAM block stores the results, the other RAM block sends the results to PC and vice versa. After a RAM block becomes full, it sends a

signal to PC to indicate that it can send the result. After detecting that event, the PC reads from the FPGA and stores the results in a file.

A C++ program was written for detecting the RAM block full event, and after detecting this event, data is read through the Universal serial bus (USB) port and is stored in a file for later analysis. Dev-C++, an integrated development environment (IDE) for C++ programs, is used to generate an executable (exe) file of the C++ program [26]. The exe program not only downloads the Spartan-3 FPGA bit file to the XEM 3001 board, but also upon detecting the memory block full event, reads data through the USB port and stores the results in a file.

4.5.2 FPGA design and verification tools

For FPGA programming, Xilinx ISE foundation v6.2 software was used. The hardware description language called VHDL (VHSIC Hardware Description Language) was used and Xilinx software converts the high level VHDL language into a bit file, which can be used to configure the FPGA. The VHDL programs are shown in Appendix A. All the VHDL programs were simulated using ModelSim software [28].

4.5.2.1 Additional functionalities implemented in FPGA

- The ratio of pulse height found from a wedge anode output end to the sum of the pulse heights found from both the outputs of the same wedge anode can be used to identify the ion. This ratio is used to increment one of the four ion counters. Four counters correspond to four ions to be detected, H^+ , He^{++} , He^+ , and O^+ . After the

detection of the presence of the pulse, the division process will be activated which computes the ratio. During the computation of ratio, the process of maximum detection will be disabled. The total time required to process the pulse along with computing the ratio requires 2250 ns. Therefore, the present implementation can process 44.4×10^4 ion events per second whereas the expected ion count rate in the present application is 10000 per second. By using a faster clock for FPGA it is possible to process more than 4.44×10^4 ions.

- An external clock generated by the PC interface board (shown in Figure 4.12) synchronizes the data transfer between the main board and the PC interface board. This clock is called the “data read clock” and is used by FPGA to send out the FIFO data.

4.5.2.2 FPGA resources used

The algorithm implementation in FPGA (for processing 4 pulse inputs coming from 2 wedge anodes) used

- 87% of the CLBs
- 6 RAM blocks
- 63 I/O pins
- 1 DCM

4.5.2.3 The digital electronics implementation requires 1.8-1.9W of power. The FPGA

power requirement has been minimized by the following techniques:

1. Only the required blocks are assigned faster clocks, and the other blocks are assigned lower frequency clocks.
2. Clock signals were routed through low-skew (delay) dedicated lines which minimize the power, compared to the clock carried on general data carrying lines.
3. Optimized digital logic was used for the entire implementation.
4. Enable signals were used to enable adder blocks, FIFO blocks, and RAM blocks.
5. Logic blocks are placed close to each other if they interact with each other, e.g. all the blocks within FIR filters are placed close to each other, and also, the adder block and FIR block shown in Figure 4.10 were placed close to each other.

Chapter Five

HARDWARE TESTING AND RESULTS

5.1 Generating pulses from anode board

Because the actual set-up to test the anode board with the ion analyzer is not readily available, the testing of the anode board was done by using the following method. A voltage pulse source (from a pulse generator), acting as an edge generator was held very near to the wedge pattern, insulated properly for avoiding any direct contacts as shown in Figure 5.1. The pulses are capacitively coupled onto the wedge pattern, and they generate millivolt pulses at its two output ends. The terminal which is close to the pulse source gets more capacitively coupled input and generates a higher amplitude output because of the wedge pattern which is shown in Figure 1.8.

If the capacitively coupled input is equal for both the terminals of the wedge pattern, then both the terminals generate equal amplitude outputs at both the output ends. Proper functioning of the wedge anode board was verified by checking that the frequency at both the output ends of wedge pattern is equal to the frequency of the pulse source. Pulse source was set to generate square wave pulses at 3 kHz, and the rise time of the input pulses was 25ns. The two outputs of the anode board were tested using a Tektronix TDS2024 [38] oscilloscope if they pulse outputs generated (as shown in Figure 4.4) were of 3 kHz frequency.

Figure 5.1: Generating pulses from anode board using pulse generator

5.2 Testing main board

The pulses generated by the anode board were input to the pulse shaping circuits, and their outputs were tested using Tektronix TDS2024 [38] oscilloscope. The pulse outputs generated by the pulse shaping circuits were tested for a rise time of 75ns and fall time of 300ns. Due to the noise problems in the pulse output, the pulse-shaping circuits were relocated to the anode board. The sources of noise were variations in the power supply and also as the distance between the channel plate detector and pulse processing electronics. The noise in the pulse reduced due to this relocation, as the distance between the channel plate and pulse shaping and amplifying circuits reduced.

5.2.1 Testing ADC (ADC08L060) [30]

This ADC converts the analog pulse input using a 20MHz sampling clock and gives out 8 bit parallel output. For a pulse input to the ADC, the 8 outputs were observed using an HP 16500B logic analyzer [34]. The logic analyzer was set up to trigger when the output of ADC is greater than 30 because the ADC output for the noise was observed to be below 23. ADC data obtained from the logic analyzer is shown in Figure 5.2 along with the ideal pulse. ADC output matches closely with the ideal pulse.

Figure 5.2: ADC output and the ideal pulse comparison

5.3 Testing the algorithm in hardware

The five regions shown in Figure 5.3 with vertical bars were tested on the wedge pattern of the anode board. Each of the test regions approximately corresponds to the region where the ions H^+ , He^{++} , He^+ , and O^+ generate electron clouds on the wedge pattern. Each of the test regions generates two outputs on the wedge pattern. The process of pulse generation on the wedge pattern has been explained in Section 5.1 of this chapter.

Figure 5.3: Test regions on the wedge pattern of the anode board

Each of the outputs are shaped and amplified by the pulse-shaping circuits. Shaped pulses are fit using the two FIR filters in FPGA. Whenever a maximum is found, the results are stored in FIFO. The results data consist of the pulse height found at the left and the right

sides of the wedge pattern along with timing and other data. The ratio of pulse height found at the left side of the wedge pattern and the sum of the pulse height found at the left and the right sides of the wedge pattern is calculated (as shown in Equation 5.1).

$$ratio = \left(\frac{\text{left side amplitude}}{\text{left side amplitude} + \text{right side amplitude}} \right) \quad (5.1)$$

Some of the results obtained are shown in Table 5.1.

Repeatability test	Test region 1 Ratio	Test region 2 Ratio	Test region 3 ratio	Test region 4 ratio	Test region 5 Ratio
1	0.365942	0.456905	0.510168	0.552212	0.655617
2	0.369217	0.457627	0.509251	0.550269	0.656772
3	0.372332	0.458022	0.510955	0.551341	0.657724
4	0.369740	0.454976	0.510168	0.550373	0.655042
5	0.365693	0.456848	0.508818	0.551661	0.652863
6	0.368672	0.457467	0.510097	0.551887	0.657169
7	0.369740	0.457611	0.508326	0.550467	0.655327
8	0.369841	0.458409	0.511013	0.549565	0.654562
9	0.368280	0.457741	0.508850	0.549227	0.656648
10	0.364391	0.458266	0.510526	0.549128	0.657968
Average	0.3688	0.4573	0.5097	0.5508	0.6557

Table 5.1: Some results of the five test regions

From table 5.1 we can observe that as the test region is moved from right to left, the result value increases. Figure 5.4 shows the result values for all five tests.

Figure 5.4: Plot summarizing the results obtained from the five tests

From the above plot, we can deduce that as the test region is moved from right to left, the ratio left side/(left + right side output) increases. This ratio can be used to identify the ion as shown in Table 5.2.

Ratio	Ion
< 0.45	O ⁺ ion
0.45> and <0.5	He ⁺ ion
0.5> and <0.55	He ⁺⁺ ion
>0.55	H ⁺ ion

Table 5.2: Ratio values for corresponding ions

5.3.1 Testing of PC interface board

The PC interface board has been discussed in Section 4.5 of chapter 4. The Xilinx program for the Spartan-3 FPGA chip on this board and the C++ program to control USB operations are listed in Appendix C.

This board was tested for the following signals.

- Data read clock was tested using the oscilloscope for a clock period of 125ns as well as for the gap of 50 μ s between every ten clocks.
- Frame Sync signal was tested using the oscilloscope to have a clock period of 200 μ s.
- Data read line is a results data line which was tested by analyzing the results stored in a file in PC.

Chapter Six

CONCLUSIONS AND FUTURE WORK

6.1 Conclusion

Energetic ion analyzers and mass spectrometers commonly produce electronic pulses whose amplitude and time depend on the energy, mass, and direction of an incoming ion. The pulses generated will have an unknown amplitude, unknown starting time, and also, added random noise. Algorithm called pulse height fitting with two fit curves has been developed to determine the amplitude and starting of these pulses.

Pulse height fitting with two fit curves algorithm has been simulated in Matlab, and implemented in hardware to process the pulses coming from a wedge anode board to determine their heights. The simulation tests included detecting pulses at different noise levels, fitting the pulse to find the best-fit result, determining the accuracy of pulse height fit for varying noise levels, and finding the exact starting time of the pulse.

Two PCBs were designed to generate and process the pulses. Algorithm was implemented in Xilinx Spartan-3 FPGA. VHDL language and Xilinx software tools were used to implement the algorithm in Xilinx Spartan-3 FPGA, on one of these boards. The third commercial board XEM3001 [31] from Opal Kelly was used for transferring results to PC. C++ and VHDL languages were used to send the results from this board to PC and store in a file. Eagle PCB CAD soft was used for design and implementation of PCBs

[33]. The current study focused on detecting the presence of pulses and estimating their approximate height to identify the ions.

Applications of the algorithm include

- To find the energy of ions entering an ion analyzer [12]
 - This algorithm can be used to find the energy of an ion when the pulse generated by that ion, in an ion analyzer has its height proportional to the ion energy.
- Pulse position sensing to determine the mass of an ion, in mass spectrometers utilizing static $\vec{E}$ and $\vec{B}$ fields.

The results of the simulation tests of the algorithm indicate that the

- Algorithm can process pulses of width as small as 500ns.
- Best-fit result of the algorithm matches closely the shape of the pulse.
- Algorithm can fit the pulse with DC offset closely.
- Algorithm can detect the presence of the pulse even when the SNR in the signal is roughly equal to 3dB.
- Pulse height found varies by a maximum of 0.2 (i.e. varies from 1.2-0.8) from the expected pulse height of 1 when the SNR in the signal is more than 7dB.
- Algorithm can determine the arrival of the pulse to within 6ns when the RMS of the noise level is 5% of the pulse height.

An FPGA is used for implementing the digital pulse height technique in hardware. The results indicate that the

- Algorithm implementation in hardware can process more than 10000 events per second (the expected pulse count rate).
- Implementation can identify the ions based on the position of the electron cloud on the wedge pattern.
- Digital electronics implementation requires 1.9W of power.

6.2 Future Work

The present research can be improved to provide the following additional information.

- In time of flight mass spectrometers, [22] mass-to-charge ratio of ions is measured by measuring the time of arrival of ions.
 - Electron generated by an ion (when an ion hits a thin foil releasing an electron) is detected, when it hits a detector and generates a pulse. Ion is also detected when it hits the same detector but after some delay. By measuring the time difference between the arrival of electron and ion at that detector the velocity of ion is determined and then its mass-to-charge ratio is calculated. Thus, the circuit could also be used for time of flight mass spectrometers [22].
- Pulse position sensing to determine the angle of entrance of an ion, electron, or neutral atom by measuring its position of input on a channel plate.

- If the two pulses are lying close to each other, the present algorithm may not detect the individual pulse heights properly. If the pulses are lying close to each other, then the fitting curves may use some of the pulse sample values from the other curve, resulting in wrong pulse height. For such events, the present algorithm should be modified to find the individual pulses. This will help in gathering more information rather than rejecting the piled-up events.
- Energetic ions can cause errors in the logic by causing SEU (single event upset) errors. Because Xilinx FPGAs are not designed to overcome single event upset errors, Actel FPGAs or non-RAM-based other FPGAs which are designed to handle such errors can be used for the future project. If such FPGAs are used, periodic reconfiguration of FPGAs is not needed. This transformation to a different FPGA can be done by changing the code.

References

- [1] Kermit K. Murray, Robert K. Boyd, Marcos N. Eberlin, G. John Langley, Liang Li and Yasuhide Naito, "Standard Definitions of Terms Relating to Mass Spectrometry," International union of pure and applied chemistry analytical chemistry division 2006.
- [2] Margaret G. Kivelson and Christopher T. Russell, *Introduction to Space Physics*. New York: Cambridge university press, 1995.
- [3] Daniel S Babb, *Pulse circuits: switching and shaping*. Englewood Cliffs, NJ: Prentice-Hall, 1964.
- [4] Itamar Elhanany, Shimson Jacobi, Michael Kahane, "A Novel Architecture for Digital Pulse Height Analysis with Application to Radiation Spectroscopy," Proceedings of the 7th Mediterranean Conference on Control and Automation (MED99) Haifa, Israel, pp. 2143-2151, June 1999.
- [5] Paulo. C. P. S. Simoes, Jose C.Martins, and C. M. B. A. Correia, "A New Digital Signal Processing Technique for Applications in Nuclear Spectroscopy Techniques," IEEE Transactions on Nuclear science, vol. 43, no. 3, pp. 1804-1809, June 1996.
- [6] J.M. Los Arcos, E.Garcia-Torano, "A New Pulse Height Analysis Method for Radiation Spectroscopy," Nuclear Instruments and methods in Physics research, A339, pp. 99-101, 1994.
- [7] Valentin Jordanov and Glenn F. Knoll, "Digital Pulse Processor Using a Moving Average Technique," IEEE Transactions on nuclear science, vol. 40, no. 4, pp. 764-769, Aug. 1993.
- [8] E.Cosulich and F.Gatti, "A Digital Processor for Nuclear Spectroscopy with Cryogenic Detectors," Nuclear Instruments and methods in Physics research, A249, pp. 211-215,1992.
- [9] R.E.Chrien and R.J.Sutter, "Noise and Pile-up Suppression Techniques by Digital Signal Processing," Nuclear Instruments and methods in Physics research, A249, pp. 421-425, 1986.
- [10] Joseph Ladislav Wiza, "Microchannel Plate Detectors," Nuclear Instruments and Methods, vol. 162, pp. 587-601,1979.

- [11] D.W. Walton, A.M. James, J.A. Bowles and A.D. Johnstone “ High-speed 2-D imaging for Plasma Analyzers using Wedge-and-Strip-Anodes,” Robert F.Pfaff, Joseph E. Borovsky, David T.Young, editors, *Measurement Techniques in space plasmas particles*, page-295, Mullard space science laboratory, University college London, U.K, American geophysical union, Washington DC 1998.
- [12] S.M. Ritzau, H.O.Funsten and J.E.Borovsky “Solid state detection of low energy ions and electrons for constellation missions,” V.Angelopoulos and P.V.Panetta, editors, *Science closure and enabling technologies for constellation class missions*, pp 131- 135, University of California, Berkley 1998.
<http://sprg.ssl.berkeley.edu/ConstellationClassMissions/Ritzau.pdf>
- [13] <http://astro.uni-tuebingen.de/groups/orfeus/detector.shtml>
Last visited on November 22, 2006.
- [14] R. M. Tromp, M. Copel, M. C. Reuter, M. Horn von Hoegen, and J. Speidell “A New Two-Dimensional Particle Detector for a Toroidal Electrostatic Analyzer,” *Review of Scientific Instruments*, vol. 62, Issue 11, pp. 2679-2683, Nov. 1991.
- [15] J.S.Lapington “Charge Position Readout Devices for Imaging Photon Detectors,” *IEEE Colloquium on Satellite Instrumentation*, pp. 4/1-4/3, Jan. 1988.
- [16] O. H. W. Siegmund, M. Lampton, J. Bixier, S. Chakrabarti, J. Vallerger, S. Bowyer, and R. F. Malina, “Wedge and Strip Image Readout Systems for Photon-counting Detectors in Space Astronomy,” *Journal of the Optical Society of America*, vol. 3, n 12, pp. 2139-2145, Dec. 1986.
- [17] V. Cappellini, A. G. Constantinides and P. Emiliami, *Digital Filters and their Applications*. New York: Academic Press, 1978.
- [18] Robert G, Irvine *Operational amplifier characteristics and applications*. New York: Prentice-Hall, 1994.
- [19] J. Basilio SimGes, P.C. P. S. Simoes, and C. M. B. A. Correia, “Nuclear Spectroscopy Pulse Height Analysis Based on Digital Signal Processing Techniques,” *IEEE Transactions on Nuclear science*, vol. 42, no. 3, pp. 700-704, June. 1996.
- [21] NASA, <http://www.universe.nasa.gov/xrays/programs/astroe/eng/pha.html>
Last visited on November 2, 2006.
- [22] W C .Wiley and I.H.McLaren, “Time of Flight Spectrometer with Improved Resolution,” *The Review of Scientific Instruments*, vol. 26, number 12, Dec. 1955.

- [23] Wolfram Mathworld, <http://mathworld.wolfram.com/RelativeError.html>
Last visited on November 2, 2006.
- [24] Manfred.Boehm, sub-proposal to “Twin Rockets to Investigate CUSP Electrodynamics,” West Virginia University,Morgantown,WV, Proposal to Geospace LCAS 2004.
- [25] Peled and B. Liu, “A New Hardware Realization of Digital Filters,” IEEE Trans. on Acoust., Speech,Signal Processing, vol. ASSP-22, pp. 456-462, Dec. 1974.
- [26] B.S.Grewal,*Higher Engineering Mathematics*. Delhi: Khanna Publishers, 2002.
- [27] Signal-to-noise ratio,
http://www.maxim-c.com/appnotes.cfm/appnote_number/641/
Last visited on November 2, 2006.

Parts specifications list:

- [28] ModelSim Inc, <http://www.model.com/>
Last visited on November 2, 2006.
- [29] National semiconductor Inc, <http://www.national.com/pf/LM/LMH6646.html>
Last visited on November 2, 2006.
- [30] National semiconductor Inc,
http://www.datasheetcatalog.com/datasheets_pdf/A/D/C/0/ADC08L060.shtml
Last visited on November 2, 2006.
- [31] Opal kelly Inc, <http://www.opalkelly.com/products/xem3001/>
Last visited on November 2, 2006.
- [32] Xilinx Inc,
http://www.xilinx.com/products/silicon_solutions/fpgas/spartan_series/spartan3_fpgas/index.htm
Last visited on November 2, 2006.
- [33] Eagle CADSOFT Inc,
<http://www.cadsoftusa.com/>
Last visited on November 2, 2006.
- [34] MAX1231 ADC
<http://datasheets.maxim-ic.com/en/ds/MAX1227-MAX1231.pdf>
Last visited on November 2, 2006.

- [35] AD5544 DAC
<http://www.chipcatalog.com/Datasheet/9D1581479850257E9BE4A9D3B4E3B13D.htm>
Last visited on November 2, 2006.
- [36] Mathworks Inc,
<http://www.mathworks.com/products/matlab/>
Last visited on November 2, 2006.
- [37] Astrodyne Inc,
<http://www.datasheets.org.uk/datasheet.php?article=837906>
Last visited on November 2, 2006.
- [38] Tektronix Inc,
http://www.tek.com/site/ps/0,,3G-15314-INTRO_EN,00.html
Last visited on November 2, 2006.
- [39] Bloodshed software, <http://www.bloodshed.net/devcpp.html>
Last visited on November 2, 2006.
- [40] <http://en.wikipedia.org/wiki/FPGA>
Last visited on November 2, 2006.
- [41] TI Inc, <http://focus.ti.com/lit/ml/slyl022a/slyl022a.pdf>
Last visited on November 2, 2006.

APPENDIX A: VHDL PROGRAMS OF THE MAIN BOARD

```

--This program implements the following logic components
--FIFO block
--control_rom:which generates periodic signals to control MAX1231 ADC, telemetry
pattern etc
--sends results to data transfer board
--instantiates MAX1231 ADC

--Inputs
--clk:20MHz clock
--gated_clock:data read clock
--minor_frame_sync:reconfiguration counter clock
--otherboard_sdout:Results data output line
-- max_detect_1: MAXimum detection signal a '1' bit signal (indicates maximum from
one wedge pattern)
-- max_detect_2: MAXimum detection signal a '1' bit signal (indicates maximum from
the other wedge pattern)
-- fifo_din_ch1:FIFO input (From one of the wedge pattern)
-- fifo_din_ch2:FIFO input (From the other wedge pattern)
-- major_frame: This signal is '1' when 40 minor_frame_syncs are counted
-- frame_counter: A 10 bit counter counts minor_frame_sync
-- reset_counters1: Resets one set of ion counters
-- reset_counters2: Resets other set of ion counters
-- max1231_sdin: This signal is used to get results from MAX1231 ADC
-- max1231_eoc_bar: 'eoc' end of conversion signal from MAX1231 ADC
-- chip_select_pin: FPGA select pin used to synchronize data transfer between
FPGAs
-- op_counter1_ch1,op_counter2_ch1,op_counter3_ch1,op_counter4_ch1 : 4 "10bit"
ion counters
-- op_counter1_ch2,op_counter2_ch2,op_counter3_ch2,op_counter4_ch2 : 4 "10bit"
ion counters

--Outputs
-- max1231_sclk: A MAX1231 ADC clock
-- max1231_cs_bar: Sends cs_bar signal to MAX1231 ADC
-- max1231_sdout: Sends commands to MAX1231 ADC

--coregen macros
--fifo_common,control_rom,shift_40out

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

```

```

use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

```

```

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

```

```

entity fifo_logic is

```

```

    Port (

```

```

        clk:in std_logic;
        --gated_clock_t_t:out std_logic;
        gated_clock:in std_logic;
        minor_frame_sync:in std_logic;
        otherboard_sdout:out std_logic;
        max_detect_1:in std_logic;
        max_detect_2:in std_logic;
        fifo_din_ch1:in std_logic_vector(39 downto 0);
        fifo_din_ch2:in std_logic_vector(39 downto 0);
        major_frame:inout std_logic;
        frame_counter:out std_logic_vector(9 downto 0);
        reset_counters1: out std_logic;
        reset_counters2: out std_logic;
        test_sdout:out std_logic_vector(1 downto 0);
        --reset_control_rom_fifo_outbit_t,control_rom_fifo_outbit_t :out std_logic;
        --count_4_T_t:OUT STD_LOGIC_VECTOR(3 DOWNT0 0);
        --ainit:in std_logic;
        --wr_en_t,rd_en_t,reset_m_t,:out std_logic;
        --dout_t:out std_logic_vector(39 downto 0);
        --reset_m_t,reset_delayed_t,reset_control_rom_fifo_outbit_t:out std_logic;
        --word_clock_t:out std_logic;
        --p_load_t,shift_out_en_t,gated_clock_t_t,control_rom_fifo_outbit_t:out
std_logic;
        --sdin:in std_logic;
        --wr2ram:out std_logic;
        --telemetry_data:out std_logic_vector(39 downto 0);
        -- frame_sync_t:out std_logic;
        --- temp outputs---
        -- clk_2_t,clock_11_t,clk2_40_t,rom_op_t_t,reset_m_t:out std_logic;
        -- count_gated_t:out std_logic_vector(4 downto 0);
        --reset_count_gated_t,word_clock_t:out std_logic;
        --rom_addr_t:out std_logic_vector(10 downto 0);

```

```

--rom_dout_t:out std_logic_vector(11 downto 0);
--frame_counter_t:out std_logic_vector(9 downto 0);
--p_load_t,control_rom_fifo_outbit_t,p_load_t:out std_logic;
--rd_enable_fifo_t,rd_enable_fifo_t2:out std_logic;
--p_load_t,p_load_en_t,p_load_d_t:out std_logic;
-- mux_out_temp_t,mux_out_t:out std_logic_vector(39 downto 0);
--sel_t:out std_logic_vector(1 downto 0);
--rom_addr_t:out std_logic_vector(10 downto 0);
--count_40frames_t:out std_logic_vector(5 downto 0);
--fifo_din_t:out std_logic_vector(39 downto 0);
--rd_ack_t:out std_logic;
--rom_rd_en_t:out std_logic;
--rom_sequence_T:OUT STD_LOGIC;
--gated_clock_t_t:out std_logic;
--fifo_dout_t:out std_logic_vector(39 downto 0);
--wr_en_2d_t,wr_en_d_t,wr_en_t:out std_logic;
--max2_delay2_t,max_detect_1_d_t :out std_logic;
-----
-- max 1231 ADC signals
--tx1_reg:out std_logic;
max1231_sclk:out std_logic;
max1231_sdin:in std_logic;
max1231_sdout:out std_logic;
max1231_cs_bar:out std_logic;
max1231_eoc_bar:in std_logic;
chip_select_pin:in std_logic;

--count_4_t:out std_logic_vector(3 downto 0);
--master_data:in std_logic_vector(7 downto 0);
-----
op_counter1_ch1,op_counter2_ch1,op_counter3_ch1,op_counter4_ch1:in
std_logic_vector(9 downto 0);
op_counter1_ch2,op_counter2_ch2,op_counter3_ch2,op_counter4_ch2:in
std_logic_vector(9 downto 0)

--control_rom_rdy_t:out std_logic;
--rom_access_4_t:out std_logic_vector(2 downto 0);
--control_rom_op_t:out std_logic_vector(11 downto 0)

-- shift_out_en_temp_t:out std_logic
--fifo_din_t:out std_logic_vector(39 downto 0);

--FRAME_SYNC : in std_logic
);
end fifo_logic;

```

architecture behavioral of fifo_logic is

```
-- fifo_common component instantiates FIFO
component fifo_common
    port (
        din: IN std_logic_VECTOR(39 downto 0);
        wr_en: IN std_logic;
        wr_clk: IN std_logic;
        rd_en: IN std_logic;
        rd_clk: IN std_logic;
        ainit: IN std_logic;
        dout: OUT std_logic_VECTOR(39 downto 0);
        full: OUT std_logic;
        empty: OUT std_logic;
        almost_full: OUT std_logic;
        almost_empty: OUT std_logic;
        rd_ack: OUT std_logic;
        rd_err: OUT std_logic;
        wr_ack: OUT std_logic;
        wr_err: OUT std_logic);
end component;
```

--control_rom component is a ROM which instantiates the ROM for telemetry and MAX1231 ADc

```
component control_rom
    port (
        addr: IN std_logic_VECTOR(10 downto 0);
        clk: IN std_logic;
        dout: OUT std_logic_VECTOR(11 downto 0);
        en: IN std_logic;
        nd: IN std_logic;
        rfd: OUT std_logic;
        rdy: OUT std_logic);
end component;
```

--max1231_top is used to control/operate the MAX1231 ADC

```
component max1231_top
    Port ( clk : in std_logic;
           sclk : out std_logic;
           sdin : in std_logic;
```

```

    control_rom_op:in std_logic;
    sdout : out std_logic;
    cs_bar : out std_logic;
    QSPO_CE:in std_logic;
    test_sdout:out std_logic_vector(1 downto 0);
    --sclk_out_t,sclk_in_t: out std_logic;
    --tx_reg_t:out std_logic_vector(7 downto 0);
    --read_counter_ored_t:out std_logic_vector(5 downto 0);
    --max1231_we_t:out std_logic;
    --temp_t:out std_logic_vector(15 downto 0);
    --dist_ram_en_t:out std_logic_vector(1 downto 0);
    --sclk_in_counter_t:out std_logic_vector(4 downto 0);
    --max1231_addr_t:out std_logic_vector(3 downto 0);
    --tx1_reg:out std_logic;
    max1231_dout:out std_logic_vector(11 downto 0);
    --sclk_ored_t:out std_logic;
    eoc_bar : in std_logic);
end component;

```

--shift_40out compoenet sends the 40bit results

component shift_40out

```

    port (
        CLK: IN std_logic;
        SDOUT: OUT std_logic;
        P_LOAD: IN std_logic;
        D: IN std_logic_VECTOR(39 downto 0);
        CE: IN std_logic);
end component;

```

--gated_gen_top compoenent

--component GATED_GEN_TOP

```

--Port (
--    CLK : in std_logic;
--    GATED_CLOcK:inout std_logic;
--    sdin:in std_logic;
--    wr2ram:out std_logic;
--    telemetry_data:out std_logic_vector(39 downto 0);
--    FRAME_SYNC : out std_logic);
--end component;

```

--component controlrom_addr_compute

```

--    port (
--        A: IN std_logic_VECTOR(7 downto 0);
--        S: OUT std_logic_VECTOR(7 downto 0));

```

```
--end component;
```

```
--temporary signals
```

```
signal counter_2:std_logic_vector(3 downto 0):="0000";
signal reset_m,word_clock,reset_m9:std_logic:= '0';
signal reset_control_rom_fifo_outbit:std_logic:= '0';
signal count_4:std_logic_vector(3 downto 0):="0000";
signal control_rom_fifo_outbit:std_logic:= '0';
signal p_load,p_load_en:std_logic:= '0';
signal max2_delay1,max2_delay2,wr_ack:std_logic:= '0';
signal wr_en,reset_delayed:std_logic:= '0';
signal full,empty,rd_ack,rd_err:std_logic:= '0';
signal control_rom_op:std_logic_vector(11 downto 0):=x"000";
signal rom_dout:std_logic_vector(11 downto 0):=x"000";
signal frame_counter_temp:std_logic_vector(9 downto 0):="0000000000";
signal fifo_dout,mux_out,mux_out_temp, fifo_dout_reg:std_logic_vector(39 downto
0):=x"0000000000";
signal rom_sequence:std_logic:= '0';
signal rom_access_4:std_logic_vector(2 downto 0):="000";
signal control_rom_rdy:std_logic:= '0';
signal rom_rd_en:std_logic:= '0';
signal rom_addr:std_logic_vector(10 downto 0):="00000000000";--check out??????--
should be zero
signal nd,rfd:std_logic:= '0';
signal sel:std_logic_vector(1 downto 0):="00";
signal max1231_reg1,max1231_reg2,max1231_reg3,max1231_reg4:std_logic_vector(11
downto 0):="000000000000";
signal max1231_dout:std_logic_vector(11 downto 0):="000000000000";
signal shift_reg_1,shift_reg_2,shift_reg_3,shift_reg_4:std_logic:= '0';

signal count_40frames:std_logic_vector(5 downto 0):="000000";
signal frame_counter_bitzero_2d,frame_counter_bitzero_d:std_logic:= '0';
```

```
signal control_rom_counter:std_logic_vector(2 downto 0):="000";
signal rom_access_en,rom_rd_en_d,wr_err: std_logic:= '0';
signal fifo_din: std_logic_vector(39 downto 0):=x"00000000000";
signal shift_out_en_temp,major_frame_d:std_logic:= '0';
signal hk:std_logic_vector(1 downto 0):="00";
signal otherboard_sdout_temp,major_frame_temp,wr_en_d:std_logic:= '0';
signal counter_sel:std_logic_vector(1 downto 0):="00";
signal max_detect_1_d,max_detect_2_d,max2_delay2_d:std_logic:= '0';
signal rd_enable_adc,rd_enable_fifo,reset_m_delayed:std_logic:= '0';
signal t:std_logic_vector(3 downto 0):="0000";
```

```

signal ainit: std_logic:= '0';
signal reset_fifo,r1,r2:std_logic:= '0';
--signal rd_count, wr_count,rd_count1,wr_count1:std_logic_vector(8 downto
0):="0000000000";
--signal fifo_counts:std_logic_vector(19 downto 0):="0000000000000000000000";
signal almost_full,almost_empty,p_load_d:std_logic:= '0';
--signal ila_control: std_logic_vector(35 downto 0);
--signal data      : std_logic_vector(8 downto 0);
--signal trig2     : std_logic_vector(1 downto 0);
--signal trig1     : std_logic_vector(1 downto 0);
signal count_40frames0_2d,count_40frames0_d:std_logic:= '0';

--signal gated_clock,gated_clock_t,frame_sync_t:std_logic:= '0';
--signal wr2ram:std_logic:= '0';
--signal telemetry_data:std_logic_vector(39 downto 0):=x"000000000000";

-- adc signals
--signal max1231_eoc_bar:std_logic;

--- logic to enable wr_en in multiple cycles when two maximas are detected at the same
time
begin

--gated_framesync: GATED_GEN_TOP
--    Port map(
--        CLK => clk,
--        GATED_CLOcK => gated_clock_t,
--        sdin => '1',
--        wr2ram=>wr2ram,
--        telemetry_data => telemetry_data ,
--        frame_sync => frame_sync_t);

--gated_clock<=gated_clock_t;
--gated_clock_t_t<=gated_clock_t;

--logic to delay one of the max detect signals if both of them arrive at the same time
--if max_detect1 is '1' then the other max_detect is delayed by '1' clock cycle in this
process
process(clk)
begin
    if clk'event and clk='1' then
        if max_detect_1='1' then

```



```

        max2_delay1<=max_detect_2;
    else
        max2_delay1<='0';
    end if;
end if;
end process;

```

--max_detect_2 is delayed by '1' more cycle

```

process(clk)
begin
    if clk'event and clk='0' then
        max2_delay2<=max2_delay1;
    end if;
end process;

```

--Following process is used to count gated_clocks..after 10 gated clocks is reset to zero and also when a major_frame is '1'

```

process(major_frame,reset_m_delayed,gated_clock)
variable counter:std_logic_vector(3 downto 0):="0000";
begin
    if reset_m_delayed='1' or major_frame='1' then
        counter:="0000";
    elsif gated_clock='1' and gated_clock'event then
        counter:=counter+'1';
    end if;
    counter_2<=counter;
end process;

```

--reset_m signal is used for checking whether 10 gated clocks are got
 reset_m<= ((counter_2(3))) and (not (counter_2(2))) and ((counter_2(1))) and
 (not(counter_2(0)));

--reset_m9 is '1' when counter_2 is 9

```

reset_m9<= ((counter_2(3))) and (not (counter_2(2))) and (not(counter_2(1))) and  

((counter_2(0)));

```

--reset_m is delayed by '1' clock cycle..but using 'clk' clock

```

process(clk)
begin
    if clk'event and clk='1' then
        reset_m_delayed<=reset_m;
    end if;
end process;

```

----reset_m is delayed by one more clock cycle..

```
process(clk)
begin
  if clk'event and clk='1' then
    reset_delayed<=reset_m_delayed;
  end if;
end process;
```

--a process used to count whether we got 40 gated clocks--(this is done by incrementing a counter when the gated_clock count is 9..

--when it becomes 4 i.e gated_clocks are 40..the corresponding counter_4 is reset by

reset_control_rom_fifo_outbit

--and is used to generate the signal control_rom_fifo_outbit

```
process(major_frame,gated_clock)
begin
  if major_frame='1' then
    count_4<="0000";
  elsif gated_clock'event and gated_clock='1' then
    if reset_control_rom_fifo_outbit='1' then
      count_4<="0000";
    elsif reset_m9='1' then
      count_4<=count_4+'1';
    end if;
  end if;
end process;
```

--to count whether 4 times 10 gated clocks are got.....by generating '1' when

count_4="0100"

reset_control_rom_fifo_outbit<= not(count_4(3)) and count_4(2) and (not(count_4(1)))
and (not(count_4(0)));

--After 40 gated_clocks are obtained..control_rom_fifo_outbit is made '1' which is used to enable a load

-- signal of the output shift register

```
process(gated_clock)
begin
  if gated_clock'event and gated_clock='1' then
    if count_4="0011" then
      control_rom_fifo_outbit<='1';
    else
      control_rom_fifo_outbit<='0';
    end if;
  end if;
end process;
```

```

--logic to generate the parallel load of the output shift register
p_load_en<=control_rom_fifo_outbit and reset_m_delayed;

--p_load is delayed by '1' clock cycle here..
--clock source is clk but control_rom_fifo_outbit is changed by gated_clock
--This synchronizes properly the loading of the shift_40 register
process(clk)
begin
  -- if gated_clock='0' and gated_clock'event then
  if clk'event and clk='1' then
    if p_load_en='1' then
      p_load_d<='1';
    elsif control_rom_fifo_outbit='0' then -- not so good coding ...may work with a
single variable
      p_load_d<='0';
    end if;
  end if;
end process;

-- Logic to generate wr_en signal to FIFO
--max_detect_1 and max_detect_2 are delayed '1' or 2 clock cycles

--max_detect_1 is delayed by '1' clock cycle
process(clk)
begin
  if clk'event and clk='1' then
    max_detect_1_d<=max_detect_1;
  end if;
end process;

--max_detect_2 is delayed by '1' clock cycle
process(clk)
begin
  if clk'event and clk='1' then
    max_detect_2_d<=max_detect_2;
  end if;
end process;

--max_detect_2 is delayed by '2' clock cycle
process(clk)
begin
  if clk'event and clk='1' then
    max2_delay2_d<=max2_delay2;
  end if;
end process;

```

```

    end if;
end process;

--based on the the max_detect_1 or max_detect_2 signal fifo_din will get fifo_din_ch1 or
fifo_din_ch2 value
process(clk)
begin
    if clk='1' and clk'event then
        if max_detect_1_d='1' then
            fifo_din<=fifo_din_ch1(39 downto 4) & "00" & hk; --"000" & rd_count1 & "000"
& wr_count1 & "00" & fifo_din_ch1(13 downto 4) & "0000" ;    --anode number is
"00" from first anode
        elsif (max_detect_1_d='0' and max_detect_2_d='1') or max2_delay2_d='1' then
            fifo_din<= fifo_din_ch2(39 downto 4) & "01" & hk; --"000" & rd_count1 & "000"
& wr_count1 & "00" & fifo_din_ch2(13 downto 4) & "0000" ;    --anode number is
"01" from second anode
        end if;
    end if;
end process;

```

--t is used to identify the anode number generating the maximum
t<=chip_select_pin & max_detect_1 & max2_delay2 & max_detect_2;

```

--based on 't' value anode number is generated
process(clk)
begin
    if clk='1' and clk'event then
        case t(3 downto 0) is
            when "0100" => hk(1 downto 0)<="00";
            when "0001" => hk(1 downto 0)<="01";
            when "1100" => hk(1 downto 0)<="10";
            when "0010" => hk(1 downto 0)<="11";
            when others => hk(1 downto 0)<="ZZ";
        end case;
    end if;
end process;

```

--wr_en for fifo is generated here....using max_detect_1_d or max_delay2_d or
max_detect_2_d
wr_en<=(max_detect_1_d) or max2_delay2_d or max_detect_2_d;

--wr_en_d=wr_en...if the FIFO is not full

```

process(clk)
begin
  if clk'event and clk='1' then
    if full='0' then
      wr_en_d<=wr_en; --This delay corresponds to delay due to fifo_fin
    else
      wr_en_d<='0'; --if FIFO is full writing to FIFO is disabled
    end if;
  end if;
end process;

--based on frmae_counter and chip_select_pin reading from only one of the FPGAs FIFO
is enabled
process(clk)
begin
  if clk'event and clk='1' then
    if (chip_select_pin='1' and count_40frames0_2d='1') or (chip_select_pin='0' and
count_40frames(0)='0') then --- should be uncommented
      rd_enable_fifo<=control_rom_op(2);
    end if;
  end if;
end process;

--fifo_common component is instantiated here
fifo_block : fifo_common
  port map (
    din => fifo_din , --input to FIFO
    wr_en => wr_en_d, --writing enable to FIFO
    wr_clk => clk,    --write clock
    rd_en => rd_enable_fifo, --read enable signal
    rd_clk => clk,    --read clock
    ainit => '0',     --initialisation to '0'
    dout => fifo_dout, --fifo output
    full => full,     --full indicates if FIFO is full
    empty => empty,   --indicates if FIFO is empty
    almost_full => almost_full, --indicates FIFO has one more space to write into
    almost_empty => almost_empty, --indicates FIFO has one more space to read into

    rd_ack => rd_ack, --successful read acknowledgement
    rd_err => rd_err, --read request failure
    wr_ack => wr_ack, --successful write acknowledgement
    wr_err => wr_err); --write request failure

```

-- rom_sequence is a one clock high signal(high just before next 40 bits are to be loaded into shift register)...generated to change the contro rom addresss and also to read its contents

rom_sequence<=reset_m_delayed and control_rom_fifo_outbit;

--Whenever a major_frame is '1' it clears rom_access_4

--When rom_sequence becomes '1' it will load 3 into rom_access_4, which is used to access control rom 4 times

--after 10 bits are sent rom_access_4 is decremented once till its zero

process(major_frame,clk)

begin

if major_frame='1' then

rom_access_4<="000";

elsif clk='1' and clk'event then

if rom_sequence='1' then

rom_access_4<="011";

elsif rfd='1' and rom_access_en='1' then --rfd='1' when rfd is '1' it indicates

memory is ready for new data

rom_access_4<=rom_access_4-1;

end if;

end if;

end process;

--rom_access_en is hused to count down only till zero(rom_access_4 counter)

rom_access_en<=rom_access_4(2) or rom_access_4(1) or rom_access_4(0); --rom

--control rom read enable signal

rom_rd_en<=(rom_access_en and rfd) or rom_sequence;

--delay the major frame by '1' clock so that franme counter can be assigned to the

rom_addr

process(clk)

begin

if clk'event and clk='1' then

major_frame_d<=major_frame;

end if;

end process;

--When a major_frame is '1' the control_rom_counter is incremented to next 2^N state (2,4,8,16,32...etc)

process(clk)

begin

if clk'event and clk='1' then

```

    if major_frame='1'then
        control_rom_counter<=control_rom_counter+"1";
    end if;
end if;
end process;

--When major_frame delayed by 1 clock cycle is '1' then control_rom_counter is
addressed to control_rom address data
process(clk)
begin
    if clk='1' and clk'event then
        if major_frame_d='1' then
            rom_addr<=control_rom_counter & "00000000";
        elsif rom_rd_en='1' then
            rom_addr<=rom_addr+'1'; --rom_addr is incremented when control_rom read
signal is '1'
        end if;
    end if;
end process;

nd<=rfd; --when rfd is '1' it indicates memory is ready for new data

--control_rom component is instantiated here
rom_access: control_rom -- latency of 1 clock signal
port map (
    addr => rom_addr,    --rom address
    clk => clk,          --rom clock
    dout => rom_dout,    --rom output
    en => rom_rd_en,     -- rom enable
    nd => nd,            -- new data is accpted when it is '1'
    rfd => rfd,          -- when rfd is '1' it indicates memory is ready for new data
    rdy => control_rom_rdy); --indiated valid output at control_rom output port

--rom_rd_en is delayed by '1' clock
process(clk)
begin
    if clk='1' and clk'event then
        rom_rd_en_d<=rom_rd_en;
    end if;
end process;

```

```

--rom_rd_en_d is used to assign rom output to control_rom_op signal
process(clk)
begin
    if clk'event and clk='1' then
        if rom_rd_en_d='1' then
            control_rom_op<=rom_dout;
        else
            control_rom_op<="0000000000" & control_rom_op(1 downto 0);
--otherwise control_rom_op is cleared except 2 LSB bits
        end if;
    end if;
end process;

--one set of ion counters are reset at regular intervals
process(clk)
begin
    if clk'event and clk='1' then
        reset_counters1<= not(frame_counter_temp(0)) and control_rom_op(9); -- used to reset
the ion counters in z2_next file
    end if;
end process;

--other set of ion counters are reset at regular intervals
process(clk)
begin
    if clk'event and clk='1' then
        reset_counters2<=frame_counter_temp(0) and control_rom_op(9); -- used to reset the
ion counters in z2_next file
    end if;
end process;

--
--counter_sel(1 downto 0)<=frame_counter_temp(1 downto 0);

--frame_counter(0) bit is used to select which ion counters set is sent to outputs
process(p_load_en,frame_counter_temp(0),op_counter1_ch1(9 downto
0),op_counter2_ch1(9 downto 0),op_counter3_ch1 (9 downto 0),op_counter4_ch1(9
downto 0),op_counter1_ch2(9 downto 0),op_counter2_ch2(9 downto
0),op_counter3_ch2(9 downto 0),op_counter4_ch2(9 downto 0))
begin
    if frame_counter_temp(0)='0' and p_load_en='1' then
        mux_out_temp<= op_counter1_ch1(9 downto 0) & op_counter2_ch1(9 downto 0) &
op_counter3_ch1 (9 downto 0) & op_counter4_ch1(9 downto 0);
    elsif frame_counter_temp(0)='1' and p_load_en='1' then

```



```

        mux_out_temp<= op_counter1_ch2(9 downto 0) & op_counter2_ch2(9 downto 0) &
op_counter3_ch2 (9 downto 0) & op_counter4_ch2(9 downto 0);
    end if;
end process;
```

```
-- This process is used to store FIFO output when it is a valid FIFO output..that is when
we get rd_ack='1'
```

```
-- else zero is sent to the mux input
```

```
process(clk)
```

begin

```
if clk='1' and clk'event then
```

```
if rd_ack='1' then
```

```
fifo_dout_reg<=fifo_dout;
```

```
--elseif p_load_d='0' then
```

```
-- fifo_dout_reg<="111011101110111011101110111011101110";
```

```
elseif rd_err='1' then
```

```
fifo_dout_reg<="0000000000000000000000000000000000000000"; --else
```

fifo_dout_reg is cleared

```
--fifo_dout_reg<="000000000000000000000000000000000000000000";
```

end if;

end if;

end process;

--sel(1 to 0) is used to select whether frame_counter or MAX1231 ADc or FIFO data are sent to output

```
sel(1 downto 0) <= control_rom_op(1) & control_rom_op(0);
```

--This process implements the output pattern (whether frame_counter or MAX1231 ADC or FIFO data are sent to output)

```
process(sel(1 downto
```

```
0),max1231_reg1,max1231_reg2,max1231_reg3,max1231_reg4,frame_counter_temp(9
downto 0),mux_out_temp,fifo_dout_reg,rom_addr(10 downto
0),chip_select_pin,count 40frames)
```

begin

case sel(1 downto 0) is

when "00" =>

```
--& max1231_reg4(1 downto 0)
```

```

mux_out<=frame_counter_temp(9 downto 0) & "0000000000" & "0000000000" &
max1231_reg1(1 downto 0) & max1231_reg2(1 downto 0) & max1231_reg3(1 downto
0) & max1231_reg4(1 downto 0) & "00";

```

when "01" =>

```

mux_out<=max1231_reg1(11 downto 2) & max1231_reg2(11 downto 2) &
max1231_reg3(11 downto 2) & max1231_reg4(11 downto 2);

```

```

when "10"=>
    mux_out<= mux_out_temp(39 downto 0);
when "11" =>
    mux_out<= fifo_dout_reg(39 downto 0) ;
when others =>

mux_out<="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
end case;
end process;

--chip_select_pin and frame counter (count_40frames) is used to make one FPGA to send
data out
process(chip_select_pin,count_40frames(0),count_40frames0_2d)
begin
    if chip_select_pin='1' and count_40frames0_2d='1' then
        shift_out_en_temp<='1';
    elsif chip_select_pin='0' and count_40frames(0)='0' then
        shift_out_en_temp<='1';
    else
        shift_out_en_temp<='0';
    end if;
end process;

--40 bits output shift register is instantiated here
send_to_otherboard: shift_40out
    port map (
        CLK => gated_clock,
        SDOUT => otherboard_sdout_temp,
        P_LOAD => p_load_d,
        D => mux_out,
        ce => '1');

--One of the FPGAs not selected to send data out is assigned to high impedance
--other FPGA sends its data out
process(shift_out_en_temp,otherboard_sdout_temp)
begin
    if shift_out_en_temp='1' then
        otherboard_sdout<= otherboard_sdout_temp;
    else
        otherboard_sdout<='Z';
    end if;
end process;

```

```

---MAX1231 part
-- this signal is used to enable reading from the max_1231_ram block ...
-- this is enabled when only the frame_counter(0) and chip_select_pin are same...
-- in order for each FPGA to read from it's max_RAM block only when it is sending the
data...
rd_enable_adc<=not(shift_out_en_temp) and control_rom_op(4);

```

```

--MAX1231 component is instantiated here

```

```

adc_control: max1231_top
  Port map( clk =>clk,
    sclk =>max1231_sclk,
    sdin =>max1231_sdin,
    control_rom_op =>control_rom_op(3),
    sdout =>max1231_sdout,
    cs_bar => max1231_cs_bar,
    QSPO_CE =>rd_enable_adc,
    test_sdout => test_sdout,
    -- tx1_reg => tx1_reg,
    --sclk_out_t,sclk_in_t: out std_logic;
    --tx_reg_t:out std_logic_vector(7 downto 0);
    --read_counter_ored_t:out std_logic_vector(5 downto 0);
    --max1231_we_t:out std_logic;
    --temp_t:out std_logic_vector(15 downto 0);
    --dist_ram_en_t:out std_logic_vector(1 downto 0);
    --sclk_in_counter_t:out std_logic_vector(4 downto 0);
    --max1231_addr_t:out std_logic_vector(3 downto 0);
    max1231_dout =>max1231_dout,
    --sclk_ored_t => sclk_ored_t,
    eoc_bar =>max1231_eoc_bar);

```

```

-- 4 registers are used to store the successive 4 MAX1231 outputs(each output is 10bits
wide) before they are sent to shift register(40 bits)
-- 4 registers(each one bit) are used to delay the enable control_rom_op(5,6,7,8) bit one
clock cycle..so that MAX1231 output can be assigned properly
-- to enable the upper 4 registers...

```

```

process(clk)
begin
  if clk='1' and clk'event then
    shift_reg_1<=control_rom_op(5); --control_rom_op(5) bit is delayed by '1' clock
cycle
  end if;
end process;

```

```

process(clk)
begin
    if clk='1' and clk'event then
        shift_reg_2<=control_rom_op(6);
    end if;
end process;

```

```

process(clk)
begin
    if clk='1' and clk'event then
        shift_reg_3<=control_rom_op(7);
    end if;
end process;

```

```

process(clk)
begin
    if clk='1' and clk'event then
        shift_reg_4<=control_rom_op(8);
    end if;
end process;

```

```

process(clk)
begin
    if clk='1' and clk'event then
        if shift_reg_1='1' then
            max1231_reg1<=max1231_dout; --MAX1231 output is assigned to
max1231_reg1 which is sent to output
        end if;
    end if;
end process;

```

```

process(clk)
begin
    if clk='1' and clk'event then
        if shift_reg_2='1' then
            max1231_reg2<=max1231_dout;
        end if;
    end if;
end process;

```

```

process(clk)
begin
    if clk='1' and clk'event then
        if shift_reg_3='1' then
            max1231_reg3<=max1231_dout;
        end if;
    end if;
end process;

```

```

process(clk)
begin
    if clk='1' and clk'event then
        if shift_reg_4='1' then
            max1231_reg4<=max1231_dout;
        end if;
    end if;
end process;

```

--generating major_frame signal

--A minor frame sync signal is used as clock to increment count_40frames counter

```

process(major_frame ,minor_frame_sync)
begin
    if major_frame='1' then
        count_40frames<="000000"; -- count_40frames counter is reset after every 40
frames are obtained
    elsif minor_frame_sync'event and minor_frame_sync='1' then
        count_40frames<=count_40frames+"000001";
    end if;
end process;

```

-- count_40frames(0) is delayed once and it is used to select which FPGA will send out the data

```

process(clk)
begin
    if clk'event and clk='1' then
        count_40frames0_d<=count_40frames(0);
    end if;
end process;

```

-- count_40frames(0) is delayed again and it is also used to select which FPGA will send out the data

```
process(clk)
begin
    if clk'event and clk='1' then
        count_40frames0_2d<=count_40frames0_d;
    end if;
end process;
```

--major_frame_temp is '1' when count_40frames=40
 major_frame_temp<=count_40frames(5) and not(count_40frames(4)) and
 count_40frames(3) and not(count_40frames(2)) and not(count_40frames(1)) and
 not(count_40frames(0));

--major_frame is generated using major_frame_temp

```
process(clk)
begin
    if clk'event and clk='1' then
        major_frame<=major_frame_temp ;
    end if;
end process;
```

--A major frame counter frame_counter_temp is incremented here

```
process(clk)
begin
    if clk='1' and clk'event then
        if major_frame_d='1' then
            frame_counter_temp<=frame_counter_temp + "0000000001";
        end if;
    end if;
end process;
```

--frame_counter is sent out

```
frame_counter<=frame_counter_temp;
```

----- rom data format-----

```
--16 --
--15--
--14--
--13--
--12--
--11--
```

```

--10--
--09-- reset counters--synchronous reset
--08-- enable register 4 for holding first 10 bits of max 1231 data
--07-- enable register 3 for holding first 10 bits of max 1231 data
--06-- enable register 2 for holding first 10 bits of max 1231 data
--05-- enable register 1 for holding first 10 bits of max 1231 data
--04-- read distributed RAM contents(max 1231 data)--will be one for four continuos
locations in ROM..to read four form Distributed RAM
--03-- send conversion register data so that max1231 can start conversion of 0-14+temp
data ..conversion register data is MSB when this bit is '1'
--02-- FIFO read signal should be anded with p_load or ??? will be one from 3a-to
39a...39d
--01-- select 1
--00-- select 0

```

```

--wr_en_t<=wr_en;
--wr_en_d_t<=wr_en_d;
--wr_en_2d_t<=wr_en_2d;
--rd_en_t<=rd_en;
-- reset_m_t<=WR_EN;
-- reset_delayed_t<=p_load;
--word_clock_t<=word_clock;

--rom_addr_t<= rom_addr;
--rom_dout_t<= rom_dout;

--rom_rd_en_t<= rom_rd_en_d;
--rom_sequence_t<=rom_sequence;
--control_rom_rdy_t<=control_rom_rdy;
--rom_access_4_t<= rom_access_4;
--p_load_t<=p_load;
--p_load_d_t<=p_load_d;
--p_load_en_t<=p_load_en;
--sel_t<=sel;
--shift_out_en_t<=SEL(1);
--control_rom_op_t<=control_rom_op;
--mux_out_t<=mux_out;
-- mux_out_temp_t<= mux_out_temp;
--shift_out_en_temp_t <= sclk_ored_t;
--fifo_din_t<=fifo_dOUT;
--rd_ack_t<=rd_ack;

```

```

-- count_40frames_t<=count_40frames;
--frame_sync_t<=frame_sync;
--gated_clock_t_t<=gated_clock_t;
--reset_control_rom_fifo_outbit_t<=reset_control_rom_fifo_outbit;
--control_rom_fifo_outbit_t<=control_rom_fifo_outbit;
--reset_delayed_t<=reset_delayed;
--reset_m_t<=reset_m;
--count_4_t_t<= count_4;
--fifo_dout_t<=fifo_dout;
--max_detect_1_d_t<=max_detect_1_d;
--max2_delay2_t<=max2_delay2;
--test_sdout(1 downto 0)<=rom_addr(3 downto 2);
--rd_enable_fifo_t<=rd_enable_adc;
--rd_enable_fifo_t2<=rom_rd_en;
end behavioral ;

```



```
-- This file implements the fitting logic (2 ADC outputs are fit), maximum detection logic
-- Division logic to identify ion
-- 4 ion counters (incremented using division result)
-- 40 bits to send to FIFO are combined here
-- 40 bits =(2*13bits pulse height values + 10 bits master counter values + 2 anode
identify bits + 2 ADC overflow bits)
```

```
--This file is instantiated twice in main_module.vhd file to implement the fitting logic for
two wedge patterns
```

```
--Inputs
```

```
-- clk: 20MHz clock input
-- clk_2x :40MHz clock input
-- reset_counters: to reset ion counters
-- master_data: External (timing or reconfiguration counter values) 10 bits
-- data_ch1:8 bit input from a ADC
-- data_ch2:8 bit input from a ADC (data_ch1 and data_ch2 come from acommon
wedge board)
```

```
--Outputs
```

```
-- max_detect :'1' bit output indicates the detection of maximum
-- op_counter1: "10" bit ion counter 1 output
-- op_counter2: "10" bit ion counter 2 output
-- op_counter3: "10" bit ion counter 3 output
-- op_counter4: "10" bit ion counter 4 output
-- fifo_din: 40 bits data to be sent to FIFO
```

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

```
--components instantiated from the core generator are
```

```
--add_20bit,
add_14bit,counter_4bit,counter_16,counter_10,shift_serial,fir_filter_1,fir_filter_2,shift_r
eg13,compare_con,comparator_gt,sub_19
```

```
library UNISIM;
use UNISIM.VComponents.all;
```

```
-- This entity(z2_chs_next) takes inputs from 2 ADCs (i.e from one wedge anode
detectors)
-- each input is given to two FIR filters (fir_filter_1 and fir_filter_2) and are fit and
presence of maximum is checked
```

-- Total of 4 FIR filters are instantiated in this file

entity z2_chs_next is

```
Port ( clk : in std_logic;
      --clk_2:in std_logic; --clk/2
      --clk_4:in std_logic; --clk/4
      clk_2x:in std_logic;
```

```
-----
      --fir_op_ch1_t,fir_op_ch2_t,fir_op_ch21_t,fir_op_ch22_t:out std_logic_vector(17
downto 0);
```

```
      --max_found_t,max_found1_t,max_found2_t:out std_logic; --ch2_shreg5_t,
, ch2_shreg1_t
```

```
      --fir1_ch1_data_t,fir2_ch1_data_t:out std_logic_vector(19 downto 0);
```

```
      --q_shreg5_t,q_shreg1_t:out std_logic_vector(12 downto 0);
```

```
      --group1_sum_t: out std_logic_vector(20 downto 0);
```

```
--
```

```
q_shreg2_t,q_shreg3_t,q_shreg4_t,q_shreg6_t,q_shreg7_t,q_shreg8_t,q_shreg9_t:out
std_logic_vector(11 downto 0);
```

```
      --initiate_ch2_t,cee2_t,cee1_t:out std_logic;
```

```
      --op_down_ch1_t,op_down_ch2_t:out std_logic_vector(3 downto 0);
```

```
      --A1_t,B1_t,A1_sub19_t,B1_sub19_t:out std_logic_vector(13 downto 0);
```

```
      --A1_temp_t,B1_temp_t:out std_logic_vector(13 downto 0);
```

```
      --A1_scaled_t,sum_final_scaled_t:out std_logic_vector(27 downto 0);
```

```
      --quo_t:out std_logic_vector(13 downto 0);
```

```
      --locked:inout std_logic;
```

```
      --clk_out:out std_logic;
```

```
      --delay_op_t:out std_logic;
```

```
      --fir1_ch1_maxdata_t:out std_logic_vector(19 downto 0);
```

```
      --fir2_ch1_maxdata_t:out std_logic_vector(19 downto 0);
```

```
      --fir1_ch2_maxdata_t:out std_logic_vector(19 downto 0);
```

```
      --fir2_ch2_maxdata_t:out std_logic_vector(19 downto 0);
```

```
      --quo_t:out std_logic_vector(27 downto 0);
```

```
      --op_down_ch1_t:out std_logic_vector(4 downto 0);
```

```
      --disable_1_1_t,rdy_fir1_ch1_2d_t,rdy_fir1_ch2_2d_t:out std_logic;
```

```
      --disable_1_t,disable_2_t:out std_logic;
```

```
      --max_found_t:out std_logic;
```

```
      --group1_sum_temp_t,group2_sum_temp_t:out std_logic_vector(20 downto 0);
```

```
      --ch2_maxdata_t,ch1_maxdata_t:out std_logic_vector(11 downto 0);
```

```
      --A_t,B_t:out std_logic_vector(12 downto 0);
```

```
      --sum_final_t:out std_logic_vector(14 downto 0);
```

```
      --rdy_fir1_ch2_t,rdy_fir1_ch1_t:out std_logic;
```

```
      --fir1_ch2_maxdata_t,fir1_ch1_maxdata_t:out std_logic_vector(11 downto 0);
```

```
      --A1_t,quo_t :out std_logic_vector(28 downto 0);
```

```

--max_found_t,max_found1_t,max_found2_t:out std_logic;
--chip_select_pin:in std_logic;
--delay_op_t,cntr1_t:out std_logic;
--max_add1_t,max_add2_t,max_add3_t,max_add4_t,max_add5_t:out std_logic;
--cntr1_t:out std_logic;
--delay_op_t:out std_logic;
--quo_t:out std_logic_vector(23 downto 0);
--cntr1_t,cntr2_t,cntr3_t,cntr4_t,n1gt_t:out std_logic;
-----
reset_counters: in std_logic;
max_detect:out std_logic;
master_data:in std_logic_vector(9 downto 0);
data_ch1 : in std_logic_vector(7 downto 0);
data_ch2 : in std_logic_vector(7 downto 0);
op_counter1,op_counter2,op_counter3,op_counter4:out std_logic_vector(9
downto 0);
fifo_din:out std_logic_vector(39 downto 0));

end z2_chs_next;

```

architecture Behavioral of z2_chs_next is

--"add_20bit" component adds two 20 bit numbers(signed) which are the FIR filter outputs

```

component add_20bit
  port (
    A: IN std_logic_VECTOR(19 downto 0);
    B: IN std_logic_VECTOR(19 downto 0);
    S: OUT std_logic_VECTOR(20 downto 0));
end component;

```

--"add_14bit" component adds two 14 bit numbers(unsigned).Inputs are C1 and C2
component add_14bit

```

  port (
    A: IN std_logic_VECTOR(13 downto 0);
    B: IN std_logic_VECTOR(13 downto 0);
    Q: OUT std_logic_VECTOR(14 downto 0);
    CLK: IN std_logic;
    CE: IN std_logic);
end component;

```

--"compare_3" component(compare_3.vhdl)compares the division output with 3 constants and increments one of six signals which increment one of four ion counters

```

component compare_3 is
  Port ( ip_n1 : in std_logic_vector(7 downto 0);
        n1gt : out std_logic;
        n1lt : out std_logic;
        n2gt : out std_logic;
        n2lt : out std_logic;
        n3gt : out std_logic;
        n3lt : out std_logic);
end component;

```

```

component counter_4bit
  port (
    Q: OUT std_logic_VECTOR(3 downto 0);
    CLK: IN std_logic;
    SSET: IN std_logic);
end component;

```

```

component counter_16
  port (
    Q: OUT std_logic_VECTOR(15 downto 0);
    CLK: IN std_logic;
    CE: IN std_logic);
end component;

```

--"counter_10" component counts ions..4 counter_10 are instantiated for ions

```

component counter_10
  port (
    Q: OUT std_logic_VECTOR(9 downto 0);
    CLK: IN std_logic;
    CE: IN std_logic;
    SCLR: IN std_logic);
end component;

```

--"divide_top" component(divide.vhdl) divides two 24 bit numbers(a1/(a1+b1))

```

component divide_top
  Port ( clk : in std_logic;
        dividend : in std_logic_vector(23 downto 0);
        divisor : in std_logic_vector(23 downto 0);
        start : in std_logic;
        DELAY_OP:OUT STD_LOGIC;

```

```

        quot: out std_logic_vector(23 downto 0));
end component;

```

```

component shift_serial
  port (
    CLK: IN std_logic;
    SDIN: IN std_logic;
    Q: OUT std_logic_VECTOR(2 downto 0));
end component;

```

```

component fir_filter
  port (
    ND: IN std_logic;
    RDY: OUT std_logic;
    CLK: IN std_logic;
    RFD: OUT std_logic;
    DIN: IN std_logic_VECTOR(7 downto 0);
    DOUT: OUT std_logic_VECTOR(20 downto 0));
end component;

```

--"fir_filter_1" component instantiates one FIR filter 1 with a(n) coefficients

```

component fir_filter_1
  port (
    ND: IN std_logic;
    RDY: OUT std_logic;
    CLK: IN std_logic;
    RFD: OUT std_logic;
    DIN: IN std_logic_VECTOR(7 downto 0);
    DOUT: OUT std_logic_VECTOR(19 downto 0));
end component;

```

--"fir_filter_1" component instantiates one FIR filter 2 with b(n) coefficients

```

component fir_filter_2
  port (
    ND: IN std_logic;
    RDY: OUT std_logic;
    CLK: IN std_logic;
    RFD: OUT std_logic;
    DIN: IN std_logic_VECTOR(7 downto 0);
    DOUT: OUT std_logic_VECTOR(19 downto 0));
end component;

```

--"shift_reg_13" component is used to store the 13 bit results C1 or C2 when maximum is detected

```

component shift_reg13
    port (
        D: IN std_logic_VECTOR(12 downto 0);
        Q: OUT std_logic_VECTOR(12 downto 0);
        CLK: IN std_logic;
        CE: IN std_logic;
        SCLR: IN std_logic);
end component;

--"shift_reg" component is used for maximum detection
component shift_reg
    port (
        D: IN std_logic_VECTOR(12 downto 0);
        Q: OUT std_logic_VECTOR(12 downto 0);
        CLK: IN std_logic;
        CE: IN std_logic);
end component;

--"comparator_gt" component is used for maximum detection
component comparator_gt
    port (
        A: IN std_logic_VECTOR(12 downto 0);
        B: IN std_logic_VECTOR(12 downto 0);
        A_GT_B: OUT std_logic);
end component;

--"comparator_con" component is used for maximum detection (compare with some
constant)
component compare_con
    port (
        A: IN std_logic_VECTOR(12 downto 0);
        A_GT_B: OUT std_logic);
end component;

--"sub_19" component subtracts 19 hex from both the wedge outputs when a maximum is
detected
component sub_19
    port (
        A: IN std_logic_VECTOR(13 downto 0);
        Q: OUT std_logic_VECTOR(13 downto 0);
        CLK: IN std_logic;
        CE: IN std_logic);
end component;

```

--Intermediate signals

```
signal max_found1,max_found,max_found2 :std_logic:='0';
signal n1gt,n1lt,n2gt,n2lt,n3gt,n3lt,cntr1,cntr2,cntr3,cntr4:std_logic:='0';
signal start_div:std_logic:='0';
signal disable_1,disable_1_1:std_logic:='0';
signal op_down_ch1:std_logic_vector(4 downto 0):="00000";
signal max_found_ch2,max_found_ch1:std_logic_vector(8 downto 0):="0000000000";
signal q_shreg9,q_shreg8,q_shreg7,q_shreg6,q_shreg5,
      q_shreg4,q_shreg3,q_shreg2,q_shreg1:std_logic_vector(12 downto 0):='0' &
X"000";
signal ch2_shreg9,ch2_shreg8,ch2_shreg7,ch2_shreg6,ch2_shreg5,
      ch2_shreg4,ch2_shreg3,ch2_shreg2,ch2_shreg1:std_logic_vector(12 downto 0):='0'
& X"000";
```

```
signal max_found_temp:std_logic:='0';
signal nd_fir1_ch1,rdy_fir1_ch1,rfd_fir1_ch1,rdy_fir2_ch1,rfd_fir2_ch1:std_logic:='0';
signal nd_fir2_ch1:std_logic:='0';
signal nd_fir1_ch2,rdy_fir1_ch2,rfd_fir1_ch2:std_logic:='0';
signal nd_fir2_ch2,rdy_fir2_ch2,rfd_fir2_ch2:std_logic:='0';
signal fir1_ch1_data,fir2_ch1_data,fir1_ch2_data,fir2_ch2_data:std_logic_vector(19
downto 0):="000000000000000000000000";
signal ch1_maxdata,ch2_maxdata:std_logic_vector(12 downto 0):='0' & x"000";
signal max_add1,max_add2,max_add3,max_add4,max_add5:std_logic:='0';
signal sum_final:std_logic_vector(14 downto 0):="0000000000000000";
signal
delay_op,disable_1_temp,rdy_fir1_ch2_d,rdy_fir1_ch1_d,rdy_fir1_ch1_2d,rdy_fir1_ch2
_2d:std_logic:='0';
```

```
signal group1_sum,group2_sum:std_logic_vector(20 downto
0):="000000000000000000000000";
signal group1_sum_temp,group2_sum_temp:std_logic_vector(12 downto 0):="0" &
x"000";
signal A,B:std_logic_vector(12 downto 0):='0' & X"000";
signal A1_scaled,sum_final_scaled,quo:std_logic_vector(23 downto 0):= X"000000";
signal A1,B1:std_logic_vector(13 downto 0):="0000000000000000";
signal A1_temp,B1_temp,A1_sub19,B1_sub19:std_logic_vector(13 downto
0):="0000000000000000";
signal A1_sub19_ored,B1_sub19_ored:std_logic:='0';
begin
```

-----channel 1 fir filter-----

-- fir_ch1 block implements the FIR operation between F1(n) and a ADC output.

-- Generated from the core generator fir_ch1 block has

--inputs

-- clk : 40MHz clock
 -- nd_fir1_ch1 : FIR filter waits for this signal to go high to accept new input data
 -- data_ch1 : inputs representing pulse samples(one of ADC outputs)

--outputs

-- rdy_fir1_ch1 : When a FIR output is ready this signal is made high for one clock cycle

-- rfd_fir1_ch1 : This signal indicates that FIR filter can accept new data
 -- fir1_ch1_data : 20bits of FIR filter 1 result

fir1_ch1: fir_filter_1

```
port map (
  ND => nd_fir1_ch1,
  RDY => rdy_fir1_ch1,
  CLK => clk_2x,
  RFD => rfd_fir1_ch1,
  DIN => data_ch1,
  DOUT => fir1_ch1_data);
```

nd_fir1_ch1<=rfd_fir1_ch1;

-- fir2_ch1 block implements the FIR operation between F2(n) and a ADC output.

-- Generated from the core generator fir2_ch1 block has

--inputs

-- clk : 40MHz clock
 -- nd_fir2_ch1 : When next data should be received this signal should be made high
 -- data_ch1 : Filter input data sample representing pulse samples (one of ADC outputs same input as for fir1_ch1 block)

--outputs

-- rdy_fir2_ch1 : When a FIR output is ready this signal is made high for one clock cycle

-- rfd_fir2_ch1 : This signal indicates that FIR filter can accept new data
 -- fir2_ch1_data : 20bits of FIR filter 2 result

fir2_ch1: fir_filter_2

```
port map (
  ND => nd_fir2_ch1,
  RDY => rdy_fir2_ch1,
  CLK => clk_2x,
  RFD => rfd_fir2_ch1,
  DIN => data_ch1,
```



```

        DOUT => fir2_ch1_data);

nd_fir2_ch1<=rfd_fir2_ch1;

-- Outputs of the two FIR filters are added here
-- Its a signed operation
-- inputs : fir1_ch1_data and fir2_ch1_data
-- outputs:group1_sum
group1 : add_20bit
    port map (
        A => fir1_ch1_data(19 downto 0),
        B => fir2_ch1_data(19 downto 0),
        S => group1_sum(20 downto 0));

-- Only 13 bits of group1_sum bits are stored(considered for further calculations) in
signal group1_sum_temp signal
group1_sum_temp(12 downto 0)<=group1_sum(17 downto 5);

-- Since FIR filters run at 40MHz and the maximum detection logic at 20MHz
-- Whenever FIR filter outputs start,
-- following two process logics are used to enable maximum detection logic
-- rdy_fir1_ch1 signal is delayed by one cycle in the first process
process(clk_2x)
begin
if clk_2x'event and clk_2x='1' then
    rdy_fir1_ch1_d<=rdy_fir1_ch1;
end if;
end process;

-- rdy_fir1_ch1_2d signal is made high all the time except when FIR filters are not
working
process(clk_2x)
begin
if clk_2x'event and clk_2x='1' then
    rdy_fir1_ch1_2d<=rdy_fir1_ch1_d or rdy_fir1_ch1;
end if;
end process;

-- maximum is calculated from the group1_sum_temp signal
-- below part is for finding the maximum in 9 samples ( 9 shift registers are used) coming
out of the group1 block

-- shiftregch1_1 is the first shift register

```

```

-- Input :group1_sum_temp
-- Output: q_shreg1
shiftregch1_1:shift_reg
    port map (
        D => group1_sum_temp(12 downto 0),  -- 13 bits of
group1_sum_temp are used
        Q => q_shreg1,
        CLK => clk,                        -- clk used is 20MHz
        CE=>rdy_fir1_ch1_2d);             -- ce is clock enable and is enabled all the time
except when FIR filters are not working

```

```

-- shiftregch1_2 is the second shift register

```

```

-- Input : q_shreg1
-- Output: q_shreg2
shiftregch1_2: shift_reg
    port map (
        D => q_shreg1,
        Q => q_shreg2,
        CLK => clk,
        CE=>rdy_fir1_ch1_2d);

```

```

-- shiftregch1_3 is the third shift register

```

```

-- Input : q_shreg2
-- Output: q_shreg3
shiftregch1_3: shift_reg
    port map (
        D => q_shreg2,
        Q => q_shreg3,
        CLK => clk,
        CE=>rdy_fir1_ch1_2d);

```

```

-- shiftregch1_4 is the fourth shift register

```

```

-- Input : q_shreg3
-- Output: q_shreg4
shiftregch1_4: shift_reg
    port map (
        D => q_shreg3,
        Q => q_shreg4,
        CLK => clk,
        CE=>rdy_fir1_ch1_2d);

```

```

-- shiftregch1_5 is the fifth shift register

```

```

-- Input : q_shreg4
-- Output: q_shreg5

```

```

shiftregch1_5: shift_reg
    port map (
        D => q_shreg4,
        Q => q_shreg5,
        CLK => clk,
        CE=>rdy_fir1_ch1_2d);

-- shiftregch1_6 is the sixth shift register
-- Input : q_shreg5
-- Output: q_shreg6
shiftregch1_6: shift_reg
    port map (
        D => q_shreg5,
        Q => q_shreg6,
        CLK => clk,
        CE=>rdy_fir1_ch1_2d);

-- shiftregch1_7 is the seventh shift register
-- Input : q_shreg6
-- Output: q_shreg7
shiftregch1_7: shift_reg
    port map (
        D => q_shreg6,
        Q => q_shreg7,
        CLK => clk,
        CE=>rdy_fir1_ch1_2d);

-- shiftregch1_8 is the eighth shift register
-- Input : q_shreg7
-- Output: q_shreg8
shiftregch1_8: shift_reg
    port map (
        D => q_shreg7,
        Q => q_shreg8,
        CLK => clk,
        CE=>rdy_fir1_ch1_2d);

-- shiftregch1_9 is the ninth shift register
-- Input : q_shreg8
-- Output: q_shreg9
shiftregch1_9: shift_reg
    port map (
        D => q_shreg8,
        Q => q_shreg9,
        CLK => clk,

```

```
CE=>rdy_fir1_ch1_2d);
```

```
--now comparators are used for finding the maximum
```

```
--cmp1_9 block compares 5th shift reg value with 9th shift reg value
```

```
cmp1_9:comparator_gt
```

```
    port map (
```

```
        A => q_shreg5,
```

```
        B => q_shreg9,
```

```
        A_GT_B => max_found_ch1(7));
```

```
--cmp1_8 block compares 5th shift reg value with 8th shift reg value
```

```
cmp1_8:comparator_gt
```

```
    port map (
```

```
        A => q_shreg5,
```

```
        B => q_shreg8,
```

```
        A_GT_B => max_found_ch1(6));
```

```
--cmp1_7 block compares 5th shift reg value with 7th shift reg value
```

```
cmp1_7:comparator_gt
```

```
    port map (
```

```
        A => q_shreg5,
```

```
        B => q_shreg7,
```

```
        A_GT_B => max_found_ch1(5));
```

```
--cmp1_9 block compares 5th shift reg value with 6th shift reg value
```

```
cmp1_6:comparator_gt
```

```
    port map (
```

```
        A => q_shreg5,
```

```
        B => q_shreg6,
```

```
        A_GT_B => max_found_ch1(4));
```

```
--cmp1_9 block compares 5th shift reg value with 4th shift reg value
```

```
cmp1_4:comparator_gt
```

```
    port map (
```

```
        A => q_shreg5,
```

```
        B => q_shreg4,
```

```
        A_GT_B => max_found_ch1(3));
```

```
--cmp1_9 block compares 5th shift reg value with 3th shift reg value
```

```
cmp1_3:comparator_gt
```

```
    port map (
```

```
        A => q_shreg5,
```

```
        B => q_shreg3,
```

```
        A_GT_B => max_found_ch1(2));
```

--cmp1_9 block compares 5th shift reg value with 2th shift reg value

```
cmp1_2:comparator_gt
    port map (
        A => q_shreg5,
        B => q_shreg2,
        A_GT_B => max_found_ch1(1));
```

--cmp1_9 block compares 5th shift reg value with 1th shift reg value

```
cmp1_1:comparator_gt
    port map (
        A => q_shreg5,
        B => q_shreg1,
        A_GT_B => max_found_ch1(0));
```

--cmp1_9 block compares 5th shift reg value with some constant(1A0 Hex)

```
cmpcon: compare_con
    port map (
        A => q_shreg5,
        A_GT_B => max_found_ch1(8));
```

--"AND" the output of all the comparators and if output is '1' pulse is present

```
max_found1<=(max_found_ch1(8) and max_found_ch1(7)and max_found_ch1(6)and
max_found_ch1(5)and max_found_ch1(4)
    and max_found_ch1(3)and max_found_ch1(2)and max_found_ch1(1)and
max_found_ch1(0));
```

-----Channel 1 next part -----

-- This part is used for enabling a 5 bit counter when a maximum is found

-- This counter is used to reset the register used to hold the maximum FIR output, till the counter counts to zero or till division is process is done

```
ch1_counter:process(max_found,clk)
begin
    if max_found='1' then
        op_down_ch1 <="11111"; -- load the counter value when max is found with
"11111"
    elsif clk='1' and clk'event then
        if delay_op='1' then
            op_down_ch1<="00000"; --when division is done reset the counter
        elsif disable_1_temp='1' then -- if disable_1 is '1' then max has been found and we
should start down counting but till zero
            op_down_ch1<=op_down_ch1 - 1;
        end if;
```

```

    end if;
end process ch1_counter;

-- disable_1 is generated by "OR"ing from the counter outputs
disable_1_temp<=(op_down_ch1(4) or op_down_ch1(3) or op_down_ch1(2) or
op_down_ch1(1) or op_down_ch1(0));

--D-FF is used to delay the disable_1 by 1 clock
process(clk)
begin
    if clk='1' and clk'event then
        disable_1<=disable_1_temp;
    end if;
end process;

-- If division process is done (given by delay_op=1) reset the signal disable_1_1 which is
used to reset the counter
-- Else disable_1_1 signal will get values of disable_1 (disable_1 if it is '1', it will lead to
resetting of the counter)
process(clk)
begin
    if clk'event and clk='1' then
        if delay_op='1' then
            disable_1_1<='0';
        else
            disable_1_1<=disable_1;
        end if;
    end if;
end process;

-- Register to hold the data for division process.. loading is enabled by max_found_temp
and it's kept clearing till
-- diable_1_1 is high
-- diable_1_1 is high till 5 bit counter counts to 0 or till division is going on
disable_1_disables : shift_reg13
port map (
    D => q_shreg5,
    Q => ch1_maxdata, -- maximum value of channel 1 (i.e. c)
    CLK => clk,
    CE => max_found_temp,
    SCLR => disable_1_1);

```

----- second set of FIR filters for other output of wedge anode board-----

-- fir_ch2 block implements the FIR operation between F1(n) and a ADC output.

-- Generated from the core generator fir_ch2 block has

--inputs

-- clk : 40MHz clock

-- nd_fir1_ch2 : FIR filter waits for this signal to go high to accept new input data

-- data_ch2 : 8 bit input representing pulse samples(one of ADC outputs)

--outputs

-- rdy_fir1_ch2 : When a FIR output is ready this signal is made high for one clock cycle

-- rfd_fir1_ch2 : This signal indicates that FIR filter can accept new data

-- fir1_ch2_data : 20bits of FIR filter 1 result

fir1_ch2: fir_filter_1

port map (

ND => nd_fir1_ch2,

RDY => rdy_fir1_ch2,

CLK => clk_2x,

RFD => rfd_fir1_ch2,

DIN => data_ch2,

DOUT => fir1_ch2_data);

nd_fir1_ch2<=rfd_fir1_ch2;

-- fir2_ch2 block implements the FIR operation between F2(n) and a ADC output.

-- Generated from the core generator fir2_ch2 block has

--inputs

-- clk : 40MHz clock

-- nd_fir2_ch2 : FIR filter waits for this signal to go high to accept new input data

-- data_ch2 : Filter input data sample representing pulse samples (one of ADC outputs same input as for fir1_ch2 block)

--outputs

-- rdy_fir2_ch2 : When a FIR output is ready this signal is made high for one clock cycle

-- rfd_fir2_ch2 : This signal indicates that FIR filter can accept new data

-- fir2_ch2_data : 20bits of FIR filter 2 result

fir2_ch2: fir_filter_2

port map (

ND => nd_fir2_ch2,

RDY => rdy_fir2_ch2,

CLK => clk_2x,

RFD => rfd_fir2_ch2,

```

        DIN => data_ch2,
        DOUT => fir2_ch2_data);

    nd_fir2_ch2<=rfd_fir2_ch2;

-- Outputs of the two FIR filters are added here
-- Its a signed operation
-- inputs : fir1_ch2_data and fir2_ch2_data
-- outputs:group1_sum
group2 : add_20bit
    port map (
        A => fir1_ch2_data(19 downto 0),
        B => fir2_ch2_data(19 downto 0),
        S => group2_sum(20 downto 0));

-- Only 13 bits of group1_sum bits are stored(considered for furtehr calculations) in
signal group1_sum_temp signal
group2_sum_temp<=group2_sum(17 downto 5);

-- Since FIR filters run at 40MHz and the maximum detection logic at 20MHz
-- Whenever FIR filter outputs start,
-- following two process logics are used to enable maximum detection logic
-- rdy_fir1_ch2 signal is delayed by one cycle in the first process
process(clk_2x)
begin
if clk_2x'event and clk_2x='1' then
    rdy_fir1_ch2_d<=rdy_fir1_ch2;
end if;
end process;

-- rdy_fir1_ch2_2d signal is made high all the time, except when FIR filters are not
working
process(clk_2x)
begin
if clk_2x'event and clk_2x='1' then
    rdy_fir1_ch2_2d<=rdy_fir1_ch2_d or rdy_fir1_ch2;
end if;
end process;

```



```

----- channel 2 fir filter 1 next part -----
-- maximum is calculated from the group2_sum_temp signal
-- below part is for finding the maximum in 9 samples (9 shift registers are used) coming
out of the group2 block

-- shiftregch2_1 is the first shift register
-- Input :group2_sum_temp
-- Output: ch2_shreg1
shiftregch2_1:shift_reg
  port map (
    D => group2_sum_temp(12 downto 0), --13 bits of group2_sum_temp are used
    Q => ch2_shreg1,
    CLK => clk,
    CE => rdy_fir1_ch2_2d); -- ce is clock enable and is enabled all the time
except when FIR filters are not working

-- shiftregch1_2 is the second shift register
-- Input : ch2_shreg1
-- Output: ch2_shreg2
shiftregch2_2: shift_reg
  port map (
    D => ch2_shreg1,
    Q => ch2_shreg2,
    CLK => clk,
    CE => rdy_fir1_ch2_2d);

-- shiftregch1_2 is the second shift register
-- Input : ch2_shreg2
-- Output: ch2_shreg3
shiftregch2_3: shift_reg
  port map (
    D => ch2_shreg2,
    Q => ch2_shreg3,
    CLK => clk,
    CE => rdy_fir1_ch2_2d);

-- shiftregch1_4 is the fourth shift register
-- Input : ch2_shreg3
-- Output: ch2_shreg4
shiftregch2_4: shift_reg
  port map (
    D => ch2_shreg3,
    Q => ch2_shreg4,
    CLK => clk,

```

```

        CE=>rdy_fir1_ch2_2d);

-- shiftregch1_5 is the fifth shift register
-- Input : ch2_shreg4
-- Output: ch2_shreg5
shiftregch2_5: shift_reg
    port map (
        D => ch2_shreg4,
        Q => ch2_shreg5,
        CLK => clk,
        CE=>rdy_fir1_ch2_2d);

-- shiftregch1_6 is the sixth shift register
-- Input : ch2_shreg5
-- Output: ch2_shreg6
shiftregch2_6: shift_reg
    port map (
        D => ch2_shreg5,
        Q => ch2_shreg6,
        CLK => clk,
        CE=>rdy_fir1_ch2_2d);

-- shiftregch1_7 is the seventh shift register
-- Input : ch2_shreg6
-- Output: ch2_shreg7
shiftregch2_7: shift_reg
    port map (
        D => ch2_shreg6,
        Q => ch2_shreg7,
        CLK => clk,
        CE=>rdy_fir1_ch2_2d);

-- shiftregch1_8 is the eighth shift register
-- Input : ch2_shreg7
-- Output: ch2_shreg8
shiftregch2_8: shift_reg
    port map (
        D => ch2_shreg7,
        Q => ch2_shreg8,
        CLK => clk,
        CE=>rdy_fir1_ch2_2d);

-- shiftregch1_9 is the ninnth shift register
-- Input : ch2_shreg8
-- Output: ch2_shreg9

```

```

shiftregch2_9: shift_reg
    port map (
        D => ch2_shreg8,
        Q => ch2_shreg9,
        CLK => clk,
        CE=>rdy_fir1_ch2_2d);

--now comparators are used for finding the maximum
--cmpch2_9 block compares 5th shift reg value with 9th shift reg value
cmpch2_9: comparator_gt
    port map (
        A => ch2_shreg5,
        B => ch2_shreg9,
        A_GT_B => max_found_ch2(7));

--cmpch2_8 block compares 5th shift reg value with 8th shift reg value
cmpch2_8:comparator_gt
    port map(
        A => ch2_shreg5,
        B => ch2_shreg8,
        A_GT_B => max_found_ch2(6));

--cmpch2_7 block compares 5th shift reg value with 7th shift reg value
cmpch2_7:comparator_gt
    port map (
        A => ch2_shreg5,
        B => ch2_shreg7,
        A_GT_B => max_found_ch2(5));

--cmpch2_6 block compares 5th shift reg value with 6th shift reg value
cmpch2_6:comparator_gt
    port map (
        A => ch2_shreg5,
        B => ch2_shreg6,
        A_GT_B => max_found_ch2(4));

--cmpch2_4 block compares 5th shift reg value with 4th shift reg value
cmpch2_4:comparator_gt
    port map (
        A => ch2_shreg5,
        B => ch2_shreg4,
        A_GT_B => max_found_ch2(3) );

--cmpch2_3 block compares 5th shift reg value with 3th shift reg value
cmpch2_3:comparator_gt

```

```

        port map (
            A => ch2_shreg5,
            B => ch2_shreg3,
            A_GT_B => max_found_ch2(2) );

--cmpch2_2 block compares 5th shift reg value with 2th shift reg value
cmpch2_2:comparator_gt
    port map (
        A => ch2_shreg5,
        B => ch2_shreg2,
        A_GT_B => max_found_ch2(1));

--cmpch2_1 block compares 5th shift reg value with 1th shift reg value
cmpch2_1:comparator_gt
    port map (
        A => ch2_shreg5,
        B => ch2_shreg1,
        A_GT_B => max_found_ch2(0));

--cmp1_9 block compares 5th shift reg value with some constant(1A0 Hex)
cmpch2_con:compare_con    --- MSB bit should be '1' and also some other bit in the
12 bits
    port map (
        A => ch2_shreg5,
        A_GT_B => max_found_ch2(8));

--"AND" the output of all the comparators and if output is '1' pulse is present
max_found2<=(max_found_ch2(8) and max_found_ch2(7)and max_found_ch2(6)and
max_found_ch2(5)and max_found_ch2(4)
    and max_found_ch2(3)and max_found_ch2(2)and max_found_ch2(1)and
max_found_ch2(0));

--max_found_temp is "OR" of max_found1 and max_found2
max_found_temp<=max_found1 or max_found2;

--maximum found is stored in a register...max_found is made high for '1' clock
cycle...even if there is a immediate max from
--other end of wedge output..it won't cause max_found to go high
process(clk)
begin
    if clk'event and clk='1' then
        max_found<=max_found_temp and not(disable_1_temp);
    end if;

```

end process;

--This is not required since or result of both the max_found1 and max_found2 is used to get max_found...

-- max_found can be used to start a counter which can disable the two max storing registers till division is completed

-----second channel next part -----

--ch2_counter:process(max_found,clk_2)

-- begin

-- if max_found='1' then

-- op_down_ch2 <= "11111"; -- should be load_data ?

-- elsif clk_2='1' and clk_2'event then

-- if disable_2='1' then

-- op_down_ch2 <= op_down_ch2-1;

-- end if;

-- end if;

--end process ch2_counter;

--disable_2<=(op_down_ch2(4) or op_down_ch2(3) or op_down_ch2(2) or op_down_ch2(1) or op_down_ch2(0)) ;

--process(clk_2)

-- begin

-- if clk_2'event and clk_2='1' then

-- if start_div='0' then

-- disable_2_1<=disable_2;

-- else

-- disable_2_1<='0';

-- end if;

-- end if;

--end process;

-- Register to hold the data for division process..loading is enabled by max_found_temp and it's kept clearing till

-- diable_1_1 is high

-- diable_1_1 is high till 5 bit counter counts to 0 or till division is going on

disable_2_disables:shift_reg13

port map (

D =>ch2_shreg5,

Q =>ch2_maxdata,

CLK =>clk,

```

        CE => max_found_temp,
        SCLR => disable_1_1);

--Now max_found is dlayed by '1' clock cycle each time and are used to enable some
adder or subtractor blocks
--max_add1 is a delayed max_found by '1' clock cycle
process(clk)
begin
    if clk'event and clk='1' then
        max_add1<=max_found;
    end if;
end process;

--group1: add_12bit
--      port map (
--          A => fir1_ch1_maxdata,
--          B => fir2_ch1_data_r,
--          Q => sum_group1,
--          CLK => clk,
--          CE => max_add1);

--group2: add_12bit
--      port map (
--          A => fir1_ch2_maxdata,
--          B => fir2_ch2_data_r,
--          Q => sum_group2,
--          CLK => clk,
--          CE => max_add1);

--maximum found from channel 1 and 2 are stored in A and B respectively using
max_add1 as enable signal
process(clk)
begin
    if clk'event and clk='1' then
        if max_add1='1' then
            A<= ch1_maxdata(12 downto 0);
        end if;
    end if;
end process;

process(clk)
begin

```

```

    if clk'event and clk='1' then
        if max_add1='1' then
            B<=ch2_maxdata(12 downto 0);
        end if;
    end if;
end process;

```

--max_found is dealed by 2 clock cycles which gives max_add2

```

process(clk)
begin
    if clk'event and clk='1' then
        max_add2<=max_add1;
    end if;
end process;

```

-- '0' is concatenated to A and 'B' for subtraction(19) purposes

```

A1_temp<='0' & A(12 downto 0);
B1_temp<='0' & B(12 downto 0);

```

--max_add2 is used as enable signal for subtracting "19" from A1_temp

```

a1_blk: sub_19
    port map (
        A => A1_temp,
        Q => A1_sub19,
        CLK => clk,
        CE => max_add2);

```

--All the bits of A1_sub19 "ORed" to check if it less than or equal to zero(in that case it is replaced by '1')..

```

A1_sub19_ored<=A1_sub19(11) or A1_sub19(10) or A1_sub19(9) or A1_sub19(8) or
A1_sub19(7) or A1_sub19(6) or A1_sub19(5) or A1_sub19(4) or A1_sub19(3) or
A1_sub19(2) or A1_sub19(1) or A1_sub19(0);

```

--max_add2 is used as enable signal for subtracting "19" from B1_temp

```

b1_blk : sub_19
    port map (
        A => B1_temp,
        Q => B1_sub19,
        CLK => clk,
        CE => max_add2);

```

--All the bits of B1_sub19 "ORed" to check if it less than or equal to zero(in that case it is replaced by '1')..

```

B1_sub19_ored<=B1_sub19(11) or B1_sub19(10) or B1_sub19(9) or B1_sub19(8) or
B1_sub19(7) or B1_sub19(6) or B1_sub19(5) or B1_sub19(4) or B1_sub19(3) or
B1_sub19(2) or B1_sub19(1) or B1_sub19(0);

```

```

process(clk)
begin
    if clk'event and clk='1' then
        if max_add3='1' then
            if A1_sub19(13)='1' or a1_sub19_ored='0' then
                A1<="00" & x"001";    --A1 is made '1' if A1_sub19 less than or equal to
zero
            else
                A1<=A1_sub19;        --otherwise A1=A1_sub19
            end if;
        end if;
    end if;
end process;

```

--max_found is dealed by 3 clock cycles which gives max_add3

```

process(clk)
begin
    if clk'event and clk='1' then
        max_add3<=max_add2;
    end if;
end process;

```

```

process(clk)
begin
    if clk'event and clk='1' then
        if max_add3='1' then
            if B1_sub19(13)='1' or B1_sub19_ored='0' then
                B1<="00" & x"001";    --A1 is made '1' if B1_sub19 less than or equal to zero
            else
                B1<=B1_sub19;        --otherwise B1=B1_sub19
            end if;
        end if;
    end if;
end process;

```

--max_found is dealed by 3 clock cycles which gives max_add4

```

process(clk)

```



```

begin
    if clk'event and clk='1' then
        max_add4<=max_add3;
    end if;
end process;

--A1 and B1 are added and the sum is used for division process
add_twofirs: add_14bit
    port map (
        A =>A1,
        B =>B1,
        Q => sum_final,
        CLK =>clk,
        CE =>max_add4);

----- division process -----

process(clk)
begin
    if clk'event and clk='1' then
        max_add5<=max_add4;
    end if;
end process;          -- redundant delay...gave a delay more..

--Divison enable signal is obtained by delaying max_found signal by '5' times
process(clk)
begin
    if clk'event and clk='1' then
        start_div<=max_add5; --division enable signal
        max_detect<=max_add5; --This is the output from this progrm which is used to
enable FIFO write
    end if;
end process;

A1_scaled<='0'& A1 & "0" & X"00";    --A1 is scaled by multiplying by 2000(hex) or
8192(dec)
sum_final_scaled<="0" & X"00" & sum_final;    --sum_final_scaled=sum_final

--Division process is carried out..output quo=A1_scaled/(A1+B1)

```

```

div1:divide_top
  Port map (
    clk => clk, --run it at 20MHz itself not 10MHz
    dividend => A1_scaled, --dividend
    divisor => sum_final_scaled, --divisor
    start => start_div, --division start signal
    delay_op => delay_op, --division output indicator
    quot => quo); -- division output

--output of division process quo is compared here(with some constants) and is used to
increment one of the ion counters
compare3:compare_3
  port map( ip_n1 => quo(11 downto 4), -- 11 to 4 bits are compared here
    n1gt => n1gt,
    n1lt => n1lt,
    n2gt => n2gt,
    n2lt => n2lt,
    n3gt => n3gt,
    n3lt => n3lt);

--Following 4 signals are enable signals to enable one of the four ion counters
cntr1<=n1lt and delay_op;
cntr2<=n1gt and n2lt and delay_op;
cntr3<=n2gt and n3lt and delay_op;
cntr4<=n3gt and delay_op;

----- 4 counters -----
--10 bit ion counter 1
c1:counter_10
  port map (
    Q => op_counter1,
    CLK => clk, --- same clock as the division block
    SCLR => reset_counters, --clears the counter 1 data
    CE => cntr1);

--10 bit ion counter 1
c2:counter_10
  port map (
    Q => op_counter2,
    CLK => clk, --- same clock as the division block
    SCLR => reset_counters, --clears the counter 1 data
    CE => cntr2);

--10 bit ion counter 1

```

```

c3:counter_10
    port map (
        Q =>op_counter3,
        CLK => clk, --- same clock as the division block
        SCLR => reset_counters, --clears the counter 1 data
        CE =>cntr3);

--10 bit ion counter 1
c4:counter_10
    port map (
        Q =>op_counter4,
        CLK =>clk, --- same clock as the division block
        SCLR => reset_counters, --clears the counter 1 data
        CE =>cntr4);

-----ADC saturation-----*

-----*

-----latching counter data-----+
--It is implemented in fifo_logic
-----+

-----anode number----- %
---anode number indicates from which wedge ( 1of 4) maximum was detected----
--implemented in fifo_logic program-----
----- %

-----FIFO section -----
--fifo signal =A register data + B register data + 10 bits master_count data + 2 bits ADC
saturation data + anode number data
fifo_din<=A(12 downto 0)& B(12 downto 0) & master_data(9 downto 0)& "0000";

--process(clk)
-- begin
--   if clk'event and clk='1' then
--     if max_add5='1' then
--       fifo_din<=A(12 downto 0)& B(12 downto 0) & master_data(9 downto 0) &
"0000";
--     end if;

```

```

--    end if;
--end process;
-----FIFO ends here-----

-----temp signals -----
--fir1_ch1_maxdata_t<=fir1_ch1_data;
--fir2_ch1_maxdata_t<=fir2_ch1_data;

--fir1_ch2_maxdata_t<=fir1_ch2_data;
--fir2_ch2_maxdata_t<=fir2_ch2_data;
--fir1_ch1_data_t<=fir1_ch1_data;
--fir2_ch1_data_t<=fir2_ch1_data;

--max_found1_t<=max_found1;
--max_found2_t<=max_found2;
--group1_sum_temp_t<=group1_sum_temp;
--group2_sum_temp_t<=group2_sum_temp;

--disable_1_1_t<=disable_1_1;
--ch1_maxdata_t<=ch1_maxdata;
--ch2_maxdata_t<=ch2_maxdata;

--ch2_shreg5_t<=ch2_shreg5;
--ch2_shreg1_t<=ch2_shreg1;

--q_shreg1_t<=q_shreg1;
--q_shreg2_t<=q_shreg2;
--q_shreg3_t<=q_shreg3;
--q_shreg4_t<=q_shreg4;
--q_shreg5_t<=q_shreg5;
--q_shreg6_t<=q_shreg6;
--q_shreg7_t<=q_shreg7;
--q_shreg8_t<=q_shreg8;
--q_shreg9_t<=q_shreg9;

--group1_sum_t<=group1_sum;
--max_found_t<=max_found;
--disable_1_temp_t<=disable_1_temp;
--op_down_ch1_t<=op_down_ch1;
--rdy_fir1_ch1_2d_t<=rdy_fir1_ch1_2d;
--sum_final_t<=sum_final;

--max_add1_t<=max_add1;

```

```

--max_add2_t<=max_add2;
--max_add3_t<=max_add3;
--max_add4_t<=max_add4;
--max_add5_t<=max_add5;

--A_t<=A;
--B_t<=B;
--A1_t<=A1;
--B1_t<=B1;
--A1_sub19_t<=A1_sub19;
--B1_sub19_t<=B1_sub19;
--quo_t<=quo;
--n1gt_t<=n1lt;
--cntr1_t<=cntr1;
--cntr2_t<=cntr2;
--cntr3_t<=cntr3;
--cntr4_t<=cntr4;
--delay_op_t<=delay_op;

--A1_temp_t<=A1_temp;
--B1_temp_t<=B1_temp;
--A1_scaled_t<=A1_scaled;
--sum_final_scaled_t<=sum_final_scaled;
--cntr1_t<=cntr1;
--delay_op_t<=delay_op;
--rdy_fir1_ch2_t<=rdy_fir1_ch2;
--rdy_fir1_ch1_t<=rdy_fir1_ch1;
--rdy_fir1_ch2_d_t<=rdy_fir1_ch2_d;
--rdy_fir1_ch2_2d_t<=rdy_fir1_ch2_2d;
--cntr1_t<=cntr1;
--delay_op_t<=delay_op;
end Behavioral;

```

```

-- This is the main program..
-- It integrates all the modules required

-- Inputs are
--  fpga_gclk: is 20MHz input clock
--  gated_clock_ext: is input clock from PC interface board..also called data read clock
--  minor_frame_sync: is the clock to the reconfiguration counters
--  set1_datach1: 8 bit ADC input from one of the ADC..representing one end output of
anode board (first anode board)
--  set1_datach2: corresponding output from the other end of the anode board output
(first anode board)
--  set2_datach1: 8 bit ADC input from one of the ADC..representing one end output of
anode board (second anode board)
--  set2_datach2: corresponding output from the other end of the anode board
output(second anode board)
--  master_data: 10 bit reconfiguration counter value
--  max1231_sdout: MAX1231 ADC input data
--  max1231_eoc_bar: MAX1231 ADC "eoc" signal
--  chip_select_pin: chip_select_pin='1' or '0' is input to FPGA

-- Outputs from the FPGA are
--  adc_clk: 20MHz output clock to ADCs
--  otherboard_sdout: Results data line (1 bit)
--  max1231_sclk: Max231 ADC output clock
--  max1231_cs_bar: MAX1231 ADC
--  max1231_sdin: ADC results data output line
--  dac_cs_bar: AD5544 cs_bar input of

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

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.

library UNISIM;
use UNISIM.VComponents.all;

```

--Signals inside the entity listed represent the FPGA I/O signals

entity main_module_top is

```

Port ( fpga_gclk : in std_logic;
      gated_clock_ext:in std_logic;
      minor_frame_sync:in std_logic;
      adc_clk:out std_logic; -- clock for adc08l060
      set1_datach1: in std_logic_vector(7 downto 0);
      set1_datach2: in std_logic_vector(7 downto 0);
      set2_datach1 : in std_logic_vector(7 downto 0);
      set2_datach2 : in std_logic_vector(7 downto 0);
      master_data:in std_logic_vector(9 downto 0);
      otherboard_sdout:out std_logic;
      max1231_sclk:out std_logic;
      max1231_sdin:in std_logic;
      max1231_sdout:out std_logic;
      max1231_cs_bar:out std_logic;
      max1231_eoc_bar:in std_logic;
      chip_select_pin:in std_logic;
      dac_cs_bar:out std_logic;
      dac_sdout:out std_logic;
      dac_ldac_b:out std_logic;
      rd_enable_fifo_t,rd_enable_fifo_t2:out std_logic;
      test_sdout:out std_logic_vector(1 downto 0);
      dac_sclk_out:out std_logic);
end main_module_top;
```

architecture Behavioral of main_module_top is

--z2_chs_next component is defined here and it initializes the fitting blocks...

--Each instantiation of this component processes two pulses coming from two outputs coming out of wedge pattern

--This block is instantiated twice in this file...

component z2_chs_next

```

Port ( clk : in std_logic;
      --clk_2:in std_logic;
      clk_2x:in std_logic;
      max_detect:out std_logic;
      master_data:in std_logic_vector(9 downto 0);
      data_ch1 : in std_logic_vector(7 downto 0);
      data_ch2 : in std_logic_vector(7 downto 0);
      op_counter1,op_counter2,op_counter3,op_counter4:out std_logic_vector(9
downto 0);
      reset_counters: in std_logic;
```

```

        fifo_din:out std_logic_vector(39 downto 0));
end component;

```

--This component instantiates DCM and also generates 40MHz clock

```

component clk_z2_chs
  Port ( fpga_gclk : in std_logic;
        locked_t:out std_logic;
        clk_2x:out std_logic;  -- clock *2 times
        clk:out std_logic      -- clock same as input
        --clk_2:out std_logic    -- clock divided by 2
        -- clk_4:out std_logic   -- clock divided by 4
        );
end component;

```

--This component is the clock buffer

```

component BUFG
port (   O : out STD_ULOGIC;
        I : in STD_ULOGIC);
end component;

```

--This component initializes FIFO block,MAX1231 ADc,Control rom etc

```

component fifo_logic
  Port (
    clk:in std_logic;
    GATED_CLOcK:in std_logic;
    minor_frame_sync:in std_logic;
    otherboard_sdout:out std_logic;
    max_detect_1:in std_logic;
    max_detect_2:in std_logic;
    fifo_din_ch1:in std_logic_vector(39 downto 0);
    fifo_din_ch2:in std_logic_vector(39 downto 0);
    major_frame:inout std_logic;
    -- major_frame_d:inout std_logic;
    frame_counter:out std_logic_vector(9 downto 0);
    reset_counters1: out std_logic;
    reset_counters2: out std_logic;
    --master_data:in std_logic_vector(9 downto 0);
    --max1231_dout:in std_logic_vector(11 downto 0);
    max1231_sclk:out std_logic;
    max1231_sdin:in std_logic;
    max1231_sdout:out std_logic;
    max1231_cs_bar:out std_logic;
    max1231_eoc_bar:in std_logic;
    rd_enable_fifo_t,rd_enable_fifo_t2:out std_logic;

```



```

    test_sdout:out std_logic_vector(1 downto 0);
    --ainit:in std_logic;
    -- control_rom_op_max1231:out std_logic;
    --qspo_ce:out std_logic;
    chip_select_pin:in std_logic;
    op_counter1_ch1,op_counter2_ch1,op_counter3_ch1,op_counter4_ch1:in
std_logic_vector(9 downto 0);
    op_counter1_ch2,op_counter2_ch2,op_counter3_ch2,op_counter4_ch2:in
std_logic_vector(9 downto 0)
    );
end component;

```

--DAC component is used to send data to AD5544 DAC

component dac_top

```

    Port (   clk : in std_logic;
            cs_bar : out std_logic;
            sdout : out std_logic;
            sclk_out : out std_logic;
            ldac_b:out std_logic;
            major_frame:in std_logic;
            -- major_frame_d:in std_logic;
            frame_counter:in std_logic_vector(6 downto 0);  --only 7 bits are needed
address the 128 bytes stored in DAC RAM
            chip_select_pin:in std_logic
    );
end component;

```

--temporary signals are declared here (can be initialized also)

```

signal clk,locked,clk_2: std_logic:='0';

```

```

signal

```

```

op_counter1_ch1,op_counter2_ch1,op_counter3_ch1,op_counter4_ch1:std_logic_vector(
9 downto 0):="0000000000";

```

```

signal

```

```

op_counter1_ch2,op_counter2_ch2,op_counter3_ch2,op_counter4_ch2:std_logic_vector(
9 downto 0):="0000000000";

```

```

signal fifo_din_ch1,fifo_din_ch2:std_logic_vector(39 downto 0):=x"0000000000";

```

```

signal max_detect_1:std_logic:='0';

```

```

signal max_detect_2:std_logic:='0';

```

```

--signal rd_enable_fifo_t:std_logic;

```

```

signal major_frame:std_logic:='0';

```

```

signal frame_counter:std_logic_vector(9 downto 0):="0000000000";
signal dac_sclk,clk_2x:std_logic:='0';
signal gated_clock:std_logic:='0';
signal reset_counters1,reset_counters2: std_logic;

```

```

signal ila_control: std_logic_vector(35 downto 0);

```

```

--Some of the signals which are generated in the code and are used to send/get data to
ADC,DAC etc are

```

```

--designated as clocks

```

```

attribute clock_signal : string;

```

```

attribute clock_signal of clk : signal is "yes";

```

```

attribute clock_signal of clk_2x : signal is "yes"; --40MHz clock to FIR filters

```

```

attribute clock_signal of max1231_sclk : signal is "yes"; --MAX1231 clock

```

```

attribute clock_signal of dac_sclk : signal is "yes"; --DAC clock

```

```

attribute clock_signal of adc_clk : signal is "yes"; --AD08L060 clock

```

```

begin

```

```

-- clk_z2_chs component is initialized here

```

```

-- This component sets up the DCM...locked signal goes high whenever clk (DCM
output) and fpga_clk are locked

```

```

-- Whenever this locking is lost locked is made '0' by DCM

```

```

CLK_DCM:clk_z2_chs

```

```

    Port map(fpga_gclk=>fpga_gclk, -- Clock input to FPGA

```

```

        locked_t =>LOCKED,    -- Locked

```

```

        clk_2x => clk_2x,    --40MHz clock output

```

```

        clk =>clk            --20MHz clock output

```

```

        --clk_2=>clk_2

```

```

        -- clk_4=>clk_4

```

```

    );

```

```

-- clock to ADC AD08L060

```

```

adc_clk<=clk;

```

```

-----

```

--z2_chs_next component is initialized here
 --set1_datach1 and set1_datach2: inputs to FPGA(ADC outputs)
 --Four counters op_counter1_ch1,op_counter2_ch1,op_counter3_ch1,op_counter4_ch1
 are used to count ions for this compoenet (z1)
 --fifo_din_ch1: is a 40bit input to FIFO block
 --max_detect_1: indicates maximum detected...from z1
 --reset_counters1: signal is used to the four ion counters of this component (z1)
 z1:z2_chs_next

```
port map(
  clk =>clk,
  --clk_2 => clk_2,
  clk_2x=> clk_2x,    --40MHz clock
  max_detect => max_detect_1,
  data_ch1 =>set1_datach1,
  data_ch2 =>set1_datach2,
  fifo_din =>fifo_din_ch1,
  op_counter1 => op_counter1_ch1,
  op_counter2 => op_counter2_ch1,
  op_counter3 => op_counter3_ch1,
  op_counter4 => op_counter4_ch1,
  reset_counters =>reset_counters1,
  master_data => master_data(9 downto 0));
```

--z2_chs_next component is initialized again (porcesses pulses coming from the other
 wedge pattern)
 --set2_datach1 and set2_datach2: inputs to FPGA(ADC outputs)
 --Four counters op_counter1_ch2,op_counter2_ch2,op_counter3_ch2,op_counter4_ch2
 are used to count ions for this compoenet (z2)
 --fifo_din_ch2: is a 40bit input to FIFO block
 --max_detect_2: indicates maximum detected...from z2
 --reset_counters2: signal is used to the four ion counters of this component (z2)
 z2:z2_chs_next

```
port map(
  clk =>clk,
  --clk_2 => clk_2,
  clk_2x=> clk_2x, --40MHz clock
  max_detect => max_detect_2,
  data_ch1 =>set2_datach1,
  data_ch2 =>set2_datach2,
  fifo_din =>fifo_din_ch2,
  op_counter1 => op_counter1_ch2,
  op_counter2 => op_counter2_ch2,
  op_counter3 => op_counter3_ch2,
  op_counter4 => op_counter4_ch2,
```

```

        reset_counters => reset_counters2,
        master_data => master_data(9 downto 0)); --master_data => master_data(9
downto 0));

--The gated clock gated_clock_ext is given to clock buffer
gated_clock_buffer: BUFG
port map( O => gated_clock,
         I => gated_clock_ext);

--fifo_logic block is initialized here
--gated_clock is used to send results to main board
--minor_frame_sync: is a frame sync signal used to synchronise data transfer between the
FPGAs and also it is a clock for reconfiguration counters
--max_detect1_1 and max_detect_2 come from z1 and z2 blocks respectively
--fifo_din_ch1 and fifo_din_ch2 are inputs to fifo block and come from z1 and z2 blocks
respectively
--major_frame is output signal from fifo block
--frame_counter is 10bit counter data used to synchronise the data between the FPGAs
--reset_counters1:resets ion counters from z1 block
--reset_counters3:resets ion counters from z2 block
--op_counter1_ch1,op_counter2_ch1,op_counter3_ch1,op_counter4_ch1:10 bit ion
counters
--op_counter1_ch2,op_counter2_ch2,op_counter3_ch2,op_counter4_ch2:10 bit ion
counters
--max1231_sclk: clock to MAX1231 ADC
--max1231_sdin: data to MAX1231 ADC
--max1231_sdout: output from MAX1231 ADC
--max1231_cs_bar: chip select signal to MAX1231 ADC
--max1231_eoc_bar: end of conversion from MAX1231 ADC
--chip_select_pin:either '1' or '0' input ,used to synchronize data transfer between FPGAs
fifo: fifo_logic
  Port map(
    clk => clk,
    GATED_CLOcK=>gated_clock,
    minor_frame_sync =>minor_frame_sync,
    otherboard_sdout=> otherboard_sdout,
    max_detect_1=>max_detect_1,
    max_detect_2=>max_detect_2,
    fifo_din_ch1 => fifo_din_ch1,
    fifo_din_ch2 => fifo_din_ch2,
    major_frame => major_frame,
    -- major_frame_d => major_frame_d,
    frame_counter => frame_counter,

```

```

reset_counters1 => reset_counters1,
reset_counters2 => reset_counters2,
--master_data => master_data,
op_counter1_ch1 => op_counter1_ch1,
op_counter2_ch1 => op_counter2_ch1,
op_counter3_ch1 => op_counter3_ch1,
op_counter4_ch1 => op_counter4_ch1,
op_counter1_ch2 => op_counter1_ch2,
op_counter2_ch2 => op_counter2_ch2,
op_counter3_ch2 => op_counter3_ch2,
op_counter4_ch2 => op_counter4_ch2,
rd_enable_fifo_t => rd_enable_fifo_t,
rd_enable_fifo_t2 => rd_enable_fifo_t2,
test_sdout(1 downto 0) => test_sdout(1 downto 0),
--max1231_dout => max1231_dout,
max1231_sclk => max1231_sclk,
max1231_sdin => max1231_sdin,
max1231_sdout => max1231_sdout,
max1231_cs_bar => max1231_cs_bar,
max1231_eoc_bar => max1231_eoc_bar,
--ainit => '0',
--qspo_ce => qspo_ce,
chip_select_pin => chip_select_pin
);

```

---The minimum width should be 25ns high and 25ns low...maximum frequency is 20MHz.....

--clock to DAC

dac_sclk_out<=dac_sclk;

--dac_interface component is used to instantiate the DAC block

--clock and data is transferred from FPGA..

--clk:20MHz clock input

--dac_cs_bar:dac select signal to DAC

--dac_sdout:data output to DAC

--dac_ldac_b:signal to DAC indicates to load four of its input registers when this signal is low

--major_frame: is input from fifo block

--frame_counter: is used to select a particular range of DAC ROM memory

--chip_select_pin: is used to select a particular range of DAC ROM memory

dac_interface: dac_top

Port map(

clk => clk,

cs_bar => dac_cs_bar,

```

        sdout => dac_sdout,
        sclk_out => dac_sclk,
        ldac_b => dac_ldac_b,
        major_frame => major_frame,
        -- major_frame_d => major_frame_d,
        frame_counter => frame_counter(6 downto 0),
        chip_select_pin => chip_select_pin
    );
end Behavioral;

```

```

-- This file is compares the divider output and enables one of the 6 outputs which are
used to increment one of the four counters.

```

```

-- Inputs

```

```

--     ip_n1:Eight bit division logic result

```

```

-- Outputs (all single bits)

```

```

--     n1lt:

```

```

--     n1gt:

```

```

--     n2lt:

```

```

--     n2gt:

```

```

--     n3lt:

```

```

--     n3gt:

```

```

-- Components instantiated from the core generator are

```

```

--

```

```

compare_n1gt_con2,compare_n1lt_con2,compare_n2gt_con2,compare_n2lt_con2,compa
re_n3gt_con2,compare_n3lt_con2

```

```

library IEEE;

```

```

use IEEE.STD_LOGIC_1164.ALL;

```

```

use IEEE.STD_LOGIC_ARITH.ALL;

```

```

use IEEE.STD_LOGIC_UNSIGNED.ALL;

```

```

-- Entity (compare_3) shows the inputs and the outputs listed above

```

```

-- Its purely combinational logic

```

```

entity compare_3 is

```

```

    Port ( ip_n1 : in std_logic_vector(7 downto 0);

```

```

        n1gt : out std_logic;

```

```

        n1lt : out std_logic;

```

```

        n2gt : out std_logic;

```

```

        n2lt : out std_logic;

```

```

        n3gt : out std_logic;

```

```

        n3lt : out std_logic);

```

```

end compare_3;

```

--Functional part of compare_3 entity starts from here
 architecture Behavioral of compare_3 is

--The below component (compare_n1lt_con2) compares if A is less than constant(51 Hex)

--based on the result a_gt_b signal will be '1' or '0'
 component compare_n1gt_con2

```

    port (
      A: IN std_logic_VECTOR(7 downto 0);
      A_GT_B: OUT std_logic);
  end component;
```

--The below component (compare_n1lt_con2) compares if A is less than constant(51 Hex)

--based on the result a_lt_b signal will be '1' or '0'
 component compare_n1lt_con2

```

    port (
      A: IN std_logic_VECTOR(7 downto 0);
      A_LT_B: OUT std_logic);
  end component;
```

--The below component (compare_n2gt_con2) compares if A is greater than constant(82 Hex)

--based on the result a_gt_b signal will be '1' or '0'
 component compare_n2gt_con2

```

    port (
      A: IN std_logic_VECTOR(7 downto 0);
      A_GT_B: OUT std_logic);
  end component;
```

--The below component (compare_n2lt_con2) compares if A is less than constant(82 Hex)

--based on the result a_lt_b signal will be '1' or '0'
 component compare_n2lt_con2

```

    port (
      A: IN std_logic_VECTOR(7 downto 0);
      A_LT_B: OUT std_logic);
  end component;
```

--The below component (compare_n3gt_con2) compares if A is greater than constant(B5 Hex)

--based on the result a_gt_b signal will be '1' or '0'
 component compare_n3gt_con2

```

    port (
      A: IN std_logic_VECTOR(7 downto 0);
      A_GT_B: OUT std_logic);
end component;

```

--The below component (compare_n3lt_con2) compares if A is greater than constant(B5 Hex)

--based on the result a_lt_b signal will be '1' or '0'

component compare_n3lt_con2

```

    port (
      A: IN std_logic_VECTOR(7 downto 0);
      A_LT_B: OUT std_logic);
end component;

```

begin

--"compare_n1gt_con2" component is instantiated here

--result variable n1gt is '1' if ip_n1 is greater than 51 Hex

n1is_gt: compare_n1gt_con2

```

    port map (
      A => ip_n1,
      A_GT_B => n1gt);

```

--"compare_n1lt_con2" component is instantiated here

--result variable n1lt is '1' if ip_n1 is less than 51 Hex

n1is_lt: compare_n1lt_con2

```

    port map (
      A => ip_n1,
      A_LT_B => n1lt);

```

--"compare_n2gt_con2" component is instantiated here

--result variable n2gt is '1' if ip_n1 is greater than 82 Hex

n2IS_GT: compare_n2gt_con2

```

    port map (
      A => ip_n1,
      A_GT_B => n2gt);

```

--"compare_n2lt_con2" component is instantiated here

--result variable n2lt is '1' if ip_n1 is less than 82 Hex

n2IS_LT: compare_n2lt_con2

```

    port map (

```



```

        A =>ip_n1,
        A_LT_B => n2lt);

--"compare_n3gt_con2" component is instantiated here
--result variable n3gt is '1' if ip_n1 is greater than B5 Hex
n3IS_GT: compare_n3gt_con2
    port map (
        A =>ip_n1,
        A_GT_B =>n3gt);

--"compare_n3lt_con2" component is instantiated here
--result variable n3lt is '1' if ip_n1 is less than B5 Hex
n3IS_LT: compare_n3lt_con2
    port map (
        A =>ip_n1,
        A_LT_B =>n3lt);

end Behavioral;

```

```

--This program implements the division process
--Output is sued to identify one of the four outputs

--inputs

```

```

--clk:20MHz clock
--dividend: 24 bit dividend
--divisor:24 bit divisor
--start: '1' clock high signal indicates to start the division process
--Outputs
--delay_op:output
--quot: 24 bit output

```

```

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

```

```

-- The algorithm folder name is "division algorithm" it's in c:\division algos folder
-- in this I have used "lab6" acrobat file ..variable names are almost same as in that file

```

```

library UNISIM;
use UNISIM.VComponents.all;

```

```

-- It implements the division process (ratio block)
-- The dividend and divisor are 24 bits
-- If the number of bits are changed then add_sub component has to be regenerated using
core generator to suit the number of bits
-- count_op variable is the number of cycles required to get the result..it should be n+2
because some clock cycles
-- are needed for initialisation etc..
-- Final output is in quot variable..

```

```

entity divide_top is
  Port ( clk : in std_logic;
        dividend :in std_logic_vector(23 downto 0);
        divisor : in std_logic_vector(23 downto 0);
        start : in std_logic;
        DELAY_OP:OUT STD_LOGIC;
        --count_op_t:out std_logic_vector(5 downto 0);
        --q_reg_t:out std_logic_vector(23 downto 0) ;
        quot: out std_logic_vector(23 downto 0));
end divide_top;

```

```

architecture Behavioral of divide_top is

```

```

-- This component does addition or subtraction based on the input c_in..x-or is used to do
add or sub
component add_sub

```

```

    port (
      A: IN std_logic_VECTOR(23 downto 0);
      B: IN std_logic_VECTOR(23 downto 0);
      C_IN: IN std_logic;
      C_OUT: OUT std_logic;
      S: OUT std_logic_VECTOR(23 downto 0));
end component;

-- signals are initialized here
signal dec,cout :std_logic:='1';
signal count_on,t,done_div,done:std_logic:='0';
signal count_op:std_logic_vector(5 downto 0):="000000";
signal x_div,divi_data,temp,q_reg,acc_reg:std_logic_vector(23 downto 0):= X"000000";
begin

-- count_on counts the number of cycles..when it reaches zero output will be present in
quot
count_on<=count_op(5) or count_op(4) or count_op(3) or count_op(2) or count_op(1) or
count_op(0);

-- a single cycle high variable indicating that division is done..
done_div<=not (count_op(5) or count_op(4) or count_op(3) or count_op(2) or
count_op(1) or (not(count_op(0)))) ;

----- down counter to count the number of cycles ..when it reaches
'1'.output division will be done....counts till zero only
process (clk, start)
begin
  if start='1' then
    count_op<="011010";      --count should be total number of bits +2 (i.e 26 for
above example 24+2)
  elsif clk='1' and clk'event then
    if count_on='1' then
      count_op<= count_op- 1;  --decreasing the count each clock cycle
    end if;
  end if;
end process;

-- dividend data is stored in se_in variable.at each iteration t is assigned to LSB of se_in
-- t is carry out of add or subtract block..initially t is zero..
-- se_in is assigned to q_reg..
-- q_reg is a signal and se_in is a variable which can be used only within shift_reg1
block..

```

-- variables are used because they are faster compared to signals

```

shift_reg1:process(clk)
variable se_in:std_logic_vector(23 downto 0):=X"000000";
begin
    if clk'event and clk='1' then
        if start='1' then
            se_in:=dividend;
        else
            if count_on='1' then
                se_in:=se_in(22 downto 0) & t;
            end if;
        end if;
    end if;
    q_reg<=se_in;
end process shift_reg1;

```

-- initially divisor is assigned to divi_data

```

shift_reg12:process(CLK)
begin
    if clk'event and clk='1' then
        if start='1' then
            divi_data<=divisor;
        end if;
    end if;
end process shift_reg12;

```

-- acc_buf is a variable which is reset at the starting and at each iteration the MSB of q_reg value is

-- shifted to acc_buf LSB

-- acc_buf variable is assigned to signal acc_reg

```

shift_regs:process(clk)
variable acc_buf:std_logic_vector(23 downto 0):=X"000000";
begin
    if clk'event and clk='1' then
        if start='1' then
            acc_buf:=X"000000"; --at the start acc_buf variable is cleared
        else
            if count_on='1' then
                acc_buf:=temp(22 downto 0) & Q_reg(23); --acc_buf=add or sub block
                output + 1 bit of q_reg MSB
            end if;
        end if;
    end if;
end process;

```

```

    acc_reg<=acc_buf; --acc_buf variable is assigned to acc_reg
end process shift_regs;

```

```

-- This is the add or sub block..does only addition...but according to previous dec value
the output may be a addition or subtraction

```

```

-- If input x_div is inverted ..it results in subtraction

```

```

-- sum is stored in signal temp

```

```

    addi : add_sub
        port map (
            A => acc_reg, --input
            B => x_div,    --input
            C_IN => dec, -- input carry
            C_OUT => cout, --output carry
            S => temp); --output

```

```

-- t is reset at the beginning and cout(add_sub block carry out) is assigned

```

```

-- at each iteration

```

```

-- dec is set at the start thus subtracting for the first time

```

```

-- at each iteration cout is assigned to dec ..if cout is '1' then dec is '1'

```

```

control_data:process(clk)

```

```

begin

```

```

    if clk'event and clk='1' then

```

```

        if start='1' then

```

```

            dec<='1'; --at the start dec='1' initialized to '1' so that x_div is inverted

```

```

            t<='0';

```

```

        else

```

```

            if count_on='1' then

```

```

                t<=cout; -- output carry of add_sub block is assigned to 't' when count_on ='1'

```

```

                dec<=cout;

```

```

            end if;

```

```

        end if;

```

```

    end if;

```

```

end process control_data;

```

```

-- If dec is '1' then divisor data are complemented and if dec is '0' then divisor data are
not changed

```

```

process(divi_data, dec)

```

```

begin

```

```

    if dec='1' then

```

```

        x_div(23 downto 0)<=divi_data(23 downto 0) xor "111111111111111111111111";

```

```

    elsif dec='0' then

```

```

        x_div(23 downto 0)<=divi_data(23 downto 0) xor "000000000000000000000000";

```

```

end if;
end process;

-- if the count goes to '1' then done is made high..
-- in the next cycle the output will be available at 'quot' signal
process(clk)
begin
    if clk'event and clk='1' then
        done<= done_div;
    end if;
end process;

-- if done is '1' then quot will have the result of the division process
-- else quot will be reset all the times
process(clk)
begin
    if clk'event and clk='1' then
        if done='1' then
            quot<=q_reg;    --at this clock the comparators compare the quot with four
constants
        else                --before the next clock comes comparator output will be ready
            quot<= X"000000";
        end if;
    end if;
end process;

process(clk)
begin
    if clk'event and clk='1' then
        delay_op<= done;    --A delayed output indicates the completion of division
    end if;                --delay_op is used as an enable signal for all the four ion counters
end process;

--count_op_t<=count_op;
--q_reg_t<=q_reg;
end Behavioral;

```

APPENDIX B: MATLAB PROGRAMS

%Program to detect pulse presence using both the FIR filter outputs

```

% Algorithm checked for three cases of noise..less than equal to and more
% than amplitude of pulse ..by changing line 6..(i.e.e changing number '7'
% in line 6 as per the needs)

% should determine the maximum value of noise that can be added without
% affecting the working of the algorithm

clear all; % clears all data in the matlab

close all; % closes all plots

clc; % clears all command window(main matlab window) data

n(1:16)=(1:16); % initialise n with 16 numbers from 1 to 16

f1(1:16)=((1-exp(-(n(1:16)-1)*0.5)).*exp(-(n(1:16)-1)*1/6)); % fit curve f1 equation

f2(1:16)=zeros; % fit curve f2 is initialised with all zeros

f2(2:16)=((1-2.7183.^(-(2:16)-2)*0.5)).*2.7183.^(-(2:16)-2)*1/6)); % fit curve f2

implementation

f1_sqr(1:length(f1))=f1.*f1; %f1_square gives the square of each number of f1 fit curve
(for a(n) and b(n) calculation)

f1_sum=sum(f1_sqr); %f1_sum gives the sum of all elements of f1_sqr (for a(n)
and b(n) calculation)

```

```
f2_sqr(1:length(f2))=f2.*f2; %similarly fit curve f2 square of individual numbers (for
a(n) and b(n) calculation)
```

```
f2_sum=sum(f2_sqr); %similarly sum of all the elements of f2_sqr (for a(n) and
b(n) calculation)
```

```
f1f2_sqr(1:length(f1))=f1.*f2; %element by elemnt multiplication of f1 and f2 fit curve
values (for a(n) and b(n) calculation)
```

```
f1f2_sum=sum(f1f2_sqr); %similarly sum of all elements of f1f2_sqr
```

```
f1f2_sqrdsum=f1f2_sum^2;
```

```
a(1:16)=((f1(1:16)*f2_sum-f2(1:16)*f1f2_sum)/(f1_sum*f2_sum-f1f2_sqrdsum));
```

```
%First FIR filter coefficients are calculated
```

```
a(1:16)=fix(a(1:16)*100); %a(n)is converted to integers after multiplying with 100
```

```
%Displays the a(n) plot
```

```
% figure,plot(n*50,a,'b');
```

```
% title('it{First fit curve a(n)}','FontSize',16);
```

```
% xlabel('n values');
```

```
% ylabel('a values');
```

```
%clear f1f2_sqr,f1f2_sqrdsum,f1f2_sum,f1_sqr,f2_sqr,f1_sum,f2_sum; This statement
not required..cleared some variables to free RAM
```



```

b(1:16)=((f2(1:16)*f1_sum-f1(1:16)*f1f2_sum)/(f1_sum*f2_sum-f1f2_sqrdsum));

%Second FIR filter coefficients are calculated

b(1:16)=fix(b(1:16)*100);    %b(n) is converted to integers after multiplying with 100


%Displays the b(n) plot

% figure,plot(n*50,b,'b');

% title('\it{Second fit curve b(n)}','FontSize',16);

% xlabel('n values');

% ylabel('b values');


% A pulse with 20ns delayed with respect to first fit curve was used

delay=20; %delay was set to 20ns

f1_d1(1:16)=0; %the variable to hold the 20ns delayed pulse was cleared


% Following for loop generates the 20ns delayed pulse

for i=2:16

    f1_d1(i)=(1-exp(-(((i-1)*50)-delay)/100)).*exp(-(((i-1)*50)-delay)*1/300);

end

```

```
f1_d1(:)=fix(f1_d1(:).*100); %the resulting pulse was also converted to integer after
multiplying with 100
```

```
% First fit curve and second fit curve are converted to integers after
```

```
% multiplying with 100
```

```
f1(1:16)=fix(f1(1:16).*100);
```

```
f2(1:16)=fix(f2(1:16).*100);
```

```
%%Following generates the F1 and F2 plots
```

```
% Figure,plot(n*50,f1,'b');
```

```
% %title('\it{first waveform f1(n)}','FontSize',16);
```

```
% xlabel('n values');
```

```
% ylabel('f1 values');
```

```
% hold on,plot(n*50,f2,'r');
```

```
% % title('\it{Second waveform f2(n)}','FontSize',16);
```

```
% % xlabel('n values');
```

```
% % ylabel('f2 values');
```

```
% Legend('First fit curve','Second fit curve');
```

%Size of noise signal is set by setting the size for t

```
t = (0:0.0016:32);
```

%noise signal yn is initialised with zeros first

```
yn = ((1*randn(size(t))))/7; % 7 should be replaced by small value if large noise levels are  
needed and vice versa
```

```
yn(:)=fix(yn(:)*100 ); % noise is converted to integer after multiplying with 100
```

% noise signal is plotted here

```
figure,plot((9968:10042)*1,yn(9968:10042));
```

```
%title('it{ noise waveform yn}','FontSize',16);
```

```
xlabel('Time in nano seconds');
```

```
ylabel('Random noise value');
```

%d variable is used to store the noise value first

```
d(1:length(yn))=yn(:);
```

% pulse (first fit curve) is added to d here at 10000

```
d(10000:10015)= d(10000:10015)+f1;
```

%A part of d is plotted here

```
figure,plot((9968:10042)*1*1,d(9968:10042)*1);
```

```
ylabel('Amplitude');
```

```
xlabel('Time in nano seconds');
```

```
%d1 is initialized to zero first
```

```
d1(1:length(d)+15)=zeros;
```

```
% dis assigned to d1 here
```

```
d1(1:length(d))=d(1:length(d));
```

```
for i=1:length(d) % fitting operation is carried for entire length of noise signal with pulse
```

```
d1
```

```
    c1(i)=fix(sum(a(1:16).*d1(i:i+15))); % First fit curve operations
```

```
    c2(i)=fix(sum(b(1:16).*d1(i:i+15))); % Second fit curve operations
```

```
end
```

```
%%C1 is plotted below
```

```
figure,plot((9968:10042)*1,c1(9968:10042));
```

```
%title('it{fit operation with a(n) results c1}','FontSize',16);
```

```
xlabel('Time in nano seconds');
```

```
ylabel('Best fit parameter C1');
```

```
%%C2 is plotted below
```

```
figure,plot((9968:10042)*1,c2(9968:10042));
```

```
%title("\it{fit operation with b(n) results c2}','FontSize',16);
```

```
xlabel('Time in nano seconds');
```

```
ylabel('Best fit parameter C2');
```

```
% fitted pulse height C is obtained by adding C1 and C2
```

```
c(1:length(c1))=(c1(1:length(c1))+c2(1:length(c1)));
```

```
% logic to determine the presence of maximum
```

```
count=0; % count counts the number of maximums found
```

```
for i=10:length(c)-10
```

```
    if ((c(i)>c(i-1)) & (c(i)>c(i-2)) & (c(i)>c(i-3)) & (c(i)>c(i-4)) & (c(i)>c(i+1)) &  
    (c(i)>c(i+2)) & (c(i)>c(i+3)) & (c(i)>c(i+4)) & c(i)>8000) % COMPARE WITH
```

```
    NEIGHBOURING 4 AND A CONSTANT WHICH NOW IS '1'
```

```
        count=count+1;
```

```
        max_c_loc(count)=i; % maximum found is stored in max_c_loc array
```

```
    end
```

```
end
```

```
%c is scaled and value is stored in c_temp
```

```

c_temp((1:length(c)))=zeros;

c_temp(:)=((c(:).*47)/9922); % 9922 is the C value for a ideal pulse without any noise

c_temp(:)=round(c_temp(:));

```

```

%fitted height C is plotted here

```

```

figure,plot((9968:10042)*1,c(9968:10042));

%title('\it{Fit operation result: fitted pulse height}','FontSize',16);

xlabel('Time delay between pulse in data and first fit curve');

ylabel('Fitted amplitude C1+C2');

```

```

%scaled height c_temp is plotted

```

```

% figure,plot((1:length(c_temp))*1,c);

% %title('\it{Fit operation result: fitted pulse height}','FontSize',16);

% xlabel('Sample points');

% ylabel('Amplitude');

```

```

%Program to get Best-Fit result

```

```

% Best-fit results

clear all;

clear all; % clears all data in the matlab

close all; % closes all plots

clc;    % clears all command window(main matlab window) data

n(1:16)=(1:16); % initialise n with 16 numbers from 1 to 16


f1(1:16)=((1-exp(-(n(1:16)-1)*0.5)).*exp(-(n(1:16)-1)*1/6)); % fit curve f1 equation


f2(1:16)=zeros;    % fit curve f2 is initialised with all zeros

f2(2:16)=((1-2.7183.^(-(2:16)-2)*0.5)).*2.7183.^(-(2:16)-2)*1/6)); % fit curve f2

implementation


f1_sqr(1:length(f1))=f1.*f1;    %f1_square gives the square of each number of f1 fit curve
(for a(n) and b(n) calculation)

f1_sum=sum(f1_sqr);            %f1_sum gives the sum of all elements of f1_sqr (for a(n)
and b(n) calculation)


f2_sqr(1:length(f2))=f2.*f2;    %similarly fit curve f2 square of individual numbers (for
a(n) and b(n) calculation)

```

```
f2_sum=sum(f2_sqr);          %similarly sum of all the elements of f2_sqr (for a(n) and
b(n) calculation)
```

```
f1f2_sqr(1:length(f1))=f1.*f2; %element by elemnt multiplication of f1 and f2 fit curve
values (for a(n) and b(n) calculation)
```

```
f1f2_sum=sum(f1f2_sqr);      %similarly sum of all elements of f1f2_sqr
f1f2_sqrdsum=f1f2_sum^2;
```

```
a(1:16)=((f1(1:16)*f2_sum-f2(1:16)*f1f2_sum)/(f1_sum*f2_sum-f1f2_sqrdsum));
```

```
%First FIR filter coefficients are calculated
```

```
a(1:16)=fix(a(1:16)*100); %a(n)is converted to integers after multiplying with 100
```

```
%Displays the a(n) plot
```

```
% figure,plot(n ,a,'b');
```

```
% title('\it{First fit curve a(n)}','FontSize',16);
```

```
% xlabel('Index');
```

```
% ylabel('Coefficient values');
```

```
b(1:16)=((f2(1:16)*f1_sum-f1(1:16)*f1f2_sum)/(f1_sum*f2_sum-f1f2_sqrdsum));
```

```
%Second FIR filter coefficients are calculated
```

```
b(1:16)=fix(b(1:16)*100); %b(n) is converted to integers after multiplying with 100
```

```
%Displays the b(n) plot
```



```

% figure,plot(n ,b,'b');

% %title('\it{Second fit curve b(n)}','FontSize',16);

% xlabel('Index');

% ylabel('Coefficient values');


% First fit curve and second fit curve are converted to integers after
% multiplying with 100

f1(1:16)=fix(f1(1:16).*100);
f2(1:16)=fix(f2(1:16).*100);


%%Following generates the F1 and F2 plots

% Figure,plot(n*50,f1,'b');

% %title('\it{first waveform f1(n)}','FontSize',16);

% xlabel('n values');

% ylabel('f1 values');

% hold on,plot(n*50,f2,'r');

% % title('\it{Second waveform f2(n)}','FontSize',16);

% % xlabel('n values');

% % ylabel('f2 values');

```

```
% Legend('First fit curve','Second fit curve');
```

```
%Size of noise signal is set by setting the size for t
```

```
t = (0:0.0016:32);
```

```
%noise signal yn is initialised with zeros first
```

```
yn =((1*randn(size(t))))/40;
```

```
yn(:)=fix(yn(:)*100 ); % noise is converted to integer after multiplying with 100
```

```
% noise signal is plotted here
```

```
% figure,plot(yn);
```

```
% %title('\it{noise waveform yn}','FontSize',16);
```

```
% xlabel('Sample points');
```

```
% ylabel('Amplitude');
```

```
%
```

```
%d variable is used to store the noise value first
```

```
d(1:length(yn))=yn(:);
```

```
% pulse (first fit curve) is added to d here at 10080
```

```
d(10080:10095)= (d(10080:10095)+f1);
```

```

%d is plotted here

% figure,plot(d);

% %title('\it{Data input: noise with pulse added}','FontSize',16);

% xlabel('Time in nano seconds');

% ylabel('Amplitude');


%d1 is initialized to zero first

d1(1:length(d)+15)=zeros;

% dis assigned to d1 here

d1(1:length(d))=d(1:length(d));


for i=1:length(d) % fitting operation is carried for entire length of noise signal with pulse
d1

    c1(i)=fix(sum(a(1:16).*d1(i:i+15))); % First fit curve operations

    c2(i)=fix(sum(b(1:16).*d1(i:i+15))); % Second fit curve operations
end


%c1 and c2 are scaled to 47

c1(:)=fix(c1(:).*47/max(c1));

```

```

c2(:)=fix(c2(:).*47/max(c2));

% c1 and c2 are plotted here

% figure,plot(c1);

% %title('\it{fit operation with a(n) results c1}','FontSize',16);

% xlabel('Sample points');

% ylabel('Amplitude');


% figure,plot(c2,'-');

% %title('\it{fit operation with b(n) results c2}','FontSize',16);

% xlabel('Sample points');

% ylabel('Amplitude');


% The fitted pulse height is calculated here

c(1:length(c1))=(c1(1:length(c1))+c2(1:length(c1)));


% C is plotted here

% figure,plot(c);

% %title('\it{Fit operation result: fitted pulse height}','FontSize',16);

% xlabel('Sample points');

% ylabel('Amplitude');

```

%below part is for calculating offset of the pulse

```
[maxc,max_loc]=max(c);
```

```
c(1:length(c1))=fix (c(1:length(c1)));
```

```
avg1=sum(d1(max_loc-22:max_loc-2))
```

```
avg2=sum(d1(max_loc+30:max_loc+50))
```

```
avg=(avg1+avg2)/40
```

%fit1 and fit2 are initialized with zeros...

```
fit1(1:length(c),1:16)=zeros;
```

```
fit2(1:length(c),1:16)=zeros;
```

%fit1=c1*f1 and fit2=c2*f2 are calculated here

```
for i=10050:length(c)-8000    %just a part of the signal is executed
```

```
    fit1(i,1:16)=(f1(1:16).*c1(i)); %fit1=c1*f1(1:16)
```

```
    fit2(i,1:16)=(f2(1:16).*c2(i)); %fit2=c2*f2(1:16)
```

```
end
```

%c1_1=fit1 and c2_1=fit2...used one mor evariable

```
c1_1(1:length(fit1),1:16)=(fit1(1:length(fit1),:));
```

```
c2_1(1:length(fit1),1:16)=(fit2(1:length(fit2),:));
```

```

% figure,plot(c1_1(10080,:), 'b');

% %title('\it{ waveform fit1=c1*f1 }','FontSize',16);

% xlabel('sample points');

% ylabel('Amplitude');

% %

% figure,plot(c2_1(10079,:), 'b');

% %title('\it{ waveform fit2=c2*f2 }','FontSize',16);

% xlabel('sample points');

% ylabel('Amplitude');


%best-fit result varaible fit is initialized with zero

fit(1:length(fit1),:)=zeros;


%c1*f1 and c2*f2 are added ..and assigned to fit

for i=1:length(fit1)

    for j=1:16

        fit(i,j)=(c1_1(i,j)+c2_1(i,j)); %fit represents the best-fit results

    end

end


%%following section is to calculate fit result data around the maximum

%including the maximum

fit21(1:12000)=0;

```

```

fit22(1:12000)=0;

fit23(1:12000)=0;

%fit22=fit22';

for i=1:16

    fit21(10078+i-1)=fit(10078,i)./32;    %(-1) to get the exact fit.

end

for i=1:16

    fit22(10079+i-1)=fit(10079,i)./32;    %(-1) to get the exact fit.

end

for i=1:16

    fit23(10080+i-1)=fit(10080,i)./32;    %(-1) to get the exact fit.

end


fit_c1_1(1:12000)=0;

for i=1:16

    fit_c1_1(10080+i-1)=c1_1(10080,i)./32;    %(-1) to get the exact fit.

end


fit_c2_1(1:12000)=0;

for i=1:16

    fit_c2_1(10080+i-1)=c2_1(10080,i)./32;    %(-1) to get the exact fit.

end

```

```

% d1(:)=d1(:)./max(d1);

% fit23(:)=fit23(:)./max(fit23);

% fit_c1_1(:)=fit_c1_1(:)./max(fit_c1_1);

% fit_c2_1(:)=fit_c2_1(:)./max(fit_c2_1);


%following plot generates the best-fit result with  $c1*f1$  and  $c2*f2$  and also

%noisy pulse

figure,plot((1:length(d1))*1,d1(1:length(d1)),'b');

hold on,plot((1:length(fit23))*1,fit23(:),'r');

xlabel('Time in nano seconds');

ylabel('Amplitude');

grid on;


hold on,plot((1:length(fit_c1_1))*1,fit_c1_1,'g');

hold on,plot((1:length(fit_c2_1))*1,fit_c2_1,'m');

legend('noisy pulse','Best fit result','C1*F1','C2*F2');

%

% figure,plot((1:length(d1))*50,d1);

% xlabel('Time in nano seconds');

% ylabel('Amplitude');

%Program to obtain the Best-fit result for a pulse with some DC offset

```


% Best-fit results of a pulse with offset

clear all; % clears all data in the matlab

close all; % closes all plots

clc; % clears all command window(main matlab window) data

n(1:16)=(1:16); % initialise n with 16 numbers from 1 to 16

f1(1:16)=((1-exp(-(n(1:16)-1)*0.5)).*exp(-(n(1:16)-1)*1/6)); % fit curve f1 equation

f2(1:16)=zeros; % fit curve f2 is initialised with all zeros

f2(2:16)=((1-2.7183.^(-(2:16)-2)*0.5)).*2.7183.^(-(2:16)-2)*1/6)); % fit curve f2

implementation

f1_sqr(1:length(f1))=f1.*f1; %f1_square gives the square of each number of f1 fit curve

(for a(n) and b(n) calculation)

f1_sum=sum(f1_sqr); %f1_sum gives the sum of all elements of f1_sqr (for a(n)

and b(n) calculation)

f2_sqr(1:length(f2))=f2.*f2; %similarly fit curve f2 square of individual numbers (for

a(n) and b(n) calculation)

f2_sum=sum(f2_sqr); %similarly sum of all the elements of f2_sqr (for a(n) and

b(n) calculation)

```
f1f2_sqr(1:length(f1))=f1.*f2; %element by elemnt multiplication of f1 and f2 fit curve
values (for a(n) and b(n) calculation)
```

```
f1f2_sum=sum(f1f2_sqr);      %similarly sum of all elements of f1f2_sqr
```

```
f1f2_sqrdsum=f1f2_sum^2;
```

```
a(1:16)=((f1(1:16)*f2_sum-f2(1:16)*f1f2_sum)/(f1_sum*f2_sum-f1f2_sqrdsum));
```

```
%First FIR filter coefficients are calculated
```

```
a(1:16)=fix(a(1:16)*100); %a(n)is converted to integers after multiplying with 100
```

```
%Displays the a(n) plot
```

```
% figure,plot(n ,a,'b');
```

```
% %title('\it{First fit curve a(n)}','FontSize',16);
```

```
% xlabel('Index');
```

```
% ylabel('Coefficient values');
```

```
b(1:16)=((f2(1:16)*f1_sum-f1(1:16)*f1f2_sum)/(f1_sum*f2_sum-f1f2_sqrdsum));
```

```
%Second FIR filter coefficients are calculated
```

```
b(1:16)=fix(b(1:16)*100);    %b(n) is converted to integers after multiplying with 100
```

```
%Displays the b(n) plot
```

```
% figure,plot(n ,b,'b');
```

```
% %title('\it{Second fit curve b(n)}','FontSize',16);
```

```
% xlabel('Index');
```

```

% ylabel('Coefficient values');

% First fit curve and second fit curve are converted to integers after
% multiplying with 100
f1(1:16)=fix(f1(1:16).*100);
f2(1:16)=fix(f2(1:16).*100);

%%Following generates the F1 and F2 plots
% Figure,plot(n*50,f1,'b');
% %title('\it{first waveform f1(n)}','FontSize',16);
% xlabel('n values');
% ylabel('f1 values');
% hold on,plot(n*50,f2,'r');
% % title('\it{Second waveform f2(n)}','FontSize',16);
% % xlabel('n values');
% % ylabel('f2 values');
% Legend('First fit curve','Second fit curve');

%Size of noise signal is set by setting the size for t

```

```
t = (0:0.0016:32);
```

```
%noise signal yn is initialised with zeros first
```

```
yn =((1*randn(size(t))))/40;
```

```
yn(:)=fix(yn(:)*100 ); % noise is converted to integer after multiplying with 100
```

```
% noise signal is plotted here
```

```
% figure,plot(yn);
```

```
% %title('\it{noise waveform yn}','FontSize',16);
```

```
% xlabel('Sample points');
```

```
% ylabel('Amplitude');
```

```
%
```

```
%d variable is used to store the noise value first
```

```
d(1:length(yn))=yn(:)+10;
```

```
% pulse (first fit curve) is added to d here at 10080
```

```
d(10080:10095)= (d(10080:10095)+f1);
```

```
%d is plotted here
```

```
figure,plot(d);
```

```

%title("\it{Data input: noise with pulse added}','FontSize',16);

xlabel('Time in nano seconds');

ylabel('Amplitude');


% d1 is initialized to zero first

d1(1:length(d)+15)=zeros;

% dis assigned to d1 here

d1(1:length(d))=d(1:length(d));


for i=1:length(d) % fitting operation is carried for entire length of noise signal with pulse
d1

    c1(i)=fix(sum(a(1:16).*d1(i:i+15))); % First fit curve operations

    c2(i)=fix(sum(b(1:16).*d1(i:i+15))); % Second fit curve operations
end


%c1 and c2 are scaled to 47

c1(:)=fix(c1(:).*47/max(c1));

c2(:)=fix(c2(:).*47/max(c2));


%c1 and c2 are plotted here

```

```

% figure,plot(c1);

% %title('\it{fit operation with a(n) results c1}','FontSize',16);

% xlabel('Sample points');

% ylabel('Amplitude');


% figure,plot(c2,'-');

% %title('\it{fit operation with b(n) results c2}','FontSize',16);

% xlabel('Sample points');

% ylabel('Amplitude');


%The fitted pulse height is calculted here

c(1:length(c1))=(c1(1:length(c1))+c2(1:length(c1)));


% C is plotted here

% figure,plot(c);

% %title('\it{Fit operation result: fitted pulse height}','FontSize',16);

% xlabel('Sample points');

% ylabel('Amplitude');


%below part is for calculating offset of the pulse

[maxc,max_loc]=max(c);

c(1:length(c1))=fix (c(1:length(c1)));

```

```

avg1=sum(d1(max_loc-22:max_loc-2))

avg2=sum(d1(max_loc+30:max_loc+50))

avg=(avg1+avg2)/40


% figure,plot(c);

% %title('\it{ fitted pulse height}','FontSize',16);

% xlabel('Sample points');

% ylabel('Amplitude');


%%%%% below part is for implementing algorithm on the pulse with some

%%%%% offset...

for i=1:1:length(d)

    for j=1:16

        c1_new_temp(j)=(d1(i+j-1).*a(j))-(avg.*(a(j)));

        c2_new_temp(j)=(d1(i+j-1).*b(j))-(avg.*(b(j)));

    end

    c1_new(i)=(sum(c1_new_temp(:)));

    c2_new(i)=(sum(c2_new_temp(:)));

end

```

%c1_new and c2_new found are scaled to pulse amplitude here

```
c1_new(:)=fix((c1_new(:).*(47+0))/max(c1_new));
```

```
c2_new(:)=fix((c2_new(:).*(47+0))/max(c2_new));
```

%c_new is found by adding c1_new and c2_new

```
c_new(1:length(c1))=(c1_new(1:length(c1_new))+c2_new(1:length(c2_new)));
```

```
% figure,plot(c_new);
```

%fit1=c1*f1 and fit2=c2*f2 variables are initialized here

```
fit1_new(1:length(c),1:16)=zeros;
```

```
fit2_new(1:length(c),1:16)=zeros;
```

%fit1=c1*f1 and fit2=c2*f2 variables are calculated here

```
for i=1:length(c)    %just a part of the signal is executed
```

```
    fit1_new(i,1:16)=(f1(1:16).*c1_new(i));
```

```
    fit2_new(i,1:16)=(f2(1:16).*c2_new(i));
```

```
end
```

%fit1_new and fit2_new are assigned to c1_1_new and c2_1_new

```
c1_1_new(1:length(fit1_new),1:16)=(fit1_new(1:length(fit1_new),:));
```

```
c2_1_new(1:length(fit1_new),1:16)=(fit2_new(1:length(fit2_new),:));
```



```

%the final fit variable is initialized to zero first

fit_new(1:length(fit1_new),:)=zeros;

%final fit variable is obtained by adding c1_1_new and c2_1_new
for i=1:length(fit1_new)

    for j=1:16

        fit_new(i,j)=(c1_1_new(i,j)+c2_1_new(i,j));

    end

end

fit23(1:length(fit1_new))=0;

%following variables give the fit results around the maximum..including the

%maximum

for i=1:16

    fit21(10079+i-1)=(fit_new(10078,i)./(32))+avg;    %(-1) to get the exact fit.

end

for i=1:16

    fit22(10079+i-1)=(fit_new(10079,i)./(32))+avg;    %(-1) to get the exact fit.

end

for i=1:16

    fit23(10080+i-1)=(fit_new(10080,i)./(32))+0;    %(-1) to get the exact fit.

```

end

```
fit23(:)=fit23(:)+avg;
```

```
%best-fit results are plotted here
```

```
figure,plot((1:length(d1))*50,d1(1:length(d1)),'b-+');
```

```
% hold on,plot(fit21(:),'r-*');
```

```
%hold on,plot(fit22(:),'r-*');
```

```
hold on,plot((1:length(fit23))*50,fit23(:),'r-*');
```

```
%title("\it{ waveform fit=c1*f1+c2*f2}','FontSize',16);
```

```
xlabel('sample points');
```

```
ylabel('Amplitude');
```

```
grid on;
```

```
legend('noisy pulse','Best fit result');
```

```
%Program to obtain the pulse height accuracy
```

```

% pulse height accuracy

clear all; % clears all data in the matlab

close all; % closes all plots

clc;    % clears all command window(main matlab window) data

n(1:16)=(1:16); % initialise n with 16 numbers from 1 to 16


f1(1:16)=((1-exp(-(n(1:16)-1)*0.5)).*exp(-(n(1:16)-1)*1/6)); % fit curve f1 equation


f2(1:16)=zeros;    % fit curve f2 is initialised with all zeros

f2(2:16)=((1-2.7183.^(-(2:16)-2)*0.5)).*2.7183.^(-(2:16)-2)*1/6)); % fit curve f2

implementation


f1_sqr(1:length(f1))=f1.*f1;  %f1_square gives the square of each number of f1 fit curve
(for a(n) and b(n) calculation)

f1_sum=sum(f1_sqr);          %f1_sum gives the sum of all elements of f1_sqr (for a(n)
and b(n) calculation)


f2_sqr(1:length(f2))=f2.*f2;  %similarly fit curve f2 square of individual numbers (for
a(n) and b(n) calculation)

```

```
f2_sum=sum(f2_sqr);          %similarly sum of all the elements of f2_sqr (for a(n) and
b(n) calculation)
```

```
f1f2_sqr(1:length(f1))=f1.*f2; %element by elemnt multiplication of f1 and f2 fit curve
values (for a(n) and b(n) calculation)
```

```
f1f2_sum=sum(f1f2_sqr);      %similarly sum of all elements of f1f2_sqr
f1f2_sqrdsun=f1f2_sum^2;
```

```
a(1:16)=((f1(1:16)*f2_sum-f2(1:16)*f1f2_sum)/(f1_sum*f2_sum-f1f2_sqrdsun));
```

```
%First FIR filter coefficients are calculated
```

```
a(1:16)=fix(a(1:16)*100); %a(n)is converted to integers after multiplying with 100
```

```
%Displays the a(n) plot
```

```
% figure,plot(n*50,a,'b');
```

```
% title('\it{First fit curve a(n)}','FontSize',16);
```

```
% xlabel('n values');
```

```
% ylabel('a values');
```

```
%clear f1f2_sqr,f1f2_sqrdsun,f1f2_sum,f1_sqr,f2_sqr,f1_sum,f2_sum; This statement
not required..cleared some variables to free RAM
```

```

b(1:16)=((f2(1:16)*f1_sum-f1(1:16)*f1f2_sum)/(f1_sum*f2_sum-f1f2_sqrdsum));

%Second FIR filter coefficients are calculated

b(1:16)=fix(b(1:16)*100);    %b(n) is converted to integers after multiplying with 100


%Displays the b(n) plot

% figure,plot(n*50,b,'b');

% title('\it{ Second fit curve b(n)}','FontSize',16);

% xlabel('n values');

% ylabel('b values');


% A pulse with 20ns delayed with respect to first fit curve was used

delay=20; %delay was set to 20ns

f1_d1(1:16)=0; %the variable to hold the 20ns delayed pulse was cleared


% Following for loop generates the 20ns delayed pulse

for i=2:16

    f1_d1(i)=(1-exp(-(((i-1)*50)-delay)/100)).*exp(-(((i-1)*50)-delay)*1/300);

end

f1_d1(:)=fix(f1_d1(:).*100); %the resulting pulse was also converted to integer after
multiplying with 100

```

```
% First fit curve and second fit curve are converted to integers after
```

```
% multiplying with 100
```

```
f1(1:16)=fix(f1(1:16).*100);
```

```
f2(1:16)=fix(f2(1:16).*100);
```

```
%%Following generates the F1 and F2 plots
```

```
f1(:)=f1(:)./(max(f1));
```

```
figure,plot(n*50,f1,'b');
```

```
%title('\it{first waveform f1(n)}','FontSize',16);
```

```
xlabel('n values');
```

```
ylabel('f1 values');
```

```
% hold on,plot(n*50,f2,'r');
```

```
% % title('\it{Second waveform f2(n)}','FontSize',16);
```

```
% % xlabel('n values');
```

```
% % ylabel('f2 values');
```

```
% Legend('First fit curve','Second fit curve');
```

```
%Size of noise signal is set by setting the size for t
```

```
t = (0:0.0016:32);
```

```
%noise signal yn is initialised with zeros first
```

```
yn(1:length(t))=zeros;
```

%d variable is used to hold the noise data

```
d(1:length(t))=zeros;
```

% pulse(20ns delayed pulse with respect to first fit curve) was added to the d variable

represents noisy pulse signal

```
d(10000:10015)=d(10000:10015)+f1_d1;
```

%d1 variable is same as d but with 15 more locations

```
d1(1:length(d)+15)=zeros;
```

% noise level is set here

```
noise_level=50;
```

% count variable and iter are initialised to zero

```
count=0;
```

```
iter=0;
```

clear f1,f2; % f1 and f2 are cleared to clear some RAM

% pulse height accuracy starts here..960 iterations were carried out and

% also less than rms(noise) to be less than 47 also checked.

```
while((iter<=960) & (rms(yn)<=47))
```

```

iter=iter+1 % each time iter variable is incremented by '1'

d1(1:length(d))=d(1:length(d)); % d is assigned to d

for i=1:length(d) % fitting operation is carried for entire length of noise signal with
pulse d1

    c1(i)=fix(sum(a(1:16).*d1(i:i+15))); % First fit curve operations
    c2(i)=fix(sum(b(1:16).*d1(i:i+15))); % Second fit curve operations
end

%%C1 and C2 are plotted below

% figure,plot(c1);
% title("\it{fit operation with a(n) results c1}','FontSize',16);
% xlabel('t values');
% ylabel('c1');

% figure,plot(c2,'-');
% title("\it{fit operation with b(n) results c2}','FontSize',16);
% xlabel('t values');
% ylabel('c2');

```



```

% fitted pulse height C is obtained by adding C1 and C2

c(1:length(c1))=c1(1:length(c1))+c2(1:length(c1));


% C fitted height is plotted here

%   figure,plot(c);

% %   title('\it{Fit operation result: fitted pulse height}','FontSize',16);

%   xlabel('t values');

%   ylabel('c');

%   count=0;


% variable to hold max location  for the current iteration is

% initialized here

max_c_loc(iter)=0;


% count variable is used to count how many max locations were obtained

% for each iteration

count=0;


% logic to detect the presence of maximum, condition changed to >= at

% two places to detect the same amplitudes at neighboring locations

for i=10:length(c)-10

```

```

    if ((c(i)>=c(i-1)) & (c(i)>c(i-2)) & (c(i)>c(i-3)) & (c(i)>c(i-4)) & (c(i)>=c(i+1)) &
(c(i)>c(i+2)) & (c(i)>c(i+3)) & (c(i)>c(i+4)) & c(i)>5000)

```

```

    count=count+1;

```

```

    max_c_loc(iter)=i; % maximum location is stored here

```

```

end

```

```

end

```

```

% absolute value of C1-C2 is calculated here

```

```

c_sub(iter,1:length(c))=abs(c1(1,:)-c2(1,:));

```

```

% maximum location is stored into loc_new

```

```

loc_new=max_c_loc(iter);

```

```

% maximum location +1 is stored into loc_new2

```

```

loc_new2=max_c_loc(iter)+1;

```

```

% noise array for each iteration is stored here to

```

```

noise_array(iter,1:length(d))=d(1,:);

```

```

% minimum in C1-C2 is used to locate the exact location of maximum

```

```

if c_sub(iter,loc_new)>c_sub(iter,loc_new2)

```

```

    loc_new=loc_new2;

```

```

end

```

```
value=c(loc_new); %amplitude value at the maximum location is stored into value
variable
```

```
% c1_t(iter,1:41)=c1(1,9980:10020); % 20points of C1 around the maximum of C
location are stored here
```

```
% c2_t(iter,1:41)=c2(1,9980:10020); % 20points of C2 around the maximum of C
location are stored here
```

```
% Amplitude found is scaled to have value around 47
```

```
max_value_iter(iter)=fix(value*47/9922);
```

```
%new noise is generated but with higher amplitude values by decreasing
```

```
%noise_value variable each time
```

```
t = (0:0.0016:32);
```

```
noise_level=noise_level-0.05;
```

```
yn =(1*randn(size(t)))/noise_level; % noise is generated here
```

```
yn(:)=fix(yn(:).*100); % noise is converted to integer after multiplying with 100
```

```
rms_noise(iter)=rms(yn); % rms of noise is found here
```

```
d=yn; % again noise is
```

```
d(10000:10015)=d(10000:10015)+f1_d1; % pulse(20ns delayed pulse with respect to
first fit curve) was added to the d variable represents noisy pulse signal
```

```
end % new iterations start here
```

```
% the resulting amplitudes found are plotted here
```

```
max_value_iter(:)=max_value_iter(:)./47;
```

```
figure,plot(max_value_iter);
```

```
xlabel('RMS noise');
```

```
ylabel('Pulse height found');
```

```
figure,plot(max_value_iter(1:852));
```

```
xlabel('RMS noise');
```

```
ylabel('Amplitude');
```

```
% the resulting maximum locations found are plotted here
```

```
figure,plot(max_c_loc(1:852))
```

```
xlabel('Iterations');
```

```
ylabel('Amplitude');
```

```
%Program to find the starting time of the pulse
```

```

% finding the starting time of the pulse by generating pulses of 1ns-49ns

% delay with respect to first fit curve

% if noise is needed, line number 115(d(1:length(yn))=zeros;) should be changed..replace
zeros by yn

clear all; % clears all data in the matlab

close all; % closes all plots

clc; % clears all command window(main matlab window) data

n(1:16)=(1:16); % initialise n with 16 numbers from 1 to 16

f1(1:16)=((1-exp(-(n(1:16)-1)*0.5)).*exp(-(n(1:16)-1)*1/6)); % fit curve f1 equation

f2(1:16)=zeros; % fit curve f2 is initialised with all zeros

f2(2:16)=((1-2.7183.^(-(2:16)-2)*0.5)).*2.7183.^(-(2:16)-2)*1/6)); % fit curve f2

implementation

f1_sqr(1:length(f1))=f1.*f1; %f1_square gives the square of each number of f1 fit curve
(for a(n) and b(n) calculation)

f1_sum=sum(f1_sqr); %f1_sum gives the sum of all elements of f1_sqr (for a(n)
and b(n) calculation)

```

```
f2_sqr(1:length(f2))=f2.*f2; %similarly fit curve f2 square of individual numbers (for
a(n) and b(n) calculation)
```

```
f2_sum=sum(f2_sqr); %similarly sum of all the elements of f2_sqr (for a(n) and
b(n) calculation)
```

```
f1f2_sqr(1:length(f1))=f1.*f2; %element by element multiplication of f1 and f2 fit curve
values (for a(n) and b(n) calculation)
```

```
f1f2_sum=sum(f1f2_sqr); %similarly sum of all elements of f1f2_sqr
```

```
f1f2_sqrdsum=f1f2_sum^2;
```

```
a(1:16)=((f1(1:16)*f2_sum-f2(1:16)*f1f2_sum)/(f1_sum*f2_sum-f1f2_sqrdsum));
```

```
%First FIR filter coefficients are calculated
```

```
a(1:16)=fix(a(1:16)*100); %a(n)is converted to integers after multiplying with 100
```

```
%Displays the a(n) plot
```

```
% figure,plot(n*50,a,'b');
```

```
% title('\it{First fit curve a(n)}','FontSize',16);
```

```
% xlabel('n values');
```

```
% ylabel('a values');
```

```
%clear f1f2_sqr,f1f2_sqrdsum,f1f2_sum,f1_sqr,f2_sqr,f1_sum,f2_sum; This statement
not required..cleared some variables to free RAM
```

```

b(1:16)=((f2(1:16)*f1_sum-f1(1:16)*f1f2_sum)/(f1_sum*f2_sum-f1f2_sqrdsum));

%Second FIR filter coefficients are calculated

b(1:16)=fix(b(1:16)*100);    %b(n) is converted to integers after multiplying with 100

```

```

%Displays the b(n) plot

% figure,plot(n*50,b,'b');

% title('\it{Second fit curve b(n)}','FontSize',16);

% xlabel('n values');

% ylabel('b values');

```

```

% First fit curve and second fit curve are converted to integers after

% multiplying with 100

f1(1:16)=fix(f1(1:16).*100);

f2(1:16)=fix(f2(1:16).*100);

```

```

%%Following generates the F1 and F2 plots

% Figure,plot(n*50,f1,'b');

% %title('\it{first waveform f1(n)}','FontSize',16);

% xlabel('n values');

```

```

% ylabel('f1 values');

% hold on,plot(n*50,f2,'r');

% % title('\it{Second waveform f2(n)}','FontSize',16);

% % xlabel('n values');

% % ylabel('f2 values');

% Legend('First fit curve','Second fit curve');

```

%Size of noise signal is set by setting the size for t

```
t = (0:0.0016:32);
```

%d variable is used to hold the noise data

```
d(1:length(t))=zeros;
```

% noise level is set here

```
noise_level=50;
```

%delay to introduce into the pulse is initialized here

```
delay=0;
```

% variable to store the maximum locations is stored here


```

max_locations(1:50)=zeros;

% 50 iterations for 50 pulses
for iter=1:1:50

    % for each iteration delay is increased by '1'ns
    delay=delay+1;

    %f1_d1 variable holds the delayed pulse values
    f1_d1(1:16)=0;

    %pulse sample values are generated here
    for i=2:16

        f1_d1(i)=(1-exp(-(((i-1)*50)-delay)/100)).*exp(-(((i-1)*50)-delay)*1/300);
    end

    f1_d1(:)=fix(f1_d1(:).*100); % pulse samples are converted to integers after
multiplying by 100

    %hold on,plot((1:16)*1,f1_d1(1:16),'g'); % plots the pulse values

    %Random noise values are initialized here
    yn =(1*randn(size(t)))/noise_level;

```

```

%Noise values are converted to integers after multiplying by 100

yn(:)=fix(yn(:).*100);

% Generates noise plot

% figure,plot(yn);

% %title('\it{noise waveform yn}','FontSize',16);

% xlabel('t values');

% ylabel('yn');

%d variable is used to hold the noise data

d(1:length(yn))=zeros; %This is for without noise test...if noise is needed replace
zeros word by yn

%delayed pulse is added to the noise signal

d(10000:10015)=d(10000:10015)+f1_d1(2:17);

%d1 is same as d but with 15 more locations compared to d

d1(1:length(d)+15)=zeros;

% d is assigned to d1

d1(1:length(d))=d(1:length(d));

```

```

for i=1:length(d) % fitting operation is carried for entire length of noise signal with
pulse d1

    c1(i)=fix(sum(a(1:16).*d1(i:i+15))); % First fit curve operations

    c2(i)=fix(sum(b(1:16).*d1(i:i+15))); % Second fit curve operations

end

%%C1 and C2 are plotted below

%   figure,plot(c1);

%   title('\it{fit operation with a(n) results c1}','FontSize',16);

%   xlabel('t values');

%   ylabel('c1');


%   figure,plot(c2,'-');

%   title('\it{fit operation with b(n) results c2}','FontSize',16);

%   xlabel('t values');

%   ylabel('c2');


% fitted pulse height C is obtained by adding C1 and C2

c(1:length(c1))=c1(1:length(c1))+c2(1:length(c1));

```

```

% C fitted height is plotted here

%   figure,plot(c);

%   title('\it{Fit operation result: fitted pulse height}','FontSize',16);

%   xlabel('t values');

%   ylabel('c');

%   count=0;


% variable to hold max location  for the current iteration is

% initialized here

max_c_loc(iter)=0;


% count variable is used to count how many max locations were obtained

% for each iteration

count=0;


% logic to detect the presence of maximum, condition changed to >= at

% two places to detect the same amplitudes at neighboring locations

for i=10:length(c)-10

    if ((c(i)>=c(i-1)) & (c(i)>c(i-2)) & (c(i)>c(i-3)) & (c(i)>c(i-4)) & (c(i)>=c(i+1)) &

(c(i)>c(i+2)) & (c(i)>c(i+3)) & (c(i)>c(i+4)) & c(i)>5000) % COMPARE WITH

NEIGHBOURING 4 AND A CONSTANT WHICH NOW IS '1'

        count=count+1;

        max_c_loc(iter)=i; % maximum location is stored here

```

```

    end

end

% absolute value of C1-C2 is calculated here
c_sub(iter,1:length(c))=abs(c1(1,:)-c2(1,:));

% maximum location is stored into loc_new
loc_new=max_c_loc(iter);

% maximum location +1 is stored into loc_new2
loc_new2=max_c_loc(iter)+1;

% noise array for each iteration is stored here to
noise_array(iter,1:length(d))=d(1,:);

% minimum in C1-C2 is used to locate the exact location of maximum
if c_sub(iter,loc_new)>c_sub(iter,loc_new2)
    loc_new=loc_new2;
end

value=c(loc_new); %amplitude value at the maximum location is stored into value
variable

```

```

%c is scaled to 47 here

c_temp(1:length(c))=zeros;

c_temp(:)=fix((c(:).*47)./(9922));

%pulse location is stored in max_c_temp_loc which is used in delay

%equations

max_c_temp_loc=loc_new

% following three equations implement the delay equation

time(iter)=100*(log(c1(max_c_temp_loc)+c2(max_c_temp_loc)*1.9477)-
log(c1(max_c_temp_loc)+c2(max_c_temp_loc)*1.1814));

time32(iter)=100*(log(c1(max_c_temp_loc)+c2(max_c_temp_loc).*1.9477)-
log(c1(max_c_temp_loc)+c2(max_c_temp_loc).*1.1814));

time_2(iter)=100*(log(c1(max_c_temp_loc)+c2(max_c_temp_loc)*1.947742786288900
7748922546337458)-
log(c1(max_c_temp_loc)+c2(max_c_temp_loc)*1.1813617330197360295984794350891
))

% max location found is stored in max_locations array

max_locations(iter)=max_c_temp_loc;

% c1, c2, c and c_temp variable values are stored in following variables

```

```

% for each iteration

c1_t(iter,1:20001)=(c1(:)');
c2_t(iter,1:20001)=(c2(:)');
c_t(iter,1:20001)=(c(:)');
c_temp_t(iter,1:20001)=(c_temp(:)');


%folloing statements clear the variables(f1_d1,d, d1..etc)..please keep it..all these are
initialized again

clear f1_d1;

clear d;

clear d1;

clear c;

clear c1;

clear c2;

clear c_temp;

clear yn;

iter

end


%end

% end

% t1(1:iter)=(50-time(1:iter));

% plot(t1);

```

```

%

% t2(1:iter)=(50-time_2(1:iter));

% figure,plot(t2);


% The delays calculated are plotted here

figure,plot(time_2);

%title('\it{Plot of calculated delay}','FontSize',12);

xlabel('Delay between the pulse and the sampling time in ns');

ylabel('Obtained delay values');


% The delays calculated are plotted here

figure,plot(time32);

%title('\it{Plot of calculated delay}','FontSize',12);

xlabel('Delay between the pulse and the sampling time in ns');

ylabel('Obtained delay values');


% time_mod variable is used..if a proper max location is not found this

% variable is manually changed for these locations

time_mod(1:49)=time32(1:49);


% calculating percentage error

perc(1:49)=(time_mod(1:49)./(1:49));

perc(1:49)=(perc(1:49)-1).*100;

```



```

perc=perc';

% time_mod variable is plotted here along with actual delay line
figure,plot(time_mod(1:49))

xlabel('Delay between the pulse and the sampling time in ns');

ylabel('ð- obtained delay values');

hold on,plot(1:49,'k');

time_mod=time_mod';

```

APPENDIX C: C++ AND VHDL PROGRAMS OF THE PC INTERFACE

BOARD

```

//C++ PROGRAM

//First inorder for this s/w to work the two dll files msvcp71.dll and msucr71.dll should
be copied and pasted in c:\windows\system32
// The files are in C:\Program Files\Opal Kelly\FrontPanel

// working

//First the XEM 3001 board is initialized with PLL and other information
//bit file is sent to the FPGA..ater successful configuration
//infinite loops are used to check if there is an event (RAM block full)

```

```

//after the detection of the event the RAM block is read
//one more infinite loop is used to detect RAM block full event from the second RAM
block of the FPGA
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <iomanip.h>
#include <string>
#include <conio.h>
#include <stdlib.h>
#include <conio.h>

#include "okFrontPanelDLL.h"
#define DES_FPGA_CONFIGURATION_FILE "memory_example_top_old_s.bit"
//bit file to be sent to FPGA is declared here
// bit file with name memory_example_top_old_s should be saved in file c:\dev-cpp
folder

// This function can be used to display data on screen at some location given by x and y
void gotoxy(int x, int y)
{
    COORD point;
    point.X = x; point.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE),
                           point);
}

//This function detects the RAM block full event and reads the RAM block contents
//Both the RAM blocks are read by this function
bool
performDES(okUSBFRONTPANEL_HANDLE xem)
{
    unsigned short int buf[8400],buf1[8192],buf23[8192]; // --- for reading 2048 bytes buf
    should be 3072...1.5 times 2048
    char str[5],str2[5]=" ",str1[5]=" ",str4[5]=" ",str3[5]=" ";
    long len;
    int a,i,k, j,words,count=0,en1=0 ;
    // std::fstream f_out; //f_out file is declared here
    //std::ofstream f_out2;

    // std::ofstream f_write;

```

```

FILE *CarInventory = fopen("cars.txt", "w");//cars.txt file is opened here for writing
anbd pointer is *CarInventory
fprintf(CarInventory, "%s\n", "      "); //First line in the cars file is blank

//  f_write.open("numbers2.txt",std::ios::out); //numbers2.txt file is also opened

//  if (false == f_write.is_open()) {

//      printf("Error: Output file cars could not be opened.\n");
//      return(false);
//  }

//100 times the both the RAM blocks are read..can be changed to any number
while (count<100) //count variable is increased by '2' each time both the RAM blocks
are read
{

    //now wait for the trigger signal from fpga in an infinite loop
    //loop is broken when we get the signal
    //signal indicates that first ram block is full....
    while (1)
    {
        okUsbFrontPanel_UpdateTriggerOuts(xem);
        if (okUsbFrontPanel_IsTriggered(xem, 0x60,0x01)==1) //check
for the signal
            break;
    }
    // printf(" ram 1 received rdy_tx1='1'\n");

    //now trigger is got make wr_en for the first ram block zero

    //start reading from the first block
    //reading pattern
    //16 bits are read from the okUsbFrontPanel_ReadFromPipeOut() function at each
time
    //to read one location (40 bits) 4*16 bits are read (more info in VHDL code)
    // i.e. for each FPGA memory location (40 bits) 64 bits or 4 times memory is
read(therefore 2048*4=8192 total readings)
    // 8192*2=16384 to convert the data read in char* format to int format
    //data is read into buf variable

```

```

    int numbytes=okUsbFrontPanel_ReadFromPipeOut(xem, 0xA0,16384,(unsigned
char*) buf);

    //print into file that data is read from the first RAM block
    fprintf(CarInventory, "\n from first ram block \n");
    //printf("\n from first ram block \n");

    //The read data is stored into proper format here
    for(int p=0;p<=8191;p=p+4) // since p is int p<=8191
    {
        i=p+1; //p is incremented by '1' because first time 0 is read into buf variable
        fprintf(CarInventory, " "); //print blank space in the file
        //fprintf(CarInventory, "%-4x  ", buf[i++]); //To display "f11f"
        fprintf(CarInventory, "%-4x  ", buf[i++]); //store the data in file
        fprintf(CarInventory, "%-4x  ", buf[i++]);
        fprintf(CarInventory, "%-4x  \n", buf[i]); //40 bits are stored including the
skipped first data(i=p+1)
        //          if ((i+1)%32==0)
        //      fprintf(CarInventory, "%s\n\n", " ");
        } //repeat for 8192/4=2048 times

    fprintf(CarInventory, "%s\n\n", " "); //store blank and leave some blank lines
    //Now wait for the other RAM block to become full
    while (1)
    {
        okUsbFrontPanel_UpdateTriggerOuts(xem);
        if (okUsbFrontPanel_IsTriggered(xem, 0x61,0x01)==1) //check for the
RAM block full event from the second RAM block
            break;
    }

    //read again the second RAM block data into variable buf23
    numbytes=okUsbFrontPanel_ReadFromPipeOut(xem, 0xA1, 16384,(unsigned
char*) buf23);
    fprintf(CarInventory, "\n from second ram block \n"); //store first that dat is now
from second RAM block before it becomes full
    // printf("\n from second ram block \n");
    //Now store the data into the file
    for(int p1=0;p1<=8191;p1=p1+4)
    {
        i=p1+1;
        fprintf(CarInventory, " ");
        // fprintf(CarInventory, "%-4x  ", buf23[i++]); //To display "f11f"
        fprintf(CarInventory, "%-4x  ", buf23[i++]);
        fprintf(CarInventory, "%-4x  ", buf23[i++]);
    }

```

```

        fprintf(CarInventory, "%-4x  \n", buf23[i]);
        //if ((i+1)%32==0)
        //    fprintf(CarInventory, "%s\n\n", " ");
    } // writing to file ends

    // writing to file ends
    fprintf(CarInventory, "%s\n\n", " "); //again leave some blank lines in the file
    count=count+2; //increase the count by 2
} /// count ends after incrementing by two
fprintf(CarInventory, "%s\n", " ");
//    f_write.close();
    return(true);
} //function ends here

//This function is used to initialize the FPGA with the bit file
//If the FPGA is initialized properly then there will not be any error message
bool
initializeFPGA(okUSBFROTPANEL_HANDLE xem)
{
    okPLL22150_HANDLE pll; // required command..initializes pll on the XEM3001
board

    // Open the first XEM.
    okUsbFrontPanel_Open(xem, 0);
    if (false == okUsbFrontPanel_IsOpen(xem)) {
        printf("XEM could not be opened. Is one connected?\n"); //If the XEM board
is not connected this message will be displayed
        return(false);
    }

    // Get some general information about the XEM.
    char str[40];
    printf("XEM firmware version: %d.%d\n", //displays some information about the
software and board
        okUsbFrontPanel_GetDeviceMajorVersion(xem),
        okUsbFrontPanel_GetDeviceMinorVersion(xem));
    okUsbFrontPanel_GetSerialNumber(xem, str);

    printf("XEM serial number: %s\n", str); //displays serial number of the board
    okUsbFrontPanel_GetDeviceID(xem, str); //
    //printf("XEM device ID: %s\n", str); //device id can be displayed here
    // Setup a new PLL profile:
    // P=400, Q=48 --> VCO output frequency = 400 MHz.

```

```

// DIV1N = 8
// Output 1 uses DIV1BYN (50 MHz).
pll = okPLL22150_Construct(); //PLL is initialised to generate 16MHz clock
okPLL22150_SetVCOParameters(pll, 400, 48);
okPLL22150_SetDiv1(pll, DivSrc_VCO, 25); // 25 makes output 16MHz -- clk for
generating gated_clock and frame_sync
okPLL22150_SetOutputSource(pll, 0, ClkSrc_Div1ByN);
okPLL22150_SetOutputEnable(pll, 0, true);

// needs to set the clk frequency as well as source // shld be 200KHz for 200Kbits/s i.e
0.2 MHz,80 replaced by 2000
// okPLL22150_SetDiv2(pll, DivSrc_VCO, 2000); // 5 MHz output --clock is clk //
change clk to higher or lower frequency and verify whether it works
// okPLL22150_SetOutputSource(pll, 0, ClkSrc_Div1ByN); // i.e it is generating
5MHz
// okPLL22150_SetOutputEnable(pll, 0, true);

// okPLL22150_SetDiv2(pll, DivSrc_VCO, 4000); // 0.1MHz(100KHz) output --
clock is clk3 // change clk to higher or lower frequency and verify whether it works
// okPLL22150_SetOutputSource(pll, 2, ClkSrc_Div1ByN); // i.e it is generating
5MHz
// okPLL22150_SetOutputEnable(pll, 2, true);

// Configure the XEM PLL.
okUsbFrontPanel_SetPLLConfiguration(xem, pll); //send the clock information to
it

// Download the configuration file.
if (false == okUsbFrontPanel_ConfigureFPGA(xem,
DES_FPGA_CONFIGURATION_FILE)) {
    printf("FPGA configuration failed.\n");
    return(false);
}

// Check for FrontPanel support in the FPGA configuration.
if (okUsbFrontPanel_IsFrontPanelEnabled(xem))
    printf("FrontPanel support is enabled.\n");
else
    printf("FrontPanel support is not enabled.\n");

return(true);
}

```

```

//main program
int
main()
{
    char infilename[128];
    char outfilename[128];
        int i, j;
    okUSBFROTPANEL_HANDLE xem; //get a handle for the XEM board

    printf("---- Opal Kelly ---- FPGA-Application v1.0 ----\n");

    // Initialize the DLL and load all entry points.
    if (FALSE == okFrontPanelDLL_LoadLib()) {
        printf("ERROR: FrontPanel DLL could not be initialized.\n");
        return(-1);
    }

    // Initialize the FPGA with the configuration bitfile.
    xem = okUsbXEM3001v2_Construct();
    if (initializeFPGA(xem) == false) {
        printf("FPGA could not be initialized.\n");
        return(-1);
    }

    // Now the reading from the RAM blocks is performed
    if (performDES(xem) == false) {
        printf("DES process failed.\n");
        return(-1);
    } else {
        printf("DES process succeeded yes !\n");
    }
    getch();
    //return(0);
}

```

--SINGLE_MEM.VHDL

--This program is interface to the PC software

--gets the 40 bit results from gated_gen_top.vhdl and stores in one of the dual memory blocks

```

--Opal kelly PC interface signals
  --hi_clk,hi_cs,hi_rdwr,hi_busy,hi_irq,hi_addr,hi_data: signals which are controlled by
the Opal kelly software
      -- when the component okHostInterface is instantiated

--Inputs
  --clk:16MHz clock
  --sdin: Results input line coming from main board

--Outputs
  --gated_clock:results read clock
  --frame_sync: frame sync signal for reconfiguration counters

--coregen components
  --dual memory

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_misc.all;
use IEEE.std_logic_unsigned.all;
use IEEE.numeric_std.all;

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity memory_example_top_old_s is

  port (

        hi_clk          : in  STD_LOGIC;
        hi_cs           : in  STD_LOGIC;
        hi_rdwr         : in  STD_LOGIC;
        hi_busy         : out STD_LOGIC;
        hi_irq          : out STD_LOGIC;
        hi_addr         : in  STD_LOGIC_VECTOR(3 downto 0);
        hi_data         : inout STD_LOGIC_VECTOR(15 downto 0);
        clk             :in std_logic;  --for generating data 200kbits/s
        --clk2          :in std_logic;  -- generating gated_clk and frame_sync
        --xclk1         :out std_logic;  --rdy_tx_t
        -----

```



```

--rdy_tx1_2d_t,rdy_tx2_2d_t,q_t:out std_logic;
-- pipeO_read_1_t,pipeO_read_2_t:out std_logic;
-----
sdin      :in std_logic;
frame_sync :out std_logic;
--clk_t    :out std_logic;
gated_clock :out std_logic
-- pipeO_data_1_t, pipeO_data_2_t:out std_logic_vector(15 downto 0);
-- addra_t  :out std_logic_vector(11 downto 0);
-- q_t      :out std_logic

);
end memory_example_top_old_s ;

architecture Behavioral of memory_example_top_old_s is

--dual_memeory component is used to store the results and send them to PC when itis full
--two such components are instantiated
component dual_memory
    port (
        addra: IN std_logic_VECTOR(10 downto 0);
        addrb: IN std_logic_VECTOR(10 downto 0);
        clka: IN std_logic;
        clkb: IN std_logic;
        dina: IN std_logic_VECTOR(39 downto 0);
        doutb: OUT std_logic_VECTOR(39 downto 0);
        ena: IN std_logic;
        enb: IN std_logic;
        wea: IN std_logic);
end component;

--BUFG component is global clock buffer
component BUFG
port (O : out STD_ULOGIC;
I : in STD_ULOGIC);
end component;

--JK flip-flop
component jkff_TOP
    Port ( Clk : in std_logic;
           j:in std_logic;
           k:in std_logic;

```

```

        Q :out std_logic);
end component ;

```

--gated_gen_top component generates the gated_clock and frame_sync and also reads the data coming from the main board

```

component GATED_GEN_TOP Port (
    CLK : in std_logic;
    GATED_CLOcK:inout std_logic;
    sdin:in std_logic;
    wr2ram:out std_logic;
    telemetry_data:out std_logic_vector(39 downto 0);
    FRAME_SYNC : out std_logic);
end component;

```

----- Opal Kelly components -----

--okHostInterface is a PC to FPGA interface component

```

component okHostInterface port (
    hi_clk           : in  std_logic;
    hi_rdwr          : in  std_logic;
    hi_cs            : in  std_logic;
    hi_irq           : out  std_logic;
    hi_busy          : out  std_logic;
    hi_addr          : in  std_logic_vector(3 downto 0);
    hi_data          : inout std_logic_vector(15 downto 0);
    ti_clk           : out  std_logic;
    ti_control       : out  std_logic_vector(12 downto 0);
    ti_data          : inout std_logic_vector(15 downto 0));
end component;

```

--okWireIn component is used to send signal(high or low) to FPGA which can be used to set or reset a signal

```

component okWireIn port (
    ti_clk          : in  std_logic;
    ti_control      : in  std_logic_vector(12 downto 0);
    ti_data         : in  std_logic_vector(15 downto 0);
    ep_addr         : in  std_logic_vector(7 downto 0);
    ep_dataout      : out  std_logic_vector(15 downto 0));
end component;

```

--okWireOut component is used to send some signals to PC(make high or low which can be detected by Opalkelly software in PC)

```

component okWireOut port (
    ti_clk          : in  std_logic;

```

```

        ti_control : in  std_logic_vector(12 downto 0);
        ti_data    : out std_logic_vector(15 downto 0);
        ep_addr    : in  std_logic_vector(7  downto 0);
        ep_datain  : in  std_logic_vector(15 downto 0));
    end component;

```

--okPipeOut component is used to read the RAM block data into PC

```

    component okPipeOut port (
        ti_clk      : in  std_logic;
        ti_control  : in  std_logic_vector(12 downto 0);
        ti_data     : out std_logic_vector(15 downto 0);
        ep_addr     : in  std_logic_vector(7  downto 0);
        ep_read     : out std_logic;
        ep_datain   : in  std_logic_vector(15 downto 0));
    end component;

```

--okTriggerIn component is used to trigger the FPGA components..trigger will be a one clock high signal

```

    component okTriggerIn port (
        ti_clk      : in  std_logic;
        ti_control  : in  std_logic_vector(12 downto 0);
        ti_data     : in  std_logic_vector(15 downto 0);
        ep_addr     : in  std_logic_vector(7  downto 0);
        ep_clk      : in  std_logic;
        ep_trigger  : out std_logic_vector(15 downto 0));
    end component;

```

--okTriggerOut component is used to trigger PC..trigger will be a one clock high signal (is used to inform PC that a RAM block is full)

```

    component okTriggerOut port (
        ti_clk      : in  std_logic;
        ti_control  : in  std_logic_vector(12 downto 0);
        ti_data     : out std_logic_vector(15 downto 0);
        ep_addr     : in  std_logic_vector(7  downto 0);
        ep_clk      : in  std_logic;
        ep_trigger  : in  std_logic_vector(15 downto 0));
    end component;

```

--temporary signals

```

--signal ena,enb:std_logic:='1';
signal dina:std_logic_vector(39 downto 0):=x"000000000000";
signal addr1_a,addr2_a,addr1_b,addr2_b:std_logic_vector(10 downto 0):="000000000000";

```

```

signal rdy_tx2,rdy_tx1:std_logic:='0';

--signal wea2,switch_wr_en ,clk_temp:std_logic:='0';

--signal pipeO_data_temp:std_logic_vector(39 downto 0):=x"000000000000";
signal WireIn10,pipeO_data_1,pipeO_data_2:std_logic_vector(15 downto 0):=x"0000";

signal reset2 :std_logic:='0';
signal ti_clk:std_logic:='0';

signal ti_control:std_logic_vector(12 downto 0):="000000000000000";
signal ti_data:std_logic_vector(15 downto 0):=x"0000";
signal pipeO_read_1,pipeO_read_2:std_logic:='0';
signal TrigOut60,TrigOut61 : STD_LOGIC_VECTOR(15 downto 0):=x"0000";
signal frame_sync_t: std_logic:='0';
signal rom_addr_t:std_logic_vector(13 downto 0);

signal wr2ram,q:std_logic:='0';

signal telemetry_data,pipeO_data_1_temp,pipeO_data_2_temp:std_logic_vector(39
downto 0):=x"000000000000";

signal count_4,count_4_2:std_logic_vector(1 downto 0):="00";
signal read_en_1b,read_en_2b,gated_clock_t:std_logic:='0';
signal reset5,rdy_tx1_1d,rdy_tx2_1d,rdy_tx1_2d,rdy_tx2_2d:std_logic:='0';
signal reset_addr_1a_2b,reset_addr_1b_2a,reset_addr2a,reset_addr2b:std_logic:='0';
signal reset_2_d,reset_5_d:std_logic:='0';

begin

    rdy_tx1_2d_t<=TrigOut60(0);
    rdy_tx2_2d_t<=TrigOut61(0);
    q_t<=q;

    --Switching between the two RAM blocks is done by the JK- Flip-Flop
    switch_rams_we:jkff_TOP
        port map ( Clk =>clk,
            j =>rdy_tx1_2d,
            k =>rdy_tx2_2d,
            Q =>q);

    --clk_t<=clk;

```

--if the wr2ram is '1' it indicates that 40 bits have been read..

```
process(clk)
begin
    if clk'event and clk='1' then
        if wr2ram='1' then
            dina<=telemetry_data; --telemetry_dat aread from main board is assigned to dina
            which will be stored into one of RAM block
        end if;
    end if;
end process;
```

--reset2 is used to enable write signal to ram_block1..

```
process(clk)
begin
    if clk'event and clk='1' then
        reset2<= not(q) and wr2ram; ---q is JK-Flip-Flop output...reset2 will be high for one
        clock cycle
    end if;
end process;
```

--dual_memory component is instantiated here

```
RAM_BLOCK1: dual_memory
    port map (
        addra => addr1_a , --write clock address
        addrb => addr1_b, --read clock address
        clka => clk,      --write clock
        clkb => ti_clk,   --read clock
        dina => dina,      --write data output port
        doutb => pipeO_data_1_temp, --read data output port
        ena => '1',        --write enable signal
        enb => read_en_1b,  --read enable
        wea => reset2);    --write enable signal
```

--dual_memory component is instantiated here (second memory block)

```
RAM_BLOCK2: dual_memory
    port map (
        addra => addr2_a ,      --write clock address
        addrb => addr2_b,      --read clock address
        clka => clk,           --write clock
        clkb => ti_clk,        --read clock
        dina => dina,          --write data output port
```

```

        doutb => pipeO_data_2_temp ,    --read data output port
    ena => '1',        --write enable signal
    enb => read_en_2b,    --read enable
    wea => reset5);      --write enable signal

--reset5 is used to enable write signal to ram_block2..
process(clk)
begin
    if clk'event and clk='1' then
        reset5<= q and wr2ram;  ---q is JK-Flip-Flop output...reset5 will be high for one
    clock cycle
    end if;
end process;

--reset_addr2b<=(addr1_b(10) and addr1_b(9) and addr1_b(8) and addr1_b(7) and
addr1_b(6) and addr1_b(5) and addr1_b(4) and addr1_b(3) and addr1_b(2) and
addr1_b(1) and not(addr1_b(0)));

--this signal goes high when addr1_a="11111111000"..used to reset addr2_a and
addr1_b(write address of second RAM block and read address of first RAM block)
reset_addr1_b_2a<=(addr1_a(10) and addr1_a(9) and addr1_a(8) and addr1_a(7) and
addr1_a(6) and addr1_a(5) and addr1_a(4) and addr1_a(3) and not(addr1_a(2)) and
not(addr1_a(1)) and not(addr1_a(0)));

--when all the bits of addr1_a(write address of first RAM block) are '1'..indicates first
RAM block is full
rdy_tx1<=(addr1_a(10) and addr1_a(9) and addr1_a(8) and addr1_a(7) and addr1_a(6)
and addr1_a(5) and addr1_a(4) and addr1_a(3) and addr1_a(2) and addr1_a(1) and
(addr1_a(0)));

--reset2 is delayed by '1' clock cycle here (used to trigger PC)
process(clk)
begin
    if clk'event and clk='1' then
        reset_2_d<=reset2;
    end if;
end process;

--trigger out signal is assigned here..indicates to PC that a RAM block is full
process(clk)
begin
    if clk'event and clk='1' then
        TrigOut60(0)<=reset_2_d and rdy_tx1;  --one clock high signal
    end if;
end process;

```

```

    end if;
end process;

```

--rdy_tx1 is delayed by two cycles here..which is used to change the state of JK-Flip-Flop which then switches the RAM blocks

```

process(clk)
begin
    if clk'event and clk='1' then
        rdy_tx1_1d<=rdy_tx1;
    end if;
end process;

```

```

process(clk)
begin
    if clk'event and clk='1' then
        rdy_tx1_2d<=rdy_tx1_1d; -- rdy_tx1_2d goes to JK-Flip-Flop
    end if;
end process;

```

--write address of first RAM block is reset here and also incremented when reset2='1'

```

process(clk,reset_addr_1a_2b)
begin
    if reset_addr_1a_2b='1' then
        addr1_a<="000000000000";
    elsif clk'event and clk='1' then
        if reset2='1' then
            addr1_a<=addr1_a+"00000000001"; --whenver reset2 signal is '1' it is
incremented
        end if;
    end if;
end process ;

```

--similarly the write address of second RAM block is reset here and also incremented when reset5='1'

```

process(clk,reset_addr_1b_2a)
begin
    if reset_addr_1b_2a='1' then
        addr2_a<="000000000000";
    elsif clk'event and clk='1' then

```

```

        if reset5='1' then
            addr2_a<=addr2_a+"00000000001";    --whenver reset5 signal is '1' it is
incremented
        end if;
    end if;
end process ;

--read address of the first RAM block is reset here and also incremented when
read_en_1b='1'
-- read_en_1b is high when PC is reading the RAM block contents through the pipeout
component
process(ti_clk,reset_addr_1b_2a)
begin
    if reset_addr_1b_2a='1' then
        addr1_b<="000000000000";
    elsif ti_clk'event and ti_clk='1' then
        if (read_en_1b='1') then
            addr1_b<=addr1_b+"00000000001";
        end if;
    end if;
end process;

--since the results are sent in 16bits each time
--a counter is used to send the 40bits of a RAM location data in three clocks(8 zero bits
are sent as MSB are also sent. so totally 48 bits are sent)
process(reset_addr_1b_2a,ti_clk)
begin
    if reset_addr_1b_2a='1' then
        count_4<="11";    --when reading from the RAM block count_4 will be 3
    elsif ti_clk'event and ti_clk='1' then
        if pipeO_read_1='1' then    --after that it will be incremented..since it is a two bit
counter next state will be 00
            count_4<=count_4+"01";    --in 00,01 and 10 states data is sent to PC
        end if;
    end if;
end process;

--read_en_1b is a read enable signal of the first RAM block and it is '1' when both the bits
of count_4 are '1'
read_en_1b<=count_4(1) and count_4(0) and pipeO_read_1;

--based on the count_4 counter state results are sent
process(count_4(1 downto 0),pipeo_data_1_temp)
begin
    case (count_4(1 downto 0)) is

```



```

        when "00" => pipeO_data_1<=x"00" & pipeO_data_1_temp(39 downto 32); -
--in "00" state 8 zeros + 8 MSB bits of the 40bit result are sent
        when "01" => pipeO_data_1<=pipeO_data_1_temp(31 downto 16) ;    --in
"01" state 31-16 bits are sent
        when "10" => pipeO_data_1<=pipeO_data_1_temp(15 downto 0) ;    --in "10"
state 15-0 bits are sent
        when "11" => pipeO_data_1<=x"f11f";                            --in "11" state
dummy number "F11F" are sent but are not displayed in the results file
        when others => pipeO_data_1<=x"f11f";                            -- this state is never
reached.."others" state is required with the case statement
    end case;
end process;

```

```

--this signal goes high when addr2_a="1111111000"..used to reset addr1_a and addr_2b
(write address of first RAM block and read address of second RAM block)
reset_addr_1a_2b<=(addr2_a(10) and addr2_a(9) and addr2_a(8) and addr2_a(7) and
addr2_a(6) and addr2_a(5) and addr2_a(4) and addr2_a(3) and not(addr2_a(2)) and
not(addr2_a(1)) and not(addr2_a(0)));

```

```

--this signal is high when addr2_a all the bits are '1'..signifies that second RAM block is
full
rdy_tx2<=(addr2_a(10) and addr2_a(9) and addr2_a(8) and addr2_a(7) and addr2_a(6)
and addr2_a(5) and addr2_a(4) and addr2_a(3) and addr2_a(2) and addr2_a(1) and
not(addr2_a(0)));

```

```

--after the last write operation of the second RAM block ..delayed reset5 signal is used to
generate trigger(rigOut61(0))to PC

```

```

process(clk)
begin
if clk'event and clk='1' then
    reset_5_d<=reset5;
end if;
end process;

```

```

process(clk)
begin
if clk'event and clk='1' then
    TrigOut61(0)<=reset_5_d and rdy_tx2; --trigger is generated here..one clock high
signal
end if;
end process;

```

--rdy_tx2 is delayed by 2 clock cycles in order to change the state of the JK Flip_Flop
process(clk)

```
begin
  if clk'event and clk='1' then
    rdy_tx2_1d<=rdy_tx2;    --rdy_tx2 is delayed by one clock cycle here
  end if;
end process;
```

```
process(clk)
begin
  if clk'event and clk='1' then
    rdy_tx2_2d<=rdy_tx2_1d;    --rdy_tx2 is delayed by one more clock cycle here
  end if;
end process;
```

--read address of the second RAM block is reset here and also incremented when
read_en_2b='1'
-- read_en_2b is high when PC is reading the second RAM block contents through the
pipeout component

```
process(ti_clk,reset_addr_1a_2b)
begin
  if reset_addr_1a_2b='1' then
    addr2_b<="000000000000";
  elsif ti_clk'event and ti_clk='1' then
    if ( read_en_2b='1') then
      addr2_b<=addr2_b+"00000000001";    --address incrementing when
read_en_2b is '1'
    end if;
  end if;
end process ;
```

--reset_addr_1a_2b signal resets count_4_2 counter which is used to control the output
format from the second RAM block

```
process(reset_addr_1a_2b,ti_clk)
begin
  if reset_addr_1a_2b='1' then
    count_4_2<="11";    --counter is set "11"
  elsif ti_clk'event and ti_clk='1' then
    if pipeO_read_2='1' then    --when PC is reading its contents the counter is
incremented each time
      count_4_2<=count_4_2+"01";
    end if;
  end if;
end process;
```

```

        end if;
    end if;
end process;

--read_en_2b is a second RAM block read enable signal..it is generated when count_4_2
bits are both ones and also second block is being read(i.e. when pipeo_read_2='1')
read_en_2b<=count_4_2(1) and count_4_2(0) and pipeO_read_2;

----based on the count_4_2 counter state results are sent
process(count_4_2(1 downto 0),pipeo_data_2_temp)
begin
    case (count_4_2(1 downto 0)) is
        when "00" => pipeO_data_2<=x"00" & pipeO_data_2_temp(39 downto 32); --
in "00" state 8 zeros + 8 MSB bits of the 40bit result are sent
        when "01" => pipeO_data_2<=pipeO_data_2_temp(31 downto 16) ;      --in
"01" state 31-16 bits are sent
        when "10" => pipeO_data_2<=pipeO_data_2_temp(15 downto 0) ;      --in
"10" state 15-0 bits are sent
        when "11" => pipeO_data_2<=x"f22f";                          --in "11" state
dummy number "F11F" are sent but are not displayed in the results file
        when others => pipeO_data_2<=x"f22f";                        -- this state is never
reached.."others" state is required with the case statement
    end case;
end process;

----- gated clock generation-----
--gated_gen_top component is instantiated here
gated_framesync: GATED_GEN_TOP
    Port map(
        CLK => clk ,
        GATED_CLOcK => gated_clock_t,
        sdin =>sdin,
        wr2ram=>wr2ram,
        telemetry_data => telemetry_data ,
        frame_sync => frame_sync_t);

gated_clock<=gated_clock_t;  --gated clock and frame sync outputs are assigned here
frame_sync<=frame_sync_t;

-----
--okHostInterface component is instantiated here..which interfaces FPGA and PC
okHI : okHostInterface port map (hi_clk =>hi_clk,
        hi_cs => hi_cs,
        hi_rdwr => hi_rdwr,
        hi_irq => hi_irq,
```

```

        hi_busy => hi_busy,
        hi_addr => hi_addr,
        hi_data => hi_data,
        ti_clk => ti_clk,
        ti_control => ti_control,
        ti_data => ti_data);

--okWireIn componenet is instantiated here...not used in the current implementation
ep10 : okWireIn port map (
    ti_clk => ti_clk, ti_control => ti_control, ti_data => ti_data,
    ep_addr => x"10", ep_dataout => WireIn10);

--okTriggerOut component is instantiated here...used to trigger PC(indicates that first
RAM block is full)
ep60 : okTriggerOut port map (
    ti_clk => ti_clk, ti_control => ti_control, ti_data => ti_data,
    ep_addr => x"60", ep_clk => ti_clk, ep_trigger => TrigOut60); --Trigout60 --
signal indicates that first RAM block is full and ready to send data

--okTriggerOut component is instantiated here...used to trigger PC(indicates that second
RAM block is full)
ep61 : okTriggerOut port map (
    ti_clk => ti_clk, ti_control => ti_control, ti_data => ti_data,
    ep_addr => x"61", ep_clk => ti_clk, ep_trigger => TrigOut61); ----Trigout61
--signal indicates that second RAM block is full and ready to send data

--okPipeOut componenet with address x"A0" and pipeo_data_1(16 bits) read the contents
of the first RAM block
epA0 : okPipeOut port map (
    ti_clk => ti_clk, ti_control => ti_control, ti_data => ti_data,
    ep_addr => x"A0", ep_read => pipeO_read_1, ep_datain => pipeO_data_1);

--okPipeOut componenet with address x"AA" and pipeo_data_2(16 bits) read the
contents of the second RAM block
epA1 : okPipeOut port map (
    ti_clk => ti_clk, ti_control => ti_control, ti_data => ti_data,
    ep_addr => x"A1", ep_read => pipeO_read_2, ep_datain => pipeO_data_2);

-----
end Behavioral;
```

```

--GATED_GEN_TOP.VHDL
--This program generates the gated_clock and the frame_sync signal
--Reads the data line to get the results from the main board
--it uses cbare_10.vhdl, cb2re_160.vhdl and cb2re_40.vhdl programs

--Inputs
  --clk:16MHz clock
  --sdin:results data line

--Output
  --wr2ram: when high indicates that 40 bits have been read
  --telemetry_dtaa:40 bits results
  --frame_sync:minor frame sync signal.increments the reconfiguration counters

--Inout
  --gated_clock: clock to read the dat afrom the main board and also is used to read that
line into this board (PC interface board)

--coregen components
  --ram_as_rom and telemetry_in

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

-- Uncomment the following lines to use the declarations that are
-- provided for instantiating Xilinx primitive components.
library UNISIM;
use UNISIM.VComponents.all;

entity GATED_GEN_TOP is
  Port ( CLK : in std_logic;
        GATED_CLOcK:inout std_logic;
        sdin:in std_logic;
        wr2ram:out std_logic;
        telemetry_data:out std_logic_vector(39 downto 0);

```

```

--tc_160_t,tc_40_t,clock_11_t:out std_logic;
--q_160_t_t:out std_logic_vector(7 downto 0);
--q_40_t_t:out std_logic_vector(5 downto 0);
--rom_op:out std_logic_VECTOR(1 downto 0);
--- temp outputs--
-- clk_2_t,clock_11_t,clk2_40_t,rom_op_t_t,reset_m_t:out std_logic;
-- count_gated_t:out std_logic_vector(4 downto 0);
-- reset_count_gated_t,word_clock_t:out std_logic;
-- rom_addr_t:out std_logic_vector(13 downto 0);
-- gated_clock_delay_t,reset_t,rom_op_temp:out std_logic;
--count_40_t:out std_logic;
--temp_t:out std_logic_vector(39 downto 0);
-- control_rom_fifo_outbit_t,p_load_t,mem_we_t,geded_clock_delay_t:out
std_logic;
-- count_4_T:OUT STD_LOGIC_VECTOR(3 DOWNT0 0);
-----
FRAME_SYNC : out std_logic);
end GATED_GEN_TOP;

```

architecture Behavioral of GATED_GEN_TOP is

```

--temporary signals are declared and also initialized here
signal clk_2,clock_11,clk2_40,clk2_160,frame_sync_d1,frame_sync_d2:std_logic:='0'; --
clock divided by 2
signal tc_40,tc_160,reset:std_logic:='0';
signal ce_160,ce_40:std_logic:='1';
signal q_160_t: std_logic_vector(7 downto 0):=x"00";
signal q_40_t: std_logic_vector(5 downto 0):="000000";
signal rom_op_t: std_logic_VECTOR(0 downto 0):="0";
--signal rom_op_t: std_logic:='0';
signal rom_addr:std_logic_VECTOR(13 downto 0):="0000000000000000";
signal counter_1:std_logic_vector(7 downto 0):=x"00";
signal mem_we:std_logic:='0';
signal sdout1,count_40,gated_clock_delay,count_40_d:std_logic:='0';
signal count_40clocks:std_logic_vector(5 downto 0):="000000"; -- just to count 40
gated_clocks
signal temp:std_logic_vector(39 downto 0);
signal q_10:std_logic_vector(3 downto 0):="0000";

```

--ram_as_rom block is used to generate the gated_clock and frame_sync signals at periodic intervals

component ram_as_rom

port (

```

        addr: IN std_logic_VECTOR(13 downto 0);
        clk: IN std_logic;
        dout: OUT std_logic_VECTOR(0 downto 0));
end component;

```

--JK-FF component is used to divide the input by 2
 component jkff_TOP

```

    Port ( Clk : in std_logic;
          j:in std_logic;
          k:in std_logic;
          Q :out std_logic);
end component ;

```

--cb2re program is a divide by 10 program
 component cb2re

```

    Port ( Clk : in std_logic;
          CEO:out std_logic;
          CEO1:inout std_logic;
          q:out std_logic_vector(3 downto 0);
          CE:in std_logic );
end component ;

```

--cb2re_160 component is a divide by 160 program
 component cb2re_160

```

    Port ( Clk : in std_logic;
          CEO1:out std_logic;
          q:out std_logic_vector(7 downto 0);
          CE1:in std_logic );
end component ;

```

--cb2re_40 component is a divide by 40 program
 component cb2re_40

```

    Port ( Clk : in std_logic;
          Q :out std_logic_vector(5 downto 0);
          CE1:in std_logic );
end component ;

```

--telemetry_in component is a serial in shift register stores 40bits
 component telemetry_in

```

    port (
        CLK: IN std_logic;
        SDIN: IN std_logic;
        Q: OUT std_logic_VECTOR(39 downto 0));
end component;

```

```

begin

--divide the 16MHz clock by 2
div_2:jkff_TOP
  port map ( Clk =>clk,
             j =>'1',
             k =>'1',
             Q =>clk_2);

--divide the 8MHz clock by 10
div_10:cb2re
  port map (Clk =>clk,
            CEO =>clock_11, --used to generate frame_sync
            CEO1 =>clk2_160, --one of the clock enable signals for divide by 160
component
  q => q_10,      --output counter
  ce =>clk_2);

--generate the clock enable signals for divideby 160 and divide by 40 components
ce_160<=not(clk2_160) and not(clk_2);
ce_40<=not(clk2_160) and not(clk_2) and (clk2_40);

--divide by 160 component
div_160:cb2re_160
  port map (Clk =>clk, --16MHz clock
            CEO1 =>clk2_40, --clock to divide by 40 signal
            q => q_160_t, --output counter
            CE1 =>ce_160); --clock enable signal

--generate the ram_as_rom component address
rom_addr<=q_40_t & q_160_t;

--access the rom
access_ROM: ram_as_rom
  port map (
    addr => rom_addr,
    clk => clk,
    dout => rom_op_t); --rom output

-- divide by 40 block
div_40:cb2re_40
  port map (Clk =>clk,

```



```

    q => q_40_t, --output counter
    CE1 => ce_40); --clock enable signal for

---eliminatng the spike ---logic-----
process(reset,clk)
    variable counter:std_logic_vector(7 downto 0):=x"00";
    begin
        if clk='1' and clk'event then
            if reset='1' then    --if reset=1 then (reset =1 after 160 times(batch of ten) gated
clocks are generated
                counter:="000000"; --counter is reset
            elsif rom_op_t(0)='1' then    --when rom output is '1' increment the counter by '1'
                counter:=counter+'1';
            end if;
        end if;
        counter_1<=counter;
    end process;

reset<= ((counter_1(7))) and (not(counter_1(6))) and counter_1(5) and ((counter_1(4)))
and (not(counter_1(3))) and (not(counter_1(2))) and (not(counter_1(1))) and
(not(counter_1(0))); --160
--check whether gated clock and frame sync are generated properly
--reset<=counter_1(4) and (not(counter_1(3))) and ((counter_1(2))) and
(not(counter_1(1))) and (not(counter_1(0)));

--generate the gated_clock to send to main board
process(clk)
    begin
        if clk='1' and clk'event then
            gated_clock<=(not(clk_2)) and rom_op_t(0) and (not(reset));
        end if;
    end process;

--delay the gated clock to read the results from the main board
process(clk)
    begin
        if clk='1' and clk'event then
            gated_clock_delay<=gated_clock;
        end if;
    end process;

--counter is used to check if 40 bits are read
process(count_40,gated_clock_delay)

```

```

begin
  if gated_clock_delay='1' and gated_clock_delay'event then
    if count_40='1' then
      count_40clocks<="000001";
    else
      count_40clocks<=count_40clocks+'1';
    end if;
  end if;
end process;

--signal to indicate 40
count_40<=count_40clocks(5) and not(count_40clocks(4)) and (count_40clocks(3)) and
not(count_40clocks(2)) and not(count_40clocks(1)) and not(count_40clocks(0));

--shift in register to read the results data line..results coming from the main board
get_telemetrydata: telemetry_in
  port map (
    CLK => gated_clock_delay,
    SDIN => SDIN,
    Q => temp); --40 bits are read into this signal

--assign the 40 bits to output
telemetry_data<=temp;

--delay the count_40 signal
process(clk)
begin
  if clk='1' and clk'event then
    count_40_d<=count_40;
  end if;
end process;

--generate the signal to indicate that 40 bits ahve been read (a clock high signal)
process(clk)
begin
  if clk='1' and clk'event then
    if count_40='1' and count_40_d='0' then
      mem_we<='1';
    else
      mem_we<='0';
    end if;
  end if;
end process;

```

```

--assign that to the output signal
wr2ram<=mem_we;

-- rom_op<=rom_op_t;

--check if 160 counts have ben reached
tc_160<=q_160_t(7) and (not(q_160_t(6))) and (not(q_160_t(5))) and q_160_t(4) and
q_160_t(3) and q_160_t(2) and q_160_t(1) and q_160_t(0);

--check if 40 counts have been reached
tc_40<=q_40_t(5) and not(q_40_t(4)) and not(q_40_t(3)) and q_40_t(2) and
q_40_t(1) and q_40_t(0);

--generate the minor frame sync
frame_sync_d2<=(clock_11) and tc_160 ;

--delay the frame sync signal so that all the gated clocks are sent out
process(clk)
begin
    if clk'event and clk='1' then
        frame_sync_d1<=frame_sync_d2 ;
    end if;
end process;

process(clk)
begin
    if clk'event and clk='1' then
        frame_sync<=frame_sync_d1; --actual frame sync is generated here
    end if;
end process;

--temp signals ---

-- clk_2_t<=clk_2;
-- clock_11_t<=clock_11;
-- clk2_40_t<=clk2_40;
-- rom_op_t_t<=rom_op_t(0);
-- count_gated_t<=counter_1;
-- reset_count_gated_t<=reset;
-- rom_addr_t<=rom_addr;
-- q_40_t_t <=q_40_t ;
-- q_160_t_t <=q_160_t ;

```

```

-- count_40_t<=count_40;
-- mem_we_t<=mem_we;
-- geted_clock_delay_t<=gated_clock_delay;
-- temp_t<=temp;
-- word_clock_t<=word_clock;
-- reset_m_t<=reset_m;
-- control_rom_fifo_outbit_t<= control_rom_fifo_outbit;
-- count_4_t<=count_4;
-- p_load_t<=p_load;
-- rd_en_t<=rd_en;

--tc_160_t<=tc_160;
--tc_40_t<=tc_40;
--clock_11_t<=clock_11;

-- gated_clock_delay_t<=gated_clock_delay;
-- reset_t<=clk_2;
-- rom_op_temp<=rom_op_t(0);

end Behavioral;

```

```

--CB2RE.VHDL

--This program generates a divide by 10 output

--Inputs
  --clk:16MHz clock
  --ce:clock enable input

--Outputs
  --ceo:high when count is 10
  --q: 4 bit counter used to count till 9

--Inout
  --ceo1:high all the time except when count is zero

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.std_logic_misc.all;
use IEEE.numeric_std.all;

entity cb2re is
  Port ( Clk : in std_logic;
        CEO:out std_logic;
        CEO1:inout std_logic;
        Q :out std_logic_vector(3 downto 0);
        CE:in std_logic);
end cb2re;

```

architecture Behavioral of cb2re is

```
--terminal_count is used to count till 10
constant TERMINAL_COUNT : std_logic_vector(3 downto 0):=x"9";

--temporary signals r
signal r,ceo2:std_logic;
signal tc:std_logic:='0';
--signal ce:std_logic:='1';

signal Q_temp :std_logic_vector(3 downto 0):=x"0"; --q_temp is 3 bit counter used to
count till 10

begin

process( CLK)
begin
    if (CLK'event and CLK='1') then --check for clk'event and clk='1'
        if (CE='0') then    --CE is zero all the time
            if (R='1') then  --R is '1' when 10 clocks are counted (can be replaced with TC)
                Q_temp <= (others => '0'); --then q_temp is reset to zero
            else
                Q_temp <= Q_temp+"1";    -- else increment the counter values
            end if;
        end if;
    end if;
end process;

process(Q_temp)
begin
    if (Q_temp = TERMINAL_COUNT) then    --compare if counter=10
        TC <='1';                        --if yes then TC=1 and r=1
        r<='1';
    else
        TC <='0';                        --else tc=0 and r=0
        r<='0';
    end if;
end process;

process(clk)
begin
    if clk'event and clk='1' then
```

```

        if (TC='1') then          --whenever a terminal count has been reached make
ceo2=1 after '1' clock cycle
        ceo2<='1';              --it is used to generate ceo2 (once ceo2 becomes high it
will be high all the time)
        end if;
        end if;
    end process;

CEO<=TC and CEO2;              --ceo='1' when tc is reached..ceo2 is used to disable ceo for
first ten clock cycles
ceo1<=((Q_temp(3) or Q_temp(2) or Q_temp(1) or Q_temp(0))) or (not(CEO2)); --ceo1
is generated here..ceo2 is used to disable ceo1 for first ten clock cycles
q<=q_temp;                    --counter values are assigned to q
end Behavioral;

```

-- CB2RE_160.VHDL

-- This program is a divide by 160 which is used to access the ram_as_rom component
(address)

--Inputs

--clk:16MHz clock
--ce1:enable signal

--Outputs

--ceo1:high when 160 counts are reached
--q: 8 bit counter

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.std_logic_misc.all;
use IEEE.numeric_std.all;

```

```

entity cb2re_160 is
    Port ( Clk : in std_logic;
        -- CE2:in std_logic;
        CEO1:out std_logic;
        Q :out std_logic_vector(7 downto 0);
        CE1:in std_logic);
end cb2re_160;

```

architecture Behavioral of cb2re_160 is

```
--constant variable terminal_count is used to check whether 160 clocks are obtained
constant TERMINAL_COUNT : std_logic_vector(7 downto 0):=x"A0";

--
signal r,tc:std_logic;
--8 bit counter
signal Q_temp :std_logic_vector(7 downto 0):=X"00";

begin

  process(r,CLK)
  begin
    if (R='1') then --if 160 counts are reached the counter values are reset
      Q_temp <= (others => '0');
    elsif (CLK'event and CLK='1') then --else on clk transition from zero to 1
      if (ce1='1') then
        -- if ((CE1='0') and (CE2='0'))then
          Q_temp(7 downto 0) <= Q_temp(7 downto 0)+"1"; --counter is incremented
        end if;
      end if;
    end process;

  process(Q_temp)
  begin
    if (Q_temp(7 downto 0) = TERMINAL_COUNT) then --if the terminal count
has been reached
      TC <='1'; --make tc=1 and r=1
      r<='1';
    else
      TC <='0'; --else keep tc and r=0
      r<='0';
    end if;
  end process;
  --CEO<=TC ;--and CE;
  --ceo1<= Q_temp(7) or Q_temp(6) or Q_temp(5) or Q_temp(4) or Q_temp(3) or
Q_temp(2) or Q_temp(1) or Q_temp(0);

  ceo1<= Q_temp(7) and (not Q_temp(6)) and (not(Q_temp(5))) and Q_temp(4)
and Q_temp(3) and Q_temp(2) and Q_temp(1) and Q_temp(0); --ceo1='1'
when q_temp is 160
```



```

q<=q_temp(7 downto 0); --apply the counter values to output signal q
end Behavioral;

```

```

--CB2RE_40.VHDL

```

```

--This is a divide by 40 program

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.std_logic_misc.all;
use IEEE.numeric_std.all;

```

```

entity cb2re_40 is
  Port ( Clk : in std_logic;
        Q :out std_logic_vector(5 downto 0);
        CE1:in std_logic);
end cb2re_40;

```

```

architecture Behavioral of cb2re_40 is
--constant variable terminal_count is 40
constant TERMINAL_COUNT : std_logic_vector(5 downto 0):="101000";
signal r,tc:std_logic;

```

```

--6 bit counter
signal Q_temp :std_logic_vector(5 downto 0):="000000";

```

```

begin

--
  process(r,CLK)
  begin
    if (R='1') then      --if required counts are reached reset the counter
      Q_temp <= (others => '0');

      elsif (CLK'event and CLK='1') then
        if (ce1='1') then  --else if ce1='1'
          Q_temp <= Q_temp+"1"; -- increment the counter
        end if;
      end if;
    end process;

    process(Q_temp)
    begin
      if (Q_temp = TERMINAL_COUNT) then  --check if terminal count has been
reached
        TC <='1';          --if yes make tc and r='1'
        r<='1';
      else
        TC <='0';          --else keep tc and r=0
        r<='0';
      end if;
    end process;

    q<=q_temp;             --apply the counter values to the output signal
  end Behavioral;

```

--This program is a JK Flip-Flop

--Inputs

--J: J input to the Flip-Flop

--K: K input of the Flip-Flop

--clk: clock input

--Output

--Q: Flip-Flop output

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

entity jkff_TOP is

Port (Clk : in std_logic;

j:in std_logic;

k:in std_logic;

Q :out std_logic);

end jkff_TOP;

architecture Behavioral of jkff_TOP is

```
--temporary signal temp
signal temp:std_logic:='0';
```

```
--coding starts after the begin keyword
begin
```

```
--JK F-F implementation in the process block
```

```
process (Clk )
```

```
begin
```

```
    if (Clk'event and Clk='1') then      --At the positive clock edge
```

```
        if (J='0') then
```

```
            if (K='1') then              --check if j=0 and k=1
```

```
                temp <= '0';              --in that cse reset the Flip-Flop
```

```
            end if;
```

```
        else
```

```
            if (K='0') then              --else check for j=1 and k=0
```

```
                temp <= '1';              --set the Flip-Flop
```

```
            else                          --else j=1 and k=1
```

```
                temp <= not temp;          --toggle the state of the Flip-Flop
```

```
            end if;
```

```
        end if;
```

```
    end if;
```

```
end process;
```

```
q<=temp;                                --Assign the signal temp to Flip-Flop output
```

```
end Behavioral;
```