

Fall 1-31-2010

A FPGA/DSP design for real-time fracture detection using low transient pulse

Akash Mathur
New Jersey Institute of Technology

Follow this and additional works at: <https://digitalcommons.njit.edu/theses>



Part of the [Electrical and Electronics Commons](#)

Recommended Citation

Mathur, Akash, "A FPGA/DSP design for real-time fracture detection using low transient pulse" (2010).
Theses. 49.
<https://digitalcommons.njit.edu/theses/49>

This Thesis is brought to you for free and open access by the Electronic Theses and Dissertations at Digital Commons @ NJIT. It has been accepted for inclusion in Theses by an authorized administrator of Digital Commons @ NJIT. For more information, please contact digitalcommons@njit.edu.

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

A FPGA/DSP DESIGN FOR REAL-TIME FRACTURE DETECTION USING LOW TRANSIENT PULSE

**by
Akash Mathur**

This work presents the hardware and software architecture for the detection of fractures and edges in materials. While the detection method is based on the novel concept of Low Transient Pulse (LTP), the overall system implementation is based on two digital microelectronics technologies widely used for signal processing: Digital Signal Processor (DSP) and Field Programmable Gate Array (FPGA). Under the proposed architecture, the DSP carries out the analysis of the received baseband signal at a lower rate and hence can be used for large number of signal channels. The FPGA's master clock runs at a higher frequency (62.5MHz) for the generation of LTP signal and to demodulate the passband ultrasonic signals sampled at 1MHz which interrupts the DSP at every 1 μ s. This research elaborates on designing a Quadrature Amplitude Modulator - demodulator (QAM) on the FPGA for the received signal from the ultrasound and edge detection on the DSP processor to detect the presence of edges/fractures on a test Sawbone plate. In this work, the LTP technology is applied to determine the location of the Sawbone plate edges based on the reflected signals to the receivers. This signal is then passed through a QAM to get the maxima (peaks) at the received signal to study the parameters in the DSP. This work successfully demonstrates the feasibility of modular programming approach across the two platforms. The dual time scale platform readily accommodates higher temporal resolution needed for the generation of Low Transient Pulses and the processing of real time baseband signals on the DSP for various test conditions.

**A FPGA/DSP DESIGN FOR REAL-TIME FRACTURE DETECTION
USING LOW TRANSIENT PULSE**

by
Akash Mathur

**A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering**

Department of Electrical and Computer Engineering

January 2010

Blank Page

APPROVAL PAGE
A FPGA/DSP DESIGN FOR REAL-TIME FRACTURE DETECTION
USING LOW TRANSIENT PULSE

Akash Mathur

1/11/2010

Dr. Timothy Chang, Thesis Advisor
Interim Chairperson, Professor of Electrical and Computer Engineering, NJIT

Date

1/11/10

Dr. Durgamadhab Misra, Committee Member
Graduate Advisor, Professor of Electrical and Computer Engineering, NJIT

Date

1/11/10

Dr. Edwin Hou, Committee Member
Associate Chair for Undergraduate Studies, Associate Professor of Electrical and
Computer Engineering, NJIT

Date

BIOGRAPHICAL SKETCH

Author: Akash Mathur

Degree: Master of Science

Date: January 2010

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering,
New Jersey Institute of Technology, Newark, NJ, 2010
- Bachelor of Technology in Electronics and Communications Engineering,
Kurukshetra University, Haryana, India, 2006

“100 yrs from now no one will ever remember what clothes you wore, which car you drove, which perfume you put, which country you lived.....

But what will be remembered from this thesis is that blessed are those whose parents are like mine and the love I have for them is not something so easy to define.....

I dedicate my thesis to my parents

-- Akash

ACKNOWLEDGMENT

I am heartily thankful to my thesis advisor, Dr. Timothy Chang, whose encouragement, guidance, lot of novel ideas and support throughout my research and in writing my thesis is unquestionable. His extreme patience, forgiveness, inspiration, enthusiasm, and immense knowledge are the primary reasons for the success of my research.

Alongside my adviser I would like to thank my thesis committee, Dr. Durgamadhab (Durga) Misra and Dr. Edwin Hou, for their support and encouragement. I am really honored and thankful to them for being in the committee for my defense.

I am indebted to many of my colleagues to support me. Dr. Biao Cheng has helped me selflessly throughout the thesis and his contribution for my completion is undeniable. Ashish Ratnakar and Lan Yu have supported and helped me in more than one ways.

Lastly but most importantly, I will like to thank my parents and my sister for supporting me in which ever way they could and because of whom I am able to complete my thesis and hence achieve the degree of Master of Science.

TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION.....	1
1.1 Overview	1
1.2 Hardware Architecture	4
1.3 DSP Starter Kit.....	5
1.4 Daughter Board.....	5
1.5 Low Transient Pulse.....	7
1.6 Benefits.....	8
1.6.1 TMS320C6416T.....	8
1.6.2 SPARTAN IIE (XC2S300E).....	11
1.7 RTDX (Real Time Data Exchange).....	11
2 TMS32C6416: TI's DSP PROCESSOR	13
2.1 Digital Signal Processor.....	13
2.2 Memory Map.....	14
2.3 Interrupts.....	15
2.4 Timers.....	17
2.5 Software Programming.....	19
2.5.1 Project File (.pjt).....	20
2.5.2 Linker File.....	20
2.5.2.1 Memory.....	21
2.5.2.2 Section.....	21

TABLE OF CONTENTS
(Continued)

Chapter	Page
2.5.3 FPGA (.out) File.....	22
2.5.4 main.c Program.....	22
2.5.4.1 Initialization.....	22
2.5.4.1.1 ADC(THS1206).....	22
2.5.4.1.2 Flag Setup.....	24
2.5.4.1.3 Miscellaneous.....	25
2.5.4.2 Loading the FPGA.....	26
2.5.4.3 Free Run.....	27
2.5.4.3.1 Moving Average Filter.....	28
2.5.4.3.2 Edge Detection.....	29
3 SPARTAN 2E:FPGA.....	31
3.1 Hardware.....	31
3.2 System Design.....	32
3.2.1 Programming... ..	32
3.2.3 Simulation Analysis.....	33
3.3 Programming FPGA.....	34
3.3.1 Quadrature Amplitude Modulator-demodulator(QAM).....	36
3.3.1.1 Since and Cosine Generator.....	36
3.3.1.1.1 Simulation Results.....	37
3.3.1.1.2 Register Transistor Logic.....	39

TABLE OF CONTENTS
(Continued)

Chapter	Page
3.3.1.1.3 Synthesized Output.....	40
3.3.1.2 Infinite Impulse Response (IIR) Filter.....	40
3.3.1.2.1 Simulation Results and Design.....	41
3.3.1.2.2 Register Transistor Logic.....	44
3.3.1.3 Square Root.....	45
3.3.1.3.1 Simulation Results.....	47
3.3.1.3.2 Register Transistor Logic.....	47
3.3.1.4 Full QAM.....	49
3.3.1.5 Frequency Spectrum.....	49
3.3.2 Low Transient Pulse.....	51
3.3.2.1 Register Transistor Logic.....	51
3.3.3 Clock Divider.....	52
3.3.3.1 Register Transistor Logic.....	52
3.3.4 Interface (FPGA- DSP).....	53
3.3.4.1 Register Transistor Logic: Decoder.....	54
3.3.5 ADC Module.....	57
3.3.5.1 Register Transistor Logic.....	57
3.3.6 UnsignedtoSigned and SignedtoUnsigned Converters Module.....	58
3.3.7 Complete FPGA System Design.....	58
4 ALGORITHM AND RESULTS.....	60

TABLE OF CONTENTS
(Continued)

Chapter	Page
4.1 Algorithm.....	60
4.2 Experimental Verifications.....	61
4.2.1 Symmetric Placement.....	62
4.2.2 Asymmetric Placement.....	63
5 CONCLUSION.....	65
5.1 Conclusion.....	65
5.2 Future Work.....	66
APPENDIX A RESOLUTION.....	67
APPENDIX B DSP MEMORY.....	69
APPENDIX C SAMPLING TIME.....	71
REFERENCES.....	74

LIST OF TABLES

Table		Page
1.1	Comparison of various DSP processors in the market.....	9
2.1	Variables used in the FPGA program.....	26
3.1	System Information.....	31
3.2	Available IOs with various Spartan 2E packages.....	31
3.3	Filter Parameters defined in Matlab and the corresponding poles and zeros.....	41
3.4	Memory map of the EMIF for the interface between DSP and FPGA.....	54
4.1(a)	Estimation of longitudinal parameter of the plate, symmetric placement.....	63
4.1(b)	Passband maxima detection: summary of plate position errors.....	63
4.1(c)	Envelope maxima detection: summary of plate position errors.....	63
4.2(a)	Estimation of longitudinal parameter of the plate, asymmetric placement.....	64
4.2(b)	Passband maxima detection: summary of plate position errors.....	64
4.2(c)	Envelope maxima detection: summary of plate position errors.....	64

LIST OF FIGURES

Figure	Page
1.1 Experimental ultrasonic crack detection system.....	3
1.2 Block diagram to show the interface between the DSP and FPGA.....	4
1.3 Block diagram for the on board components and their interfaces on the DSK...	5
1.4 Low Transient Pulse.....	7
1.5 Millions instruction per second for various processors.....	10
2.1 Architectural overview of TMS320C6416 CPU.....	13
2.2 Memory map of C6416T DSP Starter Kit (DSK).....	15
2.3 THS1206 configuration flow.....	23
2.4(a) Envelope signal before filtering.....	29
2.4(b) Envelope signal after filtering.....	29
2.5 Snapshot of the value displayed in the Watch window of the CCS.....	30
3.1(a) Matlab/Simulink model for the QAM system which is implemented on the FPGA.....	33
3.1(b) Matlab/Simulink simulation results for the QAM system.....	34
3.2 Design flow of FPGA.....	35
3.3 Schematic displaying the design of the sine and cosine generator.....	36
3.4 Matlab simulation result for the generated data for the sine wave generator.....	37
3.5(a) Simulation results for the sine and cosine generator for $i= 1$ to $i=11$	38
3.5(b) Simulation results for the sine and cosine generator for $i=11$ to $i= 21$	38
3.6 RTL logic for the sine and cosine generator.....	39

LIST OF FIGURES
(Continued)

Figure	Page
3.7 Sine generator output observed on the oscilloscope.....	40
3.8 Magnitude response of the filter with parameters from Table 3.3.....	41
3.9a Direct- II form for the implementation of an IIR filter.....	42
3.9b Direct-II form re-arranged for its implemented over the FPGA.....	43
3.10 RTL for the IIR Filter.....	45
3.11 Restoring 2n-bit square rooter, combinational implementation.....	46
3.12 Simulation result for the square rooter.....	47
3.13(a) RTL for the Square Root consisting on onemod and subtractor modules.....	48
3.13(b) RTL for the sub block onemod consisting modules.....	48
3.13(c) RTL for the sub-sub-block (A) consisting a 2's complement, full adder, multiplexer and an inverter modules.....	48
3.14 Schematic of the QAM. Different sub-blocks port mapped to form one whole system.....	49
3.15 Received signal and its FFT.....	50
3.16 Envelope signal and its FFT.....	51
3.17 RTL for the LTP.....	52
3.18 RTL for the Clock Divider.....	53
3.19 RTL decoder schematic for the interface between the FPGA and the DSP.....	55
3.20 RTL schematic for the read enable for the ADC initialization.....	56
3.21 RTL logic for the write enable for the DAC1 initialization.....	57
3.22 RTL logic for the write enable for the DAC2 initialization.....	57

LIST OF FIGURES
(Continued)

Figure	Page
3.23 RTL schematic for the retrieval of sampled data from the ADC.....	58
3.24 Top level schematic of the system on the FPGA.....	59
4.1 Sawbone plate with its dimensions with the edges.....	62
4.2 Received signal and its envelope signal observed on the oscilloscope with symmetric placement of the transducers.....	63
4.3 Received signal and its envelope signal observed on the oscilloscope with the asymmetric placement of transducers.....	64
C.1 Output on the oscilloscope showing the step size of 3.10 μ s.....	71
C.2 Output on the oscilloscope showing the step size of 2.50 μ s.....	72
C.3 Output on the oscilloscope showing the step size of 1.66 μ s.....	72
C.4 Output on the oscilloscope showing error for any sampling period above 0.5 μ s (2MHz).....	73

CHAPTER 1

INTRODUCTION

1.1 Overview

Ultrasound has been widely used in applications for the detection and location of the cracks and fracture on various materials i.e. bone, composite structures, and metals etc.[2-4]. Other detection methods are based on the monitoring of the nonlinear elastic material behavior of damaged material [1], guided waveguides [2], magnetic leakage flux [2] and laser based techniques [3] . This thesis applies the low transient pulse method [4] for the determination of the crack in the Sawbone which is a synthetic composite material designed to emulate the acoustic properties of the cortical bone.

The recent generation of 3-D X-ray bone densitometry and peripheral quantitative CT (pQCT) has made major advances. However, they are expensive and emit ionizing radiation. Therefore it is of significant interest to explore quantitative ultrasound (QUS) because ultrasound is also highly sensitive to elastic properties and defects of bone materials. In addition, ultrasound is safe and easy to deploy so that ultrasonic equipment can be made portable and relatively inexpensive [2]. Ultrasonic methods have also been used for monitoring bone fracture healing and creep in structural materials [5-8]. Although some researchers have employed ultrasonography and power Doppler ultrasonography to assess the appearance and neo-vascularization of the callus tissue during healing, the majority of the studies have utilized quantitative ultrasound techniques [5]. Ultrasonic monitoring of bone healing is discussed in detail in [5]. Many

methods have been employed for the detection of edge cracks like the Frobenius method to enable possible detection of location of the crack based on the measurement of natural frequencies [9], the wavelet theory, eddy current and many more for various surfaces [10-12].

There are various benefits of using ultrasound for Non Destruction Testing (NDT). This work addresses the use of the low transient acoustic pulses to enhance system resolution and to simplify hardware requirements. The Low Transient Pulse (LTP), as opposed to the conventional acoustic pulse, has shorter pulse duration and thus results in less phase interference and higher performance. To produce low transient acoustic pulses, it is often necessary to have a prescribed signal to drive the transmitter [4]. Nondestructive evaluation (NDE) is a term that is often used interchangeably with NDT [13]. However, technically, NDE is used to describe measurements that are more quantitative in nature. For example, an NDE method would not only locate a defect, but it would also be used to measure something about that defect such as its size, shape, and orientation. NDE may be used to determine material properties, such as fracture toughness, formability, and other physical characteristics. In ultrasonic testing, high-frequency sound waves are transmitted into a material to detect imperfections or to locate changes in material properties [13].

Field-Programmable Gate Arrays/Digital Signal Processor (FPGA/DSP) environment has been used so far for various designs like Predictive Current Control of Voltage-Source Inverters, AC-voltage regulation, neural network controller so on and so forth [14-16]. The purpose of using this setup varies, from unloading the DSP processor by performing certain part in the hardware FPGA [14] to that design which cannot be

reached using a digital signal processor (DSP) [16]. Also while some are used for cost effective implementation [15] others are mainly used for performing high frequency computation on FPGA and data processing on DSP [17]. In other words resources (FPGA/DSP) partitioning of the architecture is based upon the functionality and speed requirements [18].

FPGAs have become an extremely popular implementation technology for custom hardware because they offer a combination of low cost and very fast turnaround, but the use of FPGAs is hampered by the very large overhead of FPGA-based architectures [19]. This is the reason we are using the two technologies for their best use in our work. While we will perform the high frequency (62.5 MHz) computing with FPGA all lower frequency (>1MHz) data extensive processing will be done within the DSP. The flexibility of FPGAs provides multiprocessor and open architecture interface [20]. This flexibility helps us to interface the DSP in our system with the external components like the ADC (Analog to Digital Converter) and the DAC (Digital to Analog Converter) through its EMIF bus for high speed data communication between them.

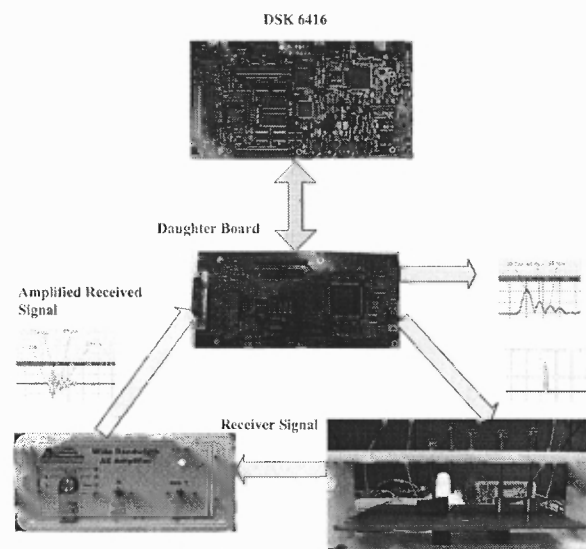


Figure 1.1 Experimental ultrasonic crack detection system

1.2 Hardware Architecture

The experimental test system shown in Figure 1.1 is implemented with DSP and FPGA where, the DSP carries out the analysis of the received signal at 1MHz while the FPGA operates at 62.5MHz for the generation of LTP signal and to retrieve the envelope of the received signal. The software for DSP is written in C while that for FPGA is written in VHDL. The FPGA executes front end processing tasks at a higher rate while the DSP analyzes data of the received baseband signal from the FPGA at a much lower rate.

As shown in Figure 1.1, the experimental hardware consists of Spectrum DSP's (TMS320C6416) DSP Starter Kit (DSK), Dalanco Spry's daughter board (FPGA, Spartan-IIe), Physical Acoustic Corporation's wide bandwidth AE amplifier and a set of ultrasonic transducer. Figure 1.2 shows the interface structure of the DSK and the FPGA and the task allocation among the two processors. The test plate is shown in lower right hand side of Figure.1.1 where a set of two Physical Acoustic 150 KHz ultrasonic transducers are mounted on the test material.

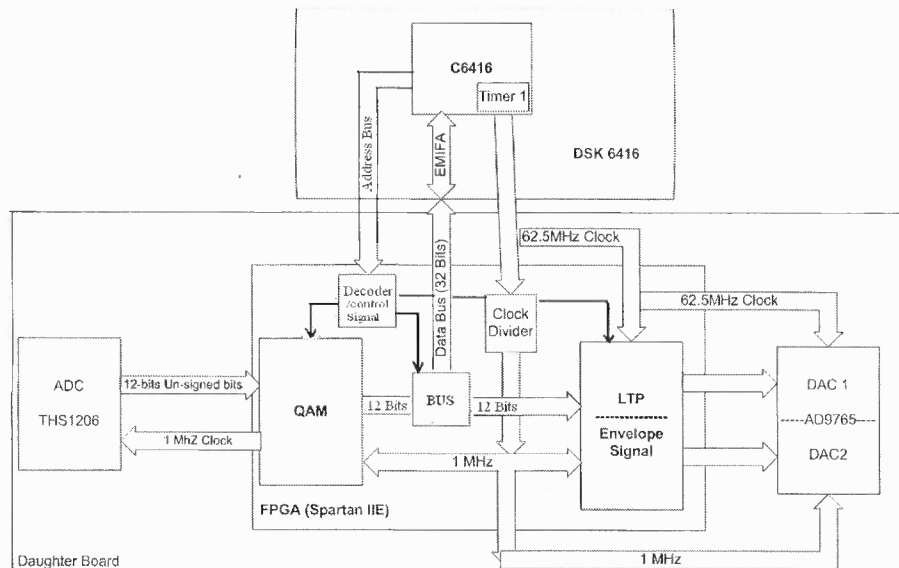


Figure 1.2 Block diagram to show the interface between the DSP and FPGA

1.3 DSP Starter Kit

The C6416 DSK is a low-cost standalone development platform that serves as a hardware reference design for the TMS320C6416 DSP. Figure 1.3 shows the schematic diagram of the DSK board with the on board components and interfaces. The C6416 DSP device operates at 1GHz clock rate, with information processing rate of about 8800 MIPS (nearly 9 billion instructions per second), three flexible multi-channel buffered serial ports and the internal DMA engine which can provide over 2Gbps of I/O bandwidth with 64 independent channels, a high performance fixed point processor [21]. The sampling rates of the ADC and DAC codec converters and the hardware interrupt are functions of the timer frequency which is software controlled. The AIC23 is designed for conventional audio processing and is not used in this experiment.

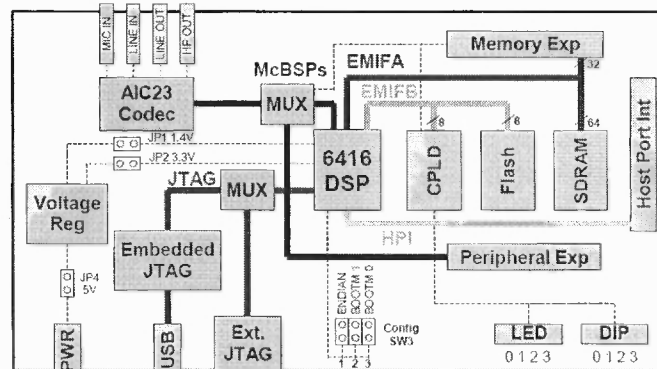


Figure 1.3 Block diagram for the on board components and their interfaces on the DSK
Source: TMS320C6416 DSK Technical Reference, 505945-0001 Rev. A April 2003

1.4 Daughter Board

Drive signal synthesis, receiver signal demodulation and interfaces between the DSK, ADC, DAC, etc. are carried out on the Dalanco Spry daughter board. The main components on the daughter board include: Spartan-IIIE FPGA, THS1206 ADC and ADC9765 DAC. The THS1206 is a 12-bit ADC which runs at 1MHz (clock from the

FPGA), has four channels with signal-to-noise ratio of 68 db [22]. The AD9765 is a dual-port, high speed, 2-channel, 12-bit CMOS DAC [23]. It integrates two high quality, 12-bit TxDAC+ cores, a voltage reference, digital interface circuitry into a small 48-lead LQFP and supports update rates up to 125 MSPS [23]. For the current system, AD9765 is running on two clocks viz. 62.5 MHz and 1MHz for two different channels. A VHDL and schematic mixed design entry is used to generate the configuration file for the FPGA. VHDL is a hardware description language for very high speed integrated circuits; it is used to describe electronic systems without any dependency on implementation [24].

The Spartan-III FPGA itself has 6,912 logical cells and 16 bits/LUT distributed RAM with a speed grade of -6. They are customized by loading configuration data into internal static memory cells [25, 26]. Unlimited reprogramming cycles are possible with this approach [25, 26]. Stored values in these cells determine logic functions and interconnections implemented in the FPGA [25, 26]. Configuration data can be read from an external serial PROM (master serial mode), or written into the FPGA in slave serial, slave parallel, or boundary scan modes [25, 26]. The configuration of the Spartan III FPGA determines the interaction of the ADC, DAC, and digital I/O with the local bus. The current experimental system utilizes 1,581 slices out of 3072 with a total gate count of 30,892 [25, 26]. Other benefits of the Daughter Board include:

- 1) Higher sampling rate: The C6416 DSK has an onboard CODEC (TLV320AIC23) that has predefined sampling rates. This CODEC has the maximum sampling frequency of 96KHz [22], which is significantly less than the required sampling frequency of 1MHz for the ADC and 62.5MHz required for the DAC (Low Transient Pulse Generation). The

Daughter Board's ADC and DAC operate in the MHz range and are adequate for this application.

2) No DC blocking: The C6416 DSK CODEC blocks DC and hence cannot be used for motion control. The Daughter Board's analog interface, on the other hand, operates from DC to 125MHz.

1.5 Low Transient Pulse

The Low transient pulse (LTP) technology is an innovative technique [4] to produce a short duration and compact acoustic pulse by means of pre-shaping the excitation signal. It has been experimentally verified that the LTP method produces a better measurement resolution and simpler hardware implementation due to less phase interference and a less complex algorithm. No modulation circuits or regenerative loops are necessary to synthesize the drive signal. Within the quantitative ultrasonography context, the LTP method improves detection resolution by minimizing aliasing of signals transmitted from soft and hard tissues. This improvement is achieved by convolving the input pulse with the impulse sequence to generate a necessary drive signal. Figure 1.4 below shows the pulse-impulse convolution scheme.

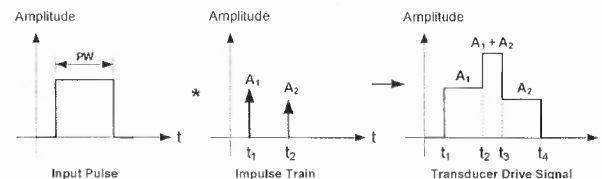


Figure 1.4 Low Transient Pulses. Detailed descriptions of the LTP parameters are given in [4].

Source: B. Cheng and T. Chang, "Enhancing ultrasonic imaging with low transient pulse shaping," *Ultrasonics, Ferroelectrics and Frequency Control*, IEEE Transactions on, vol. 54, pp. 627-635, 2007.

A typical ultrasonic transducer can be approximated by a second order underdamped transfer function as:

$$G(s) = \frac{\alpha s}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

The design parameters for a two-impulse low transient pulse shaper can be summarized as follows [4]:

$$A_1 = \frac{1}{1 + M_p^n}, \quad t_1 = 0$$

$$A_2 = \frac{M_p^n}{1 + M_p^n},$$

$$t_2 = \frac{n\pi}{\omega_n \sqrt{1 - \zeta^2}} = \frac{n\pi}{\omega_d}, \quad M_p^n = \left(e^{-\frac{\zeta\pi}{\sqrt{1 - \zeta^2}}} \right)^n, \quad n = 1, 3, 5, \dots$$

where A_1 and A_2 are the impulse amplitudes that occur at time t_1 and t_2 , respectively

while $M_p^n = \left(e^{-\frac{\zeta\pi}{\sqrt{1 - \zeta^2}}} \right)^n$ the variable n determines the oscillation cycles of the LTP acoustic signal.

1.6 Benefits

1.6.1 TMS320C6416T

- C6416 is the world's first 90nm DSP running at 1GHz. Moving transistors closer together has increased the speed of the operation, as well as it allows a higher density of on-chip memory to increase application efficiency [27]. This 90 nm process technology has eased integration of system-on-a-chip architectures by streamlining communication between the DSP, memory, peripherals, RISC processors and analog components [27].
- There are three main vendors of single DSP core processor chips: Analog Devices, Freescale, and Texas Instruments. Table 1.1 [28] compares these DSPs.

Table 1.1 Comparison of various DSP processors in market

Chip Family (Development Boards)	Price Range	Data Format	Clock speed/core	Primary Competitors	Primary Application Telecom, Infrastructure ,automotive and video Military Imaging and Infrastructure Audio Applications Motor Control, Digital Power supply Control (Automobile, Industrial) and digital power Portable Audio, Consumer Electronics like Mobiles Telecom, Infrastructure and video Automobile and consumer Electronics Audio systems
ADI Black Fin (ADSP-BF518F EZ-BOARD)	\$325	32bit fixed	400 MHz	C '55x and C'64x	
ADI Tiger Shark (ADZS-TS201S-EZLITE)	\$1,000	32bit float	500 MHz	C'67x and ADISharc C'67x and ADITiger Shark	
ADI Sharc (ADSP-21371 EZ-KIT Lite)	\$495	32bit float	266 MHz		
Free Scale '5685x (EVM100)	\$149	24Bit fixed	120 MHz	C '64 and TI C28x/24x	
TI C28x/24x (eZdsp™ F2812)	\$325	32/16bit Fixed	60/40 MHz	Freescale '5685	
TI 'C55x (C55x DSK)	\$515	16bit Fixed	200 MHz	ADI Black Fin	
TI 'C6416T (C64x DSK)	\$515	32bit Fixed	1GHz	ADI Black Fin	
ADSP-BF5xx(ADZS-BF561-EZLITE)	\$500	16bit Fixed	600 MHz	TI' C55x	
TI '67 (C67x DSK)	\$414	32bit Float	720 MHz	SHARC, CPUs	

Source: BDTI 2009; Datasheet of TMS320C6414T, TMS320C6415T, TMS320C6416T Fixed-Point Digital Signal Processors, SPRS226M, 2009; Datasheet of ADSP-BF512/BF512F ; Datasheet of ADSP-TS201S; Datasheet of ADSP-21371; 56F826 Evaluation Module Hardware User's, Freescale Manual; Technical reference eZdsp™ F2812, Spectrum Digital ; Datasheet of TMS320LC549-80; Datasheet of ADSP-BF561; Datasheet of TMS320C6713

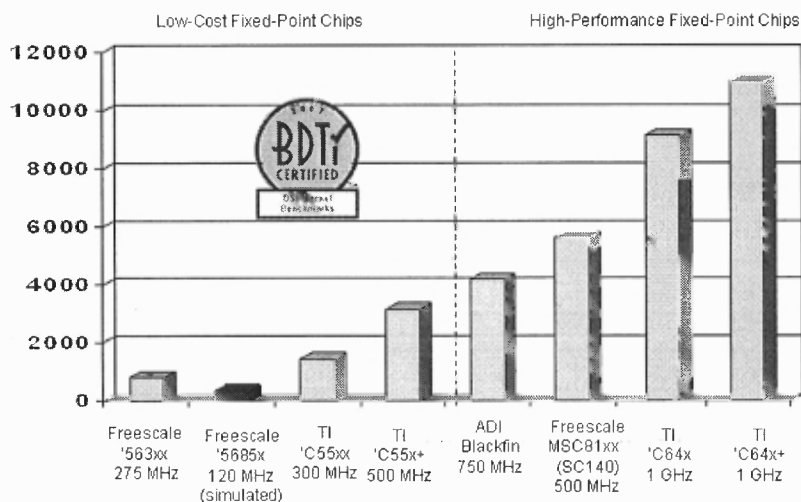


Figure 1.5 Millions Instructions per second for various processors.

Source: BDTI 2009; Datasheet of TMS320C6414T, TMS320C6415T, TMS320C6416T Fixed-Point Digital Signal Processors, SPRS226M, 2009; Datasheet of ADSP-BF512/BF512F ; Datasheet of ADSP-TS201S; Datasheet of ADSP-21371; 56F826 Evaluation Module Hardware User's, Freescale Manual; Technical reference eZdsp™ F2812, Spectrum Digital ; Datasheet of TMS320LC549-80; Datasheet of ADSP-BF561; Datasheet of TMS320C6713

Based on the above data we use TMS320C6416T chiefly for the following reasons

- C6416T running at 1GHz can execute close to approximately 10 billion instructions per second. This can facilitate in performing exhaustive data crunching of the demodulated data. RTDX which uses lot of instruction cycles for its execution for the communication between the host and the DSP can be used to display the data on the host PC in real time
- Since C6416T is a fixed point DSP, its power consumption is low hence this feature can be exploited for the future design of a hand held ultrasound device.
- Most prominently, as per our design requirement, C6416T can generate a maximum clock frequency (timer) of 62.5 MHz (16 ns) which helps in the enhanced resolution of the Low Transient Pulse (LTP) generation. For the present application, LTP width is about 3.75 μ s. Although we do get a quantization error of $(16 \times 235 = 3750 \mu\text{s})$, $3760 - 3750 = 10\text{ns}$, this error is within experimental tolerances.

1.6.2 Spartan-IIE, (XC2S300E)

- Spartan-IIE FPGAs are typically used in high-volume applications where the flexibility of a fast programmable solution adds benefits and are ideal for shortening product development cycles while offering a cost-effective solution for high volume production [26].
- It offers on-chip synchronous single-port and dual-port RAM (block and distributed form), DLL clock drivers, programmable set and reset on all flip-flops, fast carry logic, and many other features [26].
- It contains an optimal number of gates for the design, hence extremely efficient implementation for cost/transistor. It also readily interfaces, without any glitches, with the DSP, ADC and DAC with multiple clocks.
- The design of the daughter board permits programming of the FPGA through the DSP's Code Composer Studio® (CCS) and hence saves time from JTAG emulation separately for the FPGA. The process of programming through the DSP is explained in Chapter 2.

1.7 RTDX (Real Time Data Exchange)

Real-Time Data Exchange (RTDX) is a technology developed by Texas Instruments that enables real-time bi-directional communication between a digital signal processor (DSP) or microcontroller and a host application that provides real-time, continuous visibility into the way DSP applications operate in the real world [29, 30]. This property of C6416T allows us to take our design to another level. By the use of RTDX we can transfer the envelope signal received in the DSP to the host computer in real time. Since

RTDX is bi-directional, we are not restricted to receive the transducer signals but can send data back into the DSP. This allows us to perform real time analysis using tools like Matlab. Matlab's Embedded IDE Link™ CC is the tool which helps to interface the host PC with the DSP and communicate in real time. It automates debugging, project generation, and verification of object code executing on an embedded processor or on the instruction set simulator provided by the integrated development environment (IDE) [31].

CHAPTER 2

TMS320C6416: TI's DSP PROCESSOR

2.1 Digital Signal Processor

In the design, C6416DSK functions as the central processing (Edge Detection) and control unit (Initialization of ADC and FPGA, Clock Source, Interface between FPGA/DSP) of the system. From executing the analysis of the received signal to controlling all the peripherals components, C6416 acts as the key component of the system. A brief review of the C6416 system is given in the following section based on materials [21, 27-30, 32-37]

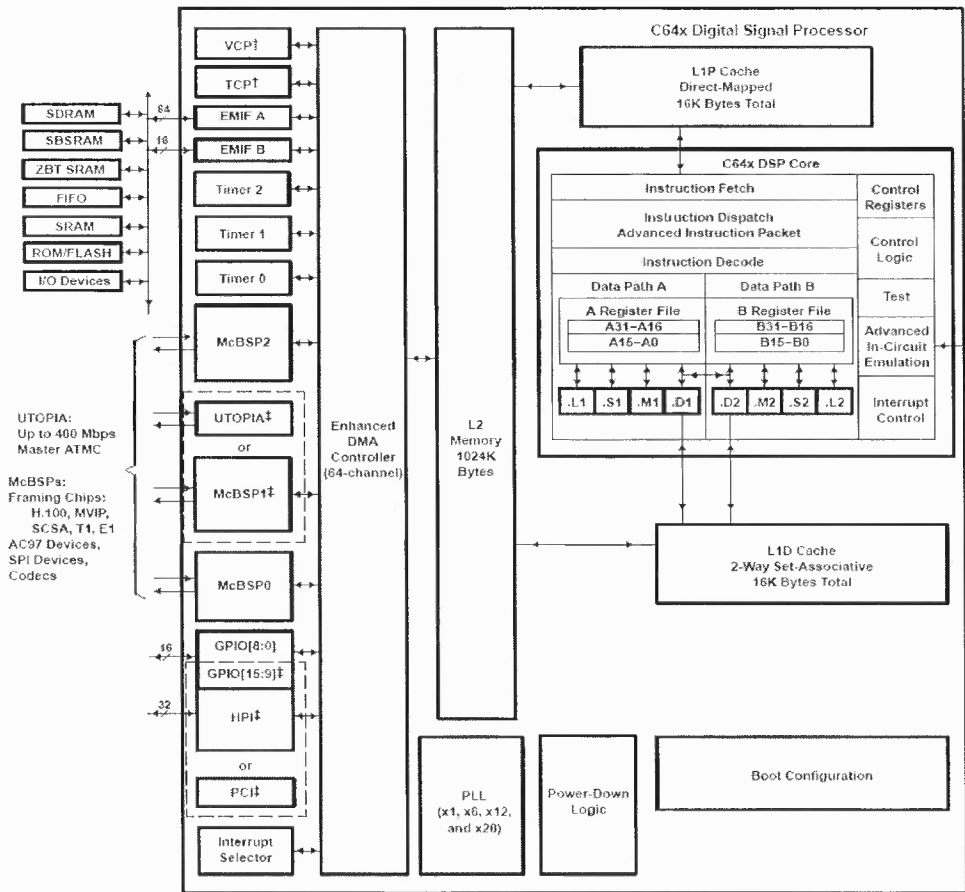


Figure 2.1 Architectural overview of TMS320C6416 DSP

Source: Datasheet of TMS320C6414T, TMS320C6415T, TMS320C6416T Fixed-Point Digital Signal Processors, SPRS226M

Figure 2.1 shows the functional unit of the C6416 and its key feature [37] are as shown below:

- Highest-Performance Fixed-Point DSPs , 1 GHz instruction cycles, 8000 MIPS
- VelociTI.2™ Extensions to VelociTI™ Advanced Very-Long-Instruction-Word(VLIW) TMS320C64x™ DSP Core
- Two External Memory Interfaces (EMIFs)
- Enhanced Direct-Memory-Access (EDMA) Controller (64 Independent Channels).
- L1/L2 Memory Architecture: 128K-Bit (16K-Byte) L1P Program Cache (Direct Mapped), 128K-Bit (16K-Byte) L1D Data Cache (2-Way Set-Associative) 8M-Bit (1024K-Byte) L2 Unified Mapped RAM/Cache (Flexible Allocation).
- Three 32-Bit General-Purpose Timers

2.2 Memory Map

The memory map of C6416 is as shown in Figure 2.2, where by default, the internal memory sits at the beginning of the address space and portions of memory can be remapped in software as L2 cache rather than fixed RAM [21].

Each EMIF (External Memory Interface) has 4 separate addressable regions called chip enable spaces (CE0-CE3) where the SDRAM occupies CE0 of EMIFA, CPLD and Flash are mapped to CE0 and CE1 of EMIFB respectively and the Daughtercards use CE2 and CE3 of EMIFA [21]. In the design, EMIFA is used to connect to the Dalanco Spry daughter board. CE2 (0xa0000000) among two chip selects (CE2 and CE3) is used for the interface with the daughter board hence with the FPGA and other components like the ADC and DAC on board.

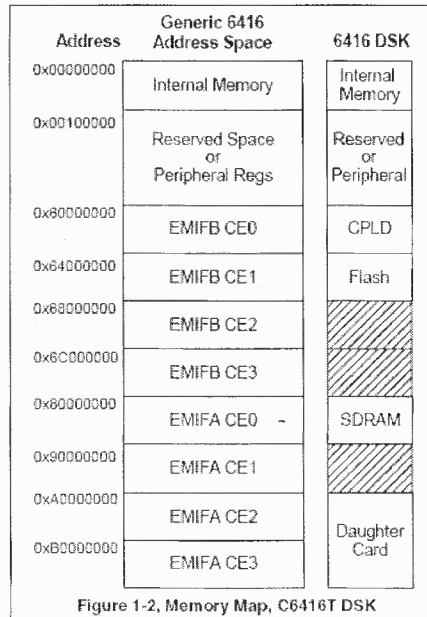


Figure 2.2 Memory Map of C6416T DSP Starter Kit (DSK)

Source: TMS320C6416 DSK Technical Reference, 505945-0001 Rev. A April 2003.

The DSK Board has two External Memory Interfaces (EMIF's)[36]:

- One 64-Bit (EMIFA) and a 16-Bit (EMIFB).
- Glueless interface to Asynchronous Memories (SRAM and EPROM) and
- Synchronous Memories (SDRAM, SBSRAM, ZBT SRAM and FIFO)
- 1280 MB Total addressable external memory space

2.3 Interrupts:

DSPs work in an environment that contains multiple external asynchronous events. These events require tasks to be performed by the DSP when they occur. Real-time data input/output handling by the C6416 includes:

- Interrupts -An interrupt stops the current CPU process so that it can perform a required task initiated by the interrupt whose source can be an ADC, a timer and so on, by redirecting the program flow to an ISR [38].

B. Polling - A polling-based program (non-interrupt driven) continuously polls or tests whether or not data are ready to be received or transmitted which makes it less efficient than the interrupt scheme [38].

- **Interrupt service Table**

A vectors file is used for the interrupt-driven program for the system. In the system designed interrupt is used to fetch the sampled data from the ADC to the DSP and send it back to the DAC. C6000 DSP Peripheral set has up to 32 interrupt sources however the CPU has 12 interrupts available for the use [32]. The interrupt service table (IST) is used when an interrupt begins, which has Fetch Packets (FP) associated with it [38]. There is an offset associated with each specific interrupt since each FP contains eight 32-bit instructions (256 bits) or 32 bytes, each offset address in the table is incremented by $(0x20)h = 32$ [38]. Below is the description of the vector.asm used for the system [38].

```

.ref    _adc_ISR
.ref    _c_int00
.sect  "vectors"

RESET_RST: mvkl  .S2    _c_int00,B0
           mvkh  .S2    _c_int00,B0
           B     .S2    B0
           NOP
           NOP
           NOP
           NOP
           NOP
           NOP
NMI_RST:
           NOP
           NOP
           .
           .
           .
           NOP
           .endloop
INT4:    mvkl  .S2    _adc_isr, B0
           mvkh  .S2    _adc_isr, B0

```

```

B .S2 B0
NOP
NOP
NOP
NOP
NOP
NOP
.....

```

- `.ref` command refers to the ISR naming used in the `main.c` program. `__adc_ISR` is the name of the ISR used for the interrupt from the ADC in C program. The function `_c_int00` is the name of the ISR which is the entry point in the BOOT program which sets up the stack and calls for `main` in the C program [38].
- `.sect` command assigns the name of this sector to be *vector* which is used to allocate it to the memory in the Linker Command (`.cmd`) file.
- All the NOPs (No operations performed) are for the FP (Fetch Packet) in the interrupt which are not used.
- In INT4 (Interrupt No.4), `mvkl.S2` moves (using the one of the eight functional unit of the CPU `.S2`) the 16 LSBs of address `__adc_isr` to 16 LSBs of `B0` and similarly `mvkh.S2` moves the rest of 16 MSBs of address `adc_isr` to `B0`. Hence `B0` contains the full 32 bit address of `adc_isr`. `B .S2 B0` then branch to the address in `B0`. For the other FPs still we don't use them we use NOPs [34, 38] . Refer to [38] Chapter 3 (3.8.2) for more information on Types of Instructions.

2.4 Timers

The timers of C6416 have two signaling modes and can be clocked by an internal or an external source with the input and output pins (TINP and TOUT) that can function as timer clock input and clock output, where an internal clock, for example the timer can signal an external ADC to start a conversion, or it can trigger the DMA controller to

begin a data transfer [33]. In the system designed, timer 1 is used to signal an external ADC, DAC and the FPGA. Since the generation of the LTP in the FPGA requires a pulse with of 3.75us, timer 1 is run at its maximum frequency of 62.5 MHz by setting up the Period to be '1' as per equation (2.1),from [33].

$$\text{Timer Frequency} = \text{CPUrate (1 GHz)} / (2 * \text{Period} * 8) \quad (2.1)$$

More detail about setting up the timer and its register can be found at [33] and Appendix C. Dividing the timer frequency (62.5MHz) for the clock of ADC (1MHz) and for other modules (like QAM) in FPGA is done in the FPGA itself which using clock diver as shown in later chapters. The system designed has timer 1 set as shown below.

```
void timer1_start()
{
    *(unsigned volatile int *)_TIMER_CTL1_ADDR &= 0xff3f; / hold the timer
    *(unsigned volatile int *)_TIMER_CTL1_ADDR |= 0x301; /0x211;
    *(unsigned volatile int *)_TIMER_PRD1_ADDR = 0x01; /set for 32 bit cnt
    *(unsigned volatile int *)_TIMER_CTL1_ADDR |= 0xC0; /start the timer
}
```

The first statement disables the counter and holds it at the current state. In the next, while the counter is disabled, we configure the timer by setting the TOUT pin to be output pin and enabling the clock mode and internal clock source (CPU clock/8). The period of the timer is set to '1'. The last statement initialize the counter register to zero and it starts counting from the next clock.

In the code at various locations we have used statements as shown below

```
*(short *) dacaddr1 = value

*(unsigned volatile int *)_TIMER_CTL1_ADDR &= 0xff3f
```

These statements are chiefly used for writing to peripherals (I/O) and setting external

registers. `(short*)dacaddr1` is the declaration of a pointer `dacaddr1` to type `short`. It just type cast `dacaddr1` to be a pointer of type `short` if not declared in the beginning of the program. Hence `*(short*)dacaddr1` is the value stored at address pointed by `dacaddr1`. Note that in the program `dacaddr1` has been defined with a hex value (address of the DAC1). So `*(short *) dacaddr1 = value` transfers the value to the external DAC1. Similarly for `*(unsigned volatile int *)_TIMER_CTL1_ADDR &= 0xff3`, `_TIMER_CTL1_ADDR` defines the address for the control register of timer1. `*(unsigned volatile int *)_TIMER_CTL1_ADDR` is the value stored at the control register and the complete statement mean ANDing it with the current value of the register.

2.5 Software Programming

Among the three different languages available for the coding of C6416 i.e. C, Linear Assembly and Assembly [38], we have used C in our design. Although the choice will have an impact on the programming and optimization efforts required but since our use of the DSP is straightforward and doesn't involve tight code integration, we have written our code in C. Also the compiler of CCS tool aids in optimization of the code into assembly very efficiently.

Careful consideration is given to the data type when writing the code. While the data communicated between the FPGA and the DSP and in the buffers for the moving average filter are declared to be `short` since there are only 12-bits of data, both retrieved from the ADC and sent to the DAC, `unsigned int` types are used for loop counters and ADC sampled data (`unsigned`) to avoid any unnecessary signed extension instruction [35].

After running the program in DSP from main, interrupt flags are set, the frontend program is loaded on to the FPGA and the ADC is initialized. The DSP then enters into an infinite while loop and waits for the hardware interrupt INT4 to occur (as defined in vector.asm). At each sampling time, the ADC module in the FPGA acquires the data from the THS1206 and also triggers an interrupt to the DSP to send this over. This interface (EMIF) between the FPGA and DSP has been explained in more detail in later chapters. The data which the DSP gets through the FPGA during the hardware interrupt is a 12-bit unsigned number and has a DC level of 0x800 or +2.7 V (More about resolution is given in Appendix A)

2.5.1 Project File (.pjt)

This is the main file and contains sub folders like Source (.c, .cmd and .asm), Library (.lib), Include (.h), DSP/BIOS Configure (.tcf) etc. Executing “Rebuild All” in the project tab of the CCS integrates all these files to .out file [35]. This file is loaded onto to the DSP for its execution.

2.5.2 Linker File (.cmd)

This file is used to map different section to the memory and it shows which section resides in which part of the memory [38]. The linker allocates the program in memory using default location algorithm and places the various sections into appropriate memory locations, where code and data reside [38]. By using a linker command file with extension .cmd, one can customize the allocation process, specifying MEMORY and SECTIONS directives within the linker command file [38]. The Linker file (.cmd) [39] used for the system is as shown below:

```

MEMORY
{
    vecs :          org=0x0          l=0x200
    ISRAM:          org = 0x200,    len=0x10000000
    SDRAM:          org=0x80500000,  len=0x4000000
    CE2:            org=0xA00000000000 , len=0x400000
}

```

```

SECTION
{
    "vectors" > vecs
    .text    >ISRAM
    .bss     >ISRAM
    .cinit   > ISRAM
    .const   > ISRAM
    .far     >ISRAM
    .stack   >ISRAM
    .cio     >ISRAM
    .systemem >ISRAM
    .mw_isrambuff >ISRAM
}

```

Linker file consists on two main parts:

2.5.2.1 Memory

This is used to declare all the memories (internal or external) or the interfaces associated with the DSP. These include the Internal SRAM (ISRAM), SDRAM (FPGA's .out file) and CE2 (EMIF, External Memory Interface). For each memory declared, the base address and size are specified. The memory *vecs* declared in the memory is used for the *vector.asm* file used for the interrupts. SDRAM declared in the memory is used to load the .out data for the FPGA. CE2 is used for the communication between the DSP and the FPGA through EMIF.

2.5.2.2 Section

This allocates the output sections into defined memory and designates the various code sections to available memory spaces[38]. All the mnemonics like *.text*, *.bss*, *.cint*, *.const*

used in the SECTION are called the Assembler Directives. An assembler directive is a message for the assembler which gets resolved during the assembling process and does not occupy memory space, since it does not produce executable code [38]. Other sections are assigned to the internal memory of the DSP. The *vectors* directive which is declared in the *vector.asm* file is assigned to *vecs* memory in the *.cmd* file. More information on different assembler directive and their use can be found in [40].

2.5.3 FPGA (.out) File

A *.hex* file is generated for the design using the Xilinx ISE. This file is converted to *.out* file using a converter provided by Dalanco Spry. In the present system, this file is loaded onto the FPGA through the DSP, so prior to running the program it is necessary to load this file onto the SDRAM which is mapped to DSP through the Chip Select 0 (CE0) by the Load option in the Code Composer Studio. This is then transferred to the FPGA as explained in section 2.5.4.2.

2.5.4 main.c Program

The basic *main.c* program provided by [39] is modified to a large extent for the design of the system as per the requirement in the FPGA-DSP environment .

2.5.4.1 Initialization

This process involves setting up the interrupt flags and initializing the ADC with values for CR0 and CR1. All the information for ADC and its setup used for the design is taken from here [22].

2.5.4.1.1 ADC (THS1206)

Figure 2.3 shows the configuration flow for the THS1206. The system designed doesn't

use the default configuration but configures the ADC as per the requirement instead.

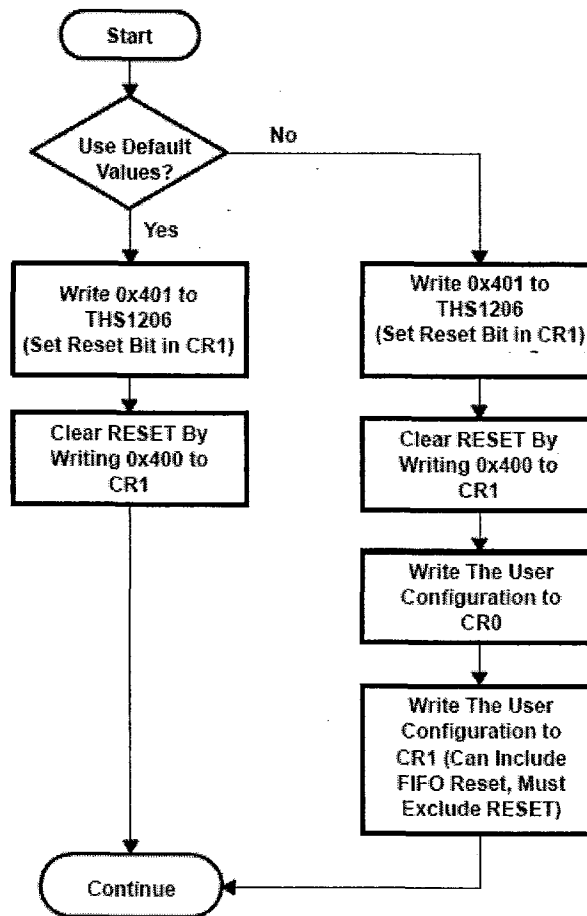


Figure 2.3 THS1206 Configuration Flow

Source: Datasheet of THS1206: 12-Bit, 6 MSPS, Simultaneous Sampling Analog-to-Digital Converter (Rev. H), Texas Instruments.

So as per the flow, 0x401 is first written to the ADC followed by 0x400. Then we set CRO. Since the ADC receives the upper 12-bits of the 16 bits register in the DSP, in the main.c we set the CRO to (0x 01A0)h which the ADC receives as (0x01A)h. This basically sets the ADC for the following from the LSB to the MSB

- Internal reference voltage selected
- Continuous conversion mode selected
- ADC made active

- Analog input BINP (ADC 4)

Then we set CR1. Same as for CR0 we set CR1 to be 0x4d20 which gets to ADC as 0x4d2. This configures the ADC for the following (LSB →MSB)

- No Reset of the ADC ‘0’
- In the write only mode to the ADC we reset the FIFO ‘1’
- Trigger level set to 0 for single channel ‘00’
- Trigger condition for DATA_AV set to Active low pulse ‘01’
- Since we are wiring to the ADC we set the bit to ‘1’
- Since we want the ADC to given an unsigned output we set this bit to ‘1’
- Normal conversion mode ‘00’
- To be set to ‘1’ by default for CR0.

For more information on the ADC and the registers refer to [22].

2.5.4.1.2 Flag Setup

Flags in the main.c file are set in the following order:

- CSR (Chip Select Register): This register is first set to a value of 0x100. This disables all the interrupts except the NMI and sets it to little endian.
- IER (Interrupt Enable Register): This register is first set to value 0x1. This resets the interrupt enable.
- ICR (Interrupt Clear Register): This register is first set to value 0xffff. This clears all the corresponding flags in the IFR (Interrupt Flag Register).

After IER, ICR and CSR are set we perform the process stated in section 2.5.4.1.2 and section 2.5.4.2 and just before going to the infinite while loop (section 2.5.4.3) we configure the flags again for the following.

- IER (Interrupt Enable Register): To set this register we OR its current value with 0x12. This enables all the nonreset interrupts and also enables interrupt No. 4 (non-NMI).
- CSR (Chip Select Register): To set this register we OR its current value with 0x1. This is for global interrupt enabler.

More information on interrupt registers can be found in [41].

2.5.4.1.3 Miscellaneous

EMIFA CE2 space control register (CECTL2, 0x 01800010) [36] is configured since we use this for our communication with the Daughter Board. This register is set with value 0x3c0cf20 for the following

- Zero number of clock cycles for AOE_N and ARE_N '00'
- Reserved '00'
- Sets EMIFA to 32 bit Asynchronous interface '010'
- Reserved '0'
- 15 clock cycles for the width of read strobe '001111'
- Reserved '11'
- 1 clock cycle for the read setup '0001'
- 0 clock cycles for write hole width and the hold time of AOE and ARE rising '00'
- 60 clock cycles for write strobe width.

Also, the program system sets the EXTPOL (External Interrupt Polarity Register, 0x19c0008) to value 0x1 which makes a high-to-low transition on an interrupt source recognized as an interrupt [32].

2.5.4.2 Loading the FPGA

The various critical variables declared with their hex values in the main.c program with their use are as shown in Table 2.1

Table 2.1 : Variables used in the FPGA program

Variable Name	Hex Value	Remark
VIRTEX_MEM	0x80500000	This is the base address of SDRAM where we load the .out file for the FPGA from the Code Composer Studio
VIRTEX_SIZE	0x393D8	This is the size of the memory of the FPGA (300K). Any data bigger than this wont be transferred to the FPGA
VIRTEX_ADDR	0xa0000000	This is the address line for sending the data to the FPGA
VIRTEX_WRCS	0xa0000004	Address for sending control bits for to the FPGA. The 2nd and the 1st bit is for Write (WR_N) and Chip Select(CS_N) bits respectively
VIRTEX_PROG	0xa0000008	Address for sending the progress control bits to the the FPGA.PROGHI and PROGLO are used to make it either high or low

A function AVR32_LoadVir () is used to load the data into the FPGA during its execution in the main program. The execution of this function is described below:

- First, all the control bits are put to high (disabled all). Then status of the control lines is checked by reading the bits on VIRTEX_STAT. If both the bits are zero, the execution goes to the next line, else it waits on the statement till both get zero.
- Second, in order to write data onto the FPGA, the active low Chip Select and Write bits are enabled by writing a 0x00 at address for VIRTEX_WRCS. The program, then checks and verifies to confirm that both the bits are '0'.
- DSP then starts transferring data from the SDRAM to the FPGA. For this to take place, the program goes to the address location VIRTEX_MEM to retrieve the 8 bits. The

data bus between the DSP and FPGA are mapped in a way that the bits are to be rearranged before they are sent e.g. the 1st of the 8 bit data from SDRAM goes to the 31st bit of the data bus on to the address location of VIRTEX_ADDR. This bits restructuring is performed as shown below.

```
d31 = ( 1 & inword ) ? BIT31 : 0;    // Virtex d0/din LSB
d23 = ( 2 & inword ) ? BIT23 : 0;    // Virtex d1
.
.
.
output = d31|d23|d22|d17|d11|d6|d5|d0;
```

The last statement for output concatenates each individual bit to form one byte to be transferred on to the FPGA. This transfer of data takes place for address VIRTEX_MEM + pointer. This pointer gets incremented after transfer of each 8 bits of data till it reaches the value equal to VIRTEX_SIZE i.e. memory size in the FPGA. Since VIRTEX_SIZE defined is little less than 300K we send 0's to the FPGA for approximately 20 times after the data from SDRAM to FPGA has been transferred.

- After the transfer is completed Chip select and Write enable are pulled back to high to stop any further erroneous transfer. These bits are then checked after they are pulled back to '11'. At the end AVR32_LoadVir function returns the control back to the main.

2.5.4.3 Free Run

Subsequent to the system being initialized and the data being loaded onto the FPGA, the program then enters an infinite while loop to wait for the interrupt from the ADC. When the interrupt occurs, the demodulated data for the received signal in the ADC are fetched into the DSP. Inside the DSP we execute two main functions.

2.5.4.3.1 Moving Average Filter

As observed on the oscilloscope, the envelope (demodulated) signal of the receiver has a high frequency noise component as shown in Figure 2.4(a). This is a problem for detecting the 3rd and the 4th maxima/peak during the estimation of plate parameter. To remove this high frequency noise we use a Moving Average Filter (MAF). For the design of this filter the demodulated data from the QAM is captured in the oscilloscope and is used as the input to the MAF filter first designed in Matlab. Simulated results show improved and desirable envelope with a MAF (window width=16) by removing all high frequency noise and giving observable peaks of the demodulated signal. This is implemented by constructing a FIFO in the DSP. In this, on each interrupt, the 12-bit input from the QAM is first converted to signed and then stored at the 1st location of , the data stored on the previous clock is shifted to 2nd , so on and so forth. The data on the 16th location is discarded. Then, all the signed values in the FIFO are added and later divided by 16 to give the desired MAF output. The improvement on the received data from the QAM is as shown in Figure 2.4(b).

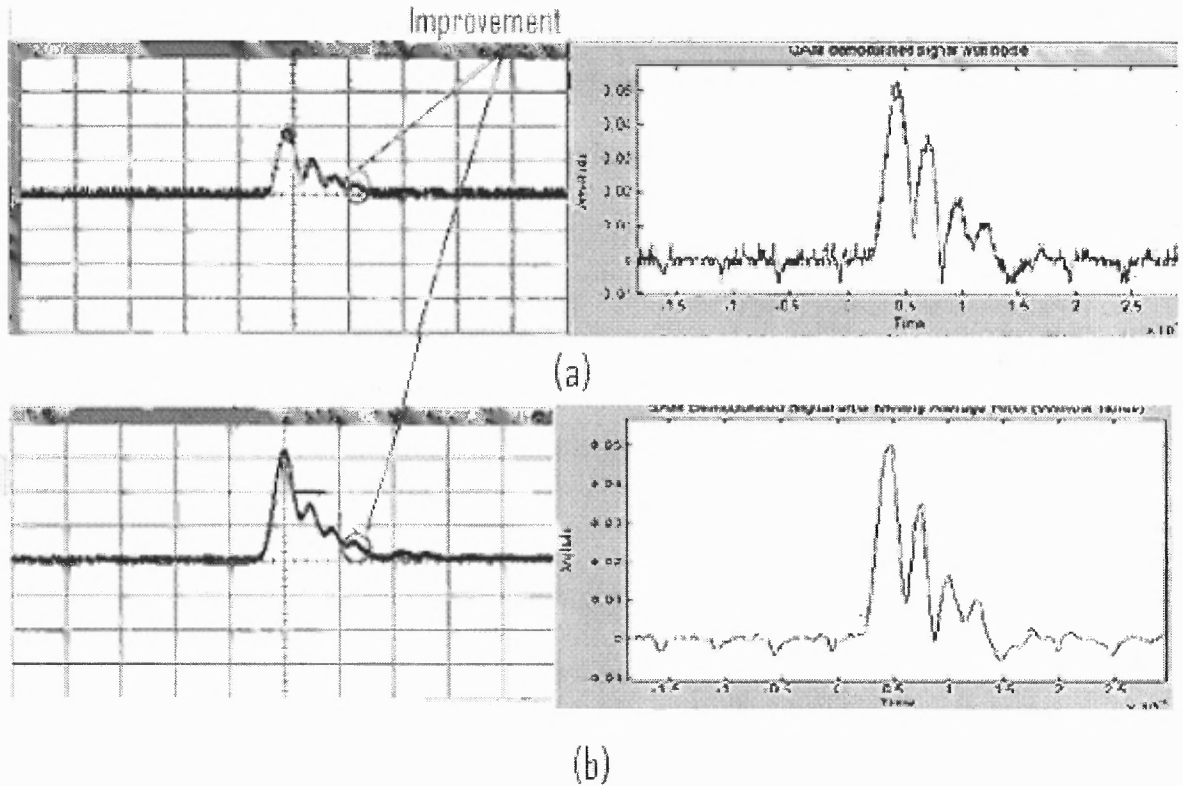


Figure 2.4 Envelop signal (a) before filtering and (b) after filtering.

2.5.4.3.2 Edge Detection

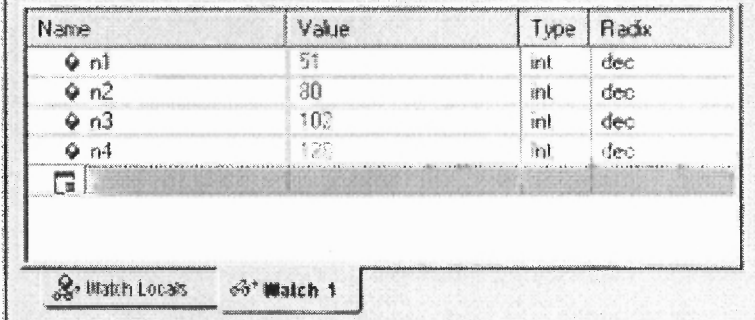
To detect the maxima or peaks from the QAM demodulated signal, an edge detection algorithm has been implemented to detect the maxima and then based on the time difference between the 1st peak and the subsequent peaks, it determines the parameter of the plate since the speed of ultrasounds in Sawbone is known. The captured, demodulated filtered data are first used to design and optimize an algorithm in Matlab which is later implemented on to the DSP.

For the detection of 1st maximum, we store the current value into a buffer from the QAM demodulator if it is greater than the previous value. Also, in the main program, a counter is incremented on each interrupt which starts from time $t=0$ and on detection of the 1st maximum, stores its value. This stored integer indexes the time for the first

maxima in μs .

For the 2nd maximum, two conditions are taken into account. Firstly, the current value should be smaller than the 1st maximum. Secondly, unlike for finding the 1st maximum, the last value stored should be less than the current value. The second condition is important because it prevents any false reporting of 2nd maximum from the 1st parabola as many points over it will be greater than the maxima from the 2nd parabola. Therefore we check for the rising curve. When both of the conditions are satisfied, the counter value during that interrupt is stored and displayed in the watch window of the CCS. For subsequent maxima, we follow the same approach as for 2nd maximum.

As shown in Figure 2.5, watch window displays the occurrence for the 1st, 2nd, 3rd and 4th maxima at time n1 (=51 μs), n2(=80 μs), n3(=102 μs) and n4 (=128 μs) respectively.



Name	Value	Type	Radix
n1	51	int	dec
n2	80	int	dec
n3	102	int	dec
n4	128	int	dec

Watch Locals Watch 1

Figure 2.5 Snapshot of the value displayed in the watch window of CCS

CHAPTER 3

SPARTAN 2E: FPGA

3.1 Hardware

A brief review of the Spartan-IIIE system architecture is given in this chapter. Details are given in [33, 34]. This is followed by a description of the QAM demodulation loop implementation.

The Spartan-IIIE family provides programmable I/O pins that support 19 commonly used I/O standards with system performance supported beyond 200 MHz [25, 42]. These standards include 16 single-ended standards and three differential standards and these I/O standards allow a single FPGA to directly interface with backplanes, memories, microprocessors, and other devices [25, 42]. It offers densities ranging from 93,000 to 300,000 system gates [25, 42] with functional elements that are interconnected by a hierarchy by routing channel, as shown in Table 3.1. The number of available I/Os for Spartan 2E are shown in Table 3.2.

Table 3.1 System Information

Device	Logic Cells	Typical System Gate Range (Logic and RAM)	CLB Array (R x C)	Total CLBs	Maximum Available User I/O	Maximum Differential I/O Pairs	Distributed RAM Bits	Block RAM Bits
XC2S300E	6,912	93,000 - 300,000	32 x 48	1,536	329	120	98,304	64K

Source: Datasheet of Spartan-IIIE 1.8V FPGA Family: Introduction and Ordering Information: Xilinx, Inc., 2001.

Table 3.2 Available IOs with various Spartan 2E packages

Device	Maximum User I/O	Available User I/O According to Package Type			
		TQ144	PQ208	FT256	FG456
XC2S300E	329	-	146	182	329

Source: Datasheet of Spartan-IIIE 1.8V FPGA Family: Introduction and Ordering Information: Xilinx, Inc., 2001.

3.2 SYSTEM DESIGN

3.2.1 Programming

The programming development is divided into 5 steps:

- 1 The Quadrature Amplitude Modulation-demodulation (QAM) architecture is first examined and optimized with a high level simulation of the complete system including the transfer functions of the ultrasonic transducers in SIMULINK using floating point accuracy [43]. The transfer function is estimated from the datasheet of the ultrasonic transducers. Based on the required output from the QAM, the filter parameters are changed.
- 2 Then different sub-blocks (IIR Filter, square root and sine/cosine generator) of the baseband processor are then translated into Matlab for hardware implementation where only fixed point arithmetic operations are used [43].
- 3 Step 2 serves two purposes: first it validates the VHDL program from the simulator (Modelsim®) output and second it verifies the experimental hardware output. Additional validation in the analog domain will be performed with a spectrum analyzer at the system [43].
- 4 A maximum detection algorithm is designed to operate on the captured and optimized envelope data in the memory of the DSP to calculate the distance of receiver from the edge of the sawbones.
- 5 This is then designed and implemented on the DSP with calibration due to noise from the received signal.

Steps 1-3 are also followed for the generation of the LTP signal.

3.2.3 Simulation Analysis

The test system implemented in Matlab is shown in Figure 3.1a, where the white blocks implement the Transducer Transfer Functions (TTFs) while the grey blocks implement the Quadrature Amplitude Modulation-demodulation (QAM). The signal from the TTF blocks are then synchronously multiplied with the sine and cosine values (quadrature) followed by a Butterworth 2nd order IIR filter, squarer (multiplier), adder and a square root calculator. Since the carrier frequency is 150 KHz, the cut-off frequency of the filters is set at 75 KHz to cut out the carrier frequency as well as any other noise. The input signal, receiver signal and envelope signal are shown in Figure 3.1b.

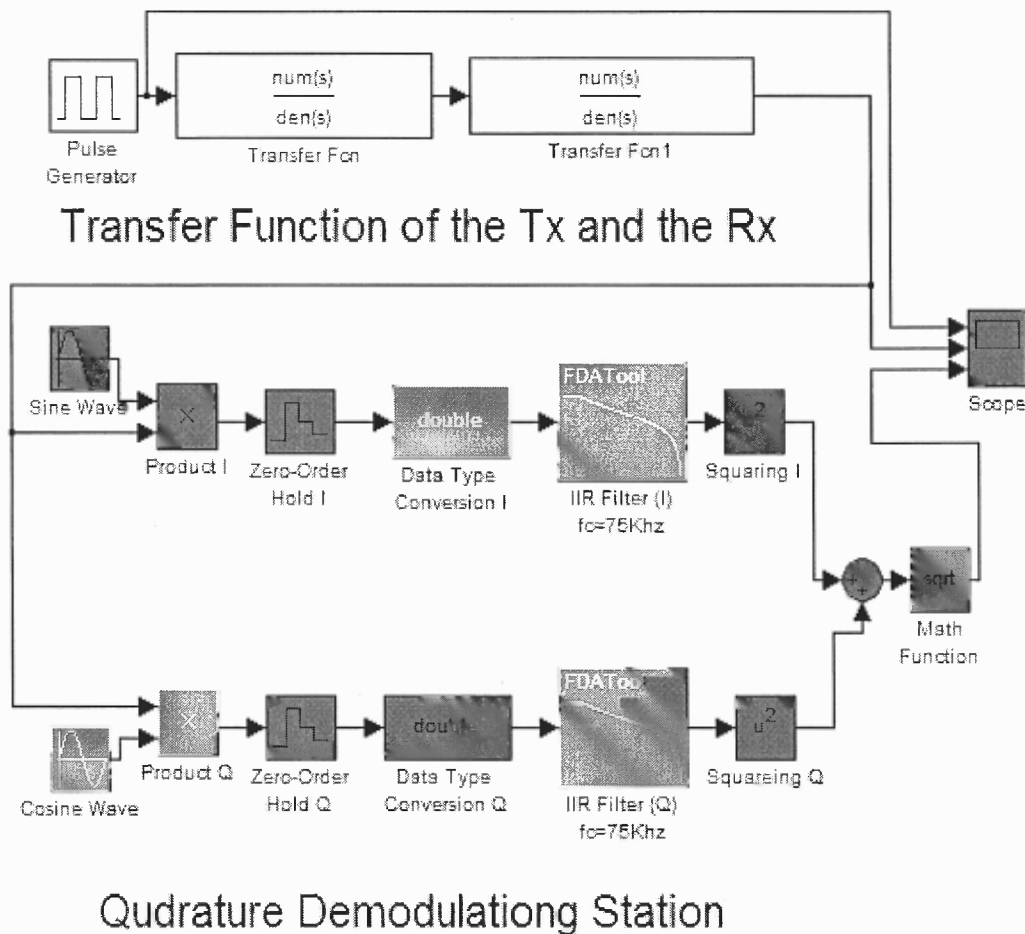


Figure 3.1a Matlab/Simulink model for the QAM system which is implemented on the FPGA.

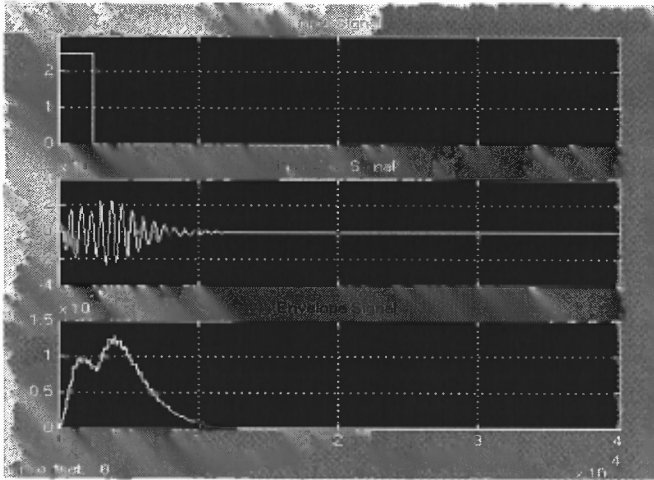


Figure 3.1b Matlab/Simulink simulation results for the QAM system.

Parameters of the transducers applied in Matlab are as shown below:

$$G(s) = \frac{\alpha s}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

Where

$$\zeta = 0.0858;$$

$$\alpha = 5000;$$

$$\omega_n = 8.3939 \times 10^5;$$

3.3 Programming the FPGA

The FPGA is programmed using VHDL in Xilinx ISE. Medium Scale Integration (MSI) approach is used for the implementation of the QAM, LTP and other interfaces i.e. the DSK, ADC, DAC etc. The design flow for the FPGA is shown in Figure 3.2

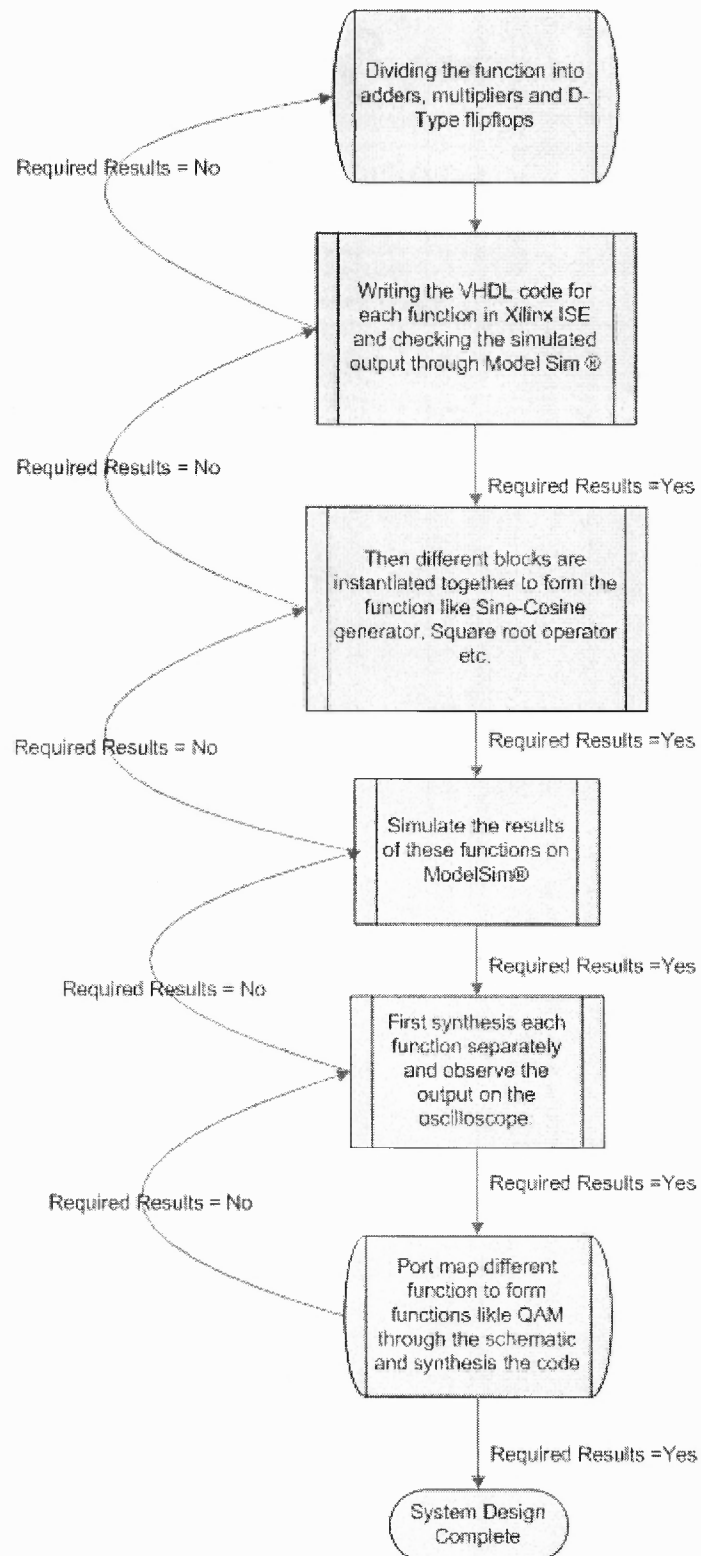


Figure 3.2 Design flow for FPGA

Source: X. Inc., ISE 9.1 In-Depth Tutorial: Xilinx Inc., 2007 ; D. L. Perry, VHDL: Programming by Example: McGraw-Hill.

3.3.1 Quadrature Amplitude Modulator-demodulator (QAM)

Each Matlab sub-block as shown in Figure 3.1a is implemented exclusively and later port mapped into the schematic. The QAM basic blocks consist of Sine and Cosine generator, IIR Filter, Multiplier and the Square Root.

3.3.1.1 Sine and Cosine Generator

The sine and cosine generators are realized by constructing a circular buffer with fixed point values from Matlab for a sine and cosine wave at 150 KHz sampled at 1MHz. At each clock pulse the buffer gives an output and increments the pointer. Mathematically, after three periods ($(1\text{MHz}/150\text{KHz}) \cdot 3 = 20$ samples) or 20 samples, the values will repeat from 1st period. Therefore, in the VHDL program a circular buffer is designed with a length of 20 (20 patterns of data) and on each clock it increments the pointer to the next value and outputs it from the generator. It is determined that a 6 bit data length is adequate for the present application. The overall buffer structure is shown in Figure 3.3

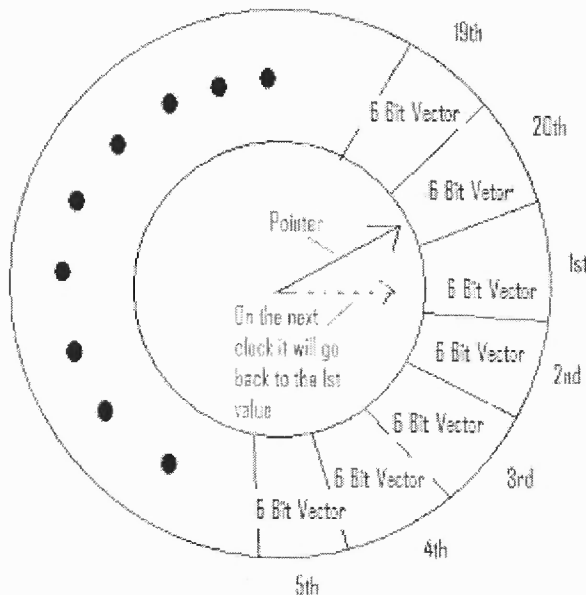


Figure 3.3 Schematic presenting the design of the sine and cosine generator

Since there are 6 bits among which the MSB is the signed, the remaining 5 bits gives sine wave with amplitude +/- 31 (011111-100001). The Matlab program below is for the generation of data which are plotted in Figure 3.4.

```
%Matlab Program for the generation of the array
%f=150 KHz
%fs=1 MHz
for i=1:20
```

```
y(i)=31*sin(2*pi*(150000/1000000)*i);
end
y=round(y);
```

```
25,29,10,-18, -31,-18,10,29,25,0,-25,-29,-10,18,31,18,-10,-29,-25,0
```

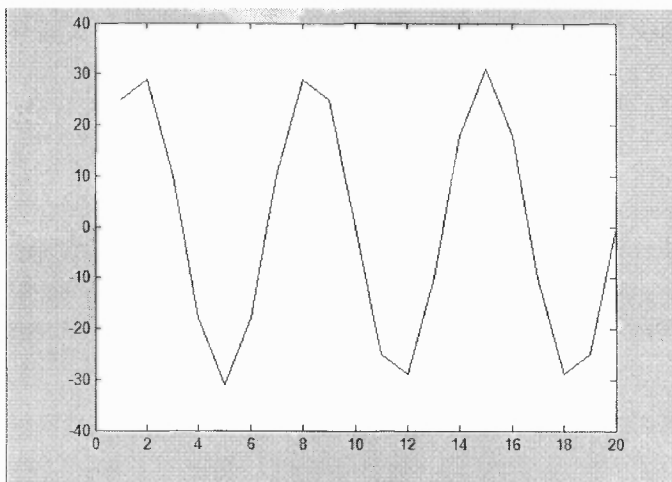


Figure 3.4 Matlab simulation result of the data produced by the sine wave generator.

3.3.1.1.1 Simulation Results

After converting the above data from decimal to 6 bits binary (signed), VHDL is used to design a circular buffer which outputs a value at each clock cycle. The same procedure applies to the cosine table as well. As shown in Figure 3.5a the system is initialized (reset) which gives a *sineout* of 000000. The *sineout* from the next clock starts from the first value in the Look Up Table (LUT) which is 25 (011001). The pointer gets

incremented at each clock till it arrives at the last value (20th) on the table which is 0 (000000), subsequently the pointer moves back to the first value of 25 as shown in Figure 3.5b in a manner consistent with a circular buffer.

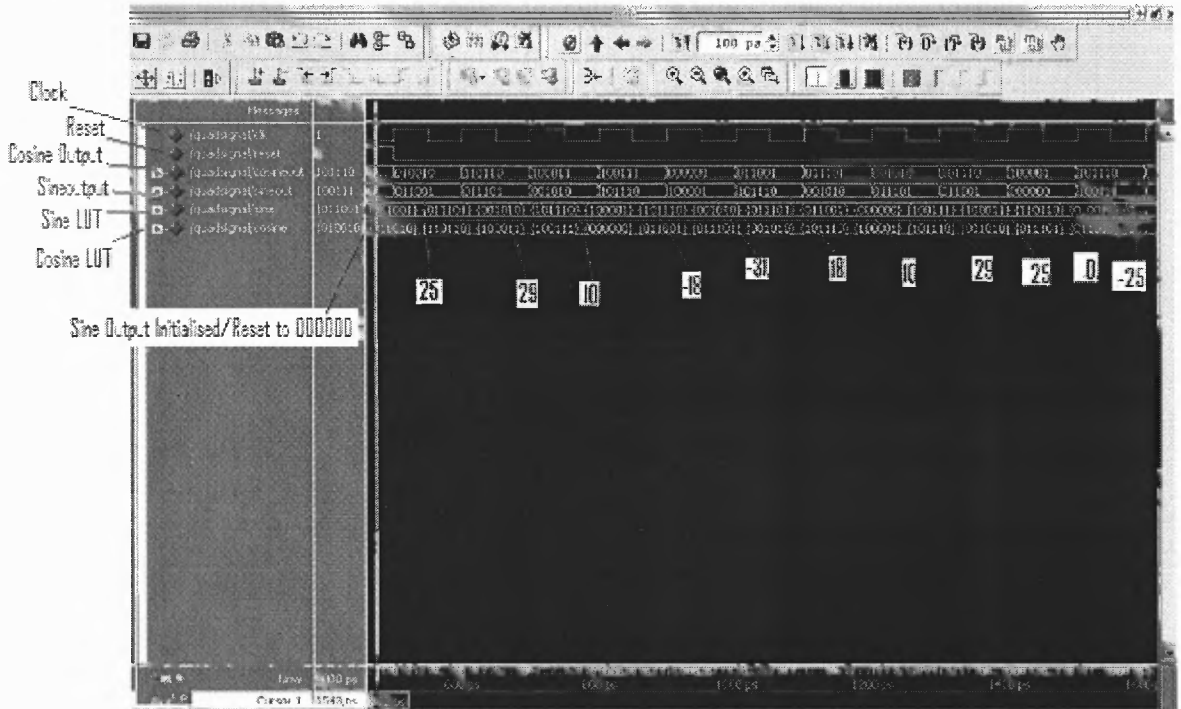


Figure 3.5a Simulation results for the sine and cosine generator for i= 1 to i=11

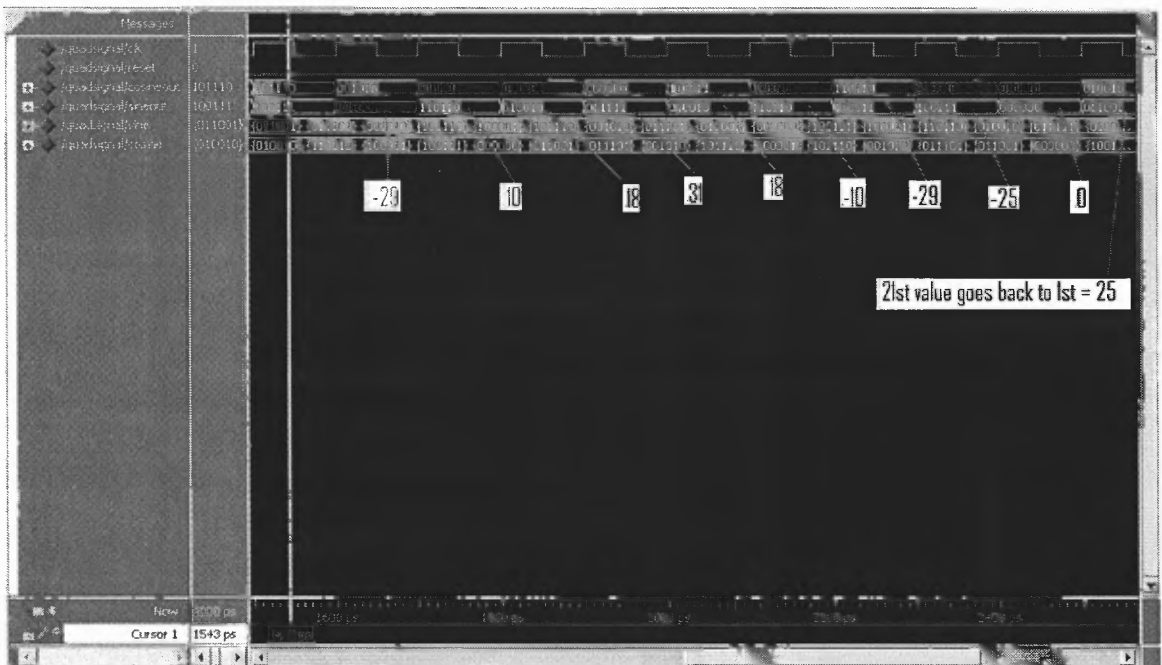


Figure 3.5b Simulation results for the sine and cosine generator for i=11 to i= 21

3.3.1.1.2 Register Transistor Logic (RTL)

The RTL generated for the VHDL code is shown in Figure 3.6.

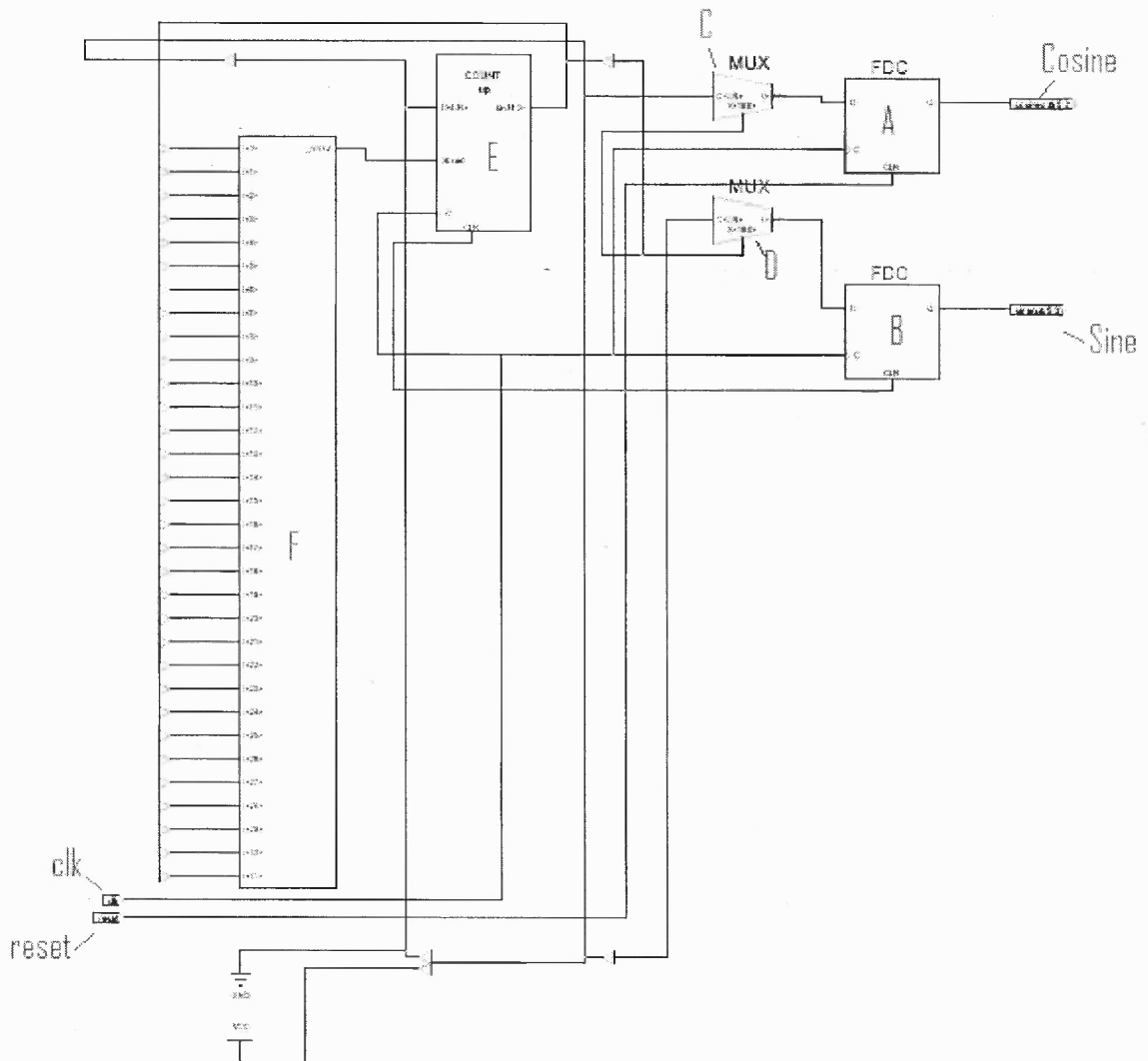


Figure 3.6 RTL for the sine and cosine generator.

The blocks consist of two multiplexers, one counter and two D-flip flops. At each clock (clk) cycle, the counter (E, Pointer) gets incremented to the next integer value (F) which is used to select the corresponding value from the LUT (multiplexer, C and D). In other words, the selection lines of the multiplexers (A and B) driven by the counter (E) are used to select one of the 20 input lines. This value then comes as an output on each rising edge of the clock (D-Flip Flop, A and B) from the sine and cosine output ports. Also, the

RTL logic has an asynchronous reset which is used to initialize the counter and clear the D- Flip flops. The clock runs at 1MHz (approx.)

3.3.1.1.3 Synthesized Output

Figure 3.7 is the output observed on the oscilloscope which demonstrates that the synthesized sine/cosine generator code running on the FPGA fulfils the requirement. In this setup, the frequency is 150.8 KHz, corresponding to a period of 6.63us while the peak to peak voltage is 14.7mv. This verifies our code and design for the sine-cosine generator.

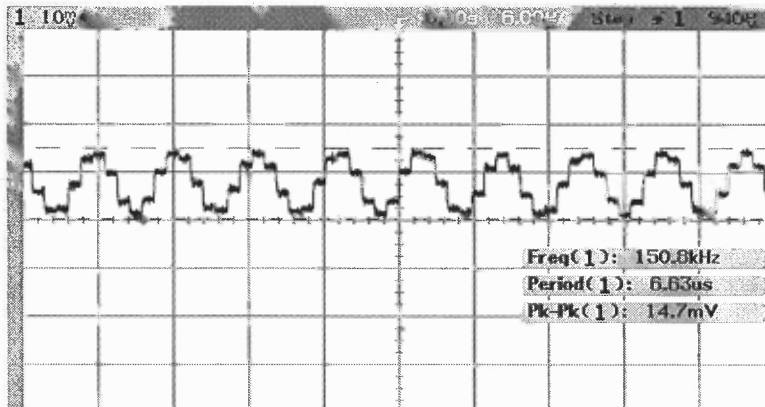


Figure 3.7 Sine generator output observed on the oscilloscope.

3.3.1.2 Infinite Impulse Response (IIR) Filter

Implementation of a low pass filter is considered in this section. A Finite Impulse Response (FIR) filter is the most commonly used format. However, the Spartan II E has lower gate count than required to implement a higher order FIR filter as per the system requirement. Due to this reason we have implemented a 2nd order Butterworth IIR filter instead, as it requires less number of gates. Since the filter is not part of a feedback loop, phase delay introduced by the IIR filter is inconsequential.

3.3.1.2.1 Simulation Results and Design

The poles and zeros of the IIR filter are generated in Matlab for the system requirement given in Table 3.3. This gives us an IIR filter with magnitude response as shown in Figure 3.8.

Table 3.3 Filter parameters defined in Matlab and the corresponding filter coefficients.

Cut off Frequency	75 KHz
Sampling rate	1 MHz
Filter	2 nd Order Butterworth Filter
Response Type	Low Pass Filter
Filter Coefficients	
Denominator	1 ; -1.3489 ; 0.51398
Numerator	1; 2 ; 1
Gain	0.041253537241720262

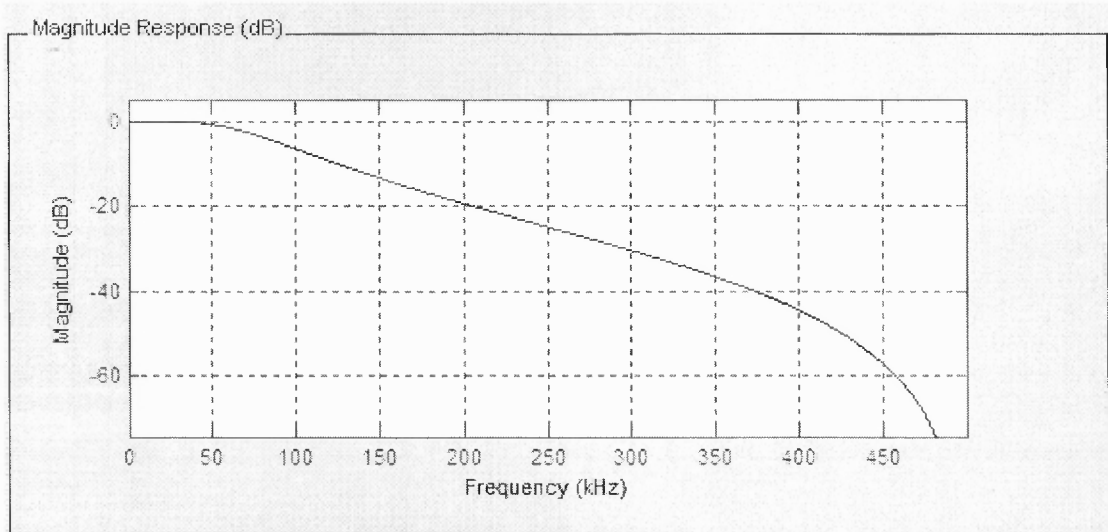


Figure 3.8 Magnitude response of the IIR filter with parameters from Table 3.3

Direct-II form as shown in the Figure 3.9a is used to implement the IIR filter on the FPGA since it requires the least memory (buffer) for its implementation as compared to the others. Here, X is the input, Y is the output, Qn are the poles and Pn are the zeros. The gain of the filter is first kept low at the beginning of the design to avoid any saturation that might occur. It is then increased based on the amplitude of the input signal

to the QAM.

Since the coefficients of the filter are fractional, extreme precaution is taken in maintaining the filter's response characteristics. This is done by appropriate scaling of the coefficients. Fixed point simulation is carried out in Matlab and compared with the Simulink model shown in Figure 3.1a. These coefficient are then broken down into multiples of $(\frac{1}{2})^n$ [44] so if input 'a' has to be multiplied with the coefficient 1.3489 it is done as shown in equation (3.0) which gives an rounding error of 0.002%:

$$a * 1.3489 = a + (a/4) + (a/16) + (a/32) + (a/256) + (a/1024) \quad (3.0)$$

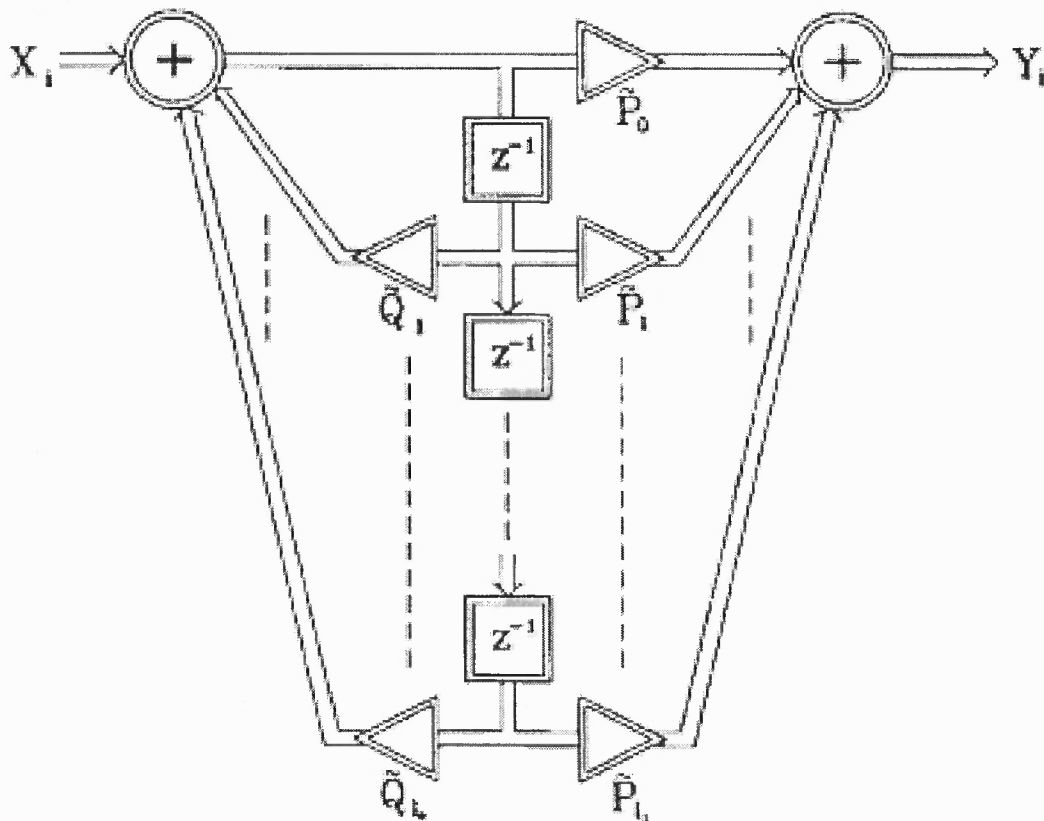


Figure 3.9a Direct- II Form for the implementation of an IIR filter. [45]

Source D. Kim and B. G. Lee, "Transform domain IIR filtering," *IEEE Transactions on Signal Processing*, vol. 43, pp. 2431-2434, 1995.

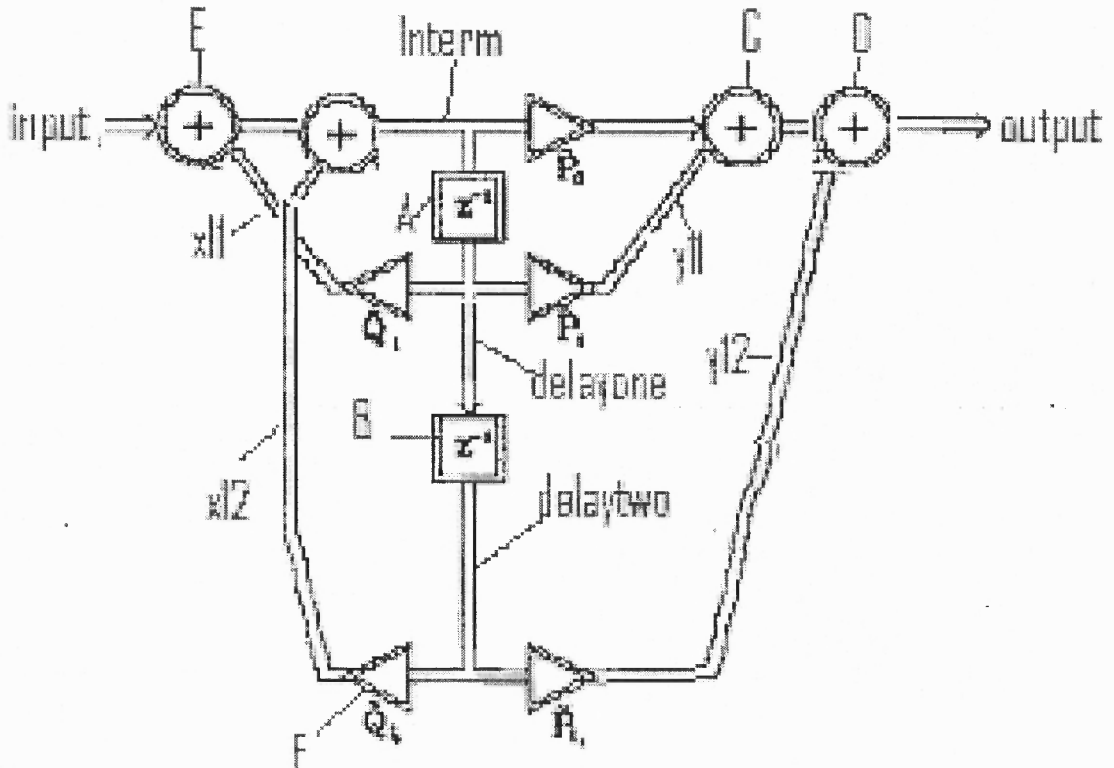


Figure 3.9b Direct-II form re-arranged for its implemented over the FPGA
 Source D. Kim and B. G. Lee, "Transform domain IIR filtering," *IEEE Transactions on Signal Processing*, vol. 43, pp. 2431-2434, 1995.

The implemented IIR Filter on the FPGA matches the filter in Figure 3.9b which is governed by the following equations:

$$\text{Interm} = x12 + x11 + \text{input}; \quad (3.1)$$

$$\text{temp} = y11 + \text{interm} \quad (3.2)$$

$$\text{Output} = \text{temp} + y12 \quad (3.3)$$

$$F = (\text{delaytwo}/1024) + (\text{delaytwo}/512) + (\text{delaytwo}/256) + (\text{delaytwo}/128) + \quad (3.4)$$

$$(\text{delaytwo}/2)$$

$$\text{delayone} = \text{interm} * z^{-1} \quad (3.5)$$

$$\text{Delaytwo} = \text{delayone} * z^{-2} \quad (3.6)$$

All the critical signals from the design (Figure 3.9b) are highlighted in red. Here, A and B are the memory of the system (D- Flip Flops) which are used to performs operations as shown in equations 3.5 and 3.6 respectively. x_{11} which is the output from multiplication of delayone and the pole gets added (E) to the input of the filter whose output is then added to x_{12} which is obtained by multiplying (Equation 3.0, F) the pole with delaytwo, this gives interm (3.1). interm goes to the D-Flip Flop (A) to get stored for one clock cycle (3.6) and also goes to the adder (C) to get added into y_{11} to get temp (3.2). temp then gets added (D) to y_{12} to give the output yout(3.3). In the meanwhile, at the output of delayone, which is the interm value from the previous clock cycle, the data gets stored in B and comes out as delaytwo in the next clock equation (3.7).

Array of adders F (3.4) on page 2 represents the function Shift and Add (F) as shown in (3.0) since the gain is fractional. This gives a better accuracy on the FPGA. While the 1MHz clock synchronizes the circuit, an asynchronous reset is used to clear and initialize the buffers (D- Flip-flop, A and B).

3.3.1.2.2 Register Transistor Logic: IIR Filter

Figure 3.10 shows the RTL of the IIR filter which has been implemented as per the design shown in Figure 3.9b. The RTL for the VHDL code is generated on 4 pages which consist of adders, multiplexers and D-type flip-flops.

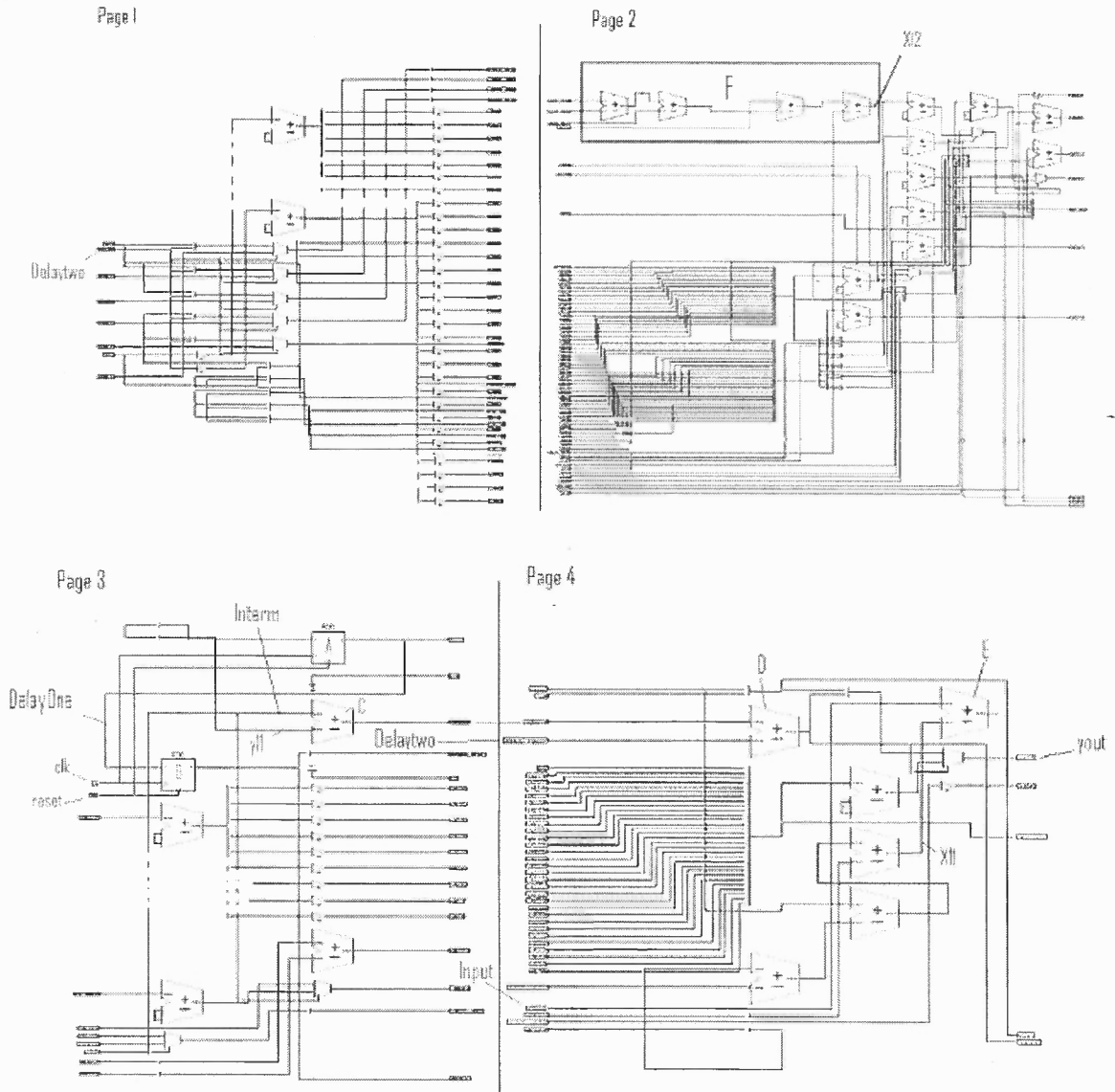


Figure 3.10 RTL for the IIR Filter.

3.3.1.3 Square Root

Square root is implemented using the Restoring Shift-and-Subtract Square Rooter method [46]. The restoring process is achieved by a multiplexer selecting the previous remainder in case of a negative result from the subtraction step where the key factor rests on the expression $P(i)$ to be subtracted from the successive remainder $R(i-1)$ and the final result $Q(-1)$ is built up by concatenation of the complemented sign bits, from $q(n-1)$ to $q(0)$ [46]. The function $P(i)$ [46] is then computed as:

$$P(i) = (4 \cdot Q(n-i) + 1) \cdot 2^{2(n-i)} \quad (3.7)$$

To achieve the above function, pseudo-operators are displayed in Figure 3.11 as shifters, they stand for the rules to be respected to connect $Q(n-i)$ to the subtractor input $P(i)$, input $P(i)$ which is made up of $Q(n-i)$, followed, from left to right, by the string '01' then by a string of $2 \cdot (n-i)$ zeros [46].

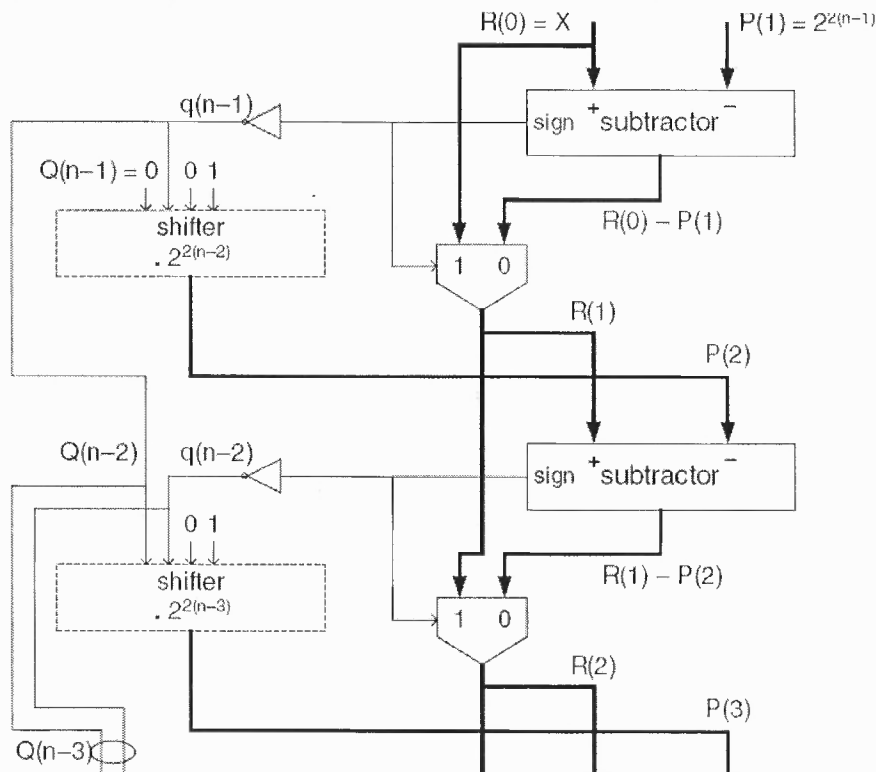


Figure 3.11 Restoring 2n-bit square rooter, combinational implementation

Source: Jean-Pierre Deschamps, G ry Jean Antoine Bioul, and G. D. Sutter, "Synthesis of Arithmetic Circuits," 2006.

For an input vector of length of $2n$ the algorithm calculates the square root of length n and is the value in $Q(-1)$ in the n th steps. This is one of the most efficient ways to perform the square root operation. Sub-blocks for adder, shifter, multiplexers and gates are designed and optimized which are later port-mapped to perform the square root operations.

3.3.1.3.1 Simulation Results

Figure 3.12 shows the simulation results for the square root generator. In this we input the binary values of 16 and get 4 at the output.

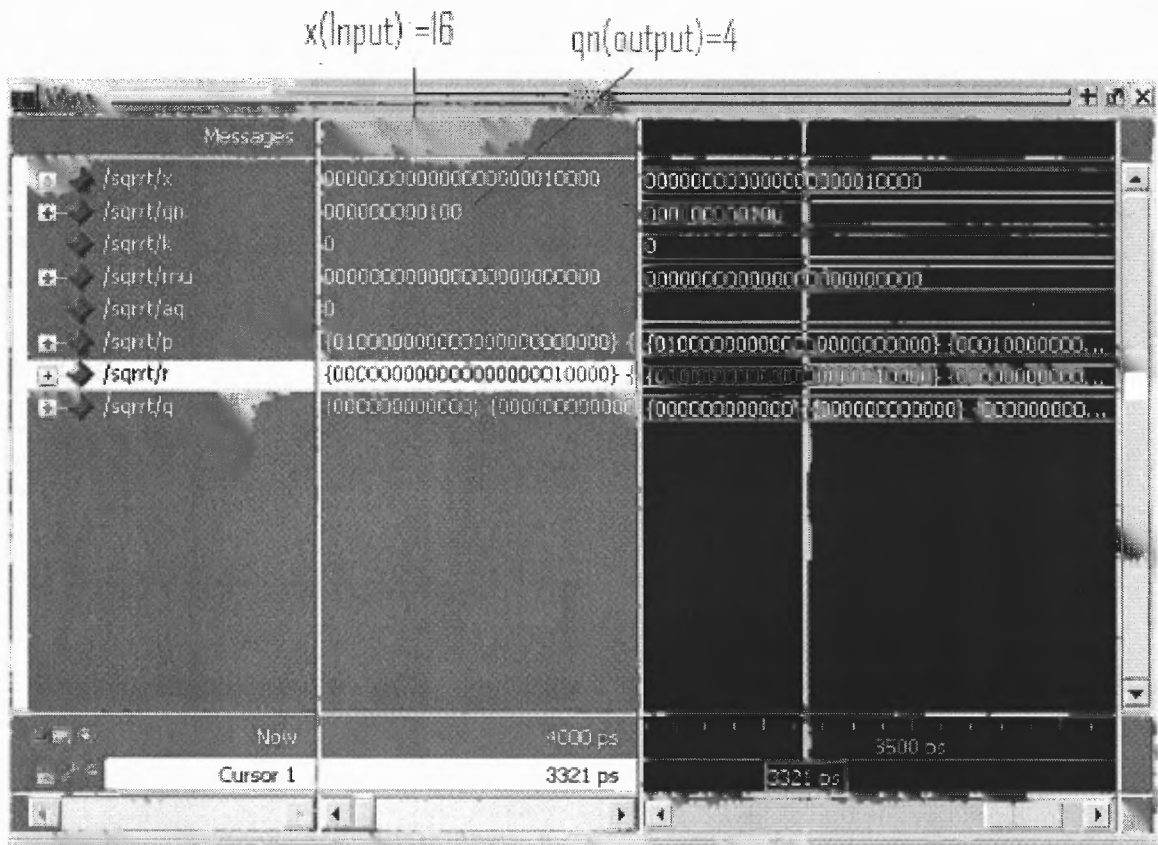


Figure 3.12 Simulation result for the square rooter.

3.3.1.3.2 Register Transistor Logic

Figure 3.13 shows the RTL logic for the square root generator. Figure 3.13a is the top level RTL which has 11 *onmod* modules and one *subtractor*. 11 *onmod* represents the n-1 steps i.e. 11 steps for the algorithm. Since we just require the square root of the input and not the remainder we just implement a *subtractor* instead of another *onmod* which saves memory also. Figure 3.13b shows the RTL of the single *onmod* (right) module compared with a single step of the algorithm (left). This RTL can be explained by naming

a part of it as $A(\text{subtractor})$ and another as $B(\text{cali})$. $P(i)$ from equation 3.7 and $R(i-1)$ which is the input to the square root calculator for the first step goes to the *subtractor* (A). Below A as shown in Figure 3.13C consists of a *subtractor* (C, complement and full adder) followed by a multiplexer which decides to output either the subtracted value ($R-R$) or R based on the sign of the subtracted value. The inverse of the MSB of the output from the adder is used as an input to the shifter in module B. B implements equation (3.7).

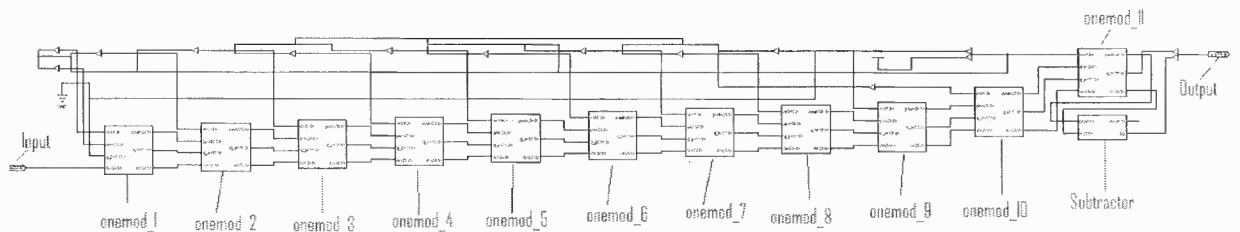


Figure 3.13a RTL for the Square Root consisting of onemod and subtractor modules

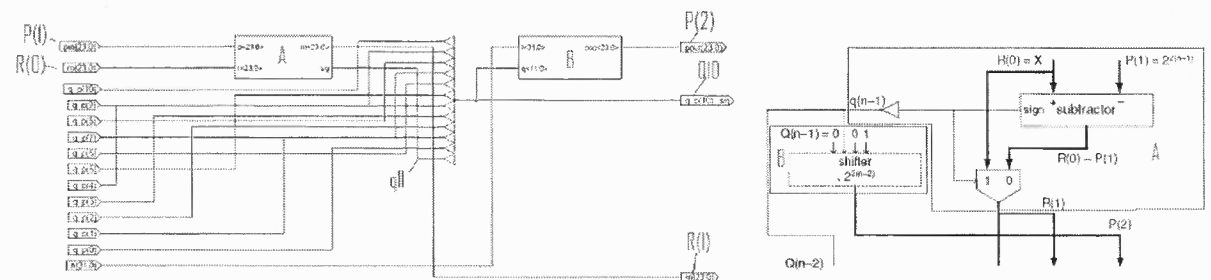


Figure 3.13b RTL for the sub block onemod consisting of submodule subtractor and cali modules

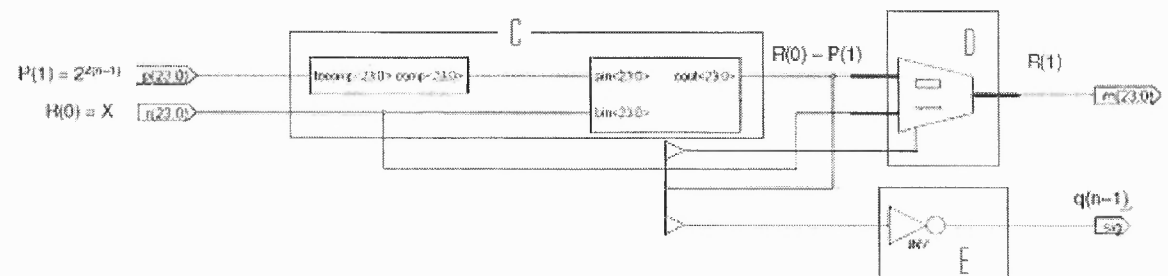


Figure 3.13c RTL for the sub-sub-block (A) consisting of a 2's complement, full adder, multiplexer and an inverter modules.

3.3.1.4 Full QAM

After the design, verification and optimization of each module is completed, port mapping is then carried out to design a system which closely matches the Matlab/Simulink model in Figure 3.1a. As shown in Figure 3.14, *Signed_Input* is the input to the QAM module. *Quadoutput* (D) module multiplies the input with the sine and cosine values respectively. These values then go to the modules *iirm* (IIR Filter, A) where any signal with frequency above 75 KHz gets removed. Squaring is performed by *mulrir* (B) followed by an adder *adderfriir*(C). The 24-bit output from the adder goes to the square root module, *sqrr* (E) to give a 12-bit output. *XLXN_30* is the demodulated 12-bit signed value for the signal received from the ADC.

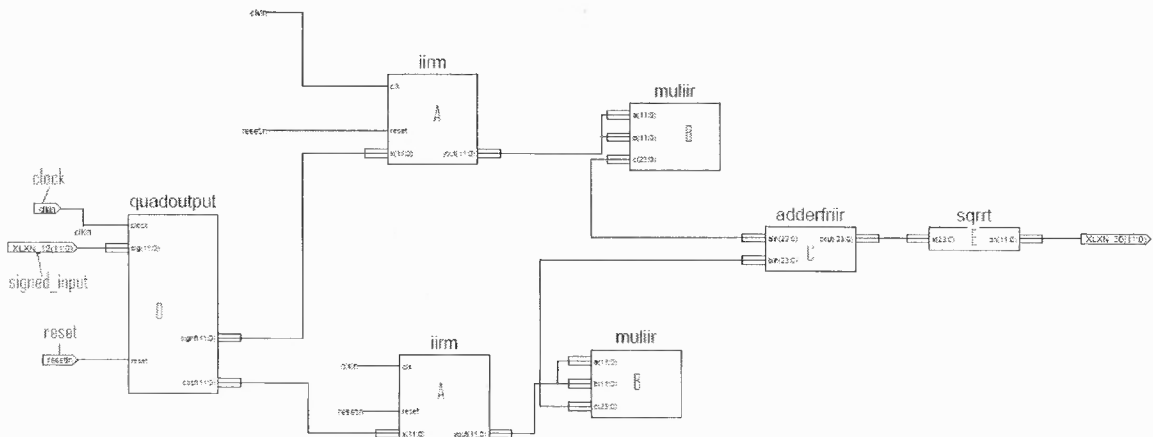


Figure 3.14 Schematic of the QAM. Different sub-blocks port mapped to form one whole system.

3.3.1.5 Frequency Spectrum

An aim for designing the QAM on the FPGA in the FPGA-DSP environment is to take the advantage of performing all high frequency design requirement on it while performing lower frequency large number crunching on the DSP.

The received signal from the transducers has frequency 150 KHz which is sampled at 1MHz. If this is directly sent to the DSP without the QAM demodulator then

an interrupt to the DSP will occur at every $1 \mu\text{s}$. So for real time system design all the calculations are required to be done in with approximately 8000 Instruction/ Interrupt. This might be good enough for small analysis algorithm to be run on the DSP but complex algorithm will require more time in real time for their execution.

This shortcoming of the system usage is overcome by performing QAM on the FPGA. The frequency spectrum of the received signal shifts from 150 KHz to 37 KHz for the QAM signal as shown in figure. This means that decimation of the demodulated signal by a factor of 4 will give same us same precision and accuracy at a reduced rate. The hardware interrupt can now be configured to occur at $4 \mu\text{s}$ which gives 32000 Instructions per interrupt. Receiver

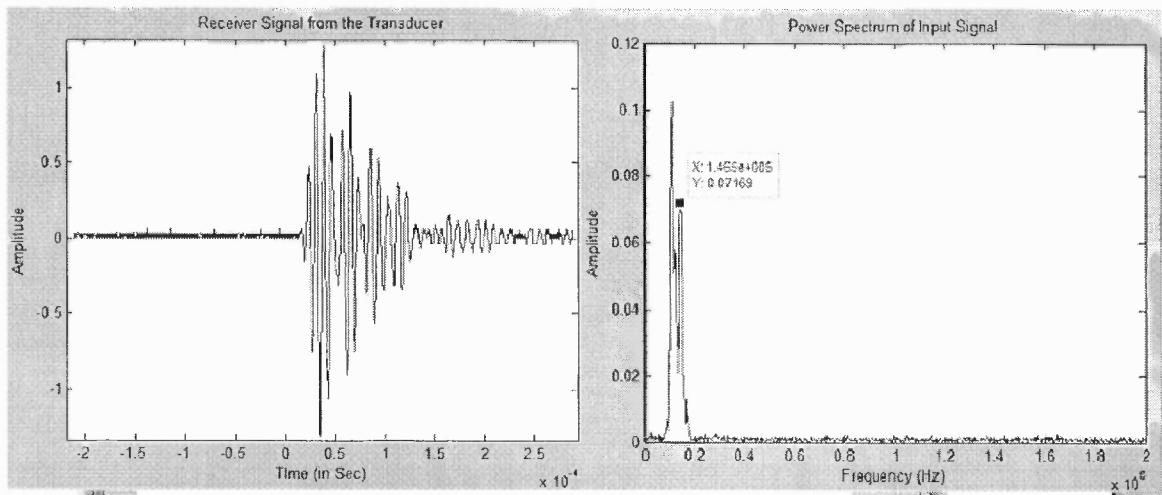


Figure 3.15 Received signal and its FFT.

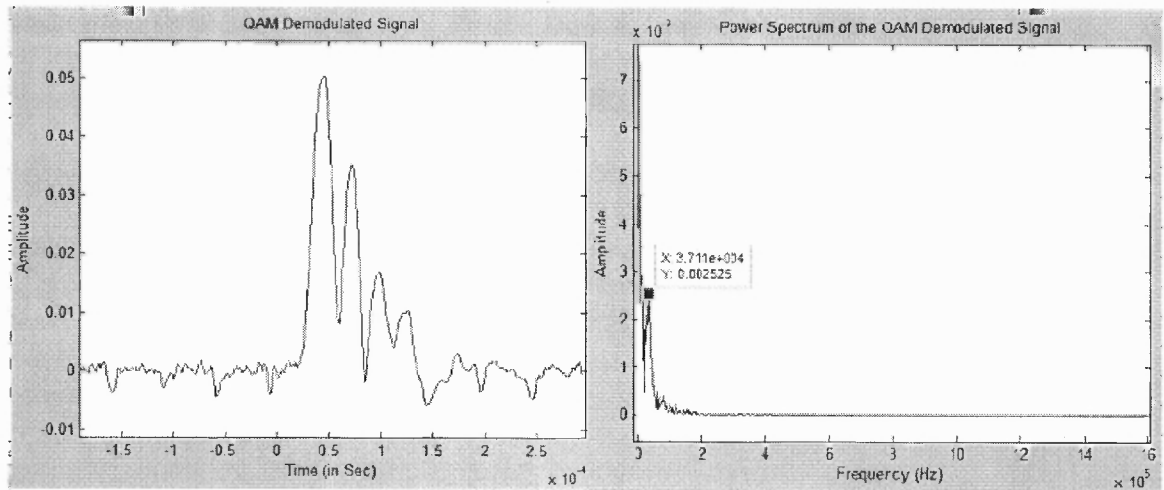


Figure 3.16 Envelope signal and it's FFT

3.3.2 Low Transient Pulse (LTP)

For the experimental system, the LTP drive signals are synthesized through the FPGA so as to achieve high temporal resolution and design flexibility [4]. Temporal resolution of the LTP is maximized by running the timer at its maximum frequency of 62.5 MHz. To generate the LTP drive signal, the FPGA is configured as a 12-bit register module pre-loaded with the corresponding binary values of each pulse level to drive the DAC converter. These values are outputted through the DAC in a circular buffer.

3.3.2.1 Register Transistor Logic of Low Transient Pulse (LTP)

The LTP function is implemented in the DAC module in FPGA shown in Figure 3.17. In this case, counter (F) gets incremented at each clock (62.5 MHz). The incremented value of the counter is checked by the less than or equal to function (A and B). For A, if the value is less than 264 then it outputs a value equal to 2.6 V ($2.6/0.0013 = 2000$) if the value of counter is greater than the value in A but less than the value in B it outputs a value equal to 1.6V ($1.6/0.0013 = 1231$). XOE and Chip Select control bits are XORed to

get write control bit for DAC1.

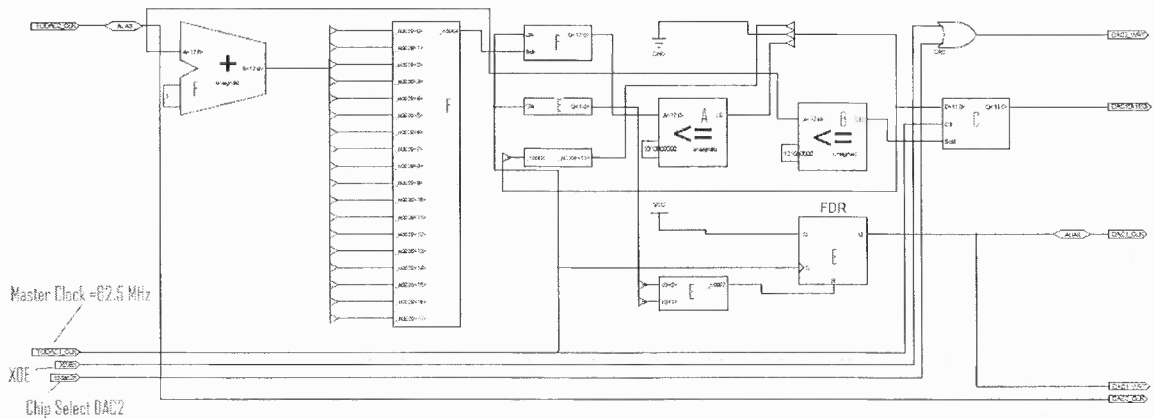


Figure 3.17 RTL for the LTP

3.3.3 Clock Divider

As the design require different module running at different clock frequency, clock divider is implemented to run the LTP and DAC at high frequency while the QAM, ADC and other modules at a relatively lower speed.

3.3.3.1 Register Transistor Logic of the Clock Divider

As shown in Figure 3.18, in the RTL of the clock divider, the counter (A) increments at each clock edge whose value is checked against a pre-loaded value of 30 (B). If the incremented value is less than 30 then input of FDR (C) gets transferred to the *clock_out* which is 1. When the value of the counter reaches 30 or more '1' is set as the output of B whose inverted value is used to trigger the reset to generate a '0' at each clock. In this way a clock input running at 62.5 MHz will gives a clock frequency output of $(62.5 \text{ MHz} / 2 * 30 = 1.041 \text{ MHz} \sim 1\text{MHz})$.

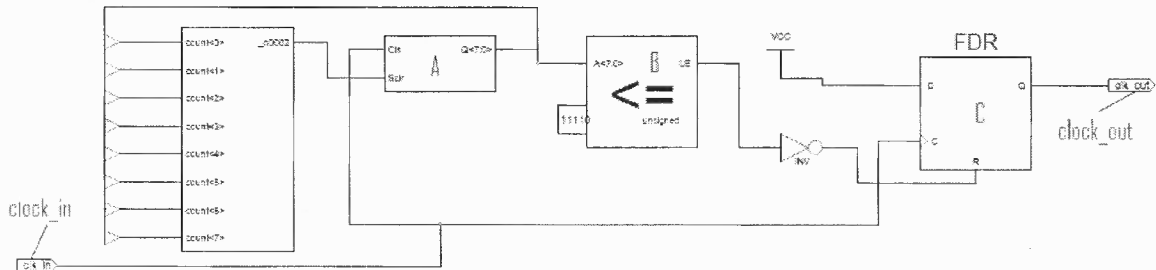


Figure 3.18 RTL for the clock divider

3.3.4 Interfacing FPGA and DSP

The C64x DSP has a diverse set of interfaces, including two glueless external memory interfaces (EMIFs): a 64-bit EMIFA and a 16-bit EMIFB that can support many glueless interfaces to a variety of external devices including external asynchronous devices like the daughter board [36]. It has a one cycle command-to-command turnaround time with at least 1 data dead cycle that is always included between commands so that read data and write data are never driven in the same cycle [36].

The DSP requests data on its bi-directional bus by communicating to the FPGA through its address lines by sending value of CSADC over the address lines. After the DSP receives the data it switched the bus from read to write (to the DAC) which also gets updated with the FPGA through its address lines by sending WRDAC1 and WRDAC2 for channel one and channel two respectively. The data from the DSP after processing are then sent back during the same period to the DAC module of the FPGA. The EMIF space control register is configured so that the setup, hold and strobe time for both read and write are 15x, 1x, and 1x respectively over the 133MHz clock cycles of the EMIF. For the current setup the turnaround time which is the time needed to switch from writing to the reading is, 13x (= 0.22 μ s). Since the interrupt is every 1us (1MHz), it is required that all the processing is done within that sampling time. Once the interrupt has been

completed the program goes to an infinite while loop and waits for the next interrupt.

Memory map of the EMIF is shown in Table. 3.4.

Table 3.4 Memory map of the EMIF for the interface between DSP and FPGA.

Source: S. D. Inc., Spectrum Digital Inc, TMS320C6416 DSK Technical Reference 505945-0001 Rev. A., 2003.; T. I. Inc., TMS320C6000 DSP External Memory Interface (EMIF) Reference Guide Literature Number: SPRU266; Texas Instruments, Inc., 2004.; T. I. Inc., Datasheet of TMS320C6414T, TMS320C6415T, TMS320C6416T Fixed-Point Digital Signal Processors, SPRS226M, 2009.

Hex Address	Name	Remarks
0xa0000000 - 0xaFFFFFFFFF	EMIFA	EMIFA CE2 memory space
0x1800010	EMIFA_CE2	Address for the EMIFA CE2 configurable registers
0xa0080000	CSADC	Address to send the initialization data for the ADC
0xa0080004	WRDAC1	or to receive the sampled data from it
0xa008000C	WRDAC2	Address to send the data to channel one for the DAC. DAC1
		Address to send the data to channel two for the DAC. DAC2

3.3.4.1 Register Transistor Logic: Decoder

As per the memory map shown in Table 3.4 above, the FPGA and the DSP communication interface at the FPGA end is implemented with the control bits and the address lines from the DSP's EMIF. The RTL developed for decoding these address lines and control bits for the synchronization of data is shown in Figure 3.19. Out of the 32 address bits, the lower 16 bits go to the FPGA. This way, for every process the design checks if the data on the data bus is intended for the FPGA, by checking the 15th, 14th and 13th bits of the address lines. If all these bits are zero then the data is for the FPGA, otherwise the data is disregarded. From the memory map bits 0 and 1 are redundant as they remain zero for all the cases so we consider address lines 15 to 2.

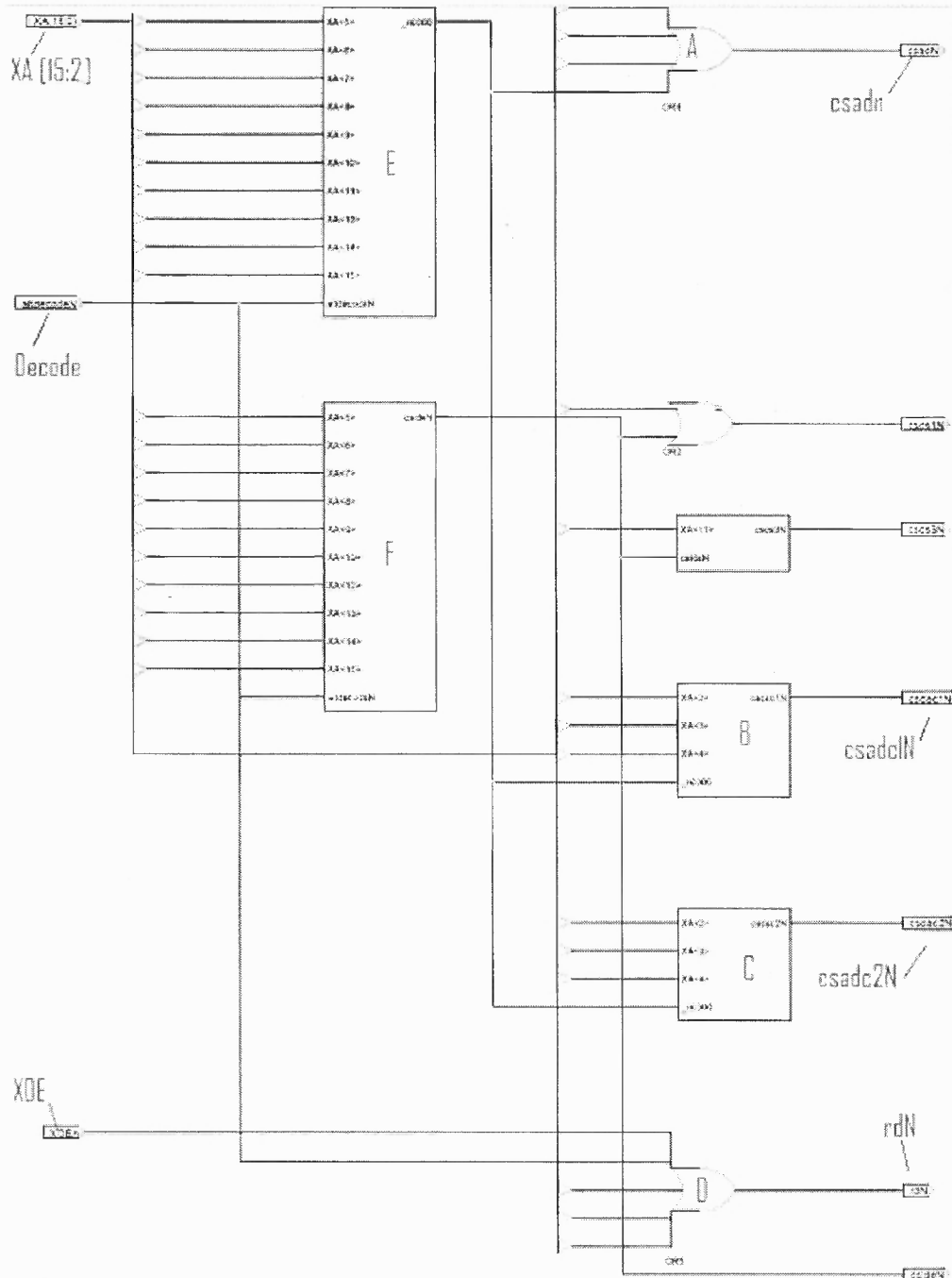


Figure 3.19 RTL decoder for the interface between the FPGA and the DSP.

During the initialization, the data bus is used to send data from the DSP to the FPGA to configure the THS1206. This communication is established if the 15th 14th and 13th address bits with the write enable and decoder enable control bits from the EMIF are

all zero. For the design as shown in Figure 3.19 these bits are ORed (D) to get *rdN* to check if it is zero or not. If *rdN* is zero then as shown in Figure 3.20, the link between the bus which sends the data to the DSP (*to_dsp*) through the *from_dsp* bidirectional bus is established and the bus *from_dsp* (alias *to_ad*) as shown in Figure 3.23 is used to send the data to the ADC through the *from_adc* bidirectional bus.

For the sampled data transfer from the ADC to the DSP, it is necessary to check if all the lower 16 address bits are zero as per the memory map (0xa0080000) along with the decoder bit. To implement the logic all the address bits with the decoder bit are ORed to give the output *csadn*, as shown in Figure 3.19. If all the address bits and the decoder bit are zero then *csadn* will go low hence the FPGA will send the sampled data to the DSP through the data line.

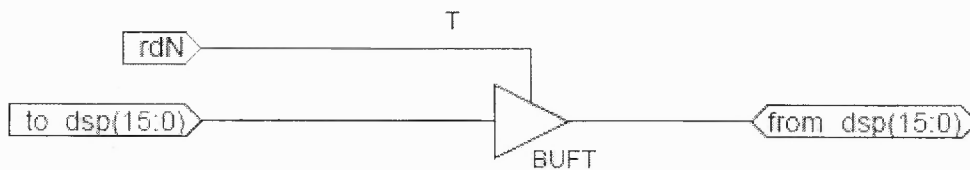


Figure 3.20 RTL schematic for the read enable for the ADC initialization

As shown in Figure 3.21, to write data on to the DAC1 from the DSP it is necessary to set to zero all the address bits in the FPGA except on the 3rd bit as the lower 16 bits of the memory location for WRDAC1 is 0x0004. Hence if *csdac1N* is zero by performing OR on all the address bit from x(15) to x(3) and inverted value of XA(2) then the data on the bi-directional bus is for DAC1. In the same way as shown in Figure 3.22, as the address for WRDAC2 is 0x000C, so if by performing OR on all the address bits from XA(15) to XA(4) with inverted values of XA(3) and XA(2) gives a low then *csdac2N* is low hence DAC2 is selected to read the data from the data bus.

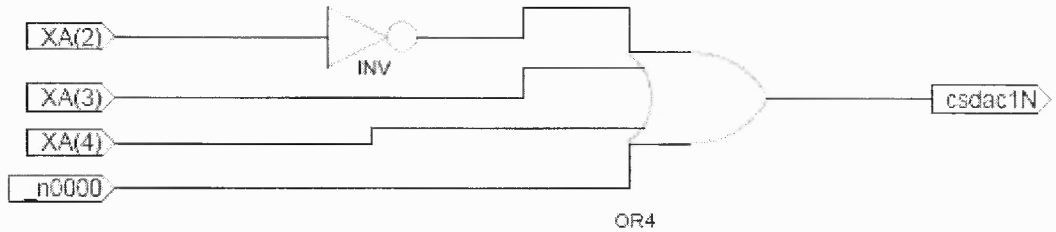


Figure 3.21 RTL schematic for the write enable for the DAC1 initialization

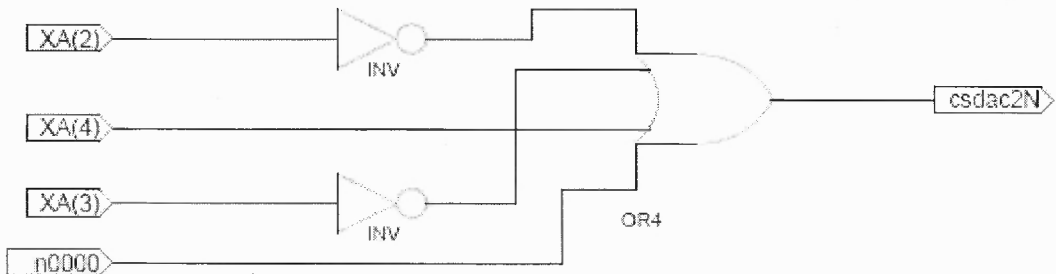


Figure 3.22 RTL schematic for the write enable for the DAC2 initialization.

3.3.5 ADC Module

The ADC RTL module in the FPGA communicates with the ADC on the daughter board. While it transfers all the sampled values from the ADC to the DSP, it is also used to initialize the ADC.

3.3.5.1 Register Transistor Logic of the ADC module

Refer to Figure 3.23, *toadclk* which runs at 1 MHz (approx.) is sent as an external clock to the THS1206. *csad* and *XWE* are ORed for the control bit *ADRW*. If this bit is set at '0' then the DSP writes data on to the data bus to initialize the ADC; otherwise this bus receives the sampled data from the ADC. *AD* (not in the Figure) is the bus between the ADC and the FPGA. This is a 12-bit bidirectional bus which receives data from the ADC to be sent to the DSP when the *ADRW* bit is high and reverses when *ADRW* is low.

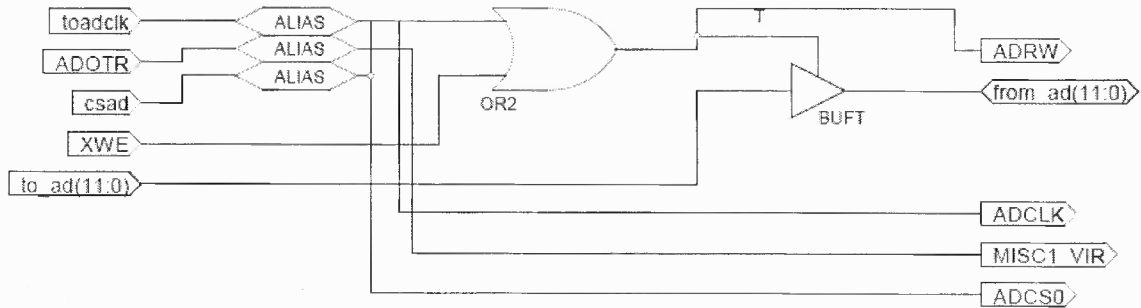


Figure 3.23 RTL schematic for the retrieval of sampled data from the ADC.

3.3.6 UnsignedtoSigned and SignedtoUnsigned Converters

Since the data coming from the ADC are unsigned, they are first converted to signed using the *unsignedtosigned* converter before sending to the QAM. In a similar way, after the data come out from the QAM, they are converted back to unsigned using the *signedtounsigned* module implements on the FPGA.

3.3.7 Complete FPGA System Design

Figure 3.24 shows the different modules for interfacing DSP to the ADC, DAC as well as for implementing the LTP and the QAM demodulator. In this realization, the ADC sends the sampled data to the ADC module (B) through the AD I/O pin of the FPGA. After being converted to *signed* (H), the data are sent to the QAM (A) for getting the envelope of the received signal. After this operation, the data are converted back to unsigned (I) before being sent to the DSP via the *dskbus* (F) module through the XD I/O port on the FPGA. After completing the operation in the DSP (Moving Average Filtering and Edge Detection), the moving average filtered data is sent back through the same I/O, XD. The data from the DSP is then sent to the *dskdac*(D) to be send to the DAC2. Simultaneously, a design in the *dskdac* generates the low transient pulse which comes out from the DAC1.

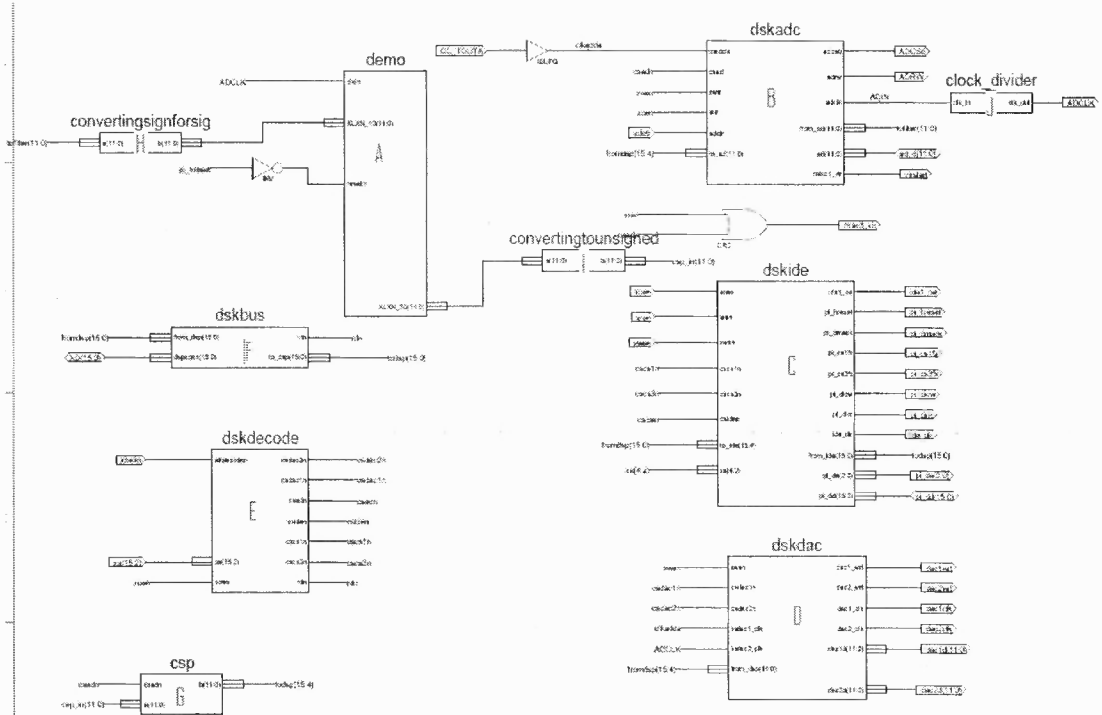


Figure 3.24 Top level schematic of the system on the FPGA.

Since we use 62.5MHz for the generation of LTP a clock divider (J) is used to run the rest of the system including DAC2 and the ADC at a lower rate of 1MHz.

CHAPTER 4

ALGORITHM AND RESULTS

4.1 Algorithm

The following algorithm is to be followed for the implementation of the crack detection on the sawbones:

- 1 LTP with a resolution of 62.5 MHz is generated from the FPGA's DAC module. 12-bit vector is sent to the DAC which converts it to pulses, each with a 3.27us duration and amplitude of $A1=2.5V$ and $A2=1.5V$. This signal is used as a drive signal for the alpha series ultrasonic transmitter.
- 2 The receiver transducer captures the acoustic signal from the test material via the PAC's wide band AE amplifier.
- 3 This signal is then passed through the THS1206 ADC which samples it at 1 MHz with a 12-bit resolution. The data array goes to the QAM module.
- 4 The 12-bit vector is then multiplied with the 5-bit vector of sine and cosine generator respectively to make an 18-bit vector for each of the two IIR filters. The gain of the filter is adjusted according to the amplitude of the input signal. The outputs of the filters give a 12-bit vector which is then squared, making it a 24-bit vector for the inphase and quadrature channels. The two channel outputs are then added to form a 25-bit sum. Using the top 24 bits for the square root hence generates a final 12-bit vector signal which is the envelope of the input receiver signal.
- 5 Hardware interrupt set at 1 MHz is used to send the QAM demodulated 12-bit data from the FPGA through the EMIF (External Memory Interface) Data Bus.

- 6 The data received by the DSP are passed through a Moving Average Filter with the window size of 16. This is done to attenuate high frequency noise in the data as shown in Figure 2.4. The maxima (peaks) are more clearly visible after the data is passed through the filter.
- 7 After the data pass through the filter, a program finds the first four maxima. This is done by first finding the 1st maximum. Then on the basis of rise and fall of the envelope signal, the program calculates the rest of the maxima i.e. when a maximum is reported, it is checked if it is incurred on the rising or the falling edge of the signal. If it is on the falling edge, the maximum is discarded but if it is on the rising edge, the maximum is reported. This can be observed in the Watch Window of the Code Composer Studio® as shown in Figure 2.5. All the values displayed are for time (in us). These values are later used to calculate the parameter of the plate by taking into account the velocity of ultrasound in Sawbone (2.5 mm/ μ s).

4.2 Experimental Verifications

Two sets of experiments are performed each with the Low Transient Pulse on the test platform to verify the performance of the system. These two experiments are

- Symmetric placement of the transducers
- Asymmetric placement of the transducers

In these experiments, a Sawbone sheet with dimensions 13 cm x 17.8 cm x 0.6 cm is used. Figure 4.1 shows the transducer placement configuration of the experiment with A, B, C, D, and E being the distances to be determined. To observe the reflections from the

edges Rx and Tx are placed along the breadth of the Sawbone plates, as shown in Figure 4.1. Errors based on the estimated maxima (EM) and observed maxima (OM) for the received and the envelope signal are then calculated.

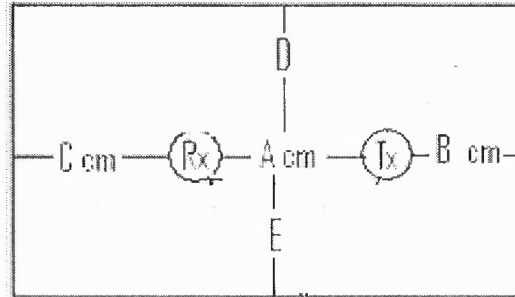


Figure 4.1. Sawbone plate with its dimensions with the edges

4.2.1 Symmetric Placement

In this test, the transducers are placed symmetrically with respect to the boundary, along the centre line of the so that $A = 3.8\text{cm}$; $B = 3.2\text{cm}$; $C = 6.0\text{cm}$; $D = 9.0\text{ cm}$; $E = 8.9\text{ cm}$. The received signals contain four maxima as shown in Figure 4.2. The values that are calculated by the maxima from the received signal and the DSP are shown in Table 4.1. As per the table there is an error for the estimation of the distance with the maxima. It is observed that due to the proximity of signals corresponding to C, D, and E paths (within ± 5 microseconds), some signal aliasing takes place, affecting the boundary estimation for C. This aliasing is also evident in Figure 4.2. It is therefore of interest to readjust the transducer placement asymmetrically to obtain complementary data for a more complete edge determination.

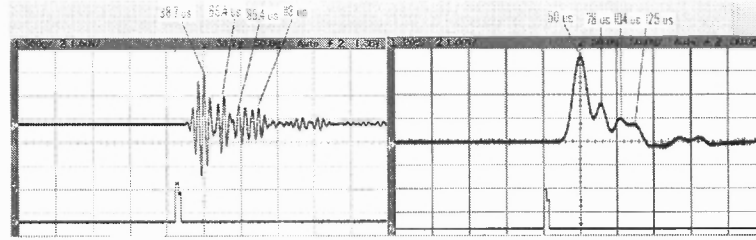


Figure 4.2. Received Signal and its envelope signal observed on the oscilloscope with symmetric placement of the transducers

Table 4.1(a) Estimation of longitudinal parameter of the plate., symmetric placement

Position	Distance (cm)	ET(us)	OT (us)	Error (us)	Error (cm)	OM (us)
A	3.8	23.2	23.6	0.4	0.1	50

Table 4.1(b) Passband maxima detection: summary of plate position errors

Position	Distance	EM (in us)	OM(in us)	Error (in us)	Error (in cm)
A	3.8	38.7		OM- EM	
B	3.2	64.3	65.4	1.1	0.27
C	6	86.7	85.4	-1.3	-0.32
D	8.9	96.30	113	16.69	4.17
E	9.1	97.86	113	15.13	3.78

Table 4.1(c) Envelope maxima detection: summary of plate position errors .

Position	Distance	EM (in us)	OM(in us)	Error (in us)	Error (in cm)
A	3.8	50		OM-EM	
B	3.2	75.6	79	3.4	0.85
C	6	98	104	6	1.5
D	8.9	107.60	125	17.39	4.34
E	9.1	109.16	125	15.83	3.95

4.2.2 Asymmetric Placement

The transducers are then placed at asymmetric positions for better determination of the plate edges. The test configurations are: A= 3.8cm, B= 3.2cm, C= 6.0cm, D=6.5cm and E=11.3cm. In this case, the aliasing problem is not evident, as shown in Figure 4.3 and

the longitudinal parameters of the plate calculated from the maxima of envelope are shown in Table 4.3. It is observed that error in estimating for C is reduced from 1.5cm to 0.75cm.

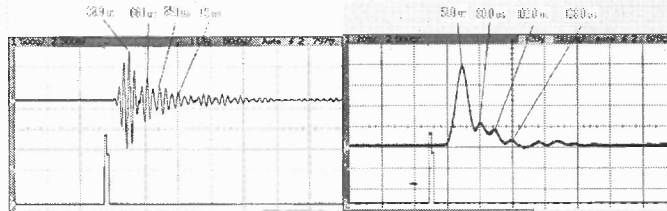


Figure.4.3. Received Signal and its envelope signal observed on the oscilloscope with the asymmetric placement of transducers.

Table 4.2 (a) Estimation of longitudinal parameter of the plate, symmetric placement.

Position	Distance (cm)	ET(us)	OT (us)	Error (us)	Error (cm)	OM (us)
A	3.8	23.2	23.7	0.4	0.125	51

Table 4.2(b) Passband maxima detection: summary of plate position errors

Position	Distance	EM (in us)	OM(in us)	Error (in us)	Error (in cm)
A	3.8	38.9		OM-EM	
B	3.2	64.5	66.1	1.6	0.4
C	6	86.9	85.1	-1.8	-0.45
D	6.5	77.87	No Data	No Data	No Data
E	11.4	115.36	113	-2.36	-0.59

Table 4.2(c) Envelop maxima detection: summary of plate position errors .

Position	Distance	EM (in us)	OM(in us)	Error (in us)	Error (in cm)
A	3.8	3.8	51		OM- EM
B	3.2	3.2	76.6	80	3.4
C	6	6	99	102	3
D	6.5	6.5	89.97	No Data	No Data
E	11.4	11.4	127.46	128	0.53

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

- In this work, an ultrasonic fracture/edge detection system has been implemented on a joint DSP/FPGA platform. The DSP carries out the analysis of the received signal at a much lower rate hence can accommodate a large number of signal channels. The FPGA runs at a higher frequency (62.5MHz) for the generation of LTP signal and to calculate the envelope of the received signal (sampled at 1MHz).
- A Sawbone plate serves as the test object from which the longitudinal dimensions are estimated. Maxima of the envelope of the received signal are used to determine the characteristics of the plate. Both symmetric and asymmetric transducer placement configurations are deployed. It is shown that proper placement of transducers significantly affect the test accuracy.
- The combination of DSP and FPGA provide an optimal solution to process passband signals. This work successfully demonstrates the feasibility of modular programming approach across the two platforms. The dual time scale platform readily accommodates higher temporal resolution needed for the generation of Low Transient Pulses and the processing of real time baseband signals on the DSP for various test conditions. This cost effective implementation enables a transition of the detection system into a hand held ultrasound device.

5.2 Future Work

- This platform can serve as the basis for future research and applications in finding better detection of cracks in Sawbones.
- Broaden the tuning of the LTP for various materials such as Titanium, Aluminum, soft tissues, etc.
- Incorporate the use of RTDX in the analysis of data in real time on the host PC
RTDX enables simultaneous real time communication between the DSP and the host PC.

APPENDIX A

RESOLUTION

Source: Datasheet of THS1206: 12-Bit, 6 MSPS, Simultaneous Sampling Analog-to-Digital Converter (Rev. H)

- Peak-to-peak voltage (ADC/DAC) = $(+2.7V) - (-2.7V) = 5.4 V$
- No. of output bits (ADC/DAC) = 12-bits.
- Therefore, the resolution (ADC/DAC) = $(5.4 / 2^{12}) = 0.013$. Hence the program will preempt any change at input equal to the resolution (0.0013v) and will show no change at the output. Any change greater than this will be observed at the output

Example: a) Hex value for an input voltage (ADC) of 1.5V and – 1.5V

b) Voltage levels at the output (DAC) through the Hex value of 0x960 & 0x460

Solution:

(a) Hex Value: This will be required to determine the hex value from the ADC for an input voltage D.C voltage.

For 1.5 V

→ Dividing with the resolution

$$1.5 / 0.0013 = 1138$$

→ Adding the DC level of 2048 = (0x800) h

$$1138 + 2048 = 3186 = \mathbf{(0xC72) h}$$

For -1.5V

→ Diving with the resolution

$$-1.5/0.0013 = -1138$$

→ Adding the DC level of 2048 = (0x800) h

$$2048 - 1138 = 910 = \mathbf{(0x38E) h}$$

(b) Voltage Levels: This will be required to determine the voltage output from the DAC for a value written on the DAC lines. It is important for tuning the LTP from the DAC for setting up the amplitude for the pulses.

For 0x960

→ Subtracting the DC level

$$(0x960) h = 2400 = 2048 - 2400 = 352$$

→ Multiplying with the resolution

$$352 * 0.0013 = \mathbf{0.4576 V}$$

For 0x460

→ Subtracting the DC level

$$(0x460) h = 1120 - 2048 = -928$$

→ Multiplying with the resolution

$$-928 * 0.0013 = \mathbf{-1.2V}$$

Since the register is 16 bits and the data received from the ADC is 12-bits we divide the incoming sample by 16. After this we subtract it by 0x800 (2048) to convert the unsigned data to sign.

APPENDIX B

DSP MEMORY

Source: Datasheet of TMS320C6414T, TMS320C6415T, TMS320C6416T Fixed-Point Digital Signal Processors, SPRS226M, Texas Instruments

For the analysis of the system many times data stored in the memory of the DSP is required to be analyzed in Matlab. The steps required for this are shown with the following example

- Example: Construct a buffer in DSP, fill it with ADC values and display it in Matlab.
- Solution: For this a buffer is first designed in the DSP program as shown below

```
value1 = (short) *(unsigned volatile int *)CSADC;
```

```
if (i<1000)
```

```
buffer_vector[i] =value1; /Global Declaration, short buffer_vector [] =0;
```

```
i++;
```

```
(cont.)
```

Once the program has been executed the value is read from this buffers in the following way. These values can be retrieved from the memory into a file for their analysis in Matlab by the following steps.

- 1) File → Data → Save
- 2) File Name: Outputdata (let)
- 3) Click Save
- 4) Storing Memory into file

Address: 0x00000200

Length: 0x9D0*

5) Click on OK

Go into the folder of your program to retrieve the file

The starting address of the memory is 0x00000200 since this the starting address for the ISRAM (check the .cdb file). The processor is byte addressable. So if our starting address is 0x200 (512) and we have 1,000 samples the last address containing the data will be $512 + (1,000 * 2) = 2512 = (0x9D0) \text{ h}$

APPENDIX C

SAMPLING TIME

Source : TMS320C6000 DSP 32-Bit Timer, Reference Guide,SPRU582B

- The sampling period for the ADC can be set from the main.c program. The line below is to be modified to change the sampling time
`*(unsigned volatile int *)_TIMER_PRD1_ADDR = 0x01; /* set for 32 bit cnt*/`
- The value to be set in the above equation for a sampling time can be calculated from equation 2.1.
- Below are few examples for setting up the clock for a system without a LTP. All these results are true for system where the master (main) clock is used through out the system. The dskbasic.out file is one such example

A. `*(unsigned volatile int *)_TIMER_PRD1_ADDR = 0xC0`

Period: 1100 0000 =int (192)

Step size : 3.10us

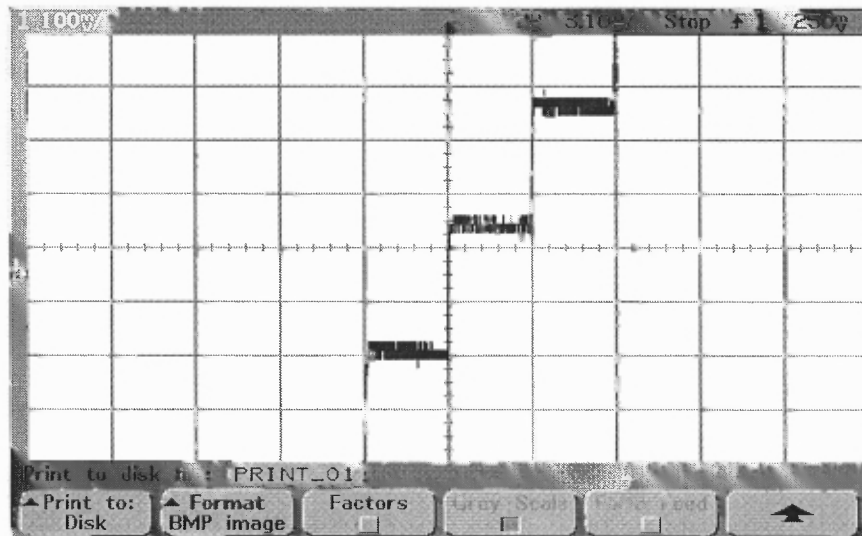


Figure C.1 Output on the oscilloscope showing the step size of 3.10 μ s

B. `*(unsigned volatile int *)_TIMER_PRD1_ADDR = 0xA0`

Period: 1010 0000 =int (160)

Step size: 2.50us

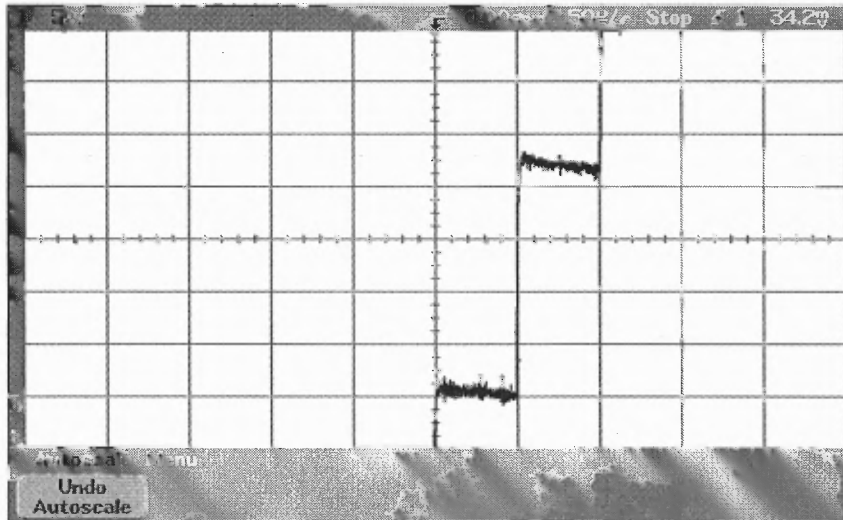


Figure C.2 Output on the oscilloscope showing the step size of 2.50 μ s

C. `*(unsigned volatile int *)_TIMER_PRD1_ADDR = 0xB`

Period: 0000 1010 =int (11)

Step size: 1.68~1.80us

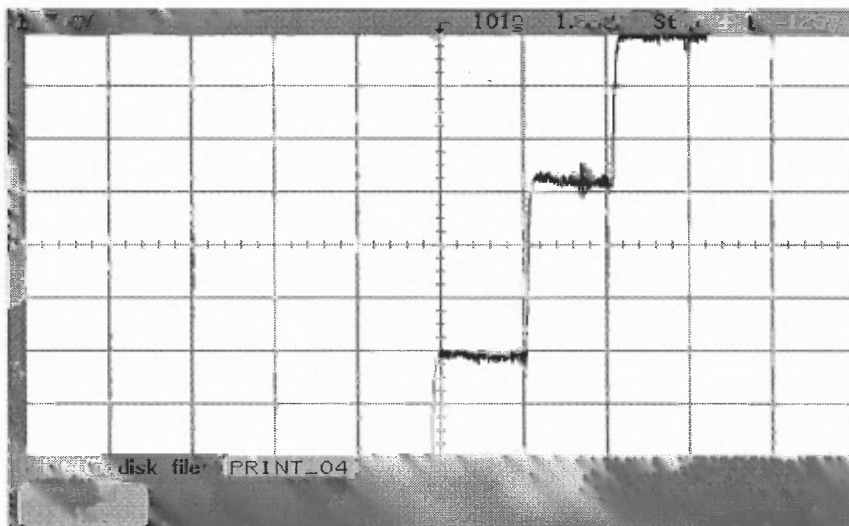


Figure C.3 Output on the oscilloscope showing the step size of 1.66 μ s

D. `*(unsigned volatile int *)_TIMER_PRD1_ADDR = 0x0A`

Period: `0000 1010 = int (10)`

Step size: **ERROR**: The signal at the output is corrupted.

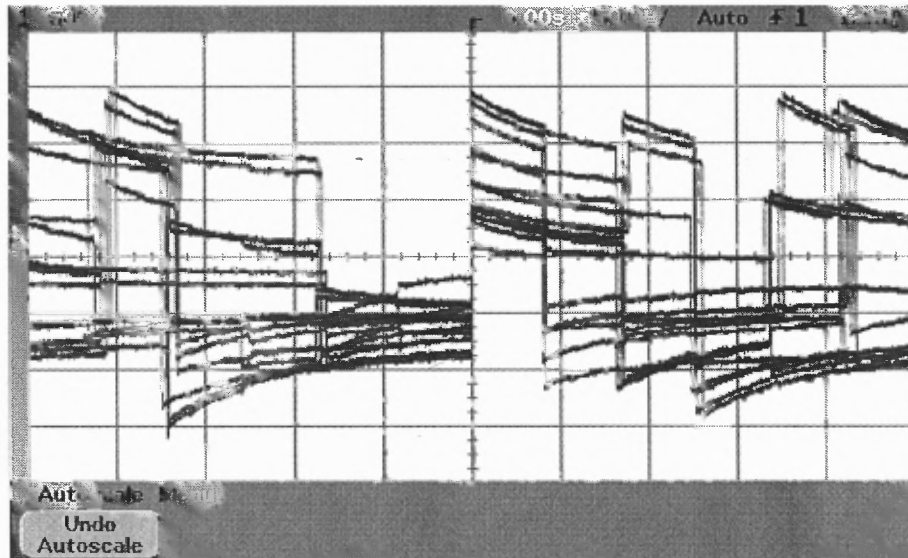


Figure C.4 Output on the oscilloscope showing error for any sampling period above $0.5 \mu\text{s}$ (2MHz)

Important Notes

- Due to the problem shown in D the ADC runs can be run to the **maximum frequency of 1MHz.**
- Since for the better LTP resolution we need the clock of the maximum timer frequency of 62.5MHz at the DAC we have designed a clock divider to run the ADC at 1 MHz.

REFERENCES

- [1] U. Polimeno and M. Meo, "Detecting barely visible impact damage detection on aircraft composites structures," *Composite Structures*, vol. 91, pp. 398-402, 2009.
- [2] P. Moilanen, "Ultrasonic guided waves in bone," *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, vol. 55, pp. 1277-1286, 2008.
- [3] A. K. Kromine, P. A. Fomitchov, S. Krishnaswamy, and J. D. Achenbach, "Laser Ultrasonic Detection of Surface Breaking Discontinuities: Scanning Laser Source Technique," *Materials Evaluation*, vol. 58, pp. 173-177, 2000.
- [4] B. Cheng and T. Chang, "Enhancing ultrasonic imaging with low transient pulse shaping," *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, vol. 54, pp. 627-635, 2007.
- [5] V. C. Protopappas, M. G. Vavva, D. I. Fotiadis, and K. N. Malizos, "Ultrasonic monitoring of bone fracture healing," *Ultrasonics, Ferroelectrics and Frequency Control, IEEE Transactions on*, vol. 55, pp. 1243-1255, 2008.
- [6] J. D. Achenbach, "Modeling for quantitative non-destructive evaluation," *Ultrasonics*, vol. 40, pp. 1-10, 2002.
- [7] J. Y. Kim, V. A. Yakovlev, and S. I. Rokhlin, "Surface acoustic wave modulation on a partially closed fatigue crack," *Journal of the Acoustical Society of America*, vol. 115, pp. 1961-1972, 2004.
- [8] C. Valle, M. Niethammer, J. Qu, and L. J. Jacobs, "Crack characterization using guided circumferential waves," *Journal of the Acoustical Society of America*, vol. 110, pp. 1282-1290, 2001.
- [9] T. D. Chaudhari and S. K. Maiti, "Modelling of transverse vibration of beam of linearly variable depth with edge crack," *Engineering Fracture Mechanics*, vol. 63, pp. 425-445, 1999.
- [10] N. Qaddoumi, E. Ranu, J. D. McColskey, R. Mirshahi, and R. Zoughi, "Microwave detection of stress-induced fatigue cracks in steel and potential for crack opening determination," *Research in Nondestructive Evaluation*, vol. 12, pp. 87-103, 2000.
- [11] K. M. Liew and Q. Wang, "Application of wavelet theory for crack identification in structures," *Journal of Engineering Mechanics*, vol. 124, pp. 152-157, 1998.
- [12] R. Albanese, G. Rubinacci, and F. Villone, "An Integral Computational Model for Crack Simulation and Detection via Eddy Currents," *Journal of Computational Physics*, vol. 152, pp. 736-755, 1999.
- [13] I. S. U. NDT Resource Center, "What is NDT?," Iowa State University.
- [14] H. Abu-Rub, J. Guzinłski, Z. Krzeminski, and H. A. Toliyat, "Predictive current control of voltage-source inverters," *IEEE Transactions on Industrial Electronics*, vol. 51, pp. 585-593, 2004.
- [15] S. Jung and S. S. Kim, "Hardware implementation of a real-time neural network controller with a DSP and an FPGA for nonlinear systems," *IEEE Transactions on Industrial Electronics*, vol. 54, pp. 265-271, 2007.
- [16] S. L. Jung, M. Y. Chang, J. Y. Jyang, L. C. Yeh, and Y. Y. Tzou, "Design and implementation of an FPGA-based control IC for AC-voltage regulation," *IEEE Transactions on Power Electronics*, vol. 14, pp. 522-532, 1999.

- [17] H. Liu, P. Meusel, N. Seitz, B. Willberg, G. Hirzinger, M. H. Jin, Y. W. Liu, R. Wei, and Z. W. Xie, "The modular multisensory DLR-HIT-Hand," *Mechanism and Machine Theory*, vol. 42, pp. 612-625, 2007.
- [18] I. P. Seskar and N. B. Mandayam, "Software radio architecture for linear multiuser detection," *IEEE Journal on Selected Areas in Communications*, vol. 17, pp. 814-823, 1999.
- [19] C. Ebeling, C. Fisher, G. Xing, M. Shen, and H. Liu, "Implementing an OFDM receiver on the RaPiD reconfigurable architecture," *IEEE Transactions on Computers*, vol. 53, pp. 1436-1448, 2004.
- [20] Z. Bielewicz, L. Debowski, and E. Lowiec, "DSP and FPGA based integrated controller development solutions for high performance electric drives," in *IEEE International Symposium on Industrial Electronics*, 1996, pp. 679-684.
- [21] S. D. Inc., *Spectrum Digital Inc, TMS320C6416 DSK Technical Reference 505945-0001 Rev. A.*, 2003.
- [22] T. Instruments, *Datasheet of THS1206: 12-Bit, 6 MSPS, Simultaneous Sampling Analog-to-Digital Converter (Rev. H)* Texas Instruments, Inc., 1999.
- [23] A. D. Inc., *Datasheet of AD9763/AD9765/AD9767* Analog Devices Inc. , 2009.
- [24] D. L. Perry, *VHDL: Programming by Example*: McGraw-Hill.
- [25] X. Inc., *Datasheet of Spartan-IIE 1.8V FPGA Family: Introduction and Ordering Information*: Xilinx, Inc., 2001.
- [26] X. Inc., *Spartan-IIE 1.8V FPGA Family: Complete Data Sheet*: Xilinx, Inc., 2003.
- [27] T. Instruments, "The World's First 90nm DSPs Running at 1GHz are Now in Volume Production," Texas Instruments Inc.
- [28] BDTI, "A Survey of Mainstream DSP Processors ": TechInsights, a Division of United Business Media LLC, 2007.
- [29] T. Instruments, *How to Write an RTDX Host Application Using MATLAB, SPRA386*: Texas Instruments, Inc., 2002.
- [30] T. Instruments, *DSP/BIOS, RTDX and Host-Target Communications, SPRA895*: Texas Instruments, Inc., 2003.
- [31] *Embedded IDE Link 4.0*: The MathWorks, Inc, 2009.
- [32] T. Instruments, *TMS320C6000 DSP Interrupt Selector, Reference Guide Spru646A*: Texas Instruments, 2004.
- [33] T. Instrument, *TMS320C6000 DSP 32-Bit Timer, Reference Guide, SPRU582B*: Texas Instrument, 2005.
- [34] T. Instruments, *TMS320C6000 Tools: Vector Table and Boot ROM Creation*: Texas Instruments, Inc., 1999.
- [35] T. Instruments, *TMS320C6000 Optimizing C Compiler Tutorial, SPRU425A*: Texas Instruments, Inc., 2002.
- [36] T. Instruments, *TMS320C6000 DSP External Memory Interface (EMIF) Reference Guide Literature Number: SPRU266*: Texas Instruments, Inc., 2004.
- [37] T. Instruments, *Datasheet of TMS320C6414T, TMS320C6415T, TMS320C6416T Fixed-Point Digital Signal Processors, SPRS226M*, 2009.
- [38] R. Chassaing, *Digital Signal Processing and Applications with the C6713 and C6416 DSK*: A John Wiley & Sons, Inc, Publication.
- [39] D. Langmann, *Dalanco-Spry Daughter Board code development, private communications*: Dalanco Spry, 2009.

- [40] T. Instruments, *TMS320C6000 Assembly Language Tools v 6.0 Beta, User's Guide, spru186p*: Texas Instruments, Inc., 2006.
- [41] T. Instruments, *TMS320C64x/C64x+ DSP, CPU and Instruction Set, Reference Guide, SPRU732H*: Texas Instruments, Inc., 2008.
- [42] A. Chotai, *Spartan-II E FPGAs, Lower I/O Cost*: Xilinx, Inc, 2003.
- [43] H. Lai and S. Boumaiza, "WiMAX baseband processor implementation and validation on a FPGA/DSP platform,"in *Canadian Conference on Electrical and Computer Engineering*, 2008, pp. 1449-1452.
- [44] U. M.-. Baese, *Digital Signal Processing with Field Programmable Gate Arrays*: Springer, 2004.
- [45] D. Kim and B. G. Lee, "Transform Domain IIR filtering," *IEEE Transactions on Signal Processing*, vol. 43, pp. 2431-2434, 1995.
- [46] Jean-Pierre Deschamps , Ge'ry Jean Antoine Bioul, and G. D. Sutter, *Synthesis of Arithmetic Circuits*: A John Wiley & Sons ,Inc, 2006.