2013

An FPGA Based Controller for a MEMS Tri-mode FMCW Radar

Sabrina Zereen
University of Windsor

An FPGA Based Controller for a MEMS Tri-mode FMCW Radar

By

SABRINA ZEREEN

A Thesis
Submitted to the Faculty of Graduate Studies through Electrical and Computer Engineering in partial fulfillment of the requirements for the Degree of Master of Applied Science at the University of Windsor

Windsor, Ontario
2013

An FPGA Based Controller for a MEMS Tri-mode FMCW Radar

By
Sabrina Zereen

Approved By:

Jessica Chen
School of Computer Science

Mohammed Khalid
Department of Electrical and Computer Engineering

Sazzadur Chowdhury, Advisor
Department of Electrical and Computer Engineering

May 17,2013

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

Abstract

In this thesis a Xilinx Virtex 5 FPGA platform based signal processing algorithm has been developed and experimentally verified for use in a MEMS based tri-mode 77GHz FMCW automotive radar to determine range and velocity of targets in the vicinity of a host vehicle. The Xilinx Virtex 5 FPGA based signal processing and control algorithm dynamically reconfigures a MEMS based FMCW radar to provide a short, medium, and long-range coverage using the same hardware. The MEMS radar comprises of MEMS SP3T RF switches, microfabricated Rotman lens and a microstrip antenna embedded with MEMS SPST switches, in additional to other microelectronic components. A CA-CFAR module has been used to eliminate false targets in a multi target clutter affected scenario. The refresh rate for the current design is 2.048ms for each mode of radar which is nearly 40 times lower than the BOSCH LRR3. The maximum percent difference from analytical to HDL calculated range values was found to be 0.16% which can be lowered with further refinement. The developed FPGA based radar signal processing algorithm can be implemented as an ASIC which can be batch fabricated to lower down the production cost so that automotive radars can become a standard item for all the vehicles on the road.

A Sincere Dedication

To mom, dad, Nowreen, Azad, Ayan, and my friends

without whom this endeavor would not have been possible

Bismillaahir Rahmaanir Raheem

In the name of Allah, The Beneficent The Merciful

Acknowledgement

Firstly, I would like to express my gratitude to the Almighty Allah, who blessed me with good health and opportunity to finish my masters from a reputed University. He also helped my acquainting me with some lovely people who have helped me in this endeavour.

I would sincerely like to thank my advisor and my co-advisor Dr. Sazzadur Chowdhury and Dr. Khalid, whose relentless support and guidance have helped me to finish my work. Also I would like to express my appreciation to Dr. Jessica Chen who has agreed to become my committee member.

I am especially grateful to Karl Leboeuf and Ahmed Ridwan for their knowledge of coding, which has helped with me with the basic knowledge of Xilinx and also helped me to pull through from several tight spots.

One another person in the University of Windsor, who played a vital role in my masters and without the help of whom things would have been really difficult, was Andria Ballo. She was always there for all the engineering students with her sympathetic ears and helpful hand for every small problem. Her miraculous ability to solve almost every problem and make life for an international student like myself easy and fun is one of her most appreciative trait.

I would also want to thank my MEMS team members, for creating such an entertaining and enjoyable workplace.

To conclude I would like to thank my parents, my sisters and the rest of my family whose prayers, support, patience and never ending belief in me has helped to work through some exasperating situations during the course of my research.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

MEMS – Microelectromechanical Systems

Radar – Radio Detection and Ranging

RF – Radio Frequency

SP3T – Single Pole Triple Throw

PRF – Pulse Repetition Frequency

DSP – Digital Signal Processor(-ing)

FPGA – Field Programmable Gate Array

DAC – Digital to Analog Converter

ADC – Analog to Digital Converter

FSK – Frequency Shift Keying

LFMCW – Linear Frequency Modulated Continuous Wave

HDL – Hardware Description Language

SARA - Strategic Automotive Radar Frequency Allocation

FFT – Fast Fourier Transform

DFT – Discrete Fourier Transform

DIT – Decimation In Time

DIF – Decimation In Frequency

CA (OS)-CFAR – Constant False(Ordered Statistics) Constant False Alarm Rate

RTL – Register Transfer Level

RCS – Radar Cross-Section

CPI – Coherent Processing Interval

VCO – Voltage Controlled Oscillator

LRR – Long Range Radar

MRR – Medium Range Radar

SRR – Short Range Radar

IF – Intermediate Frequency

MSPS – Mega-Samples Per Second

LUT – Look-Up Table

FF – Flip-Flop

BUFG – Global Buffer

BUFGCTLR – Global Clock Buffer

RAM – Random Access Memory

ROM – Read-Only Memory

DSP48E – Xilinx Digital Signal Processing Slice (5th Generation)

ISE – Integrated System Environment

LPF – Low Pass Filter

AWGN – Additive White Gaussian Noise

EM – Electro-Magnetic

MMIC – Monolithic Microwave Integrated Circuits

Nomenclature

r = target range

c = speed of RF waves through air

$T_{\text{two-way}}$ = two-way travel time for RF wave from radar sensor to target and back

v_{rel} = relative velocity

v_{target} = target velocity

v_{host} = host vehicle velocity

f_{d} = Doppler frequency shift

λ = radio wave wavelength

f_{b} = beat frequency or instantaneous intermediate frequency

f_{t} = transmit signal frequency

f_{r} = received signal frequency

τ_0 = travel time for RF wave from radar sensor to target

B = LFMCW sweep bandwidth

T = LFMCW sweep duration

k = rate of change of frequency in LFMCW sweep = B / T

X_{k} = frequency domain sample

x_{n} = time domain sample

P_{fa} = probability of false alarm

T_{A} = CFAR dynamic threshold

σ = Radar Cross-Section of a target

N_{Th} = thermal noise

SNR_{Q} = quantization signal-to-noise ratio

f_{s} = sampling frequency

f_{res} = frequency resolution of FFT

CHAPTER 1: Introduction

In this chapter the research work is defined explicating the importance of active safety systems. Some of the active safety systems include adaptive cruise control (ACC), collision warning systems including automotive steering and braking intervention. Analyzing actual crash records form 2004-2008 it has been found that a forward collision warning system, using radar sensors, will palliate crashes up to 1.2 million crashes per year [1]. Although it was found that the forward collision warning features are found only in luxury vehicles because they are costly.

1.1 Problem statement

The objective of this thesis is to design a FPGA based controller for a 77GHz MEMS FMCW automotive radar using three bandwidths, for covering long, medium and short range.

Road-accidents have played a major role for the loss of millions of lives every year. This deplorable situation can be minimized by making use of a microelectromechanical (MEMS) system based sensor technology to detect the propinquity of vehicles, pedestrians and other obstacles in the vicinity of a host vehicle in real-time. Among the existing state of the art radars in the market are long range radar (LRR) from BOSCH introduced in 2009, ARS 300 by Conti in 2009 and by Denso's third generation long range radar which was introduced a year before [2]. However the current technology in the market like ultrasonic sensors and sensor arrays, lasers, cameras attached on the side mirrors and electromagnetic radar units (which are only available with high-end vehicles) are not adequate to ensure complete safety against accidents. This is because of their slow signal processing system and also for radars; they have to scan the target area mechanically. The cost of such stand-alone devices is too high for the manufacturers to integrate them in low-cost vehicles. As such the dilemma regarding road safety remains nearly same, in spite of some vehicles being equipped with collision warning or pre-crash warning devices.

Strategic Analytics a market research firm has anticipated that within the period of 2006-2011 there would be an increase of the usage of the long-range distance radar for pre-collision warning will increase by 65% annually, the demand reaching 3 million units in 2011, with 23 million of them using radar sensors. By 2014, 7 percent of all new cars will include a distance warning system, mostly in Europe and Japan [3].

Table 1.1 is from IRTAD, an International Group and Database on Road Safety Data, shows the statistics for death count per 100, 000 in habitants and per billion vehicles-km for a number of countries , which are members of IRTAD [4]. The Table shows a considerable amount of decrease in death counts over the years 1970 to 2010, although there is an increase in count for some countries. The decrease can be due to the effective safety measures that are being introduced into the vehicles now-a-days.

However, while high income countries are showing a major reduction in the death rate due to road traffics, the same cannot be said for countries where there is a rapid increase in the number of vehicles on the road. In 2010, 1.3 million people around the world died due to road crashes and more 50 million were injured. Of these death counts about 90% are in low-income and middle-income countries. In May 2011, a Decade of Action for Road Safety was launched by the United Nations with an aspiration to stabilize and then reduce the global road traffic deaths by 2020 [4].

Table 1.1 Road Traffic Deaths per 100, 000 Inhabitants / per billion vehicle - kilometers

Different Country	Death per 100, 000 inhabitants					Death per billion vehicle-kilometers				
	1970	1980	1990	2000	2010	1970	1980	1990	2000	2010
Australia	30.4	22.3	13.7	9.5	6.1	49.3	28.2	14.4	9.3	6.1
Austria	34.5		20.3	12.2	6.6	109	56.3	27.9	15.0	-
Belgium	31.8	24.3	19.9	14.4	8.8[b]	104.6	50	28.1	16.4	9.6[b]
Canada[a]	23.8	22.7	14.9	9.4	6.6[b]	-	-	-	9.3	6.5[b]
Czech Republic	20.2	12.2	12.5	14.5	7.6	-	53.9	48.3	36.7	16.2
Denmark	24.6	13.5	12.4	9.3	4.6	50.5	25	17.3	10.7	5.6
France	32.6	25.4	19.8	13.7	6.4	90.4	44	25.7	15.1	7.1
United Kingdom	14.0	11.0	9.4	6.1	3.1	37.4[c]	21.9[c]	12.7[c]	7.3[c]	3.7[c]
United States	25.7	22.5	17.9	15.3	10.6	29.6	20.9	12.9	9.5	6.8
Germany	27.3	19.3	14.0	9.1	4.5	-	37.3	20	11.3	5.2
Hungary	15.8	15.2	23.4	12.0	7.4	-	-	-	-	-
Israel	17.1	10.8	8.7	7.1	4.6	87.9	38.8	22.4	12.4	7.1
Italy	20.5	16.3	12.6	12.4	6.8	-	-	-	-	-
Japan	21	9.7	11.8	8.2	4.5	96.4	29.3	23.2	13.4	7.7[b]
Korea	11.0	17.0	33.1	21.8	11.3	-	-	-	49.5	18.7
Malaysia[a]	-	-	22.7	25.9	23.8	-	-	-	26.3	16.2

Poland	10.6	16.8	19.2	16.3	10.2	-	-	-	-	-
Spain	16.0	17.7	23.2	14.5	5.4	-	-	-	-	-

Death within 30 days. Data recorded by the police. a = accession country. b = 2009 c = Great Britain

"Radar technology is the key to building innovative driver assistance systems to help avoid automobile accidents," says Hans Adlkofer, VP and general manager of Infineon Technologies' Sense and Control business unit [3].

The radar technology initially came with the high-end luxury vehicles, but the cost has dropped down and can be used in reasonably priced vehicles. Automotive industries all over the world are utilizing the radar technology for a wide range of detection systems such as:

1. Adaptive Cruise Control with stop and go functionality.
2. Blind-spot detection.
3. Self-parking system.
4. Side-sensors for parking aids
5. Pre-crash warning system.
6. Lane Changing assistance to the driver.
7. Backup camera and sensor for impact warning.
8. Car to computer communication devices utilizing GPS tracking system.
9. Remote speed sensor.

The global economic cost for road crashes and injuries is estimated to be around US$ 518 billion annually. Unless appropriate actions is taken road traffic injuries is predicted to become the third leading contributing factor in the global burden of disease and injury. It cost European Union 180 billion euros annually for road traffic injuries which is twice the annual budget for all other activities in these countries. This huge public health and development problem kill almost 1.2 million people a year and cause injury or disability to 20 to 50 million or more people. Data from World Bank and WHO show that unless proper action is taken the injury count will increase drastically by 2020, especially in countries where there is a rapid increase of motor vehicles [5] . Hence the multi-range radar technology is a constitutive feature to enhance highway safety and mitigate loss of lives and property damage.

1.2 Automotive Radar

On 9th June, 1904 Christian Hülsmeyera German inventor was the first to use radar waves to demonstrate his "anti-ship-colliding system" [6]. He got a German patent on 2nd August, 1904 for the detection device and then later on in the same year he obtained another patent from UK for his complete design of the Telemobiloskop, or Remote Object Viewing Device [6], [7].

The use of radar is more beneficial than devices like lasers and infrared visions equipment as radar can function through bad weather condition like rain, snow or fog. The current market has radars with frequencies of 24GHz and 77GHz. The choice of these frequencies is based on the requirement of small antenna size, relative attenuation of spectrum and fast attenuation of radio signals [8]. Automotive radars are used to calculate the target range, target velocity and azimuth angle in a short time for multiple target situations.

1.3 The MEMS Radar

Automotive radar application system is analyzed in accordance with the range it covers. Long range radars (LRR) and medium range radars (MRR) are used in cruise control and collision avoidance systems and short range radar (SRR) is used in collision avoidance, crash-prevention and parking-assist systems.

Previous discussion ascertained the importance of automotive radars in reducing the numbers of road traffic accidents, however lower cost and reliable performance are also much needed to improve the road conditions all over the world. The cost of the radars needs to be decreased to ensure cars at various price levels can afford to be equipped with it in order to enhance road safety.

MEMS based design offers the advantage of fabricating low cost, batch production of RF components like RF switches, Rotman Lens that can be used to apprehend compact high performance radar in a small form factor. Such a MEMS based radar system is being developed in the University Of Windsor, Ontario, Canada. The block diagram shown is Fig 1 exhibits the major components of the existing automotive radar system.

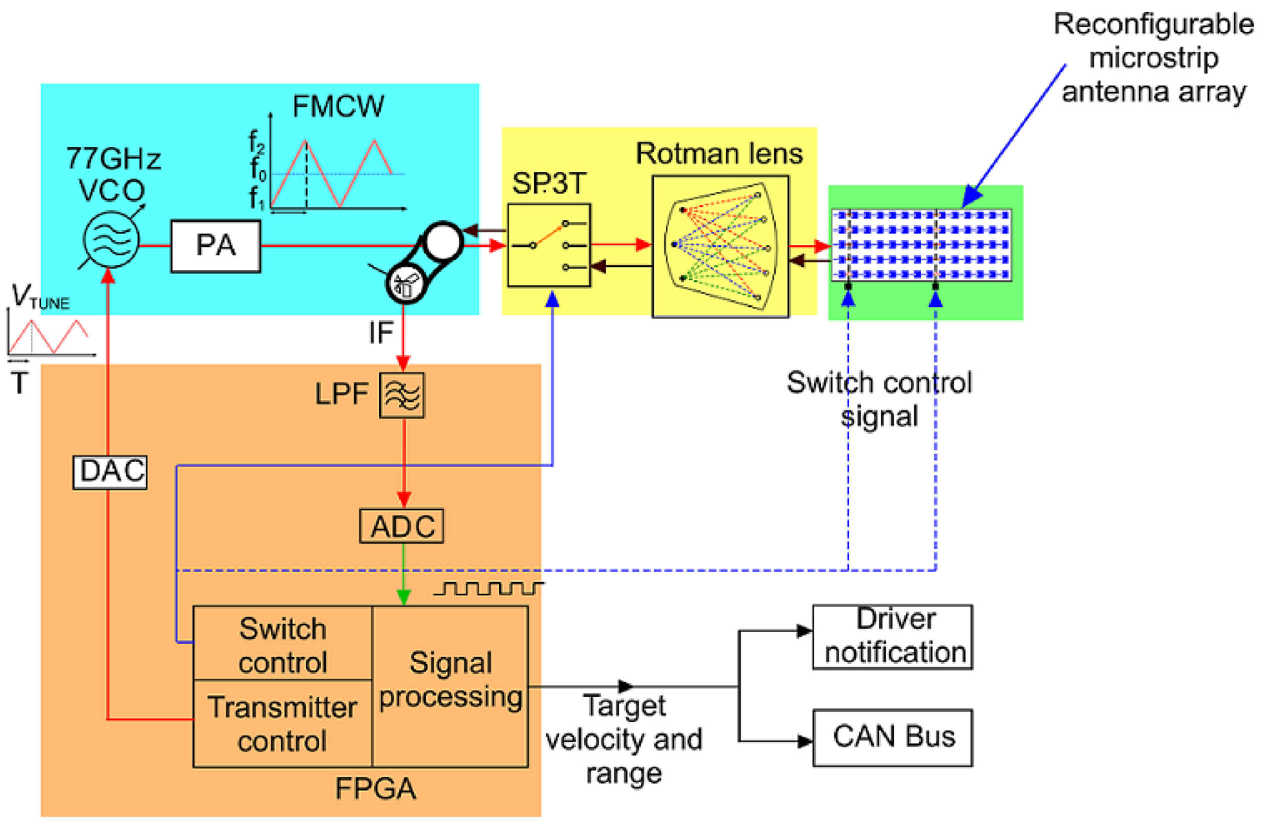


Figure 1.1 Block Diagram of the Windsor Automotive Radar System

The main components used in the Windsor Automotive Radar System are:

- FPGA/ ASIC implemented controller
- MEMS SP3T RF switches
- Microfabricated Rotman lens
- MEMS reconfigurable microstrip array antenna
- 77GHz transceiver chipset
- MEMS SPST RF switch

The tri-mode radar coverage of the current design in shown in Fig 2 (a) and (b)

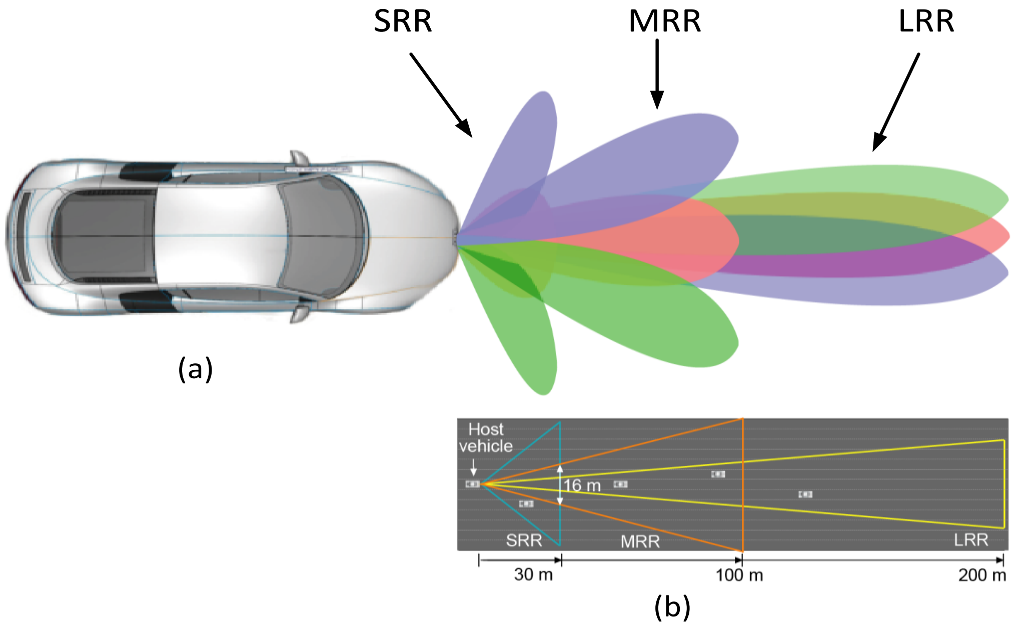


Figure 1.2 (a) The beam coverage of the Tri-mode radar and (b) the distance covered in Meter

1.4 Operating principle of the MEMS Radar:

1. A voltage controlled oscillator (VCO) is attuned by an FPGA implemented control circuit which generates a triangular signal (V_{tune}). The VCO then generates a linear frequency modulated continuous wave (LFMCW) signal with a bandwidth of 800MHz, 1400MHz and 2000MHz respectively for Long Range Radar (LRR), Medium Range Radar (MRR) and Short Range Radar (SRR) with a center frequency of 77GHz.
2. The signal generated is then fed into a MEMS SP3T switch.
3. The SP3T switch regulated by an FPGA implemented control circuit sequentially switches the LFMCW signals between the three beams of the Rotman lens.
4. The LFMCW signals after travelling through the Rotman lens cavity arrive at the array ports of the Rotman lens. The time-delayed in-phase signals are then fed into a microstrip antenna array which radiates the signal in a specific direction.

5. The microstrip has SPST switches embedded into it. The scan area of the antenna array depends on the antenna beamwidth, which in turn depends on the number of patches in the microstrip. Increasing the no of patches makes the beam narrower.
6. The beam from the Rotman lens can be steered across the target area in steps determined by a predefined angle. This steering can be done by the sequential switching of the input signal among the beamports of the Rotman lens.
7. The existing Rotman lens can transmit and receive signals. An FPGA based control circuit controls the operation of the receiver SPST switch so that the signal output at a specific beamport of the receiver Rotman lens can be mixed with the corresponding transmit signal.
8. The received signals obtained from the SP3T switch are then fed into a mixer where it is mixed with the transmitted signal to generate Intermediate Frequency (IF).
9. The IF generated are then passed into an Analog to Digital (ADC) converter, where they are sampled and converted into digital signals.
10. Lastly the output from the ADC is then processed through an FPGA implemented algorithm to calculate the range and velocity of the target detected.

The objective of this thesis is the development of an FPGA implemented algorithm to utilize a MEMS Tri-mode Radar System to detect the range and velocity of targets for three different ranges. The designed radar will certainly triumph over the use of three separate radars for long, medium and short range existing in the current market.

1.5 Research Hypothesis

The FMCW radar requires a high chirp bandwidth to improve the resolution of range and velocity and the bandwidths chosen for the design of the tri-mode radar ensures better results. Furthermore the design of the FPGA based control and signal processing algorithm can be implemented as an ASIC which offers batch fabrication of the system at a low production cost for high volume production. Accumulating the rest of the MEMS components mentioned above the target system will offer a high performance compact form of radar architecture. This will amend the highway safety situation and lower the number of road collisions which will help save lives and avoid property damage.

1.6 Research Motivation

It goes without saying that the idea of avoiding a collision is always preferable than crash protection for automotive safety scheme. The introduction of radar in the automotive industry occurred in the late 50's. In the 70's profound radar developments embarked starting at microwave frequencies. However the use of radar in automotive market became possible in the 90's where it used for safety and security purpose for the driver [9].

Radar systems in the 77GHz frequency domain give good performance in range and azimuth angle and hence can be used for several purposes to increase the efficiency of the safety system in a car [2]. It can be applied for several automotive applications like:

- For ACC and Cut-in and Stop & Go situations the short range radars make use of wider beam rather than the directive long range sensor.
- Parking assistant with more precision, longer range frequent update rates than the conventional ultrasonic systems.
- Using for Blind Spot Detection with a low cost technology.
- Pre-crash detection with fast detection rates.

The aim of the thesis is to develop MEMS tri-mode FMCW radar for the detection of range and velocity of targets for three ranges long, medium and short to enhance the automotive safety systems.

1.7 Principle Results

A LFMCW radar signal processing algorithm for FPGA/ASIC is created to control the operation of a MEMS SPST switch to dynamically alter the beamwidth of the antenna array mentioned in the radar system above to switch the radar constantly from SRR to MRR and then to LRR mode. The performance specifications achieved from the developed system are tabulated below:

Table 1.2 Radar Simulation results

Parameters	SRR	MRR	LRR
Operating Frequency	77GHz	77GHz	77GHz
Bandwidth	2000MHz	1400MHz	800MHz
Maximum Distance	30m	100m	200m
Range Resolution	0.07m	0.11m	0.19m
Maximum target range error	2cm	4cm	5cm
Maximum relative velocity (approaching and receding target)	300km/h	300km/h	300km/h
Velocity resolution	6.84km/h	6.84km/h	6.84km/h
Beam width (in degrees)	80	20	9

1.8 Thesis Organization

Chapter 1 emphasizes the importance of the work and result to this present day. It recapitulates the significance of the radar technology and their function in the automotive industry. A report of the IRTAD on road safety data is also given in this chapter in Table 1.1 tabulating the statistics of the fatality rate in different regions.

Chapter 2 encapsulates the existing literature of the radar technology and its applications in the automotive industry. The chapter also provides a good background of the proposed MEMS radar system.

Chapter 3 provides a profound mathematical and conceptual background of the radar technology, emphasizing on the LFMCW radar theory. The concept of the Cell-Averaging Constant False Alarm Rate and the digital signal processing tools used in the design is also discussed in this chapter.

Chapter 4 explicates the radar signal processing of the proposed design with block diagram and design parameters. Short descriptions of the components used in the design are also provided here.

Chapter 5 comprises of MATLAB modeling and simulation of the design. Results from MATLAB are given here with relevant graphs for single and multiple target situations for the three mode radars.

Chapter 6 constitutes the hardware implementation of the design with RTL schematics of the blocks used and table comparing the results obtained from both MATLAB modeling and hardware implementations.

Lastly Chapter 7 contributes with ideas that can be worked upon in future for further improvement of the tri-mode radar design and gives a conclusive summary of the project

CHAPTER 2: Literature Survey

○In this chapter the literature of the existing radar system will be reviewed with classification of the different radar systems available. The chapter also comprises the study of the FMCW radar over other radar types like pulsed Doppler and frequency shifting; establishing the reason for choosing FMCW radar for the target automotive radar design. The chapter covers a survey of the platform chosen for the radar design portraying its features and advantages. The state-of-the-art in the automotive radar system is also discussed in the later part of this chapter.

2.1 Literature review

The principle of radar is based on the characteristics of electromagnetic waves and the reflections properties from different materials [10]. Around 1987-88 Hienrich Hertz with his revolutionary experiments, was able to exhibit that metals and dielectric objects can reflect electromagnetic waves. In 1922 Guglielmo Marconi gave a conceptual idea about detecting object using radio signals, although it was in 1933 when he was able to demonstrate a working device [11]. Some of the applications of radar systems in the modern world are satellite radar for altitude mapping and surveillance, weather detection, synthetic aperture radar (SAR) and advanced moving target indicators (MTI) for automobiles [12]

The radar system classification can be best described using Figure 2.1

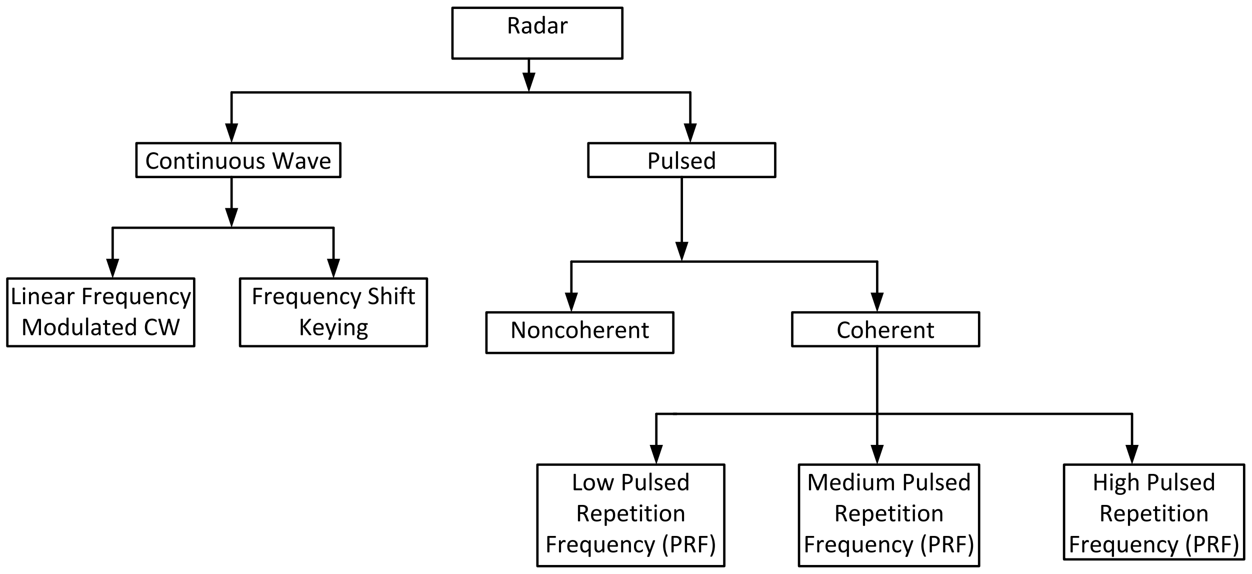

Radar
Continuous Wave
Pulsed
Linear Frequency Modulated CW
Frequency Shift Keying
Noncoherent
Coherent
Low Pulsed Repetition Frequency (PRF)
Medium Pulsed Repetition Frequency (PRF)
High Pulsed Repetition Frequency (PRF)

Figure 2.1 The Radar Classification

2.2 *Pulse Doppler Radar*

Pulsed radar transmits a high frequency modulated pulse of high power. The transmitter remains inactive for a fixed period of time during which it awaits for the received signal from the target detected. The range between the target and the radar can be calculated by measuring the round trip propagation delay between the transmitted and the received signal. The relative velocity of the target with respect to the radar can be determined from the Doppler shift frequency of the received signal. The Doppler shift frequency f_d is found by the difference between the received and the transmitted frequency. The Doppler shift is used to describe the motion of the target relative to the source from which the transmitted signal is being generated. An approaching target generates a positive frequency shift whereas the value becomes negative for a receding target. The following equations are used to calculate the range and radial velocity in a Pulsed Doppler Radar system [13], [10]:

Range : $$R = \frac{cTp}{2} \qquad (2.1)$$

Radial Velocity : $$v_r = \frac{cf_d}{2f_t} \qquad (2.2)$$

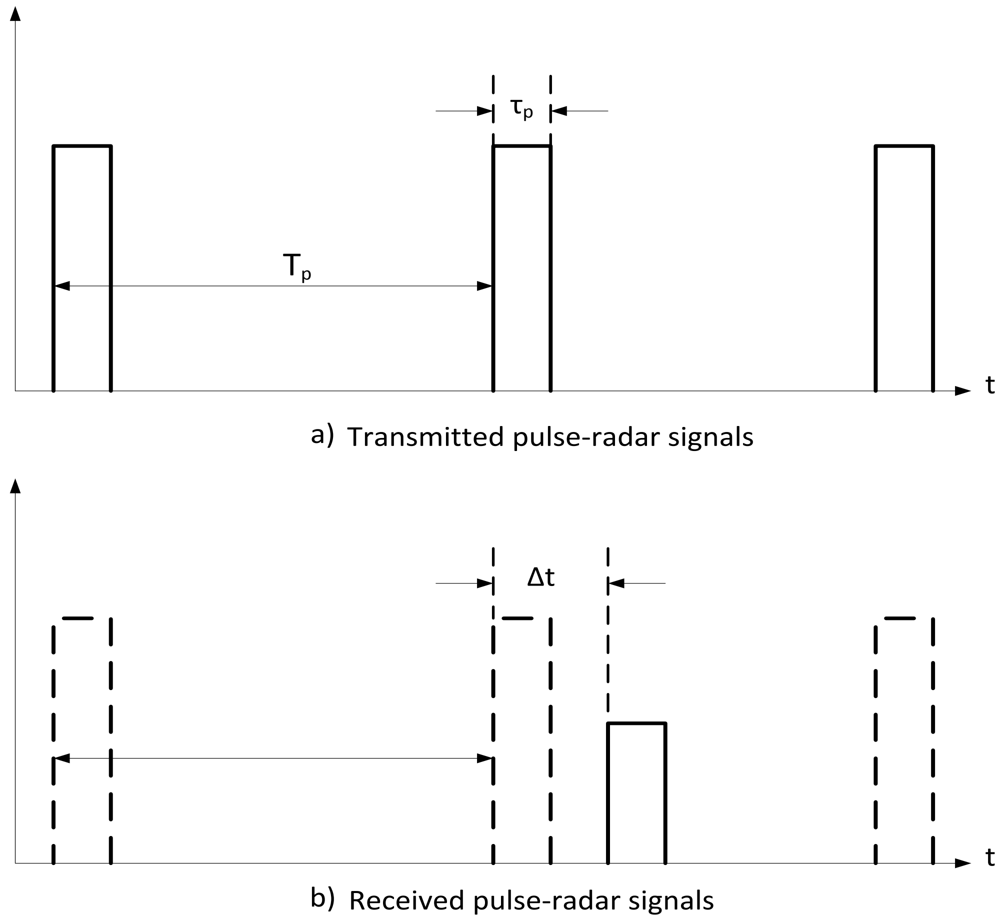
where c is the speed of light

T_p is the pulse repetition rate

f_d is the Doppler shift frequency

$f_t$is the transmitted frequency

The figure below shows the transmitted and received Pulse-Doppler radar signal where the distance to the target is calculated to be $T_p + \Delta t$. Here Tp is the pulse repetition rate, τ_p is the pulse width and Δt is the propagation delay.



τ_p
T_p
t
a) Transmitted pulse-radar signals
Δt
t
b) Received pulse-radar signals

Figure 2.2 Time dependent behavior of a) transmitted and b) received signals in a pulse-radar system

2.3 Continuous Wave Radar

Continuous Wave (CW) radar transmits a signal continuously with a selected frequency. The pure sinusoidal signal can be represented by $\cos 2\pi f_o t$ where the echo from the stationary target and the clutter will concentrate at f_o. When a moving target is detected the echoes received from it will shift the center frequency f_o by f_d, the Doppler frequency. The radial velocity of the target is then calculated by measuring the difference between f_o and f_d. Due to the continual nature of the signal emission of CW it is not possible to calculate the range unless modifications are made [13].Continuous wave can be implemented in two ways: Frequency Shift Keying (FSK) and Frequency Modulated CW (FMCW). Frequency Shift Keying modulation makes use of frequencies steps over Coherent Processing Interval (CPI) of length T_{CPI} to calculate the range. The Frequency Modulation on the other hand makes use of frequency chirp in a sinusoidal, saw-tooth or triangular form to calculate the range and velocity which will be discussed in detail in the next chapter [14]. The transmit waveforms for the two types of CW is shown in Figure 2.3.

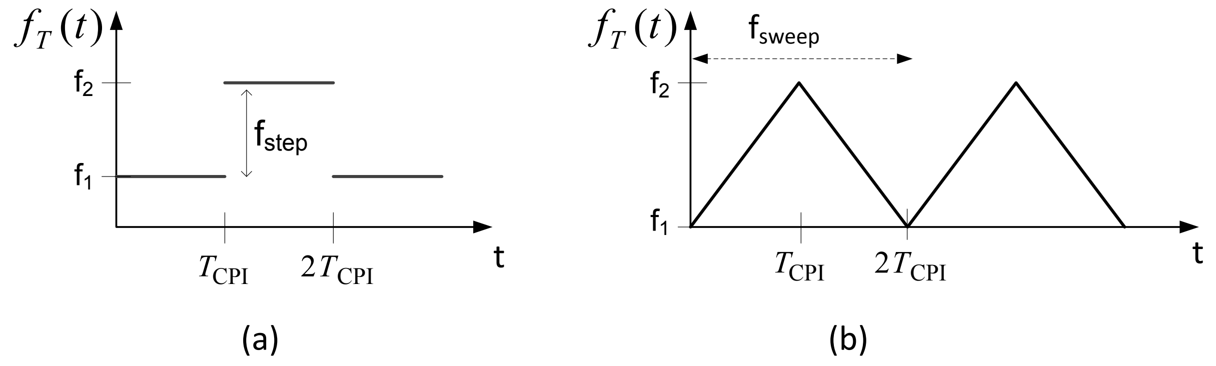


Figure 2.3 Transmit signal frequency of (a) FSK and
(b) Frequency Modulation showing a triangular waveform

The range and the radial velocity for the FSK is calculated using the following equation:

Range : $$R = -\frac{c \cdot \Delta\varphi}{4\pi \cdot f_{step}} \tag{2. 1}$$

Radial Velocity : $$v_r = -\frac{f_d \cdot \lambda}{2} \qquad (2.\ 2)$$

Here, $\Delta\phi$ is the phase difference in the two frequency peaks f_1 and f_2

f_d is the Doppler Shift frequency and

λ is the operating wavelength .

2.4 Radar type preference for the designed project

From the discussions above comprehended the two types of radar. The Pulsed Doppler, FSK and FMCW are characterized by their waveform, power at which they operate, hardware required for implementation, cost in the computation process and the applications in which they can be used. For this project Linear Frequency Modulated Continuous Wave (LFMCW) radar has been chosen. The reason for choosing this radar was reinforced by the list of disadvantages of the Pulse-Doppler Radar and the FSK radar.

Disadvantages of the Pulsed Radar for automotive application:

- The velocity calculation gets constrained when the Doppler frequency becomes equal to a multiple of the PRF causing blind speed situation.
- To avoid blind speed high PRF can be used but that results in range ambiguity.
- For selection of the PRF it has to be maintained so that blind speed is not created near the target's expected speed.
- For automotive application it requires relatively high power.
- Difficult to detect short range targets.

Disadvantages of FSK-CW radar:

- Does not allow any target resolution in range.
- It has phase discontinuities at the timings of frequency shifts which insinuate phase noise.
- Large Coherent Processing Interval (CPI) is needed to avoid range ambiguity.

Advantages of LFMCW over these shortcomings from the two types of radar are:

- Precise target range calculation.
- Determination of the velocity of the selection target.
- Low measurement time and low computation complexity.
- Less affected by clutter and atmospheric conditions.
- Power rating is lower compared with that of Pulse Doppler.
- Continual results can be obtained as opposed to Pulse Doppler.
- Is not effected by blind speed situation.
- Can also be used to detect short range targets.

Evaluation of the above points elucidates the use of Linear Frequency Modulated CW for the current design of the MEMS Tri-mode radar system.

2.5 Generating and tuning of the frequency

The RF radar signal is generating using a Voltage Control Oscillator (VCO).As it is in the case of Pulse Doppler and FSK CW radar a constant frequency pulse and CPI respectively are generated whereas for a LFMCW the VCO is attuned using a triangular modulating signal to output a frequency chirp. With the generation of Linear FM rises the concern for non-linearity errors in the output signals. The quality of the linear FM signals generated is extremely vital for radar purposes as it has influence on the radar range resolution and its accuracy to a certain degree [15]. The following equation defines the FM linearity:

Linearity $$\delta = \frac{|f_e(t)|_{\max}}{B} \tag{2. 3}$$

Here $f_e(t)$ is the difference between the instantaneous frequency of the actual output of the VCO $f_o(t)$ and the ideal linear instantaneous frequency $f_i(t)$ that is envisaged from the VCO (i.e $f_e(t) = f_o(t) - f_i(t)$). The term $|f_e(t)|_{\max}$ denotes the maximum absolute value of $f_e(t)$. B is the bandwidth of the linear FM signal.

The non-linearity in the VCO also affects the performance of the system. Generally all analog VCOs exhibit some non-linear attributes because of the varactor non-linearity. A varactor is a semiconductor device in which if a voltage is applied at the boundary of the semiconductor material and an insulator the device capacitance is affected. The following equation defines the transfer function of the VCO [16]:

$$K_o = f_e(t) - \Delta V \tag{2. 4}$$

Where ΔV is the change in the modulation voltage.

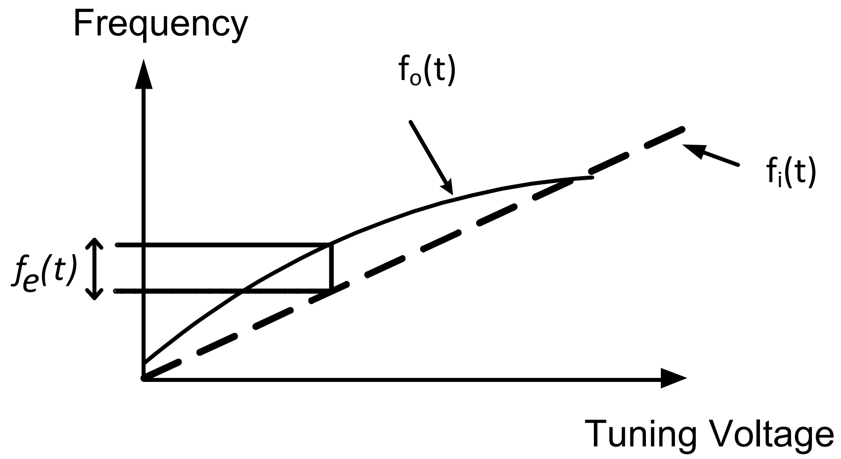


Figure 2.4 VCO Transfer function

2.6 Selection of the Development Platform for the Tri-Mode Radar Signal Processing System

The Transmitter control aggregates the radar signal generation, tuning and linearity of the waveform; attributes which are vital for the LFMCW radar as it needs linear frequency sweep. The signal generation and sweep modulation is done using a digital approach. Previously analog Phased Locked Loop (PLL) with a VCO was used for the chirp generation technique, but was replaced by digital technology. The choice to adapt a digital approach is because for radar system designs digital electronics vindicates better frequency response, excellent linearity in chirp generation and less susceptible to noise [16].

In the digital implementation of the transmitter for the proposed design the signal processing algorithm is based on Digital Signal Processor (DSP) and Field Programmable Gate Array (FPGA). The DSP part comprises of the window function, FFT, CA-CFAR and the target information calculation.

The analog portion of the design is done using the Analog to Digital Convertor (ADC) board (AD7x76/77CBZ). The throughput rate of the device is 3MSPS (Mega sample per second) and is a 12-bit 6-leadt TSOT package. It operates from a single 2.35 to 3.6V power supple. The ADC board has a low power consumption rate and also the clock speed management is very flexible. In addition to this the product comprises of a low noise, track-and-hold amplifier with a wide bandwidth which can handle input frequencies up to 55MHz. The analog input range can vary from 0 to VDD and has no delay during pipelining process. It can perform a standard successive approximation analog to digital conversion with accurate control of the data sampling with the $\overline{CS}$ pin and the serial clock, allowing the device to interface with microprocessors or DSPs [17].

A Field Programmable Gate Array (FPGA) is a semiconductor device encompassed of many logic blocks with configurable interconnections between them. The firmware is a combination of hardware and software of which the software implies to the data or program written into a ROM (Read only Memory) needed to control the hardware. The programmable function logic of the FPGA can be compiled to work as basic logic gates such as AND, OR, XOR or INVERT. It is also possible to configure it for more complex functions such as decoders and simple mathematical functions. FPGAs are capable to perform pipelining and parallelization process.

FPGA are becoming extremely popular for their exceptional performance in radar system designs. Its ability to optimize intellectual property (IP) core implementations for acute compute-intensive digital signal processing algorithms like FFT has made it become one of the building blocks in advanced radar design [16], [18].

FPGA contributes in reducing implementation area and improve throughput by accepting no-standard word-length sizes and semi- or full-parallel signal processing. In addition FPGA based emulation platform can offer real-time prototyping of ASIC (Application Specific Integrated Circuits) logic which helps with the implementation and verification of the designed system in an environment related to the target system.

In DSP design, initially the development and analyzing of the algorithm is done in a floating-point environment such as C/C++ or MATLAB. The algorithm is then converted to a fixed-point model and the unison is verified. This is done because hardware specifications are based on fixed-point representations and are also used to manually create RTL models and test benches. The RTL design associates with the method of representing a sequential circuit as a set of registers and transfer functions which describe the data flow in between the registers. The design is usually simulated at the RTL level to confirm that it is functional. A complete design environment of FPGA is offered by Xilinx (ISE), Altera and Mentor Graphics. Currently the use of intellectual property (IP) cores and customizable designware for common DSP functions like FIFO and FFT are available which helps to reduce development time and makes the system design more efficient [19].

From the study above it can be confirmed that the FPGA development platform is optimal for the radar design. A single mode radar sensor based on FPGA technology has already been designed by the MEMS team in the University of Windsor [20].

2.7 State- of- the- art Automotive Radar

The automotive radar research first started in the late 50's and by 70's radar development started at the microwave frequencies, however it wasn't until in the 90's commercialism of the automotive radars began. The aim of the automotive industry is to configure sensor for comfort and safety purpose in terms of operation, robustness, reliability and dependence on unpropitious weather conditions, cost etc. Although passive safety systems like parking aid and airbag is very helpful in the reduction of road accidents but the implementation of active safety systems like Adaptive Cruise Control (ACC) and automotive braking system further mitigates the problem as they affect the vehicle dynamics directly.

Among the first warning systems was the parking aids, collision warning systems and ACC. Greyhound installed more than 1600 24GHz radar sensors in their buses and experienced a 21% reduction in the number of accidents in 1993 compared with that in the previous year. In 1995 Japan was the first one to commercialize ACC where they favoured Lidar-ACC. In 1999 US Company Mercedes-Benz introduced the 77GHz "Distronic" into the S-class which was followed by other preeminent models like BMW 7 series, Jaguar (XKR, XK6), Cadillac (STS, XLR), Audi, A8 and Phaeton. In 2003 the Japanese Companies Toyota and Honda introduced active braking assistance system for preventing collision based on 77GHz Long Range Radar (LRR) technology. Mercedes-Benz improved their PRE-Safe break with the introduction of DISTRONIC plus in 2006 which makes use of two Short Range Radar (SRR) sensors behind the front bumper and a LRR sensor in the radiator grill to assist the driver to prevent collision by providing collision-warnings and assistance with the braking system [21]. The following figure shows the range covered by the DISTRONIC plus implemented in the new E-Class and model year 2009 S-Class Mercedes-Benz. The radar covers 200m for the LRR instead of the 150m previously used; also the new radar has Medium range coverage of 60m with 60 degree beam coverage. In addition to these the SRR sensors are still used with coverage of 80 degree beam width and range of 30m [22].

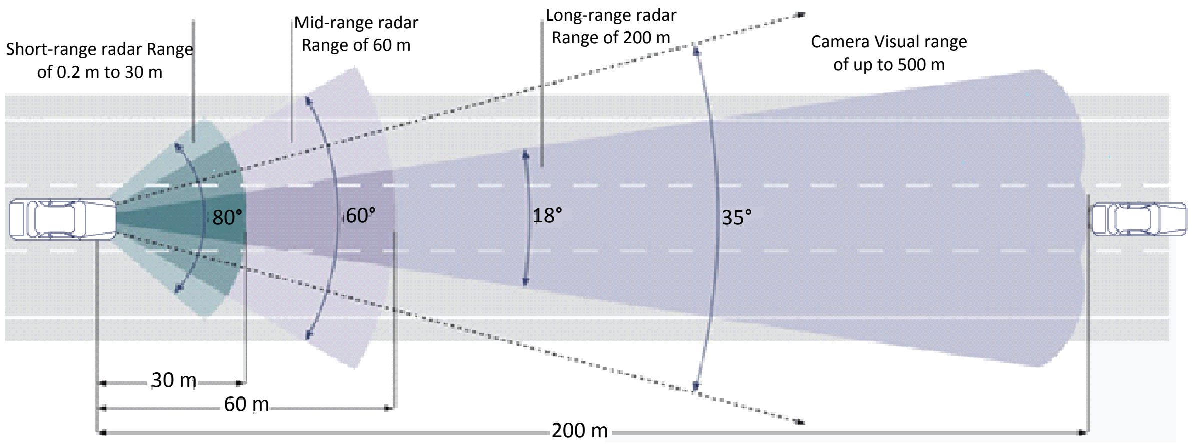

Short-range radar Range of 0.2 m to 30 m
Mid-range radar Range of 60 m
Long-range radar Range of 200 m
Camera Visual range of up to 500 m
80°
60°
18°
35°
30 m
60 m
200 m

Figure 2.5 DISTRONIC Plus

In 2004 the European Commission delineated the frequency 77GHz- 81GHz as the allocated assigned frequency for the SRR [23] .However the use of 24GHz range for the SRR till 2018 has been made official by the European Union on June, 2011 giving it time to transit it to 79GHz technology. Studies by the automotive consortium SARA member showed that using radar-based brake assistance reduced the rear-end collisions by 20 percent and a further 25 percent reduction can be seen in the severity of the accidents [24]. The following figure shows the applications of SRR implemented in Mercedes-Benz:

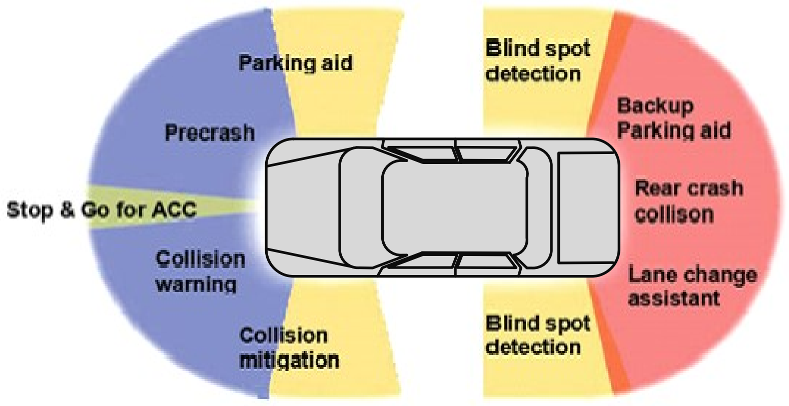


Figure 2.6 Radar applications of the SRR

The transition to 77/79GHz technology for the SRR has prompted the German Federal Ministry of Education and Research (BMBF) to fund projects KOKON (2004 to 2007) and RoCC (Radar- on-

Chips) (2008-2011) to develop a cost efficient platform. The outcome of the research is the successful development of a low cost and sturdy platform based in the SiGe technology although further technical and operational studies need to be done to operate the sensor in real-time system [23].

A crucial facet for the integration of the radar sensor in vehicles is the component size. The dimension of the automotive radar is determined by the antenna aperture. For the 77GHz technology the antenna size can be decreased for a given beamwidth requirement enabling it to achieve a better angular resolution. On the other hand in order to achieve the same performance like that of the 77GHz technology, the 24GHz technology needs to increase the antenna size by three times. In future the short range radar will be designed with an absolute bandwidth of up to 4GHz, this renders to about 5% of the relative bandwidth with the 77GHz whereas it is about 17% using the 24GHz, thus making the design of antennas and wavelength related components easier. One other advantage for the choice of the 77GHz is that it allows the combination of high transmission power (> -40 dBm/MHz) and high bandwidth (> 250 MHz) which yield long range operation together with high distance separability concurrently; whereas is not possible at 24GHz technology [2].

With the increase in driver assistance systems in the current market more functions are being insinuated like improvement in the ACC, where the speed can be automatically varied or brought to complete zero depending on the situation. In addition to this other functions like lane changing assistance (LCA) and cross traffic alert (CTA) are also being introduced. In order for these multifarious aspects to functions the use of Medium range was announced in 2011 to be used in alliance with the LRR and SRR. To specify the MRR is used for the LCA and CTA. The following figures portrait the use of the LRR and MRR followed by illustrations of various technologies existing for driving assistance [2], [10].

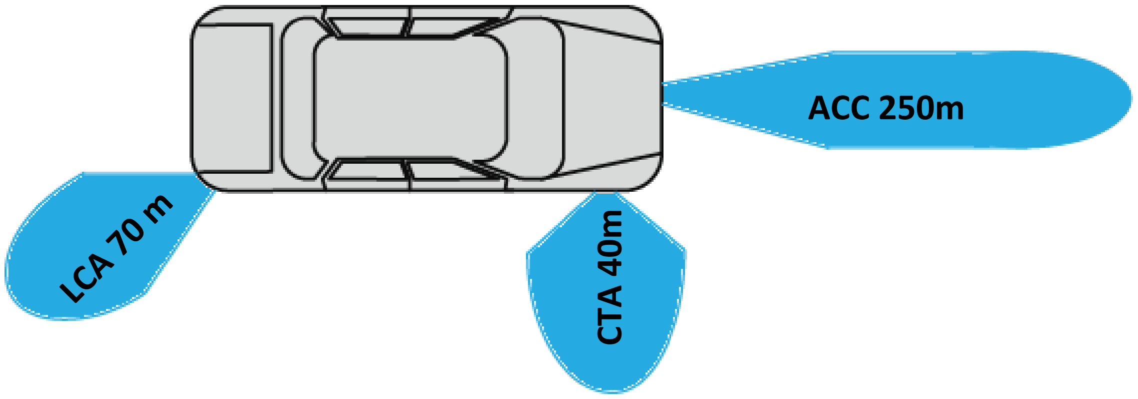


Figure 2.7 Field of view and range covered for three functions

Table 2.1 gives a brief review of the existing radar sensors in the market with significant differential factors in between them [2].

Table 2.1 Comparisons between existing sensors in the current market

Parameter	**Bosch LRR3**	**Conti ARS 300**	**Denso DNMWR004**
Dimensions	74x70x58 mm	120x90x49 mm	78x77x38 mm
R_{max}detected	250 m	200 m	150 m
Horizontal Field of View	30°	58°/17°	20°
Number of beams	4	15/17	5
Beamsteering technique	Fixed-beam	Mechanical	Electronic
Multirange capability	Single	Multiple	Single

The Bosch LRR3 [Figure 2.8(a)] sensor was in the market since 2009 with the features like dielectric lens antenna providing high gain to achieve maximum distance of 250m. The device was the first in the market to use SiGe integrated circuits at 77GHz. Bosch also launched the MRR sensor [Figure 2.8 (b)] using SiGe MMIC. Among the first radar sensors was the ARS300 [Figure 2.8 (c)] by Continental, which was used by Mercedes. It key feature was the scanning antenna which was based on a dielectric waveguide like a constant rotating drum with a special grating structure [25]. A SRR sensor with an integrated planar antenna array was introduced by IMST in Low Temperature Cofired Ceramics (LTCC) technology [Figure 2.9 (c)] operating at 24GHz [26].

(a)

(b)

(c)

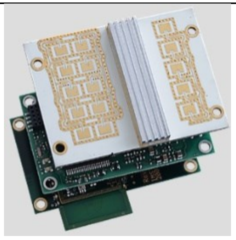

(d)

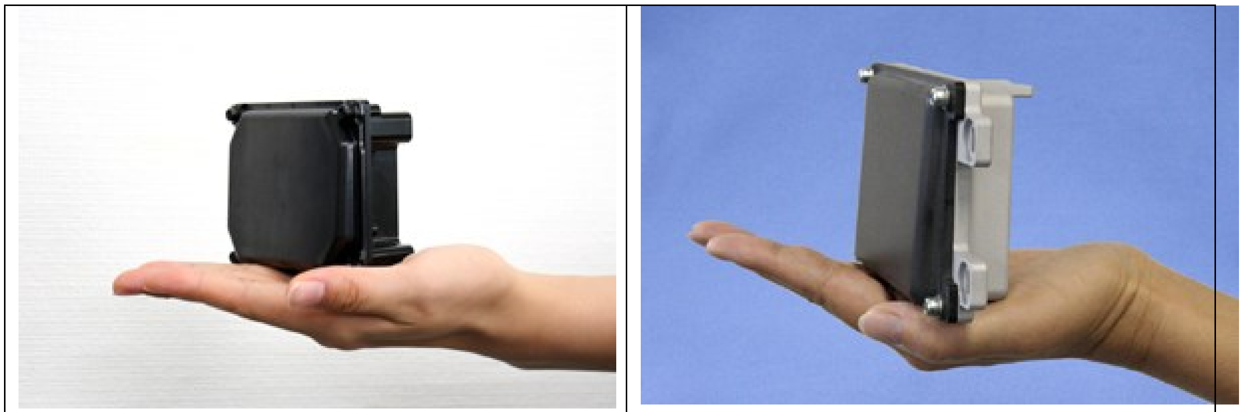
Figure 2.8 (a) Bosch LRR3 sensor (b) Bosch MRR sensor (c) Continental ARS300 (d) IMST SRR sensor

In Table 2.2 a brief overview of former generation radars are given with the name of their manufacturer and some key features; a report from Fujitsu Ten Ltd [27].

Table 2.2 Former generation radars highlighting some key features

Manufacturer	Our company	ADC	Delphi	Bosch	Honda elesys	Denso	Hitachi
Appearance							
External Dimensions (mm)	89×107×86	136×133×68	137×67×100	91×124×79	123×98×79	77×107×53	80×108×64
Modulation Method	FM-CW	FM Pulse	FM-CW	——	FM-CW	FM-CW	2-frequency CW
Detection Range	4m to 120m or greater	Approx. 1m to 150m	Approx. 1m to 150m	2m to 120m or greater	4m to 100m or greater	Approx. 2m to 150m	Approx. 1m to 150m
Horizontal Detection Angle	±8°	Approx. ±5°	Approx. ±5°	±4°	±8°	±10°	±8°
Angle Detection Method	Mechanical Scan	Beam Conversion	Mechanical Scan	Beam Conversion	Beam Conversion	Phased Array	Monopulse
EHF Device	MMIC	GUNN	GUNN	GUNN	MMIC	MMIC	MMIC

In a press release of October 2012 Fujitsu Ten Ltd declared about the development of compact 77GHz automotive radar with a 3D electronic scanning capability, which was a step up from the 2D millimeter wave radar (Press release October, 2010). However the product will be commercialized in the automotive market from 2014. The latest 3D automotive radar sensor will detect object in three dimensions i.e. it will cover "elevation" range together with distance and azimuth angle [28] .



(a) (b)

Figure 2.9 (a) 77GHz 3D Millimeter Wave Radar,(b) 76-GHz 2D Millimeter Wave Radar

A next generation driver assist technology which the three range coverage ability for better safety purpose in given the figure below:

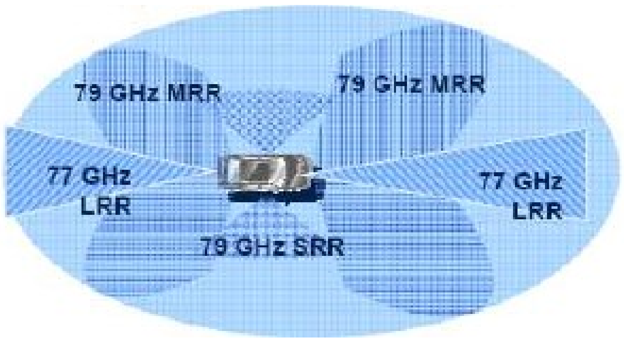


Figure 2.10 Layout of the next generation driver assistance applications

2.8 Work done in LFMCW Radar sensor design with FPGA-based platform

Some the primary manufacturers of the 77 GHz LRR sensors are ADC, BOSCH, Delphi, Denso, TRW (Autocruise), Fujitsu Ten and Hitachi. In Figure 2.11(a) a BOSCH LRR second generation radar sensor is shown which uses an analog beamforming approach. The production of the sensor started from 2004 and uses a FMCW modulation with triangular waveform. Japanese companies introduced a radar sensor with digital beamforming in 2003 (Figure 2.11 (b)). Toyota CRDL && GHz LRR sensor also makes use of digital beamforming using nine digital receiver channels (Figure 2.11 (c)) [9].

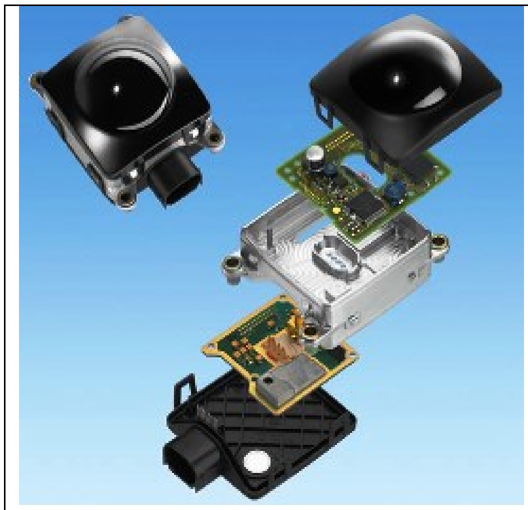
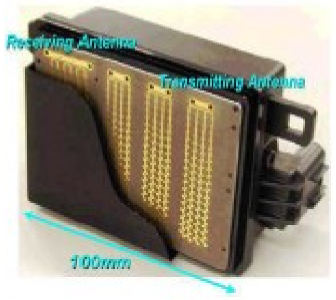
 (a)	 (b)	(c)

Figure2.11 (a) BOSCH 2nd Generation LRR (b) Denso’s 77 GHz LRR (c) Toyota CRDL 77 GHz LRR

A prior study on FPGA-based LRR radar design using LFMCW radar signal processing algorithm is done in the University of Windsor using Xilinx Virtex-5 FPGA at 100MHz. The sampling time of the system was 6.78ms and the processing time of 211.63µs. From the resources used by the developer it was seen that using Virtex-5 FPGA board only 4% of sliced registers, 23% slice LUTs, 6% of DSP48 slices and 21% of the FPGA fabric area is used [20].

The aim of this thesis is to design a tri-mode radar sensor with a faster signal processing algorithm and also optimize the accuracy of the results obtained from both the software and hardware simulations alike.

CHAPTER 3
Target FMCW Radar Design Specifications

In this chapter the mathematical models correlated with the FMCW radar is discussed in details and the relevant equations and expressions required to calculate the range and velocity of the detected targets are provided here. Detailed equation study for calculating range and velocity for both stationary and moving target scenario is discussed here. The operating parameters for the design are also identified from the mathematical studies done. A brief review is also done on additional concerns like atmospheric attenuations, temperature effects, false rate alarm, clutter removal and radar types.

3.1 Identifying operating parameters for the system design

The state-of-the-art for automotive radar system requirements for long, medium and short range radar is listed in Table 3.1 [2]. The parameters considered in this thesis are bandwidth, range coverage, range resolution, range accuracy, velocity accuracy and velocity resolution.

Table 3.1 Sensor Classification for Tri-Mode Radar System

Type	LRR	MRR	SRR
Maximum Transmit power (EIRP)	55dBm	-9dBm/MHz	-9dBm/MHz
Frequency Band	76-77GHz	77-81GHz	77-81GHz
Bandwidth	600MHz	600MHz	4GHz
Distance range Rmin..Rmax	10-250m	1-100m	0.15-30m
Distance Resolution ΔR	0.5m	0.5m	0.1m
Distance accuracy δR	0.1m	0.1m	0.02m
Velocity Resolution Δv	0.6m/s	06.m/s	0.6m/s
Velocity accuracy δv	0.1m/s	0.1m/s	0.1m/s
Angular accuracy δΦ	0.1^o	0.5^o	1^o
3dB beamwidth in azimuth ±Φmax	$\pm 15^o$	$\pm 40^o$	$\pm 80^o$
3dB beamwidth in elevation vmax	$\pm 5^o$	$\pm 5^o$	$\pm 10^o$
Dimensions	74X77X58mm	50X50X50 mm	50X50X20 mm

With regard to the above table and [20] the target tri-mode radar signal processing system is required to correlate with at least the following performance:

(a) Range: for LRR 200m, for MRR 100m and for SRR 30m

(b) Range Accuracy < 0.1m

(c) Relative velocity : -100 to 250 km/h

(d) Velocity accuracy : ± 0.1 m/s

(e) Cycle time < 6.78ms

3.2 Allocation of the LFMCW Waveform

After the detail discussion done in Chapter-2 the choice of FMCW radar was established. To be more specific the radar system chosen for this design is LFMCW which is a class of FMCW where the modulating waveform is linear. The FMCW radar however can implement other waveforms like sinusoidal, triangular and saw-tooth. The different types of radar waveforms generated are illustrated in Figure 3.1.

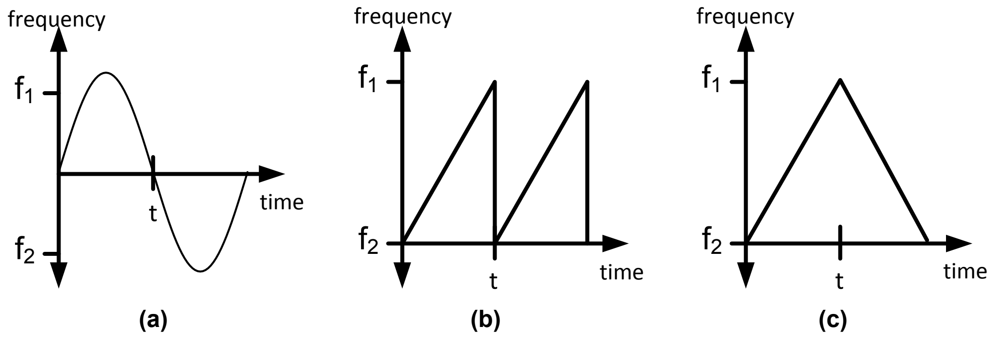


Figure 3.1 FMCW waveforms (a) Sine wave (b) Saw-tooth (c) Triangular

Sinusoidal waveform is rarely used for FMCW radar system because of the extra latency added in the computing and adjusting the wave coefficients. In addition the sine wave has less tolerance for VCO non-linearity when compared with the linear waveforms of FMCW. Although at lower operating frequencies analog modulation is possible without the need for waveform generations using digital approach.

The saw-tooth (or ramp) waveform provides only positive frequency sweep, which makes the control and electronic tuning uncomplicated. However for moving targets, the problem of range Doppler coupling arises when using the saw-tooth modulation technique [29].

The approbate technique for the design was found to be triangular modulation as the range and velocity could be calculated simultaneously. In the triangular waveform with the difference between the two equal upslope and downslope linear sweeps tantamount to twice the Doppler shift of the detected target, consequently allowing the calculation of both range and radial velocity. The other advantage of using the triangular modulation is that the different sweep directions make the system better resistant to stationary clutter and jamming signals as they have more dynamic instantaneous frequency.

3.3 The Linear Frequency Modulated Continuous Wave (LFMCW) Radar

The LFMCW technique permits the use of linear frequency sweeps (or chirp) over a selected bandwidth and measure the range and velocity using the beat frequencies from all the targets detected within the FoV (Field of View) of the radar beam. The beat frequency is the difference between the transmitted and received radar signal:

$$f_b(t) = f_t(t) - f_r(t) \tag{3. 1}$$

The chirp period and the bandwidth are key parameters for determining the refresh rate, range resolution and velocity resolution of the system. Figure 3.2 and Figure 3.3 models the triangular LFM waveform showing the beat frequency obtained from the up and down frequency sweep for a stationary and moving target.

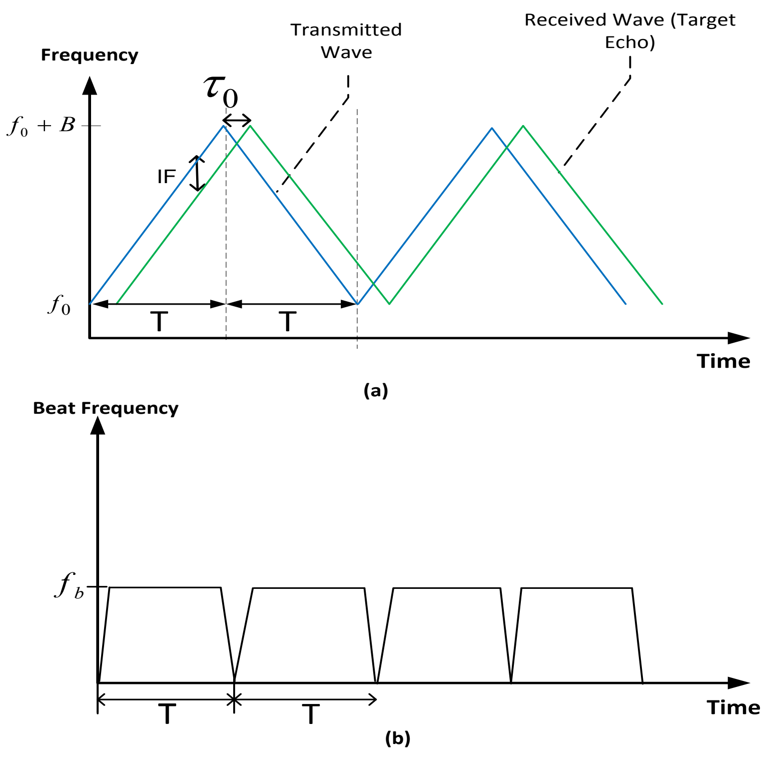


Figure 3.2 (a) Transmitted wave and received LFMCW signals (b) Beat frequency or intermediate frequency for a stationary target

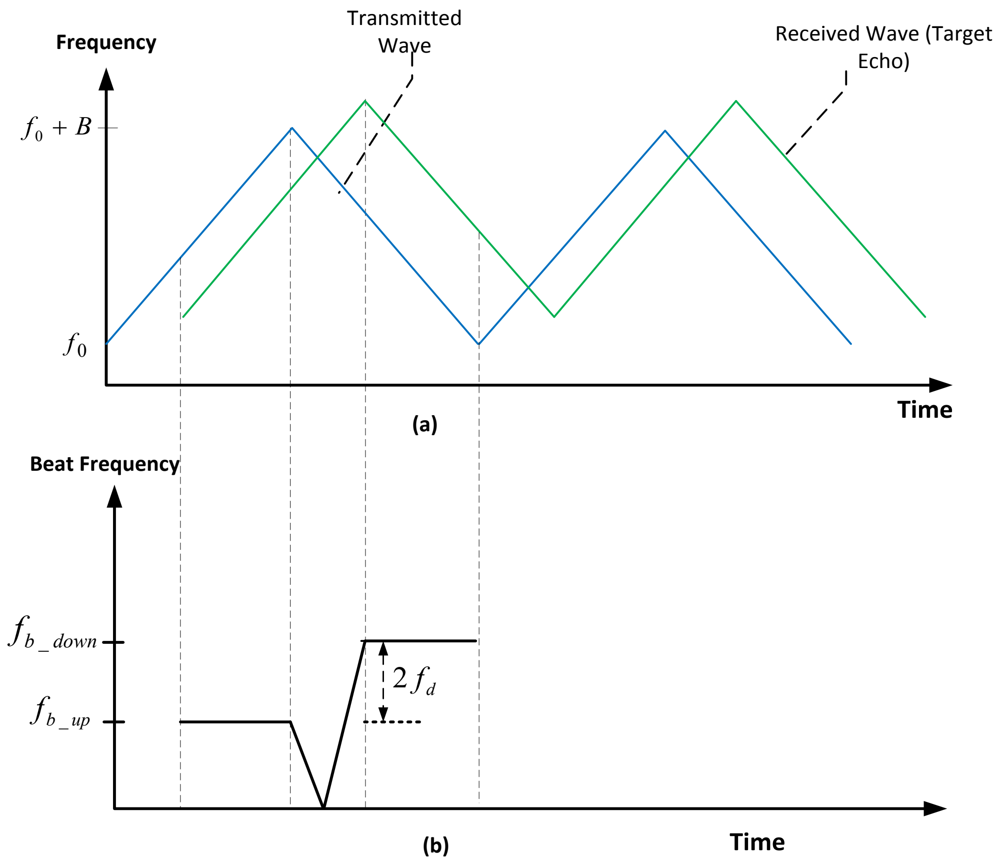


Figure 3.3(a) Transmitted wave and received LFMCW signals (b) Beat frequency or intermediate frequency for a moving target

Here,

$f_{\text{b\_up}}$	= up sweep beat frequency
$f_{\text{b\_down}}$	= down sweep beat frequency
f_{b}	= beat frequency or intermediate frequency (IF)
τ_0	= round trip delay time for the signal to be received from the target
f_{d}	= Doppler shift due to relative target velocity
f_0	= starting frequency for operation bandwidth
B	= operation bandwidth
T	= sweep duration (equal for both up and down sweep for this thesis)

3.4 *Derivation of equation for Range and Velocity*

In this section a compendious derivation of the range and velocity equations of the LFMCW radar:

$f_t(t)$	= transmitted radar signal
$f_r(t)$	= received radar signal
$k = \frac{B}{T}$	= rate of change of frequency over a single sweep

3.4.1 *Instance One: For Stationary Target Situation*

A stationary target is one which has zero radial velocity and hence has no contribution to the Doppler shift of the received signal. The transmitted signal can be defined with equation [3.2] and using the Euler's formula the complex sinusoidal equation with a base frequency of f_o modulated over a bandwidth of B Hz is given by equation (3.3) [30]:

$$v_{t1}(t) = \cos\left(2\pi f_o t + \frac{\pi B}{T} t^2\right) \tag{3. 2}$$

$$f_{t1}(t) = \exp\left(j2\pi\left(f_o t + \frac{1}{2} k t^2\right)\right) \tag{3. 3}$$

The received signal lagged by a round trip delay time of τ_0 is given by equation (3.4):

$$f_{r1}(t) = \exp\left(j2\pi\left(f_0(t-\tau_0) + \frac{1}{2} k (t-\tau_0)^2\right)\right) \tag{3. 4}$$

Mixing or multiplying the transmitted and received signals (ignoring the high frequency component) the beat frequency or intermediate frequency (IF) is obtained. When the target is stationary the beat frequency obtained is found to be equal for both up and down sweep and is represented by equation (3.5):

$$f_{b1}(t) = f_{t1}(t) \otimes f_{r1}(t) = \exp\left(j2\pi\left(f_0\tau_0 + kt\tau_0 - \frac{1}{2}k\tau_0^2 \right)\right) \tag{3. 5}$$

Taking the derivative of the phase of equation (3.5) with respect to time the instantaneous beat frequency is found which is proportional to the target range:

$$f_{\text{up}1} = \frac{d\left(f_0\tau_0 + kt\tau_0 - \frac{1}{2}k\tau_0^2 \right)}{dt} = k\tau_0 \tag{3. 6}$$

Hence for a stationary target with equal up and down seep the beat frequencies can be expressed as:

$$f_{\text{up}1} = f_{\text{down}1} = k\tau_0 = k\frac{2r}{c} \tag{3. 7}$$

Here r is the range of the target, c is the speed of the electromagnetic waves in air and k is rate of change of frequency over a single sweep. The relative range of the stationary target is calculated by the averaging the two up and down sweep instantaneous frequencies and is given equation (3.8):

$$r = \left(\frac{f_{\text{up}1} + f_{\text{down}1}}{2} \right) \times \frac{c}{2k} \tag{3. 8}$$

3.4.2 Instance two: Moving target situation

For the second situation a moving target is considered with a velocity of v_r relative to the radar sensor or host vehicle. This velocity is an additional parameter in the transmitted and received signals which occurs due to the Doppler Shift. It is denoted by $2f_0 v_r / c$ [13]. The transmitted signal is considered as the same as equation (3.3) but the received signal for the up sweep is affected by twice the amount of Doppler shift due to the two-way travel and the round trip delay of the radar wave. The resultant signal can be expressed as:

$$f_{r2}(t) = \exp\left(j2\pi\left(f_0(t-\tau_0) + \frac{1}{2}k(t-\tau_0)^2 + 2f_0\frac{v_r}{c}(t-\tau_0) \right)\right) \tag{3. 9}$$

Multiplication of the transmitted and received signals in time will give the up sweep beat frequency which is denoted by equation (3.10):

$$f_{\mathrm{b\_up}}(t) = \exp\left(j2\pi \left(\begin{array}{l} f_0\tau_0 + \left(k\tau_0 + 2f_0\frac{v_{\mathrm{r}}}{c} - 2k\tau_0\frac{v_{\mathrm{r}}}{c} \right)t - \frac{1}{2}k\tau_0^2 \\ +2\frac{k}{c}\left(v_{\mathrm{r}} - \frac{v_{\mathrm{r}}^2}{c} \right)t^2 \end{array} \right) \right) \tag{3. 10}$$

Assuming a stable computation of the instantaneous up sweep by differentiating w.r.t. time the second order and the constant terms are ignored. Hence the resultant up sweep frequency is found to be:

$$f_{\mathrm{up2}} = \frac{d\left(\left(k\tau_0 + 2f_0\frac{v_{\mathrm{r}}}{c} - 2k\tau_0\frac{v_{\mathrm{r}}}{c} \right)t \right)}{dt} = k\tau_0 + 2f_0\frac{v_{\mathrm{r}}}{c} - 2k\tau_0\frac{v_{\mathrm{r}}}{c} \approx k\tau_0 + f_{\mathrm{d}} \tag{3. 11}$$

The approximation needed to reach the expression for f_{up2} is possible as $2k\tau_0\frac{v_{\mathrm{r}}}{c} = 2k\frac{2r}{c}\frac{v_{\mathrm{r}}}{c} = 4kr\frac{v_{\mathrm{r}}}{c^2} << 1$ for bandwidths under 1GHz (LRR). The term will also produce negligible frequency values for bandwidths in tens of GHz (MRR and SRR), hence this term can be safely neglected for all the modes of radars used in this design.

For the down sweep, the Doppler shift evinces as a negative value due to the negative slope of the modulating wave. Taking into account the fact that $f_{\mathrm{d}} < B$ the beat frequency signal at the receiving end of the radar is expressed as:

$$f_{\mathrm{b\_down}}(t) = \exp\left(j2\pi \left(\begin{array}{l} f_0\tau_0 + \left(k\tau_0 - 2f_0\frac{v_{\mathrm{r}}}{c} - 2k\tau_0\frac{v_{\mathrm{r}}}{c} \right)t - \frac{1}{2}k\tau_0^2 \\ +2\frac{k}{c}\left(v_{\mathrm{r}} - \frac{v_{\mathrm{r}}^2}{c} \right)t^2 \end{array} \right) \right) \tag{3. 12}$$

Differentiation of this equation w.r.t. time the down sweep frequency for a moving target with relative velocity v_{r} is obtained.

$$f_{\text{down2}} = \frac{d\left(\left(k\tau_0 - 2f_0\frac{v_{\text{r}}}{c} - 2k\tau_0\frac{v_{\text{r}}}{c}\right)t\right)}{dt} = k\tau_0 - 2f_0\frac{v_{\text{r}}}{c} - 2k\tau_0\frac{v_{\text{r}}}{c} \approx k\tau_0 - f_{\text{d}} \tag{3. 13}$$

After studying the above dissection it can be found that the range of any target for the LFMCW radar technique can be found by adding f_{up2} and f_{down2} :

$$f_{\text{up2}} + f_{\text{down2}} = k\tau_0 + f_{\text{d}} + k\tau_0 - f_{\text{d}} = 2k\tau_0 = 2k\frac{(2r)}{c} \tag{3. 14}$$

Hence the range r can be expressed as:

$$r = \frac{(f_{\text{up2}} + f_{\text{down2}})}{2} \times \frac{c}{2k} \tag{3. 15}$$

This equation is found to be similar like equation (3.8) i.e. for a stationary target.

The relative velocity v_r can be found by subtracting f_{up2} and f_{down2} to extract the Doppler shift:

$$f_{\text{up2}} - f_{\text{down2}} = k\tau_0 + f_{\text{d}} - (k\tau_0 - f_{\text{d}}) = 2f_{\text{d}} = 4f_0\frac{v_{\text{r}}}{c} \tag*{(3. 16])}$$

Consequently the relative velocity v_r is given by:

$$v_{\text{r}} = \frac{(f_{\text{up2}} - f_{\text{down2}})}{4} \times \frac{c}{f_0} \tag{3. 17}$$

The actual target velocity relative to the host velocity v_{host} is thus found by:

$$v_{\text{target}} = v_{\text{host}} - v_{\text{r}} \tag{3. 18}$$

3.5 Generation of LFMCW radar signal using VCO

A kernel component of the state-of-the-art radar system is the use of Voltage Controlled Oscillator (VCO) to generate the FM signal. The tuning element of the VCO is a varactor diode [31]. When the VCO receives an input analog tuning voltage in the varactor diode it leads to the variation of the net capacitance which in turn contributes in the generation of an output frequency. For the current design the output frequency needs to a triangular up and down chirp for which a triangular modulating signal is required. This modulating signal can be generated using the FPGA with relative ease.

An up/down counter is needed for the modulating unit that will feed the DAC, outputting the tuning voltage to the VCO. The up and down counter is required for up sweep (zero to up) count and down sweep (back to zero) count respectively. The resolution and the refresh rates of the DAC are crucial parameters influencing the linearity of the frequency sweeps. Figure (3.4) illustrates the algorithm employed to generate the signal using a digital counter implemented in the FPGA.

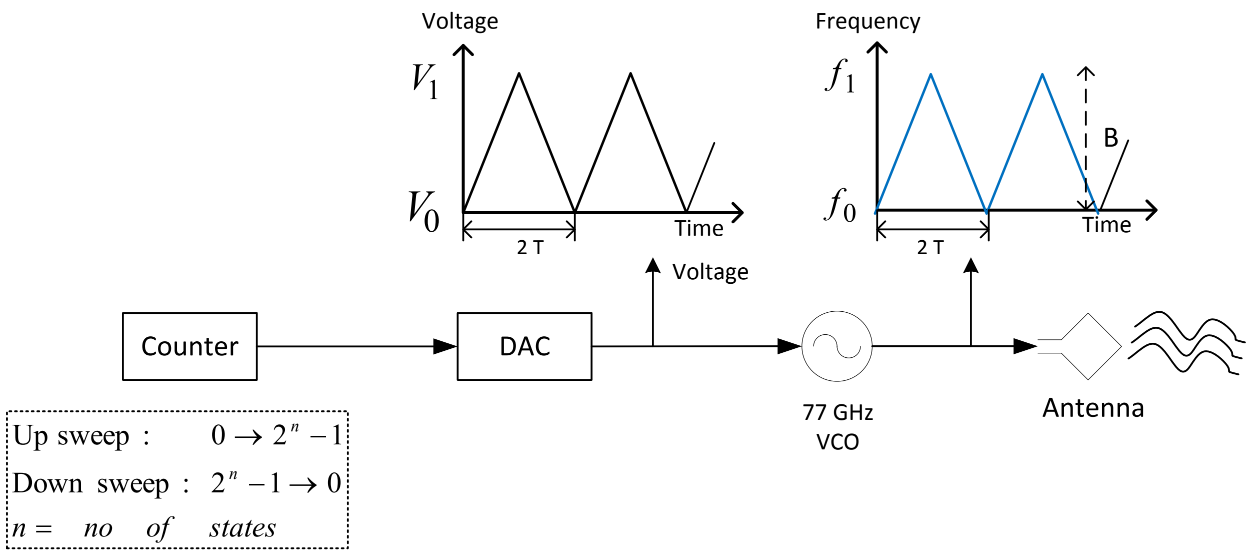


Figure 3. 4 FPGA Based Voltage Tuning To Generate Frequency Chirps By The VCO

The modulation results in a time-domain chirp signal like that shown in the following figure. An up sweep followed by an equal down sweep [20] .

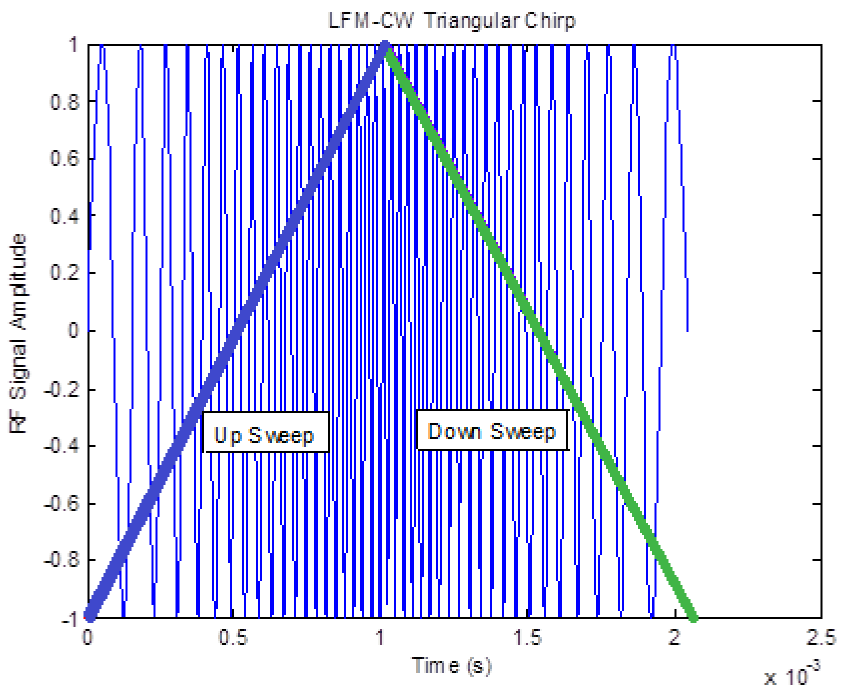


Figure 3.5 Time-domain RF signal for the LMFCW radar displaying up and down chirp

3.6 Modification of the Received signal

The echo signal received from the target needs to modified before it is passed to the digital signal algorithm. The modification is usually an analog process and is described below:

1. Low Noise Amplifier: The received signal is augmented using a low noise amplifier to oppose atmospheric and hardware attenuation.
2. Mixer: It is used to perform a time-domain multiplication of the transmitted and the received signal and output the sum and differences of these frequencies. Considering the multiplication between two signals $A_r \cos(\omega_r t)$and $A_t \cos(\omega_t t)$, the resultant IF obtained in displayed by Figure (3.5):

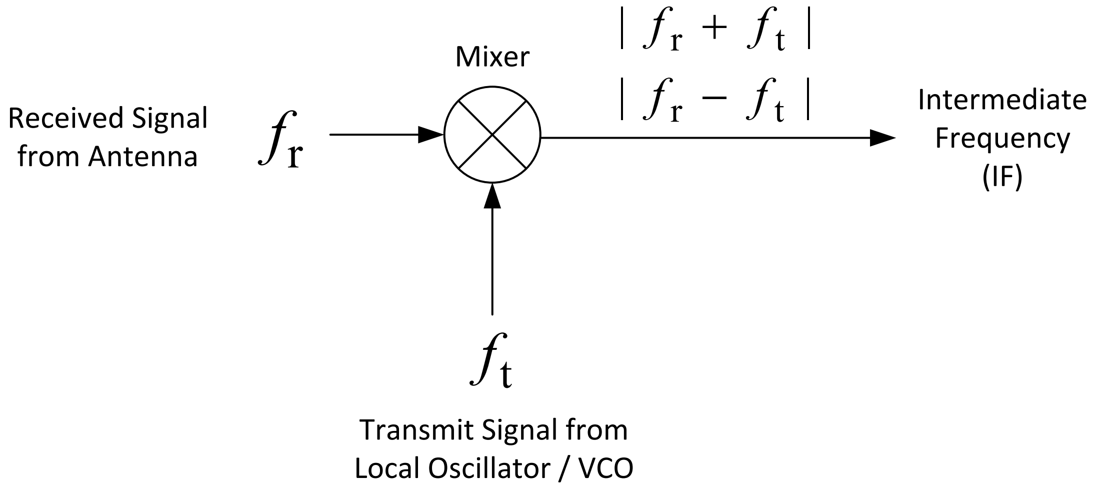

Mixer
$| f_r + f_t |$
$| f_r - f_t |$
Received Signal from Antenna
f_r
Intermediate Frequency (IF)
f_t
Transmit Signal from Local Oscillator / VCO

Figure 3.6 The Mixing Process of an RF Signal

3. Low Pass Filter: It filters out the higher frequency component to extract the desired beat frequency ($f_r - f_t$).
4. Analog to Digital Converter (ADC): The ADC samples the IF signal according to the Nyquist Theorem to avoid aliasing. The efficiency and accuracy of the overall radar processing depends on this ADC component. The output resolution of the ADC dictates the speed, accuracy of range and velocity calculation and memory usage. High resolution of the ADC ensures better accuracy, lower quantization noise and improves speed and memory requirement [32].

3.7 Digital Signal Processing for the Radar design

An outline of the cardinal digital signal processing steps needed for the system design:

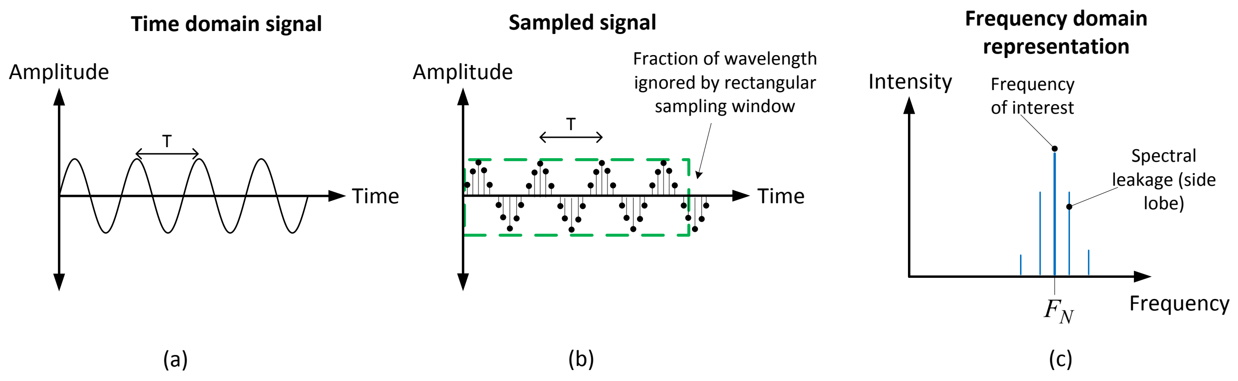
1. Windowing
2. Fast Fourier Transform for the spectral analysis
3. Cell-Averaging Constant False Alarm Rate processing

3.7.1 *Windowing*

The analog IF signal after is it was modified is sampled over a length of time by the ADC to digitize it. The sampling is done over a certain period time, e.g. t seconds. Spectral analysis, an application of window function, is used to break down a complex signal into simpler forms. This it does using a FFT, which assumes that the signal data constitutes an integral number of cycles [33].However, there is no assurance that the ADC will be sampling an integral number of wavelengths and the IF itself is distorted from the interference of noise and microwave. Sampling of the signal corresponds to the convolution with a rectangular-shaped window of uniform height. This causes spectral leakage. To avoid spectral leakage the signal has to be sampled over an infinite period of time, which is not viable. When the period of a signal is extended to a limit not equivalent to the natural period, discontinuities are observed at the boundaries. These discontinuities give rise to power leakage into the neighboring frequency bins. This can be explained using equation (3.19) and the total effect is illustration using Figure (3.6) [34], [35], [20].

$$\frac{f_{in}}{f_{sample}} \neq \frac{N_{window}}{N_{FFT}} \tag{3. 19}$$

The spectral leakage occurs when the above condition occurs, where f_{in} is the input signal, $f_{sample}$is the sampling frequency, N_{FFT} is the number of points in the FFT, N_{window} is the no of cycles in the data window.



Time domain signal
Amplitude
T
Time
(a)
Sampled signal
Amplitude
Fraction of wavelength ignored by rectangular sampling window
T
Time
(b)
Frequency domain representation
Intensity
Frequency of interest
Spectral leakage (side lobe)
F_N
Frequency
(c)

Figure 3.7 (a)Signal under investigation with period T (b) Sampled signal convolved with a rectangular window-function and, (c) Spectral leakage due to the windowing where $F_N = 1/T$.

A detail analysis is done to define the parameters and the characteristics of the different window types in order to choose the befitting window function for downsizing the spectral leakage. The real-frequency characteristics of a window is that is it a continuous spectrum consisting of a main lobe with several side lobes on either side of it. The side lobes after an interval approaches to zero. For an ideal window the ratio between the ratio between the sampling frequency and the input sinusoidal signal is exactly equal to the integral number of cycles that are within the data window i.e. (taking reference from equation (3.19)

$$\frac{f_{in}}{f_{sample}} = \frac{N_{window}}{N_{FFT}} \tag{3. 20}$$

The FFT output for such condition will display a single lobe located at the input frequency [36]. The following figure illustrates the condition described in the equation:

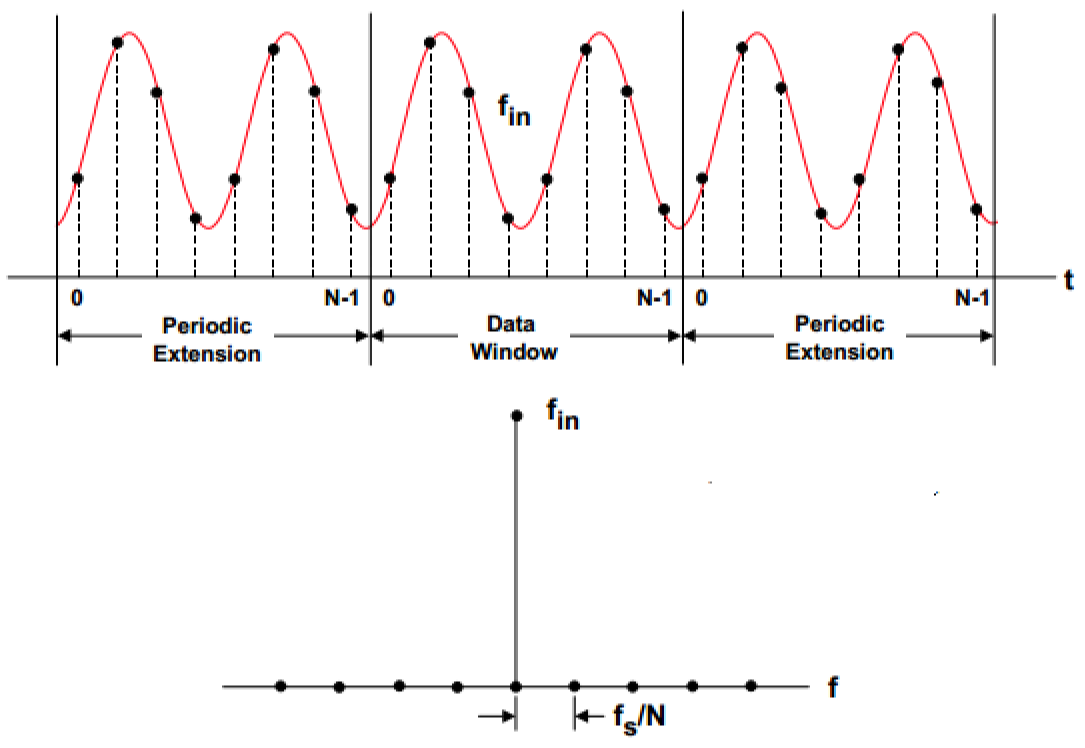


Figure 3.8 FFT Sine Wave with integral number of cycles in the data window

For situation where the sine waves do not have an integral number of cycles , discontinuities occur at the endpoints which emanate leakage in the frequency domain because of the harmonics generated. The resultant frequency spectrum will manifest side lobes with the main lobe, spread over the adjacent frequency bins. The condition can be demonstrated using the following figure:

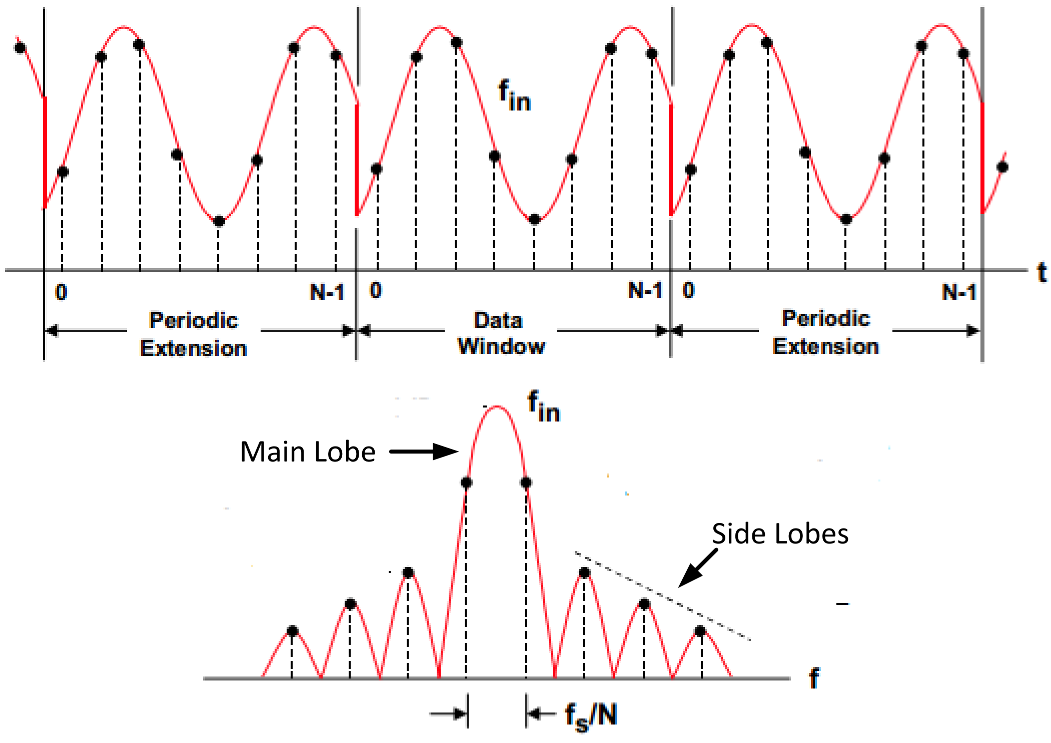


Figure 3.9 FFT Sine Wave with non- integral number of cycles in the data window

The sidelobe spectral leakage can be minimized by selecting the appropriate window function. The successive figures will elucidate four popular window functions with their respective equations and frequency response. Here $w(n)$ is the N time-domain window coefficient and the original N input signal samples are multiplied by n where $0 \le n \le N-1$ [37].

Rectangular Window [38]
$$w(n) = RECT\left[\frac{n}{0.97N}\right] \tag{3. 21}$$

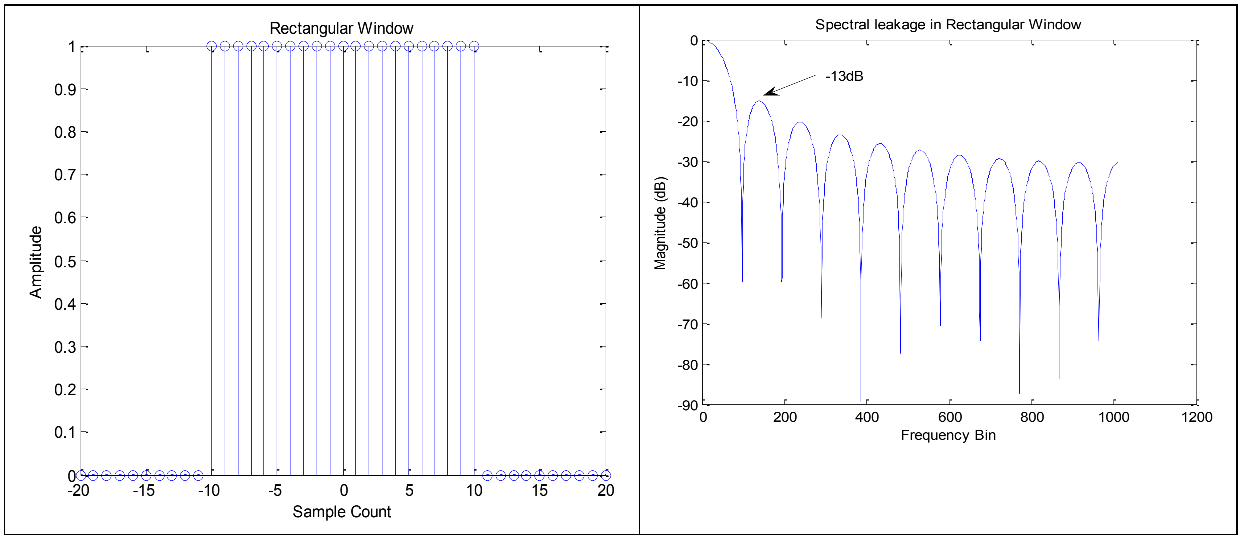


Figure 3.10 Time-domain and frequency domain representation of the Rectangular Window Function.

Triangular Window [38]:
$$w(n) = TRI\left[\frac{2n}{N}\right] \tag{3. 22}$$

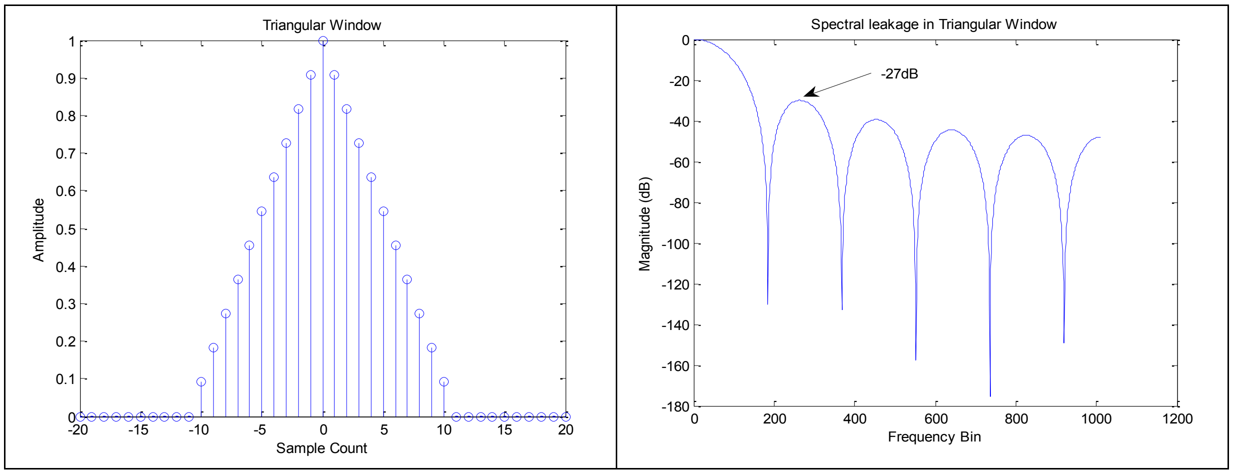


Figure 3.11 Time-domain and frequency domain representation of the Triangular Window Function.

Hanning Window: $$w(n) = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right) \qquad (3.\ 23)$$

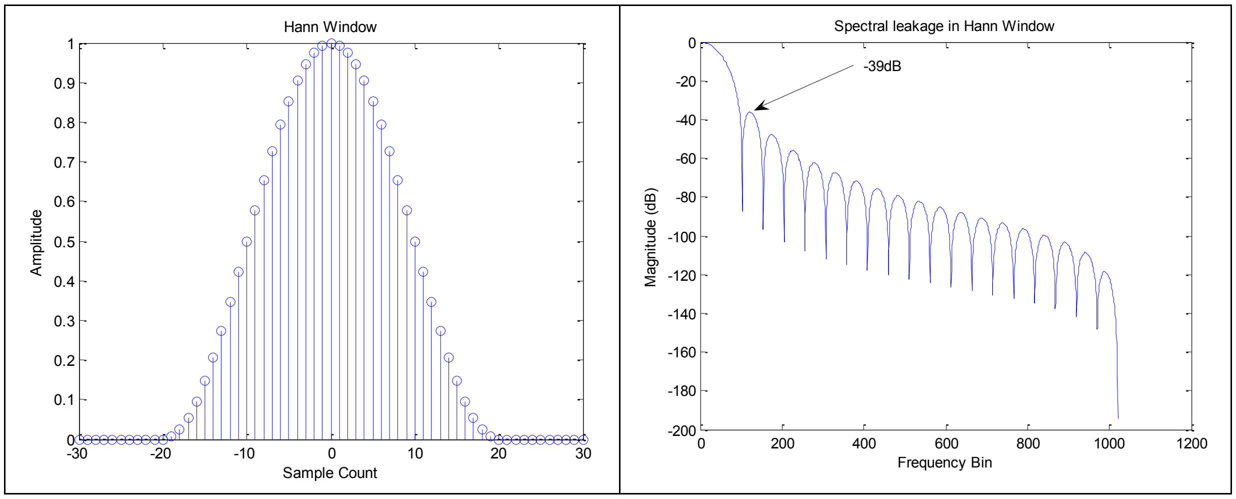


Figure 3.12 Time-domain and frequency domain representation of the Hanning Window Function.

Hamming Window: $$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) \qquad (3.\ 24)$$

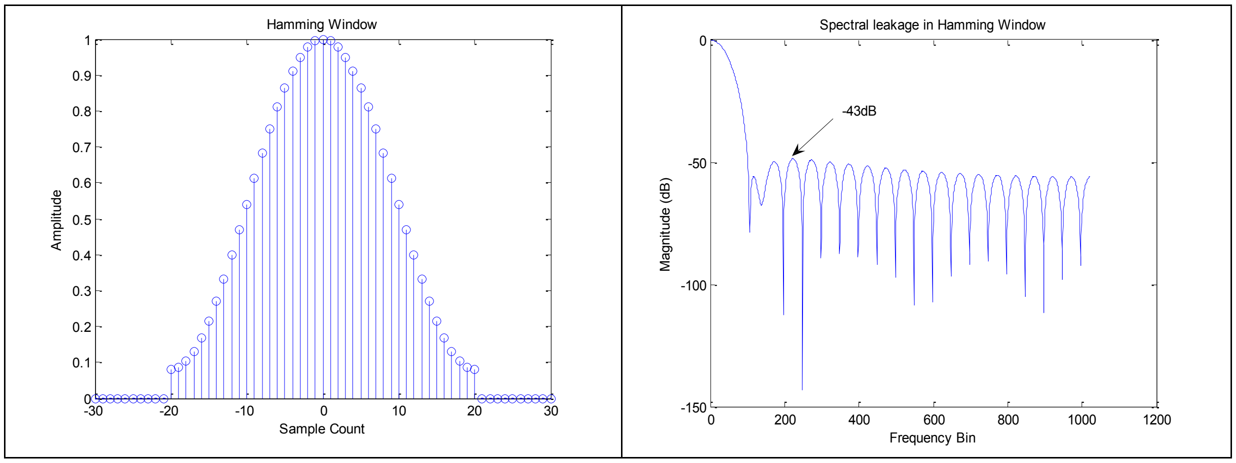


Figure 3.13 Time-domain and frequency domain representation of the Hamming Window Function.

Table 3.2 shows characteristics analysis between different window types. The ideal window is the one having a single main lobe with low side lobes and steep roll-off.

Table 3.2 Comparisons between different Window Functions

Window	Width of Main lobe (3dB) (Frequency bins)	Highest Side lobe (dB)	Side lobe Roll-off rate (dB/Octave)
Rectangular	0.89	-12	-6
Hamming	1.36	-43	-6
Triangular	1.28	-27	-12
Hanning	1.64	-39	-18
Blackman	1.68	-58	-18

From the tabulated value it can be seen that Blackman has the best side lobe attenuation and roll-off capability, although the width of the main-lobe is 1.68 indicated a wider energy spread compared with others and may cause spectral leakage. The most commonly used window in the field of communication is Hamming in spite of its roll-off rate being lower. For this project the Hamming window is being chosen for the following reasons:

1. Excellent side-lobe attenuation.
2. Ensures better accuracy even after truncating to 5 decimal places in fixed-point multiplication.
3. Favourable main-lobe width although has poor roll-off rate, but that can be attended using CFAR processing.

3.7.2 Fast Fourier Transform (FFT)

The Fast Fourier Transform (FFT) plays an important role in the digital signal processing routine. The algorithm was first discovered in the 18th century by Gauss and then again in the 1960s was redeveloped by James W. Clooney and John W. Tukey [39]. The algorithm of FFT is formulated from the principle of the decomposing of the DFT (Discrete Fourier Transform) computation into smaller sequences of DFT. This algorithm is constructed by the butterfly structure.

A simple radix-2 FFT algorithm is discussed by Cooley and Tukey and is define by the given equation:

$$X(k)=\sum_{n=0}^{N-1}x(n)e^{-i\left(\frac{2\pi}{N}\right)nk} \qquad k=0,1,\ldots.N-1 \tag{3. 25}$$

Here, $i=\sqrt{-1}$, $X(k)$ and. $x(n)$ .are a series of complex numbers and N is the transform size. The complex coefficient $e^{-i\left(\frac{2\pi}{N}\right)nk}$ is known as the twiddle factor. This algorithm expresses the $N=N_1N_2$ size DFT into smaller sizes of N_1 and N_2.

The radix affects the speed and complexity of the FFT algorithm. Two of the most common algorithms are the Radix-2 DIT and Radix-4 DIT. For a N-point Radix-2 decomposition the FFT consists of $\log_2(N)$ stages, where each stage has $N/2$ butterflies. For the same N-point FFT the Radix-4 has $\log_4(N)$ stages and each stage has $N/4$ Radix-4 butterflies.

The FFT core used in this thesis makes use of Radix-2 and Radix-4 algorithm for the DFT computing. When using the Burst I/O algorithm the decimation-in-time (DIT) is used [40]. The Radix-2 DIT is a very simple and common Cooley-Tukey algorithm. In a Radix-4 DIT algorithm, the data computation can increase up to 3 bits $(1+3\sqrt{2}=5.242)$) and for Radix-2 the growth factor is up to 2 bits $(1+\sqrt{2}=2.414)$. In this project the FFT IP core makes use of Radix-2 Lite Burst I/O architecture which is same as Radix-2 except it used shared adder/subtractor, reducing resources allocation in the process [40].

3.7.3 *Constant False Alarm Rate (CFAR) Processing unit*

The received signals from any target detected are usually corrupted with noise interference, clutter and effects of attenuation. Moreover, this noise-clutter is not analogous for every environment and as such a fixed-threshold detection technique cannot be applied to identify the false targets. To overcome this problem the Constant False Alarm Rate (CFAR) with an adaptive threshold process can be used. Based on the local noise information this adaptive process can estimate the noise power by investigating a number of reference cells. This technique of averaging a number of reference cells to detect false target is known as Cell Averaging CFAR (CA-CFAR) [41] [42]. Several other CFAR architectures have been investigated in [14]. Among the popular ones are OS-CFAR and CA-CFAR. Also further extensions for these CFAR architectures were developed. Two of them are Greatest of Selection CA-CFAR or CAGO-CFAR by Vilhelm G. Hansen and Greatest of Selection OS-CFAR or OSGO-CFAR which was developed by He You. The design architecture for the OS-CFAR is shown in the figure below:

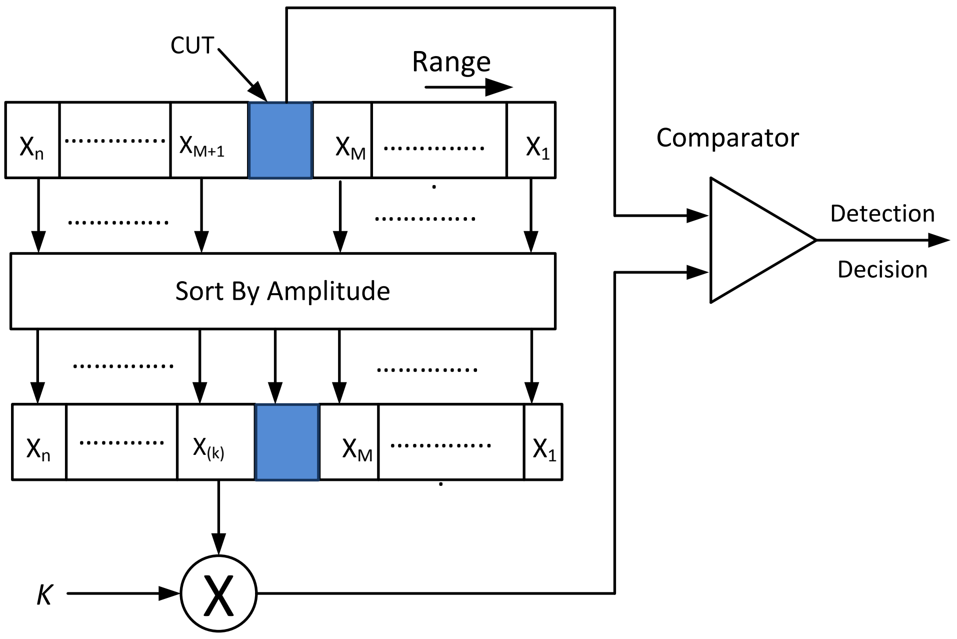


Figure 3.14 OS-CFAR architecture

Here K is the threshold value, M is the number of cells CUT is the cell under test. The factor on which the outcome is dependent is k. The samples of ranges are first arranged according to their magnitudes and the statistic Z is considered to be kth largest sample. The term $X_{(k)}$ is defined as:

$$X_{(1)} \le X_{(2)} \le \cdots \le X_{(n)}, Z = X_{(k)} \tag{3. 26}$$

For an environment where the noise-clutter interference is known, the target detection process would have been a fairly easy method. In this case the spectral intensity of the received signals would have been compared with a fixed threshold, calculated from the static noise and clutter background. If the intensity of the target in question exhibit a higher value than the threshold value then the target is considered as a valid one otherwise it is discarded as being a false one [14].

Although in real radar applications the noise-clutter interference always varies with different situations which bring back to the adaptive threshold technique. The dynamic changing of the threshold value helps the radar to be more vigilant in its detection process.

The CA-CFAR technique is the chosen CFAR technique for this design project. The operation principle of the CA-CFAR technique is abridged below with a figure of the algorithm following it:

1. Output from the FFT undergoes a square law detection process which eliminates all the negative terms, inherently computing the absolute value of the intensity of the frequency bin.

2. Guard bands are placed on either side of the Cell Under Test (CUT) to overcome the problem of spectral leakage from the CUT.
3. A total of M number of cells are considered for investigation, placing them on either side of the CUT in equal numbers i.e. $M/2$ no of reference cells on the left ($avgL$) and right side ($avgR$) of the CUT. Index k increments from 1 to $M/2$.
4. The average of the cells $avgL$ and $avgR$ are calculated and then multiplied by a constant K, a preordained value, and the adaptive threshold T_{A} is computed. The value of the term K is calculated using

$$K = P_{\mathrm{fa}}^{-\frac{1}{M}} - 1 \tag{3. 27}$$

Here P_{fa} is the Probability of False Alarm.

5. After the T_{A} is determined the CUT is compared against this value and the decision for target detected is made by the following condition [43]:

$$CUT = \begin{cases} > T_A & \text{Target detected} \\ < T_A & \text{False alarm} \end{cases}$$

This detection process runs for all FFT outputs, investigating each cell for the target detection. The CA-CFAR process can be done very fast using the FPGA as both of them can perform parallel operations.Figure (3.13) shows the CA-CFAR architecture:

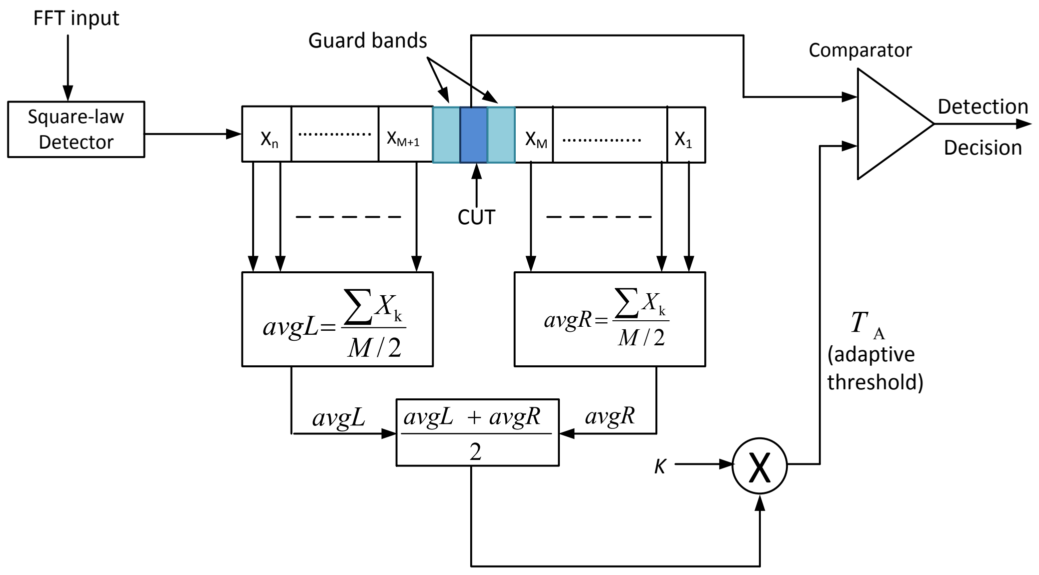


Figure 3.15 CA-CFAR Architecture Used In The Thesis

3.8 Multifarious Aspects Considered for the Radar Design

3.8.1 Radar Targets

A theory for the probability of target detection with constant target cross section or nonfluctuating targets was first analyzed by J.I. Marcum [44] of RAND Corporation which was cultivated further by another member of the corporation Peter Swerling. He elaborated the study for fluctuating targets [45] developing mathematical models for four isolated cases based on the target cross-section. These models are known as Swerling I, Swerling II, Swerling III and Swerling IV. The Swerling 0 and Swerling V is based on Marcum's Theroy as they are based on constant Radar Cross Section (RCS) case. The model variations are formed according to the chi-square probability density function (pdf). The RCS vary according to the target shape, size, dynamics and the relative motion with respect to the radar.

For electromagnetic analysis the RCS is denoted by the equation below:

$$\sigma = \lim_{r\to\infty} 4\pi r^2 \frac{|E_s|^2}{|E_i|^2} \tag{3. 28}$$

Here E_s is the scattered field intensity at distance r and E_i is the incident electric field intensity.

The maximum unambiguous range covered by the radar is one of the factors which utilize the RCS information.

The five Swerling models are classified as below [46], [47]:

Swerling 0 (also called *Swerling V*): The RCS is constant conducing to a non-fluctuating pulse amplitude with constant peak-to-peak SNR (signal to noise ratio). The reception variable which is the initial phase of each pulse is presumed to be uniformly and independently distributed in $[0,2\pi]$.

Swerling I: The RCS varies according the *chi-squared* pdf with two degrees of freedom. It is applicable for targets made up of independent scatters of nearly equal areas. Around 5 or more scattering surfaces contributes to muster this distribution. This model delineates a target with constant RCS through a single scan but differs independently from scan to scan. The pdf for this model is given as:

$$p(\sigma) = \frac{1}{\sigma_{\text{avg}}} e^{-\frac{\sigma}{\sigma_{\text{avg}}}} \tag{3. 29}$$

Here, σ is the RCS of the target and σ_{avg} is the average value of all target RCS.

Swerling II: It is similar to Swerling I except the cross-section fluctuations are independent from pulse to pulse.

Swerling III: In this model the RCS varies according to the *chi-squared* pdf but with 4 degree of freedom. It assumes one scattering surface with additional insignificant smaller scattering surfaces. Like that is Swerling I the RCS is also constant through a single scan and the pdf is expressed as:

$$p(\sigma) = \frac{4\sigma}{\sigma_{\text{avg}}^2} e^{-\frac{2\sigma}{\sigma_{\text{avg}}}} \tag{3. 30}$$

Swerling IV: The model is akin to Swerling II model except the fluctuation is from pulse t pulse. This represents a more dynamic case of Swerling III model.

The models can be illustrated by the figure below [48]:

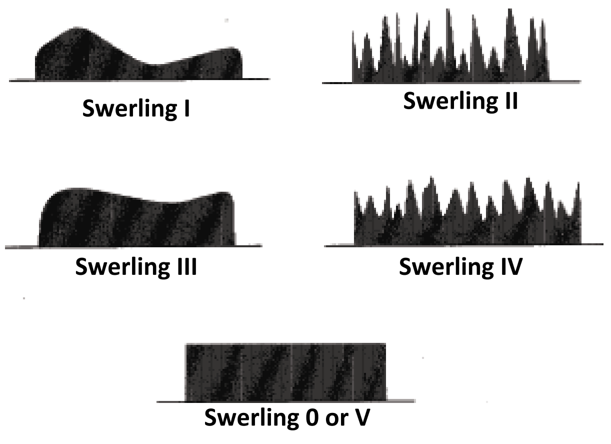


Figure 3.16 The different Swerlingfluctuating models

3.8.2 Noise

Noises are random superfluous signals which affect the received echo signalfrom the target. The sources of these noises can be from signal processing noise, RF circuits and antenna and atmospheric noise. Some crucial sources of noise are:

1. Thermal Noise: this is the noise generated by the thermal motion of the semiconductor charge carriersin the ohmic portion of the electronic and RF circuitry. The noise signal can be expressed by the following relation:

$$N_{\text{Th}} = kT_{\text{A}}B \tag{3. 31}$$

Here k is the Boltzmann's constant in Joules/Kelvin, T_{A} is the average absolute temperature in Kelvin and B is the Bandwidth in Hertz [49].

2. Background Noise: This type of noise signal is generated from environmental noise such as refection from buildings, clouds etc [42] or cosmic radiation. On other noise type is the "white" noise where all the frequencies formed by combining sounds, have equal amplitudes. It also has equal power with a fixed bandwidth at the center frequency [49].
3. "Pink" Noise: This unambiguous noise signal is also known as $1/f$ noise and a decreasing power spectral density [50]. For applications like radar where the frequency is high this type of noise has minimal effect [51].
4. Quantization Noise: When an ADC samples the IF, they quantize the amplitude of the input analog signal into binary output of finite length. This is approximately a non-linear process which results in the formation of a wide-band noise in the ADC output known as quantization noise. Two noted methods to overcome this noise effect is oversampling and dithering [52]. The signal-noise-ratio, in dB, for an ideal N-bit ADC is

$$SNR_{\text{Q}} = 6.02N + 4.77 + 20\log_{10}(L_{\text{rms}}) \tag{3. 32}$$

Here L_{rms} is the rms input voltage of the analog signal divided by the peak input voltage of the ADC.

3.8.3 Attenuation

All types of radio signals on their course through the atmosphere suffer abatement of intensity due to attenuation. The attenuation effect measured in dB/km is generally caused by the absorption or scattering of the radio signals. The scattering characteristic of the waves is based predominantly on the oxygen in the atmosphere, humidity, fog and rain. In the 70 and 80GHz bands the effect of atmospheric oxygen is considered to be negligible. Water vapor depending on the absolute humidity causes a limited loss of between 0 to 50% per km at very high humidity and temperature. Attenuation due to fog or cloud is similar to the humidity condition except it also depends on the quantity and size of liquid droplets. This loss due humidity and fog seems momentous but compared with the attenuation due to rain it is insignificant [53]. Some attenuation for different weather conditions are tabulated below [20]. In this thesis 0.8dB/km is taken as the attenuation value which falls in between light and medium rain condition with an SNR of 4.73dB.

Table 3.3 Attenuation at 70-80GHz due to atmospheric conditions

Condition	Precipitation Rate (mm/hr)	Attenuation (dB/km)
Clear, dry air	0.00	0.1
Drizzle	0.25	0.2
Light rain	1.25	0.5
Moderate rain	12.50	1.5
Heavy rain or snow	25.00	9.0

3.8.4 Clutter and Jamming

Clutter in context to the automotive scenario is associated with the scattering of radio beam due to the reflection from objects like trees, water, buildings, sign posts, road surfaces, barriers or dividers and also from the host’s bumper in addition to other sources. The target information might get obscure because of this clutter as it takes a small fraction of the total beam area. However most of these clutter are from stationary sources hence it remains fixed at a particular frequency bin over a scan sweep. It also has a low power intensity and fixed RCS. Some of the methods that can be employed to reduce clutters are narrow beamwidth, utilization of wider bandwidth, moving-target indication (MTI) etc. Although the success of clutter reduction using these techniques depend on its characteristics, radar stability and dynamic range and also on the signal processing technique utilized. Meanwhile the clutter information is recently being utilized to detect stationary objects like road boundaries, utility poles etc and also human presence for better mapping of the ambient environment. This will improve the road safety conditions by providing a real-time automotive radar that has the ability to detect both stationary and moving objects [54].

Radar jamming occurs when the entire operational bandwidth is saturated with high-power microwave signals. As a result it becomes impossible to differentiate between the real and false targets. For automotive radars jamming may occur when adjacent or near-by radar systems operate at the same frequency at an instant of time or from broadband Pulse Doppler radar generating high-power microwave signals. Noise jamming is either Spot jamming which relates to the narrowing of the jamming bandwidth to cover the radar operating bandwidth and hence cloaking it altogether. The other type is call Barrage jamming where a wide noise bandwidth is utilized to cover several radars simultaneously with one jammer [55].

Chapter 4
Signal Processing Algorithm and Control of the Radar System

This chapter gives a comprehensive summary of the algorithm used in designing the tri-mode automotive radar system. The system developed in the University of Windsor focalizes on calculating the target range, target velocity and the target angle. This thesis develops a radar system to calculate the target range and velocity using Rotman Lens, MEMS RF Switch and phased-array antenna as mentioned in Chapter-1. The signals processing section of the design is responsible for the modulation of the transmitted waves into frequency chirps and also processes the received waves after certain modification. A detail discussion of the processing algorithm and the factors affecting the received echoes (noise, attenuation etc) was discussed in the previous chapter.

This chapter describes the reasons abetting the choices made for the radar process design and elucidates individual block operation with reference to the specifications listed in Table 4.1 [20].

Table 4.1 Introductory System Specifications

Parameter	LRR	MRR	SRR
Type of Radar	LFMCW	LFMCW	LFMCW
Operating frequency	77GHz	77GHz	77GHz
VCO	TLC77xs	TLC77xs[1]	TLC77xs[1]
Target Fluctuating models examined	Swerling I, II, V (or 0)	Swerling I, II, V (or 0)	Swerling I, II, V (or 0)
Beamformer	Rotman lens	Rotman lens	Rotman lens
Beam numbers	3	3	3
Processing time per beam	2.048ms	2.048ms	2.048ms
Beam width	±4.5°	±10°	±40°
Antenna type	Phased-array antenna	Phased-array antenna	Phased-array antenna
Radar processing unit (RPU) platform	FPGA	FPGA	FPGA

4.1 Radar Transmitter Processing and Control Algorithm

The transmission section of the radar design is liable for the following responsibilities:

1. Manipulating the DAC to generate frequency chirp by tuning the VCO
2. Chirp generation synchronization on the receiving end of the signal processing inducing delay where necessary
3. Adjusting the MEMS Switch control to switch between the beam ports, changing the beam direction in the process
4. Switching between the three beams of the Rotman lens.

The transmission operation algorithm is described using the flowchart in Figure 4.1. The system starts with generation of the voltage range from $V_{\min}$ to $V_{\max}$ and then back to $V_{\min}$ which is tuned afterwards by the VCO for the frequency chirps. The target sweep duration is 1.024ms for each up and down sweep.

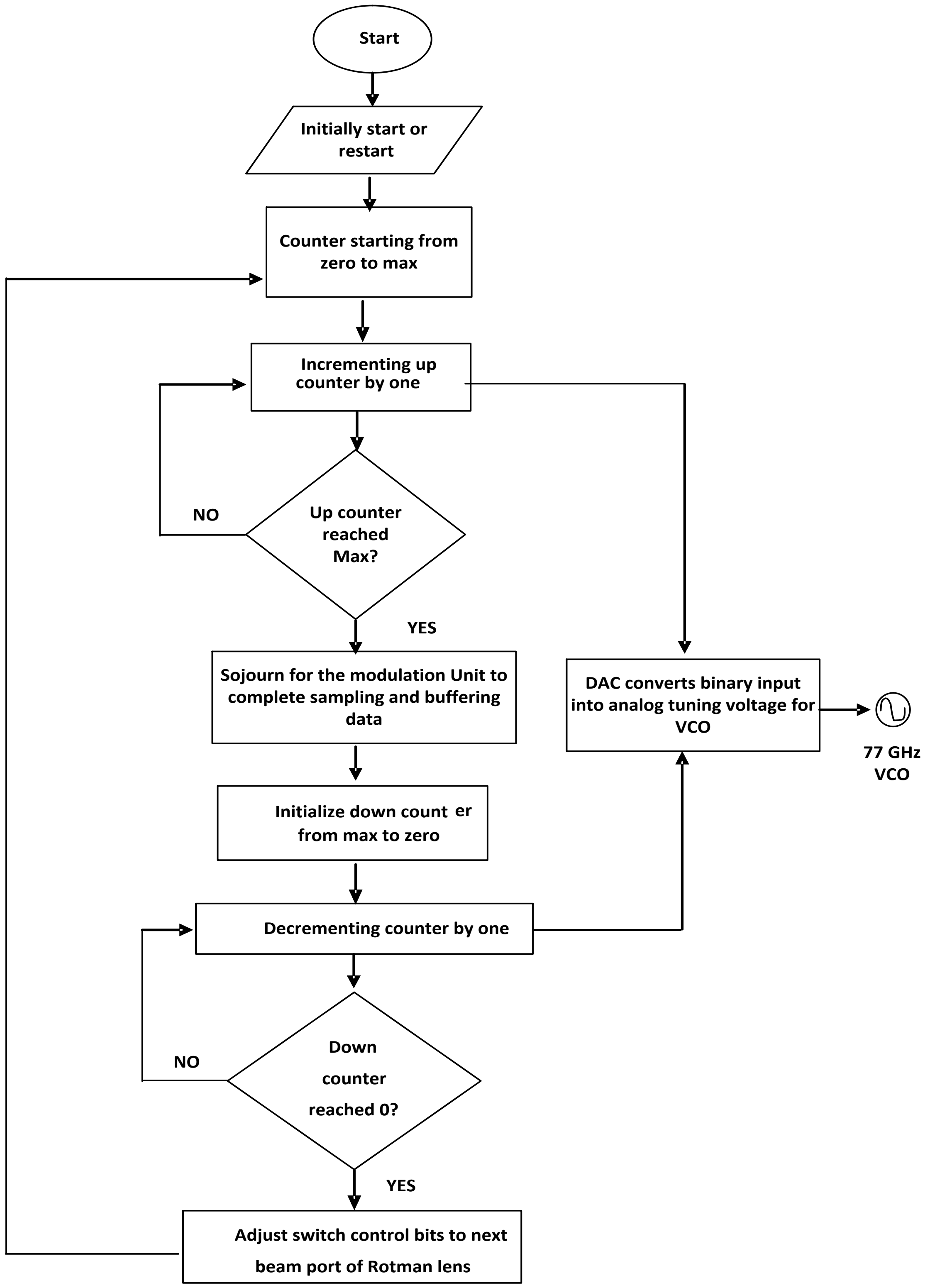

Start
Initially start or restart
Counter starting from zero to max
Incrementing up counter by one
NO
Up counter reached Max?
YES
Sojourn for the modulation Unit to complete sampling and buffering data
DAC converts binary input into analog tuning voltage for VCO
77 GHz VCO
Initialize down count er from max to zero
Decrementing counter by one
NO
Down counter reached 0?
YES
Adjust switch control bits to next beam port of Rotman lens

Figure 4 1 Radar Transmitting and Processing algorithm

4.2 Radar Receiver Processing and Control Algorithm

Once the IF signals are generated the signal processing routine is performed which is the crucial part of the radar design algorithm. The processing technique is described below followed by a block diagram of the system:

1. The Hamming window is applied to the time-domain samples collected from the ADC.
2. FFT is performed on the windowed samples.
3. The magnitude square of the FFT samples are calculated.
4. CA-CFAR algorithm is executed to eliminate the false targets disregarding noise, clutter and jamming.
5. After completion of the CFAR process peak pairing is done to calculate the target range and velocity.

The process listed above is illustrated in the figure below:

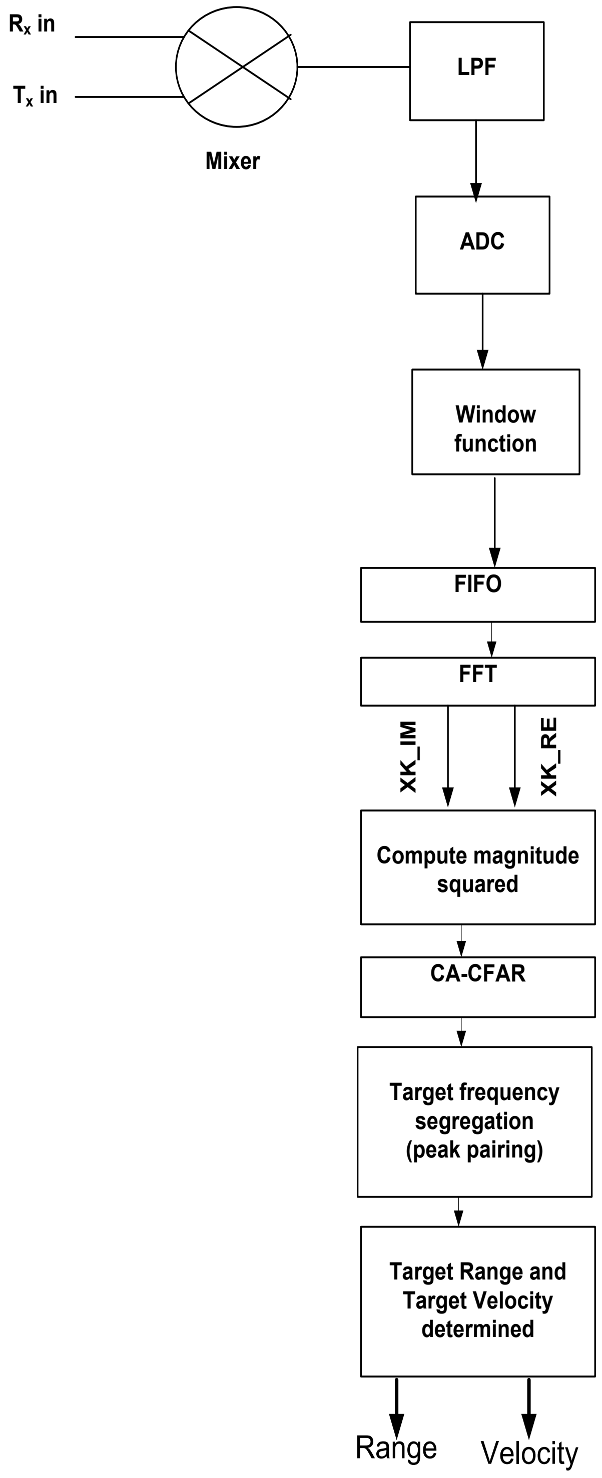

R_x in
T_x in
Mixer
LPF
ADC
Window function
FIFO
FFT
XK_IM
XK_RE
Compute magnitude squared
CA-CFAR
Target frequency segregation (peak pairing)
Target Range and Target Velocity determined
Range
Velocity

Figure 4.2 Radar Signal Processing algorithm of the design system portraying the hierarchy of the total operation

4.3 Radar Bandwidth Selection for the Tri-mode radar

The parameters chosen for the radar design is critical for improving the performance of the system. One of the cardinal criterions is the range resolution or separability of speed and distance between two targets [2]. The range resolution can be defined as the radar ability to demarcate between two or more targets on the same bearing but are at different distance [56]. Selecting the bandwidths for the tri-mode design was based on a trade-off decision between the system bandwidth and the non-linearity effects of the VCO. In addition the effect of frequency estimate in relation with the error in the distance estimate is investigated before the respective bandwidths were chosen [57]. The equations for calculating range and velocity resolution and distance estimate is given below:

Range Resolution (m) : $$\Delta R = \frac{c}{2B} \tag{4. 1}$$

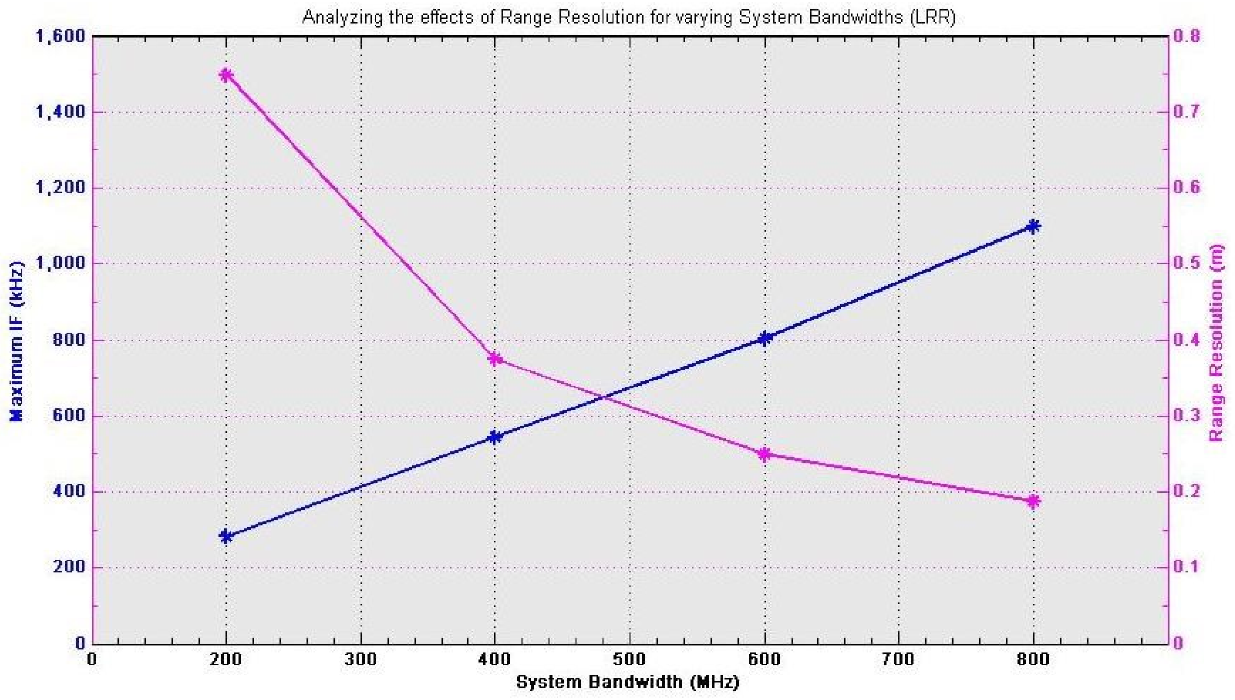
Velocity Resolution(m/s) : $$\Delta v_{\mathrm{r}} = \frac{\lambda}{2T} \tag{4. 2}$$

Distance estimate (m) : $$\delta R = \delta w \frac{c}{2k} \tag{4. 3}$$

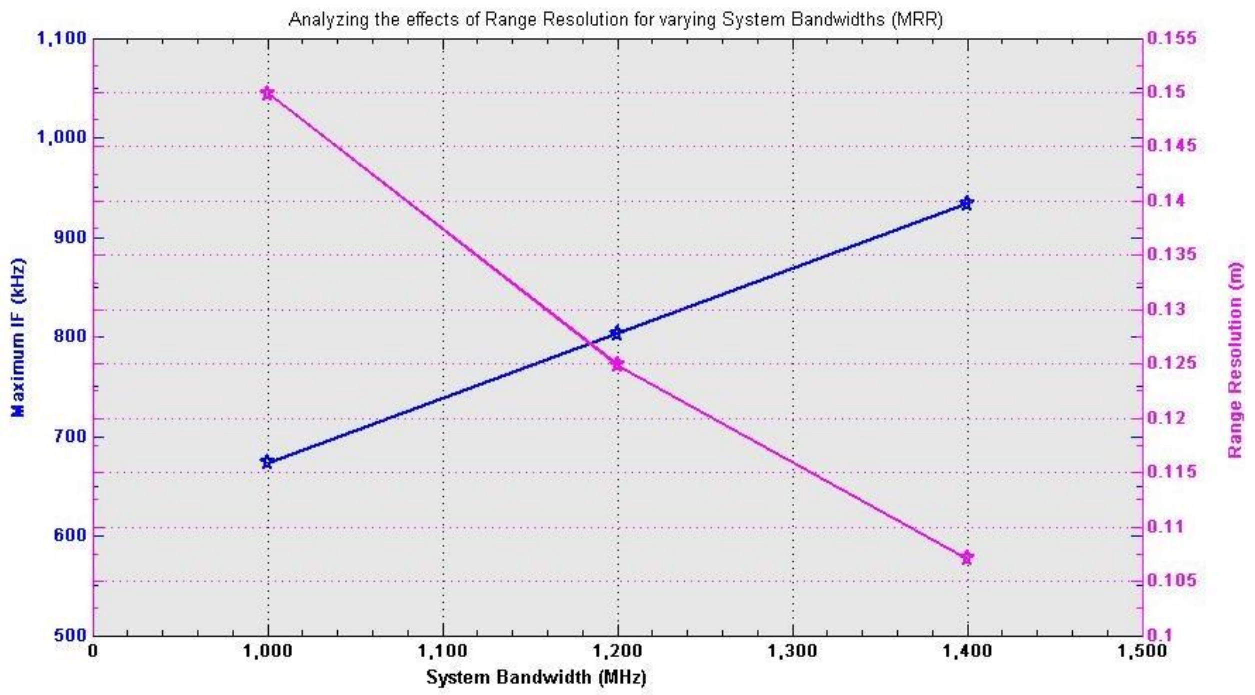
Here, c is the speed of the EM wave in air, B is the bandwidth of the system, is the radar wavelength, T is the sweep duration for up or down sweep, δw is the absolute error and k is the sweep rate of the system.

Analyzing the equations is can be deduced that increasing the sweep rate gives a better estimation of the range value and decreasing the sweep duration will improve the range resolution.

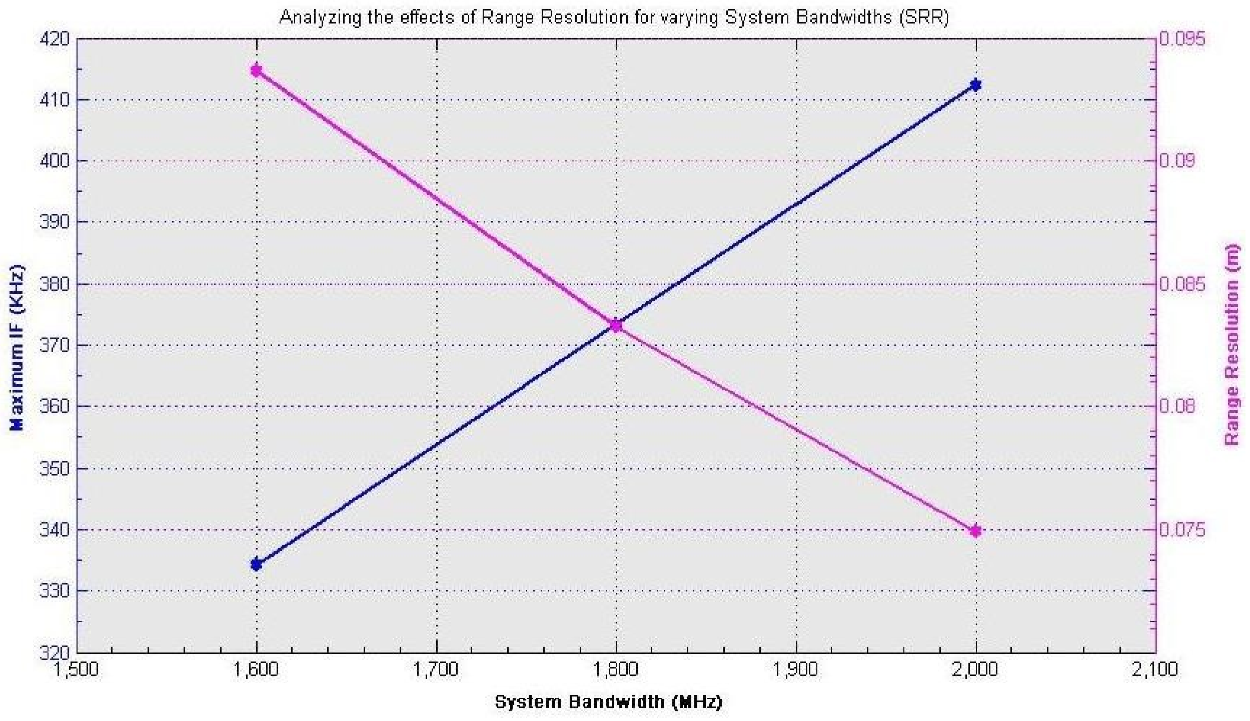
The following graphs show the effects different bandwidths have on the maximum frequency and the range resolution for the Tri-mode radar.


Analyzing the effects of Range Resolution for varying System Bandwidths (LRR)
Maximum IF (kHz)
Range Resolution (m)
System Bandwidth (MHz)
0
200
400
600
800
1,000
1,200
1,400
1,600
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
300
500
700

(a)


Analyzing the effects of Range Resolution for varying System Bandwidths (MRR)
Maximum IF (kHz)
Range Resolution (m)
System Bandwidth (MHz)
500
600
700
800
900
1,000
1,100
0.1
0.105
0.11
0.115
0.12
0.125
0.13
0.135
0.14
0.145
0.15
0.155
0
1,200
1,300
1,400
1,500

(b)



(c)

Figure 4.3 Results for Maximum IF and Range Resolution for Different Bandwidth for (a) LRR (b) MRR and (c) SRR

Surveying the above graphs it can be observed that an increase in the bandwidth meliorates the range resolution but also increases the maximum IF. However an increase in the IF means that the sampling frequency needs to be increased according to the Nyquist theorem which will lead to a compromise for the velocity resolution [equation (4.3)] and frequency resolution of the FFT [equation (4.4)].

Frequency resolution of FFT :
$$f_{\text{res}} = \frac{f_s}{N} \tag{4. 4}$$

Here f_s is the sampling frequency and N is the FFT point size which is equal to no of time-domain samples collected.

The frequency resolution is basically the minimum spacing between two frequency bins of the FFT which relates to the range resolution of the system. Also increase in the bandwidth will lead to a better estimate of the distance according to equation (4.3).

After taking everything into consideration the bandwidths chosen for the system are 800MHz, 1400MHz and 2000MHz respectively for the long, medium and short range radar. The sweep duration for each up and down sweep is taken to be 1.024ms and the ADC sampling frequency is taken to be 2 MHz (Mega Samples per Second) which would collect 2048 samples over each sweep duration. The sample count is taken as a power of 2 as a Radix-2 FFT algorithm is used.

4.4 Other System Components Configuration

ADC

From the discussion in the previous chapter the bandwidth for the tri-mode radar system was chosen to be 800MHz, 1400MHz and the 2000MHz. The sampling frequency for this thesis is kept to be 2MHz for all the bandwidths and with sweep duration of 1.024ms and exact number of 2048 samples can be collected.

FFT

For the 2048 samples collected over the given sweep rate from the ADC a Radix-2 algorithm is used to perform the Fast Fourier Transform on the signal.

CA-CFAR

The CFAR architect chosen for this design is Cell-Averaging process. The reason for the choice was discussed in detail in chapter 3. The CFAR algorithm is used to detect the targets overcoming the noise and clutter. The probability of the false alarm is 10^{-6} for this thesis and the total number reference cells considered for averaging is eight with guard cells on either side of the CUT (cell under test). The guard cells are placed to avoid spectral leakage from the CUT. The value of the constant K calculated to be:

$$K = P_{\text{fa}}^{-\frac{1}{2M}} - 1 = (10^{-6})^{-\frac{1}{2\times 8}} - 1 \approx 1.3714 \tag{4. 5}$$

Peak Pairing

After the targets were detected from the CFAR algorithm peak pairing is done according to two rules:

1. Power Intensity level: The peak intensity of the FFT output signifies the power level of the detected target. Intensity level from a target further away will create a lower power compared with a target near the host vehicle. This criterion is taken into account when pairing is done between the target detected in the up sweep and the one detected in the down sweep. A small difference in the value indicates the peak intensity is for the same target detected.

2. Spectral contiguity: With regards to the bandwidth selected and a relative velocity of 300km/h if a target detected in the up and down sweep fall within 84 bins shift of each other it can be considered to the same target. The shift in question occurs due to the Doppler shift. Hence pairing is done when targets detected from the up sweep and down sweep fall within 84 bins of each other.

4.5 Algorithm Summary

The parameters chosen in accordance with the decisions discussed so far are used to develop a design algorithm in both Matlab and HDL alike. System design was done concurrently in both floating-point (Matlab) and fixed-point (HDL) platform and simulations and performance was noted and compared in the forthcoming chapters. Summary of the parameters chosen are given in the tables below:

Table 4.2 Signal Processing Unit for the Radar System

Processing Details	Unit
DAC	16-bit, 200 MHz
ADC	12-bit, 3 MHz
Window Type/Length	Hamming/2048
FFT Type	Mixed Radix-2/4 DIT
FFT Length	2048
CFAR type	Cell- Averaging (CA)
CFAR parameters	Guard bands= 2, No of cells = 8, $P_{fa} = 10^{-6}$

Table 4.3 Specification for the Long Range Radar Design

Parameters	Symbol	Value	Unit
Bandwidth	B	800	MHz
Maximum IF	IF	1.064	MHz
Beam Width	Beam_Width	9	degrees
Chirp Time	T_{chirp}	1.024	ms
Maximum Theoretical Range Resolution.	ΔR	0.1873	m
Maximum Theoretical Velocity Resolution.	ΔV_r	1.901	m/s
Sampling Frequency	f_s	2	MHz
FFT point size	N	2048	Dimensionless
FFT Resolution	Ω	520	Hz/bin

Table 4.4 Specification for the Medium Range Radar Design

Parameters	Symbol	Value	Unit
Bandwidth	B	1400	MHz
Maximum IF	IF	325.53	MHz
Beam Width	Beam_Width	20	degrees
Chirp Time	T_{chirp}	1.024	ms
Maximum Theoretical Range Resolution.	ΔR	0.1070	m
Maximum Theoretical Velocity Resolution.	ΔV_r	1.9005	m/s
Sampling Frequency	f_s	2	MHz
FFT point size	N	2048	Dimensionless
FFT Resolution	Ω	318	Hz/bin

Table 4.5 Specification for the Short Range Radar Design

Parameters	**Symbol**	**Value**	**Unit**
Bandwidth	B	2000	MHz
Maximum IF	IF	86.57	MHz
Beam Width	Beam_Width	80	degrees
Chirp Time	T_{chirp}	1.024	ms
Maximum Theoretical Range Resolution.	ΔR	0.0749	m
Maximum Theoretical Velocity Resolution.	ΔV_r	0.3167	m/s
Sampling Frequency	f_s	2	MHz
FFT point size	N	2048	Dimensionless
FFT Resolution	Ω	85	Hz/bin

Chapter 5
Software Implementation of the Designed Project

The simulations for the thesis done in floating-point system in MATLAB platform are shown in this chapter. Based on the parameters, target specifications, mathematical concept and the system configuration a MATLAB simulation algorithm is developed and the simulated results are cumulated and compared with the HDL implementation, which will be discussed in the next chapter. Some additional toolbox were used to aid the MATLAB software e.g. Signal Processing Toolbox, DSP System Toolbox, Fixed-point Toolbox and Phased-array Toolbox.

The total signal processing in the software simulation is done according to the following sequence:

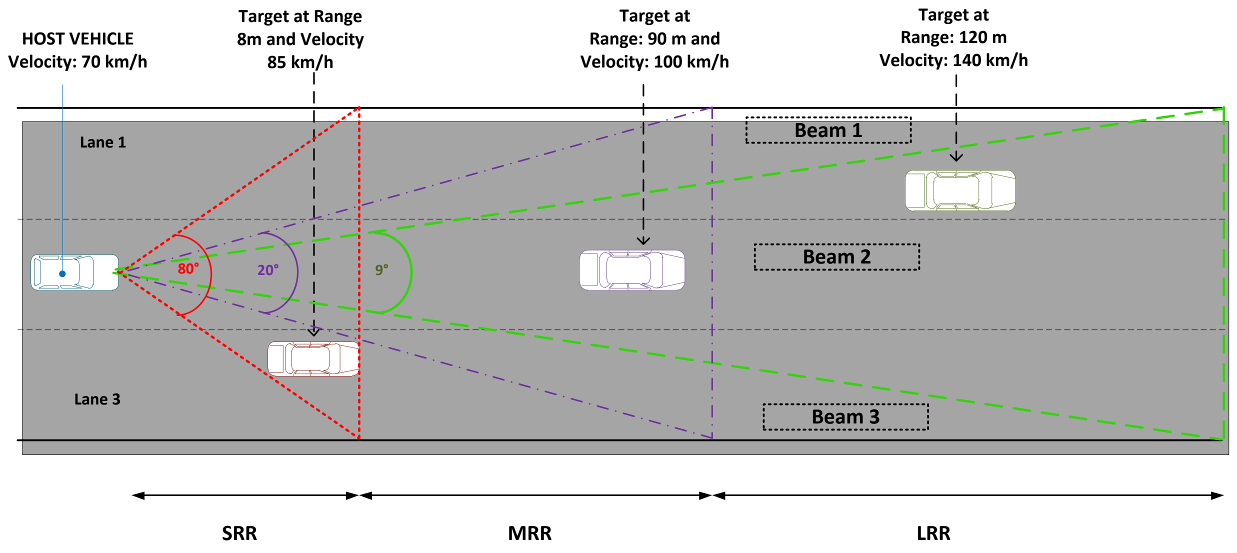
1. The IF calculated from the received echo from the detected target is sampled using a 12-bit ADC.
2. The sampled data are then processed with a hamming window to minimize the spectral leakage.
3. The resultant windowed samples are then passed through a Radix-2 FFT processing algorithm.
4. A CA-CFAR algorithm identifies the false targets and a peak paring technique is used to distinguish the relevant target signals from the up and down sweep.
5. From the information received from the peak-paring algorithm the range and velocity of the targets are calculate using the equations (3.8), (3.15), (3.17) and (3.18).

The list of the complete MATLAB code is given in Appendix 1 (a).

As the chapter is progressed relevant figures and graphs are shown to portray the simulations from each process stages described above.

5.1 *Software Simulation of the Design System*

In accordance to the research methodology listed above the simulations were done in MATLAB and the following results were observed. For the current situation single target detections were considered. The test scenario with a single beam (from the 3-beam port of the Rotman lens mentioned in Chapter 1) can be depicted by the figure below:



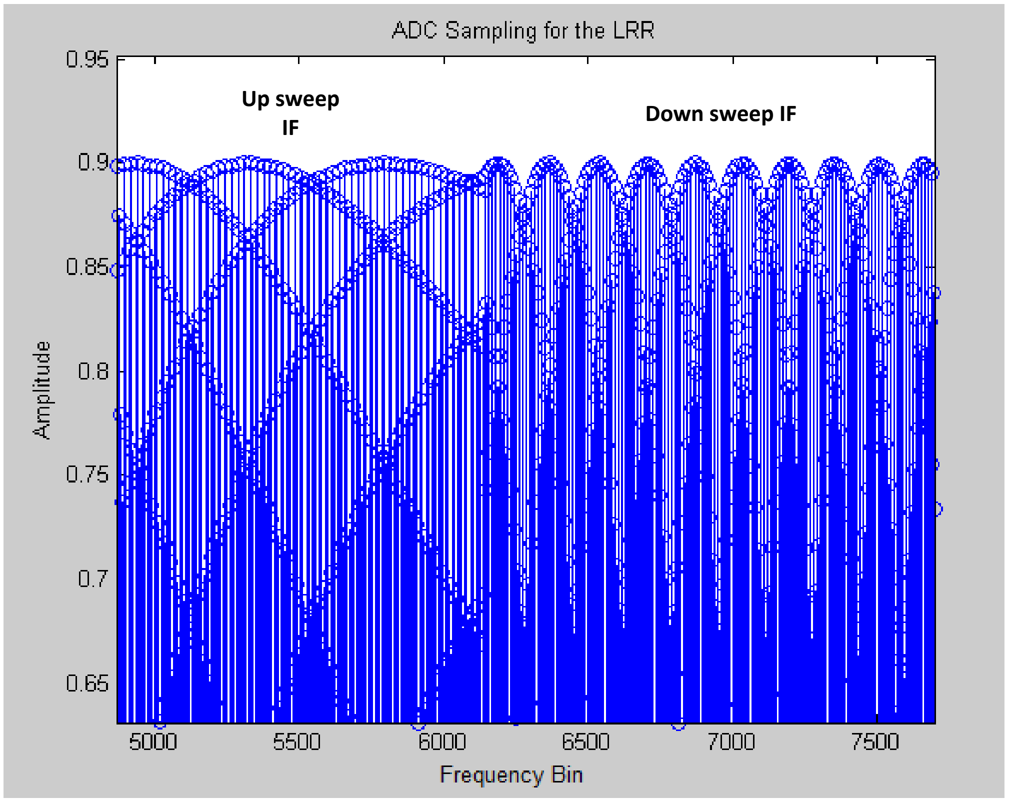
HOST VEHICLE
Velocity: 70 km/h
Target at Range 8m and Velocity 85 km/h
Target at Range: 90 m and Velocity: 100 km/h
Target at Range: 120 m Velocity: 140 km/h
Lane 1
Lane 3
Beam 1
Beam 2
Beam 3
80°
20°
9°
SRR
MRR
LRR

Figure 5.1 Test situation scenario for a single target situation

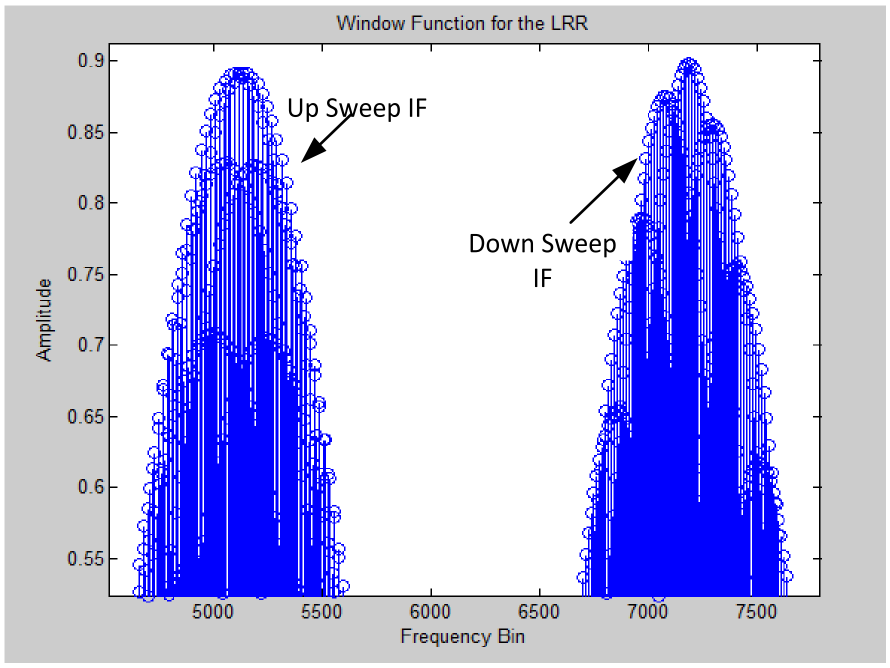
The intermediate frequencies collected from the echoes of the targets are sampled and windowed using a 12-bit ADC and Hamming window function respectively. The target echoes are presumed to add up at the receiving phased array antenna mentioned in chapter 1 as an important part in the total design of the MEMS radar block. For this test scenario it is assumed that the target detected by the LRR falls within beam 1, for MRR it falls within beam 2 and for SRR it is within beam 3. Simulations for the ADC and window functions are given below for each mode of the radar range:

Long Range Radar (LRR):

The figure below displays the ADC signal contaminated with arbitrary Additive White Noise (AWN) and also after multiplying with the Hamming window function. The signal to noise ratio is found to be 4.73dB in accordance with the attenuation scenario considered. The signal is shown for Beam 1, but for Beam 2 and 3 the signal will be similar to this. The Up and Down sweep IF is shown in the figure:



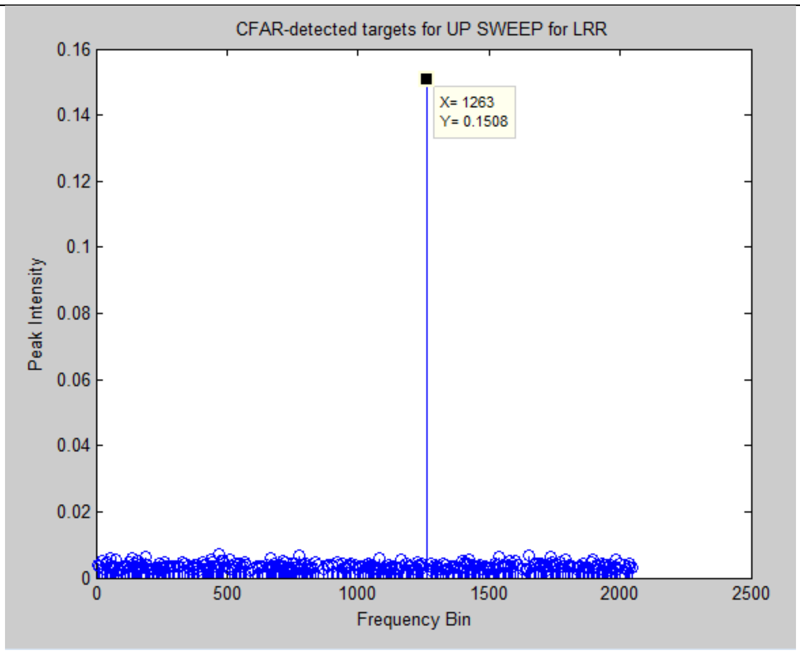
(a)



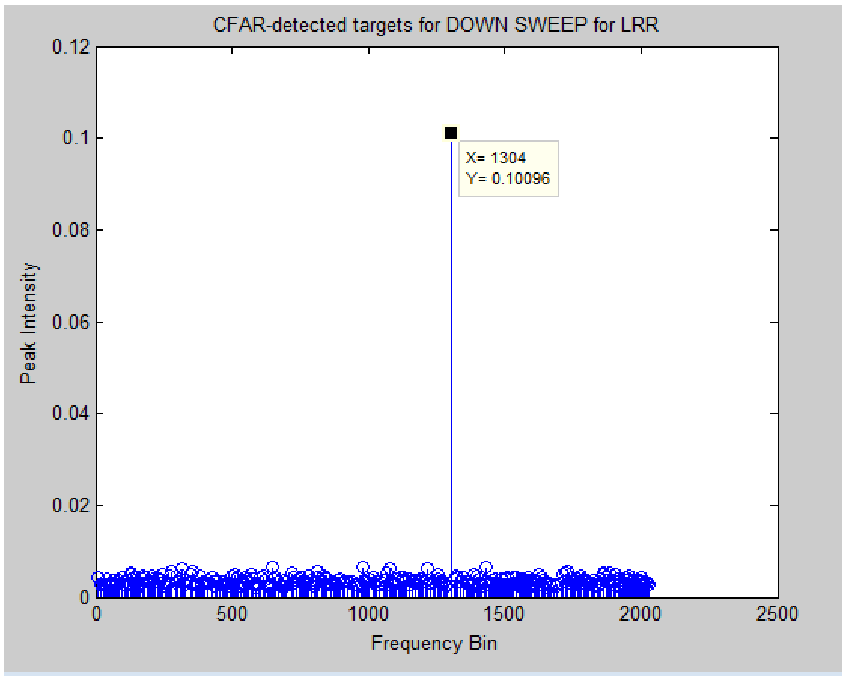
(b)

Figure 5.2 ADC Sampling and Windowing of IF from the target detected in the long range

Figure below shows the target detected in the Up and Down sweep within the beam 1 region. The target is accepted to be a valid target as the two conditions for the peak-pairing criteria was fulfilled. The Peak intensity between the two signals is 0.04984 $(Y_{UP} - Y_{DOWN} = 0.1508 - 0.10096)$ and the target signal is within the 84 bins shift $(X_{DOWN} - X_{UP} = 1304 - 1263 = 41)$. Here it is assumed that a positive value of the relative velocity indicates that the target is moving away from the host vehicle. The target is question is at a range of 120m and traveling with a velocity of 140 km/h.



(a)

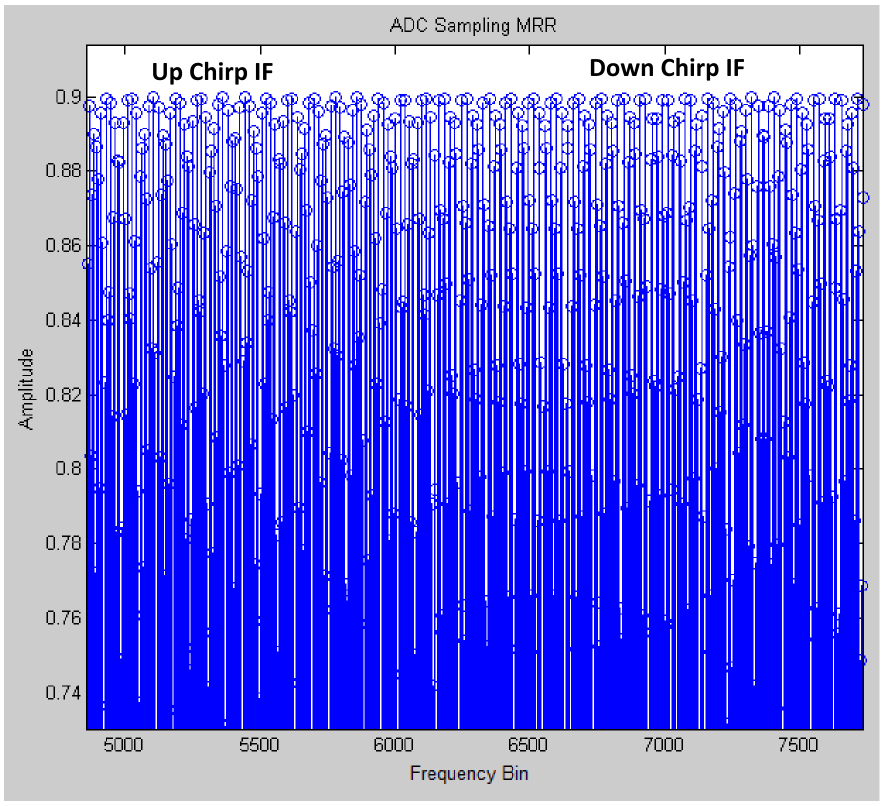


(b)

Figure 5.3 Target detection by the CFAR from the FFT output showing for both Up and Down Chirp

Mid-Range Radar (MRR):

With correspondence with the above test scenario the vehicle within the mid range is at 90m traveling with a velocity of 100km/h. Here again the process for computing the information about the detected target was done like those in LRR and the following simulation figures were obtained. The presence of Gaussian White is also noted here.



(a)

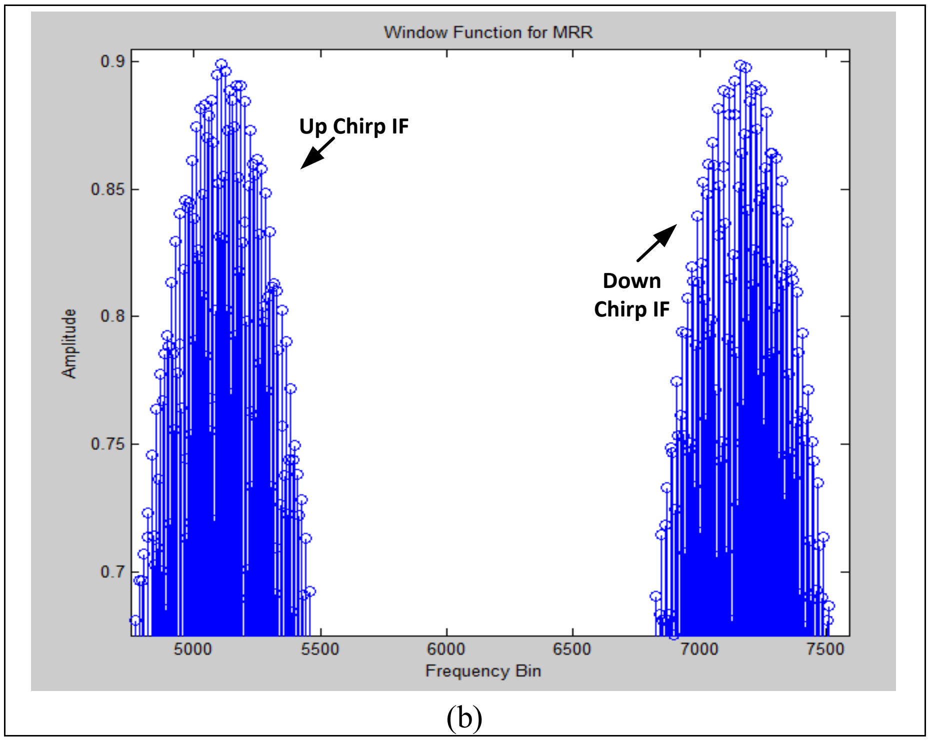
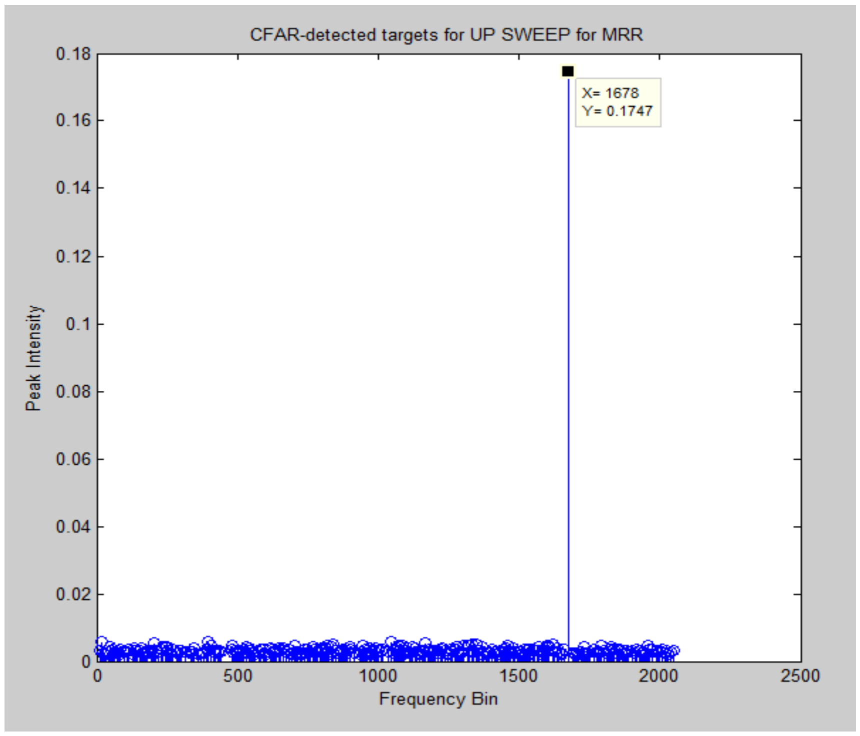
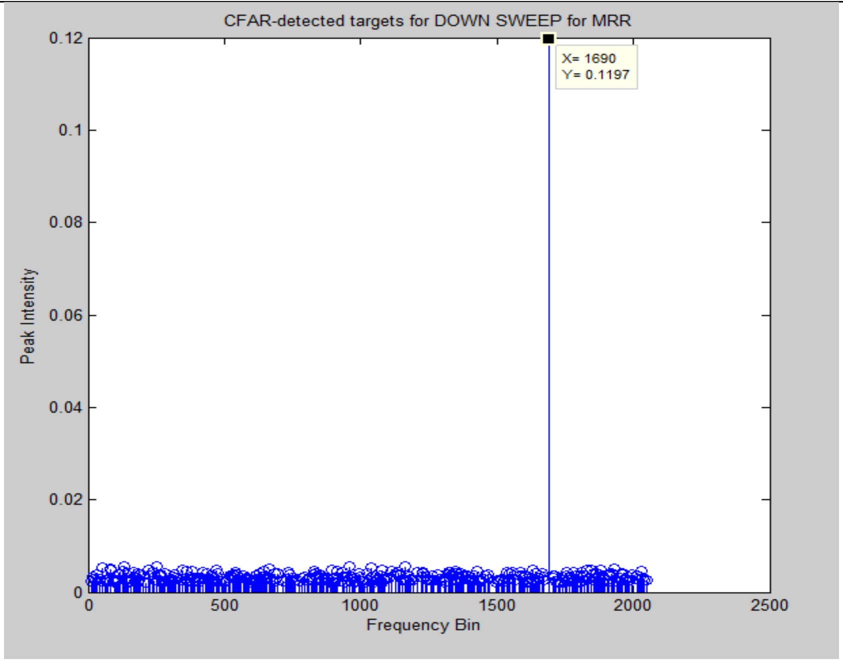


Figure 5.4 (a) ADC sampling for the Up and Down IF signal and (b) The signal after being multiplied by the Hamming Window function

The vehicle detected within the beam 2 region is verified by satisfying the condition for peak paring: the frequency bin shift and peak intensity was found to be 12 $\left(X_{DOWN} - X_{UP} = 1690 - 1678\right)$ and 0.055 $\left(Y_{UP} - Y_{DOWN} = 0.1747 - 0.1197\right)$ respectively.



(a)

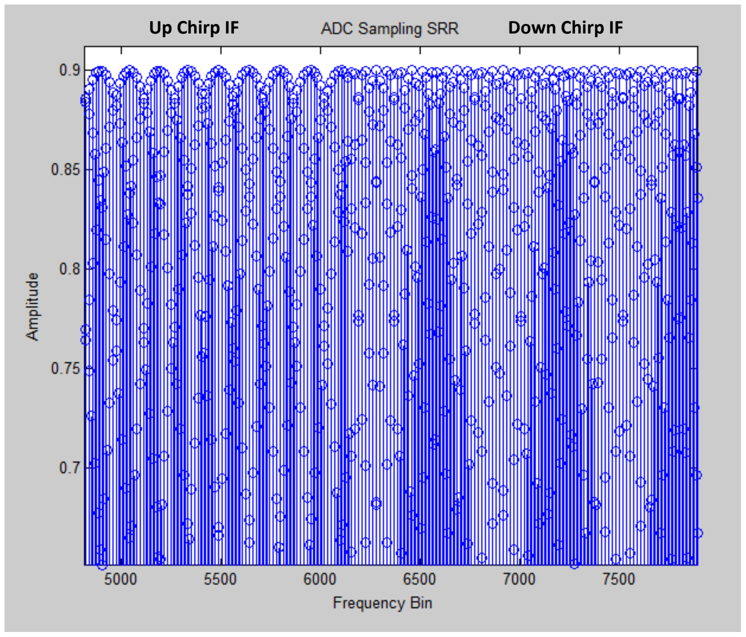


(b)

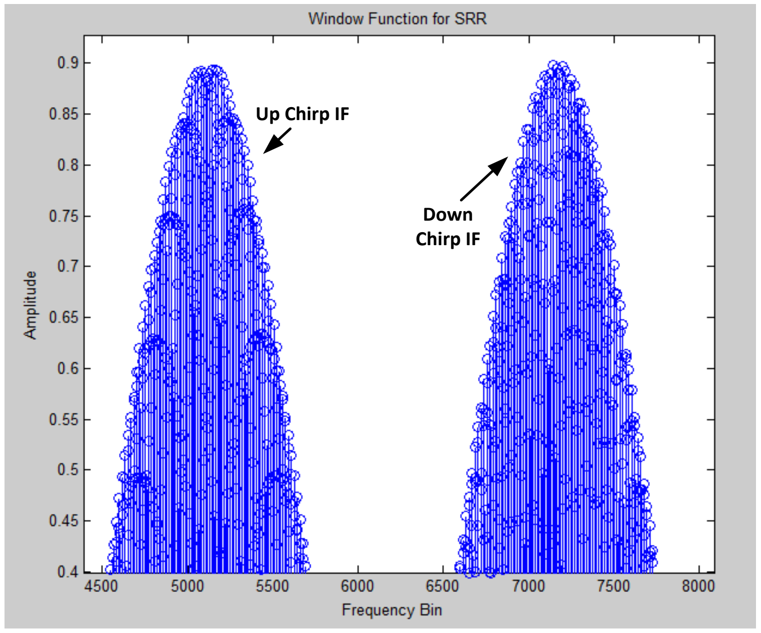
Figure 5.5 Target detection in the Mid-range shown for Up and Down Sweep

Short Range Radar (SRR):

Consistent with the test scenario shown in Figure 5.1 the target assumed with the short range is at a range of 8m travelling with a velocity of 85 km/h.



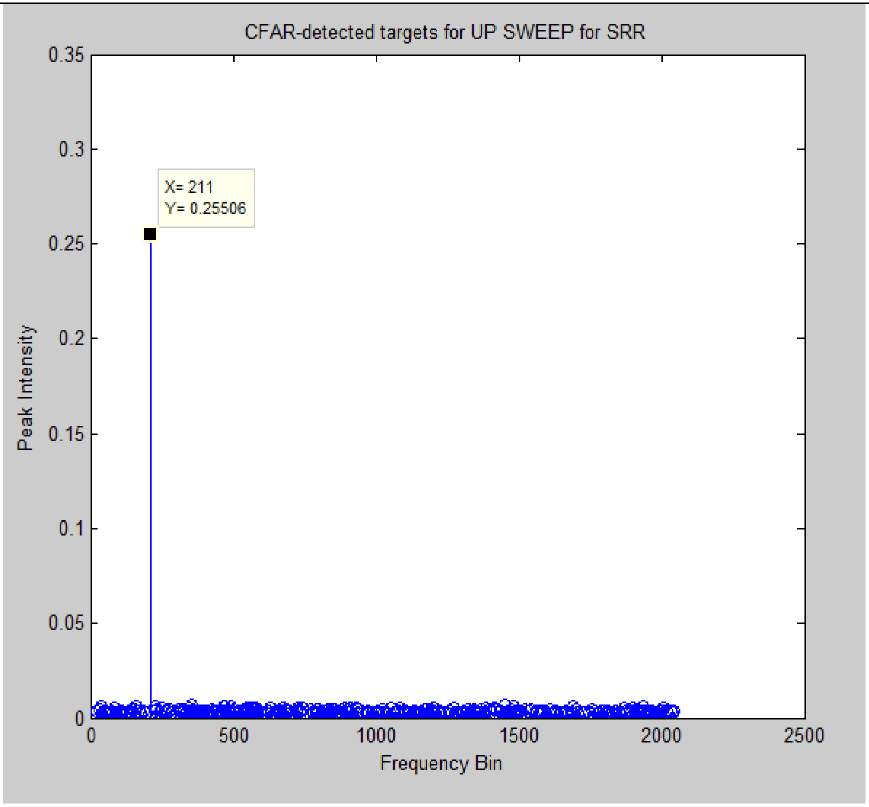
(a)



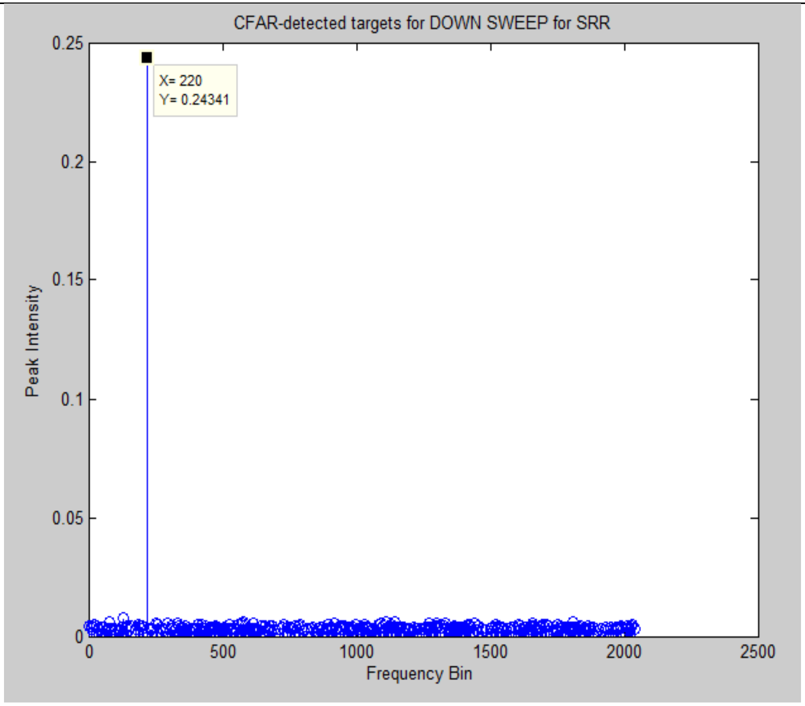
(b)

Figure 5.6 ADC Sampling of the Up and Down sweep signal and the corresponding signal after been multiplied by the Hamming Window function

Target verification with regard to the frequency bin shift and the peak intensity is also done here and found to be $9\left(X_{DOWN} - X_{UP} = 220 - 211\right)$ and $0.01165\left(Y_{UP} - Y_{DOWN} = 0.25506 - 0.24341\right)$ respectively.



(a)



(b)

Figure 5.7 Target detection within the beam 3 region of the Short Range Radar is shown and verified for the Up and Down Sweep

5.2 Verification of the Test Scenario

The results for range and velocity acquired from MATLAB for different situations in from the Tri-mode radar detection was compared against calculated values using the equations given in chapter 3. The table below provides with the assumed range and velocity values considered to verify against the MATLAB calculated values for this scenario. The range and velocities can also be calculated using the equations below which have been discussed in Chapter 3.

Table 5.1 Up and Down Frequency values calculated for the range and velocity for single target situation

Modes of Radar	f_{UP} [1] **(Hz)**	f_{DOWN} [2]**(Hz)**	**Range (r)**[3] **(m)**	**Velocity (v_r)**[4] **(km/h)**
LRR	661584	589647	120	70
MRR	836536	805705	90	30
SRR	111977	96562	8	15

Equations used for calculating the parameters are:

[1] $$f_{\mathrm{up}} = f_{\mathrm{R}} + f_{\mathrm{D}} = \frac{2kr}{c} + \frac{2f_0 v_{\mathrm{r}}}{\mathrm{c}}$$

[2] $$f_{\mathrm{down}} = f_{\mathrm{R}} - f_{\mathrm{D}} = \frac{2kr}{c} - \frac{2f_0 v_{\mathrm{r}}}{c}$$

[3] $$r = \frac{(f_{\mathrm{up}} + f_{\mathrm{down}})}{2} \times \frac{c}{2k}$$

[4] $$v_{\mathrm{r}} = \frac{(f_{\mathrm{up}} - f_{\mathrm{down}})}{4} \times \frac{c}{f_0}$$

Here the relative velocity v_r is found by calculating the difference between the target velocity and the host velocity.

Table 5.2 shows the value of the range and velocity from MATLAB simulations with corresponding Up and Down frequency sweep.

Table 5.2 MATLAB results of the range and velocity for single target situation

Modes of Radar	**Up_bins**	f_{UP}[1] **(Hz)**	**Down_bins**	f_{DOWN}[2] **(Hz)**	**Range_M (r)[3] (m)**	**Velocity_M (v_r)[4] (km/h)**
LRR	653	637695	633	618164	120.14	71.83
MRR	845	825195	839	819336	90.01	30.78
SRR	120	1171876	116	113281	8.84	13.68

[1] $$f_{\mathrm{up}} = up\_bins \times \frac{2MHz}{2048}$$

[2] $$f_{\mathrm{down}} = down\_bins \times \frac{2MHz}{2048}$$

Table 5.3 Difference between MATLAB and Actual values

Modes of Radar	Range (r)[3] (m)	Velocity (v_r)[4] (km/h)	Range_M (r)[3] (m)	Velocity_M (v_r)[4] (km/h)	Error in Range	Error in Velocity
LRR	120	70	120.1	71.83	0.1	1.83
MRR	90	30	90.06	30.78	0.01	0.78
SRR	8	15	8.84	13.68	0.84	1.32

The SRR shows the maximum error for range calculation and LRR shows the maximum error in calculating velocity.

5.3 Multiple target Test Scenario

In the multiple targets scenario multiple targets are being assumed at different range travelling with different velocities and the signal detection at each mode of radar are being observed for both up and down frequency sweep. The host vehicle velocity is again considered to be 70 km/h.

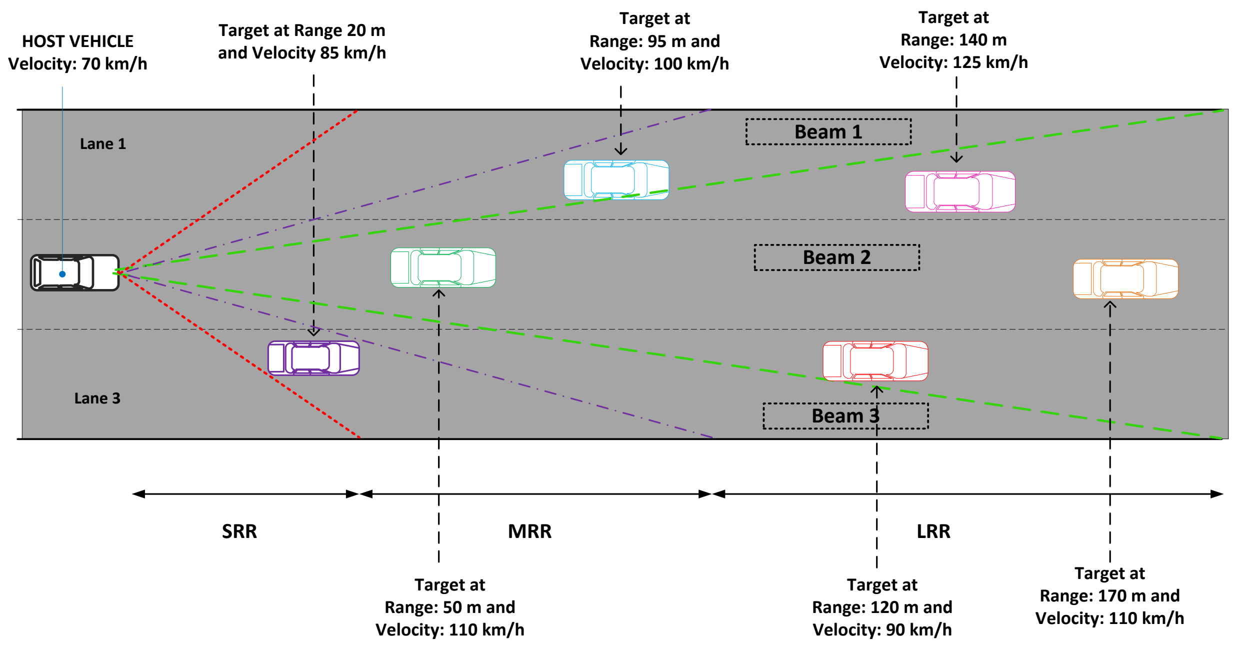
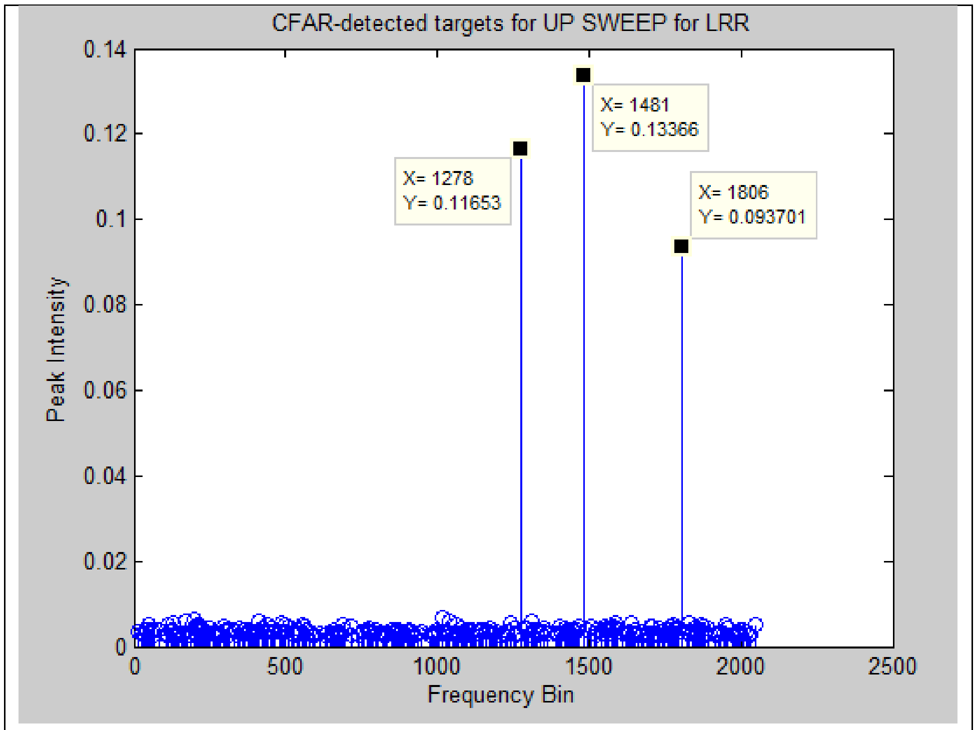
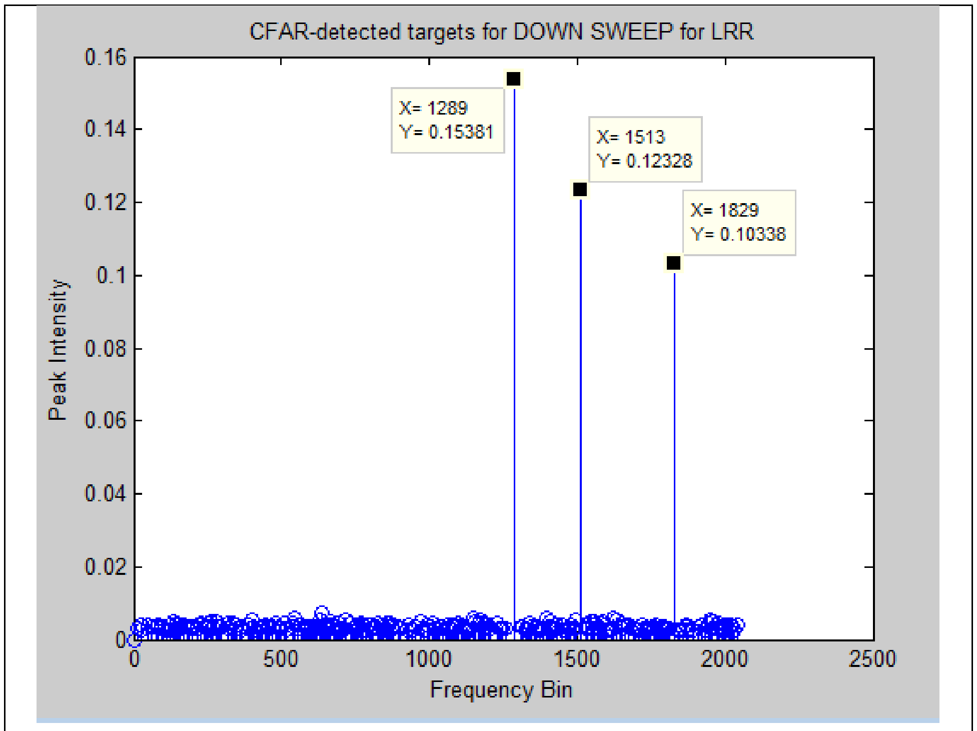


Figure 5.8 Concocted multiple targets scenario for observing the Tri-mode Radar detection efficiency

The results obtained from the CA-CFAR algorithm in the MATLAB simulation are again verified against the calculated values as done in the previous section.

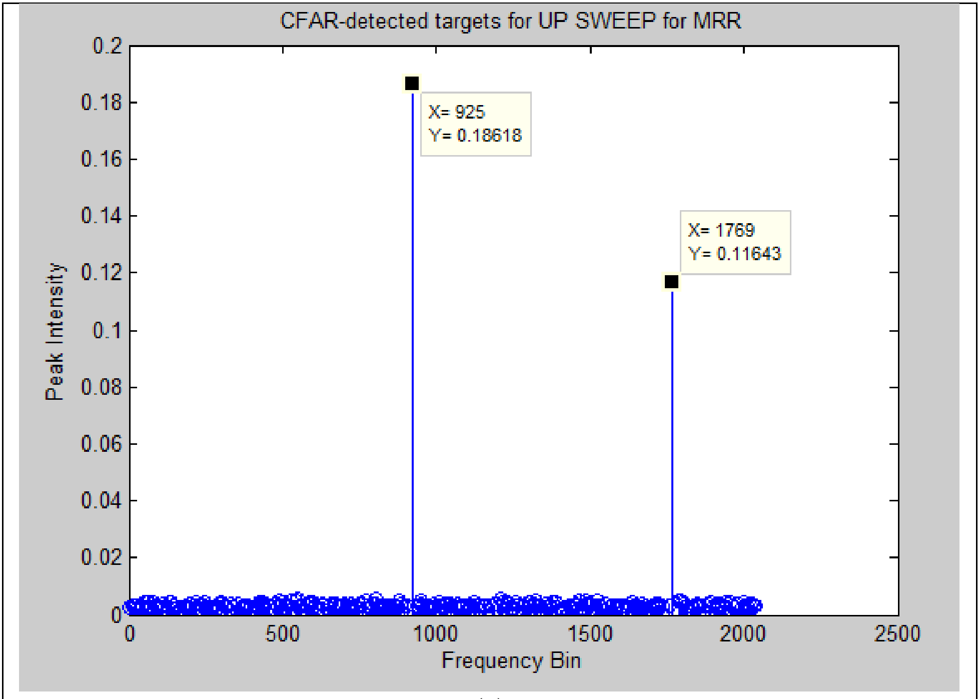


(a)

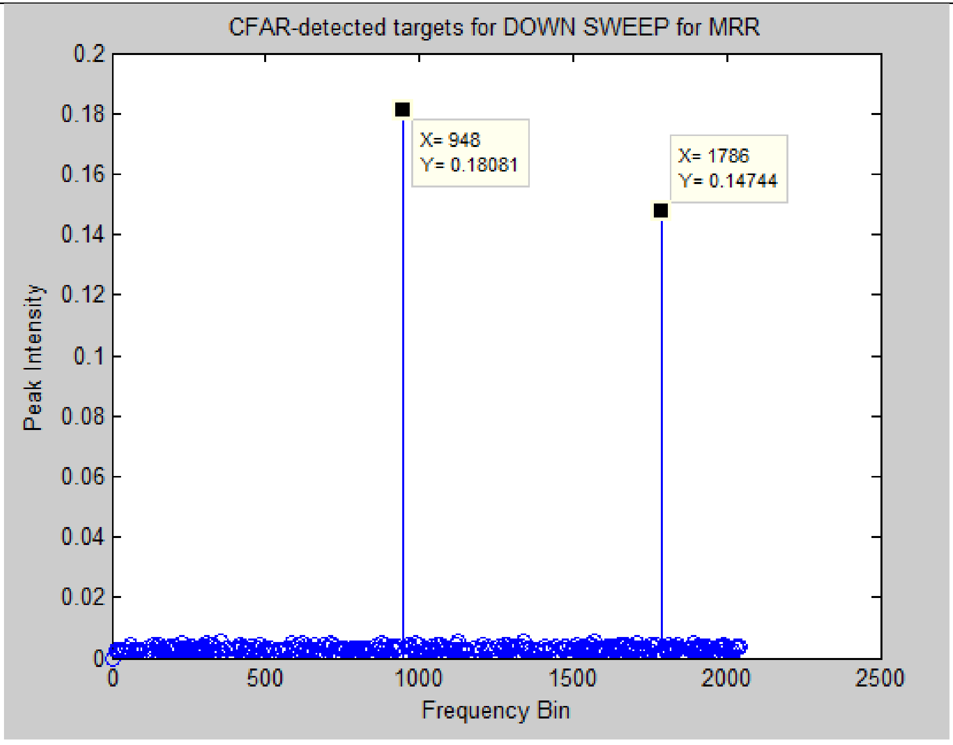


(b)

Figure 5.9 Multiple targets detected from the hypothetical scenario in the LRR mode

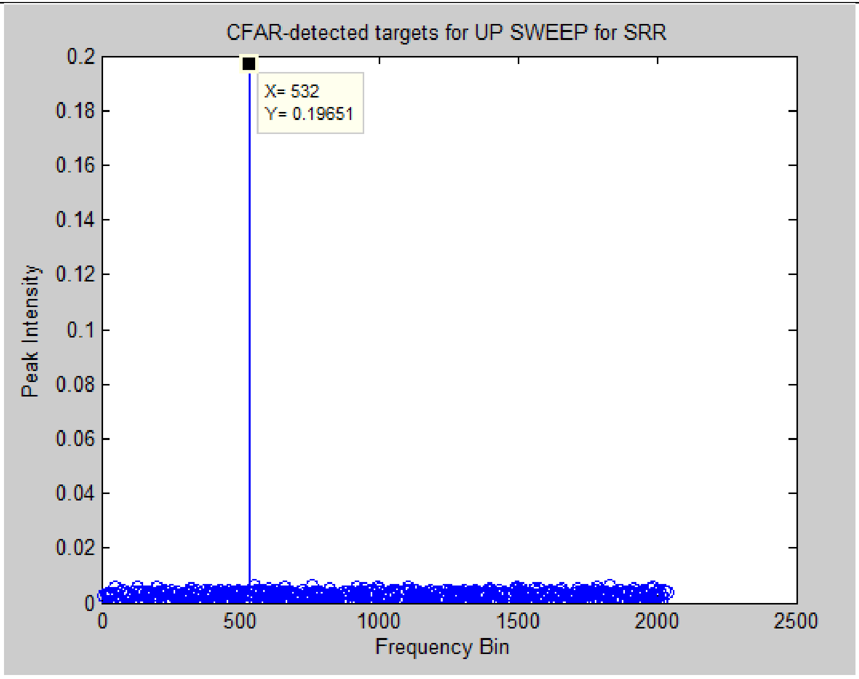


(a)

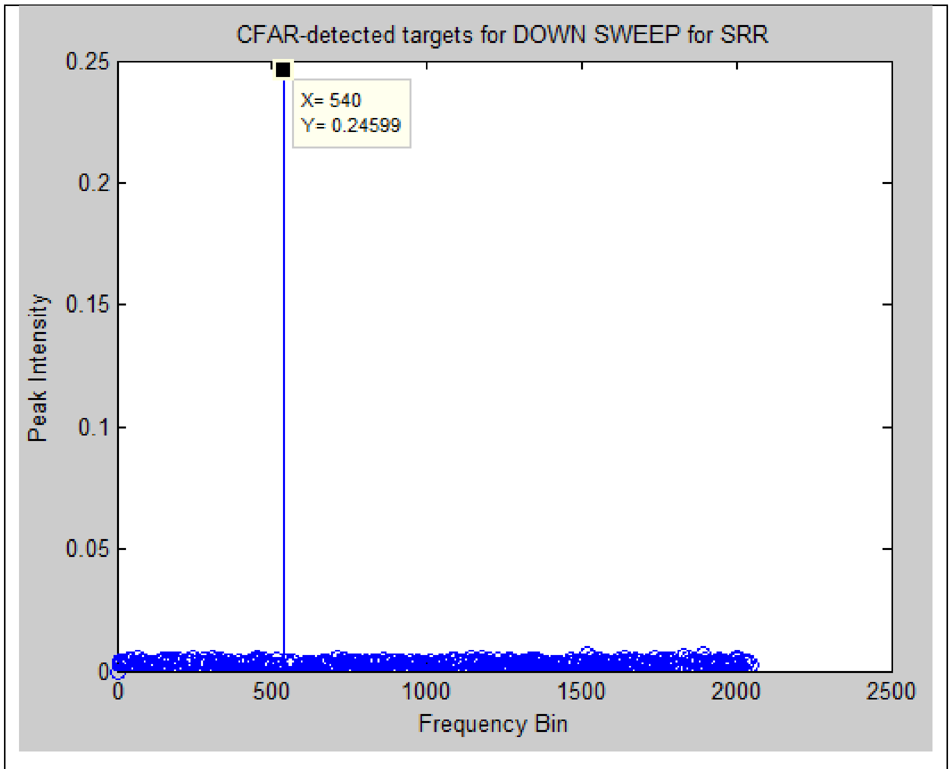


(b)

Figure 5.10 Targets detected by MRR in the multiple targets situation



(a)



(b)

Figure 5.11 Target detected by SRR sensor in the multiple targets scenario

Table 5.4 shows the results calculated for each of the target detected by the Tri-mode radar. Here again the relative velocity v_r is the difference between the host velocity and the target velocity.

Table 5.4 Up and Down frequency sweep calculated for the hypothetical situation for theTri-mode radar test scenario

Modes of Radar	f_{UP} [1] (Hz)	f_{DOWN} [2] (Hz)	Range (r)[3] (m)	Velocity (v_r)[4] (km/h)
LRR	794115	665655	140	55
	942811	829767	170	40
	671861	579370	120	20
MRR	918122	815354	95	30
	512700	399656	50	40
SRR	304350	216997	20	15

The results obtained from the MATLAB simulations are tabulated below:

Table 5.5 Results obtained from the MATLAB simulation for the given situation

Modes of Radar	**Up_bins**	f_{UP} [1] **(Hz)**	**Down_bins**	f_{DOWN} [2] **(Hz)**	**Range_M (r)[3] (m)**	**Velocity_M (v_r)[4] (km/h)**
LRR	756	738281	740	722656	140.09	54.73
	914	892578	903	881836	170.05	41.05
	644	628906	639	624023	120.05	20.53
MRR	893	872070	894	873047	95.1	30.8
	474	462891	462	451172	50.19	41.05
SRR	274	267578	270	263672	20.38	13.68

The difference in the values is shown in Table 5.6

Table 5.6 Error calculations between the actual values and the simulated values

Modes of Radar	**Range (r)[3] (m)**	**Velocity (v_r)[4] (km/h)**	**Range_M (r)[3] (m)**	**Velocity_M (v_r)[4] (km/h)**	**Error in Range (m)**	**Error in Velocity (km/h)**
LRR	140	55	140.09	54.73	0.09	0.27
	170	40	170.05	41.05	0.05	1.05
	120	20	120.05	20.53	0.05	0.53
MRR	95	30	95.1	30.8	0.1	0.8
	50	40	50.19	41.05	0.19	1.05
SRR	20	15	20.38	13.68	0.38	1.32

The maximum error for both range and velocity was found for SRR.

5.4 Observation Summary

The design algorithm of the Tri-mode radar was observed for both single target and multiple target situations. The actual results were verified against the simulated results and the difference is calculated. The maximum error for the range in both the situations was found in the SRR. The error in the velocity was found higher in LRR for the single target situation but for the multiple targets SRR showed a higher error. Although the error seem to be higher considering the situations, especially for SRR where precise location and velocity of the target is highly needed, the values found using the IF received from the targets have lower error percentage. This will be shown in the next chapter.

Chapter 6

Hardware Implementation of the Tri-mode Radar Design

In this chapter the hardware implementation of the radar signal processing algorithm is discussed highlighting the each process black discussed above. Simulations in the hardware are based on the fixed-point system while the software simulations in MATALB were done using floating-point system. The results obtained from the hardware and software simulations are tabulated and compared at the end of this chapter.

6.1 Hardware Implementation

The radar signal processing was implemented on Xilinx FPGA Virtex 5 SX50T using Verilog HDL (Hardware Descriptive Language). The IF signal from the received echoes was first passed through another platform Analog to Digital Convertor (ADC) board (AD7x76/77CBZ). The Verilog code was simulated using an ISIM simulator. The choice for the platform was discussed in detail in chapter- 2. The aim of the MEMS automotive radar designed in University of Windsor is to have smaller computation latency per sweep and also a smaller cycle time.

The Virtex-5 family provides many compelling features for advanced logic designs. The platform used in this thesis SXT is efficient for signal processing applications with advanced serial connectivity. Virtex-5 FPGAs contain many IP core blocks, enhanced clock management modules with integrated DCM (digital clock manager) and phased-locked loop (PLL) clock generators among other prominent features [58]. The device is built on a 65 nm tri-oxide technology reducing the core voltage to 1.0 V and which improves the gate-level performance bu ensuring that all transistors in the FPGA are not switched at maximum speed. Some prominent features of the Virtex-5 SXT family are:

1. Maximum Distributed RAM (in Kb) in the Configurable Logic block (CLB) - 788
2. DSP48E Slices – it has 288 slices, each slice contains a 25 x 18 multiplier, an adder and an accumulator
3. Block RAM (BRAM) – 4752 Kb. The size of the BRAM has increased to 36 Kbits from the 18 Kbits of the Virtex-4 device, making it easier for larger memory arrays to be built. Also the larger 36 Kbits can be used as independent blocks of 18 Kbits if necessary [59].

4. Look Up Table (LUT) – It contains nearly 207,360 real 6-input LUTs with over that 13 million total LUT bits [58].
5. Clock Management Tiles (CMT) – The Virtex-5 Clock is from CMT blocks which contains two DCMs and on PLL. The DCM is for controlling delay and improves noise immunity and the PLL helps to generate lower clock jitter and filter jitter. Each Virtex-5 device is provided with six CMTs each of which is capable of generating clock frequency of 550 MHz.
6. SelectIO – Contains up to 1200 packaged I/Os which can be configured from 1.2V to 3.3V.

Detail about the ADC platform used in this thesis is given in chapter 2. The total system configuration for the Tri-mode radar design using the Virtex-5 FPGA platform, ADC 7x76 evaluation platform and other required components are shown in Figure 6.1:

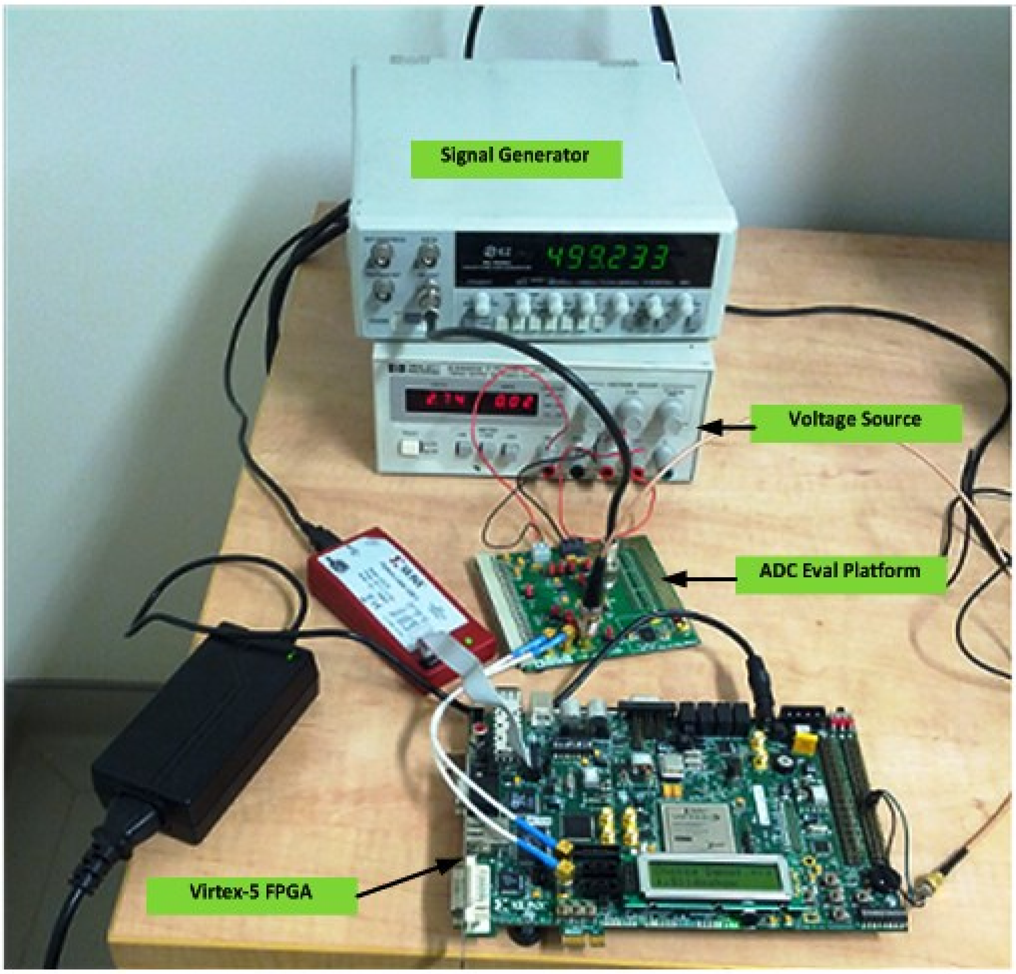


Figure 6.1 The Virtex-5 FPGA platform and ADC platform used in the design of Tri-mode radar signal processing

6.2 Radar Signal Processing Algorithm

The radar signal processing algorithm using the fixed-point system is written in Verilog code using Xilinx ISE 13.4 Design Suite. The HDL coding and processing are carried out according to the following hierarchy algorithm:

1. The global clock 100 MHz is synchronized with the 48 MHz and 64 MHz clock which are related to the ADC and DAC algorithm respectively.

2. The 48 MHz clock frequency of the ADC is synchronized with the Built-in FIFO. This is done because the ADC and FFT have different clock frequencies. The FIFO is acting as buffer storage between the window functions and FFT. The read clock frequency for the FIFO is 100 MHz while the data is written into it at 48 MHz.
3. The ADC samples are multiplied by the Hamming window function and are then passed into the FIFO. Data is written into the window multiplier at 48 MHz clock frequency analogous to ADC clock. The window process is done using the Block Memory Generator [60] .
4. The windowed samples ate collected from the FIFO generator block and passed in the FFT Core Generator at 100 MHz.
5. The FFT output (both real and imaginary) are then squared and the magnitudes of the samples are obtained.
6. Output of the FFT is passed into the CA-CFAR algorithm where the relevant signals from the detected targets are extracted from the noise and clutter.
7. The peak intensities of the CFAR outputs are then paired to match the affiliated targets from the up and down sweep.
8. Once the target information is extracted using the peak-paring algorithm the range and velocity is calculated and displayed in the LCD of the FPGA board.

The total Verilog code for the total algorithm is given in Appendix 1 (b)

6.3 The Processing Blocks Used In the Design

The block diagram for the total algorithm is given in Figure 6.2. All the process blocks are instantiated in the top module. This process stages are similar with the algorithm used in MATLAB simulation.

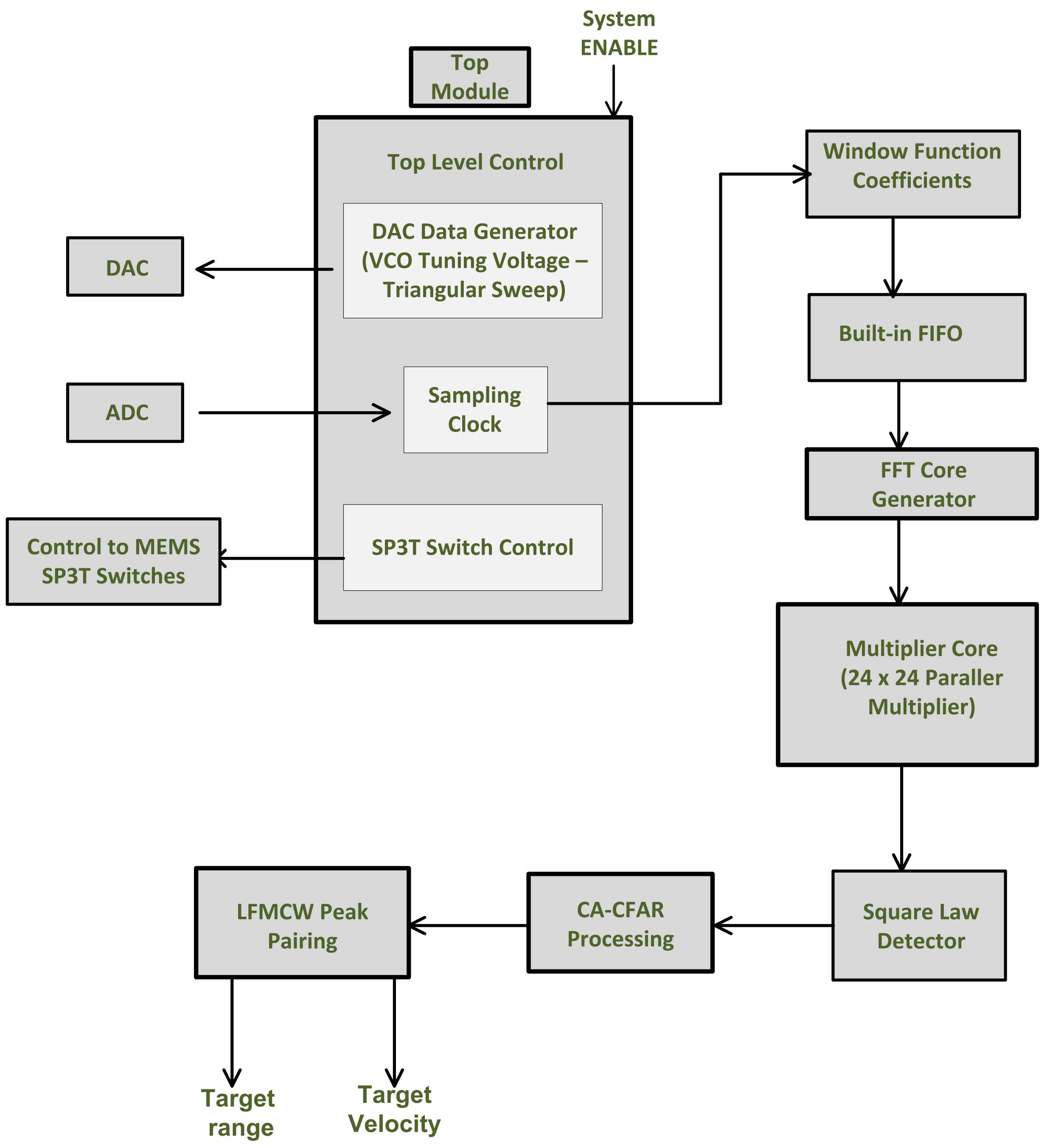


Figure 6.2 Block diagram showing the processing stages of the HDL algorithm

6.4 *The Hardware Simulation technique and the Individual Process Blocks*

The hardware implementation of the radar signal processing designed in the University of Windsor is shown in Figure 6.1. The signal generator is used to provide the IF from the targets detected. The output from the signal generator goes into the ADC evaluation board where the signal is sampled. The voltage source is used to provide the ADC with V_{DD}. The ADC can sample up to a rate of 3 Msps as mentioned in chapter 2. Sample data is passed into the FPGA through SDATA pin which serially streams data out of the ADC. The connection between ADC and FPGA is established using SDATA, SCLK and J10 , J11 pins of the ADC and the FPGA platforms respectively.

The end results i.e. target range and target velocity each are calculated using the fixed point system. The advantage of using fixed-point over floating point is that results are most accurate for a given number of bits per number and also it consumes less space. In the fixed-point algorithm a scaling factor and a particular register width is assumed. These factors decide the position of the decimal point of the resultant value.

The figure below shows a top level system block for the design.

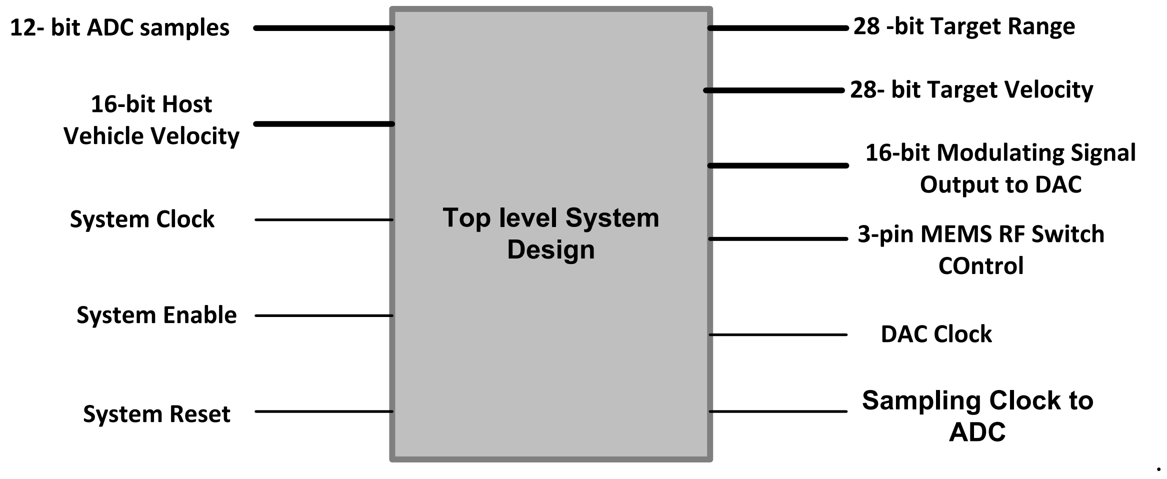


Figure 6.3 Black Box view of the Radar Signal processing algorithm. The thicker lines indicates bus and the inputs lines are on the left and the outputs are on the right

6.5 Range and Velocity Calculation Methodology

As describes above the results for target range and velocity are calculated using the fixed-point system. The 28-bit range and velocity value is divided into 12 –bit from MSB for the integer part and 16-bit from LSB for the fractional part. This ensures a precision of $1/65536$ in the calculated range and velocity values of the HDL code.

6.6 RTL Design View of the Process Blocks

For each processing algorithm RTL design view is being generated showing the pin connectivity with the corresponding bit size. To begin with the RTL view of the top module is generated and shown in Figure (6.4):

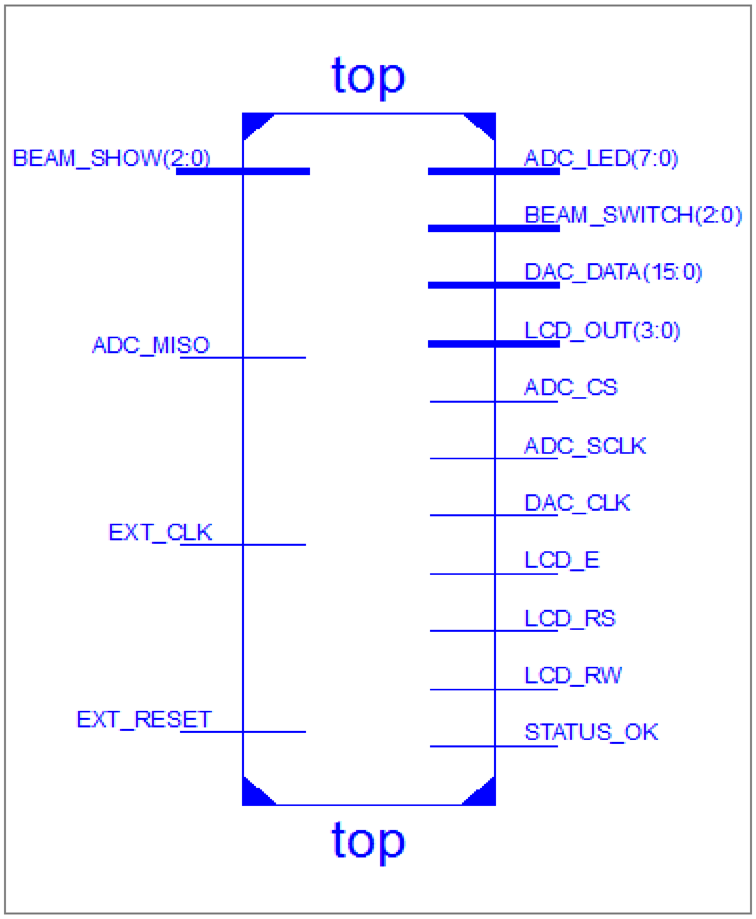


Figure 6.4 The Top Module of the HDL algorithm

Table 6.1 Details for the pins shown in the Top Module

HDL Pin	I/O	Description
ADC_MISO	Input	The ADC sampled data from the AD7x76/77CBZ)
EXT_CLK	Input	System clock at 550MHz from the ML506 board
EXT_RESET	Input	Global synchronize reset
BEAM_SHOW	Input	Displays the active beam number
ADC_LED	Output	The sampled value excluding the leading and trailing zeros
BEAM_SWITCH	Output	Controls pins to control switching between three beam ports to control the radar beam direction
DAC_DATA	Output	The tuning voltage modulated by the up and down counter
DAC_CLK	Output	Clock frequency for the Digital to Analog convertor which is 64 MHz
ADC_CS	Output	Chip Select for the ADC initiates data conversion and also manage the serial data transfer
ADC_SCLK	Output	The serial clock for the ADC which is 48 MHz
LCD_E	Output	Enabling the LCD display
LCD_RS	Output	Selecting the read enable pin of the LCD
LCD_RW	Output	Selecting the write enable pin of the LCD
STATUS_OK	Output	The input received is valid

The following figures and tables will show the RTL design view for the signal processing done to obtain the target range and velocity results before they are displayed on the LCD.

The ADC Processing:

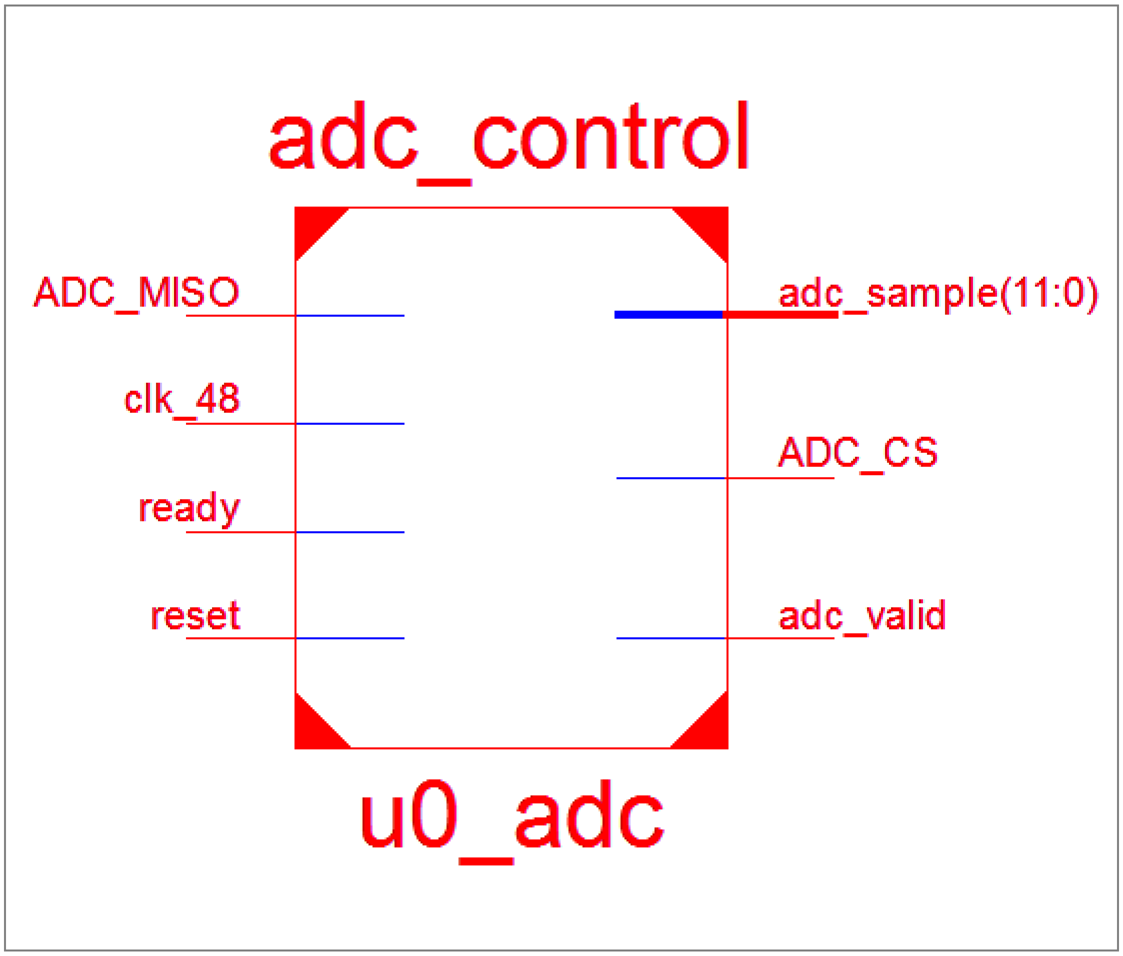


Figure 6.5 The RTL design viewer of the ADC computation

Table 6.2 Pins configuration of the ADC block

Pin name	I/O	Description
ADC_MISO	Input	The ADC sampled data from the AD7x76/77CBZ)
clk_48	Input	The clock frequency for the ADC which is 48 MHz
ready	Input	ADC values are ready with valid data
reset	Input	Global synchronous reset
adc_sample	Output	The sampled data from the ADC
ADC_CS	Output	Chip Select for the ADC initiates data conversion and also manage the serial data transfer
adc_valid	Output	The data received are valid

The Window Function:

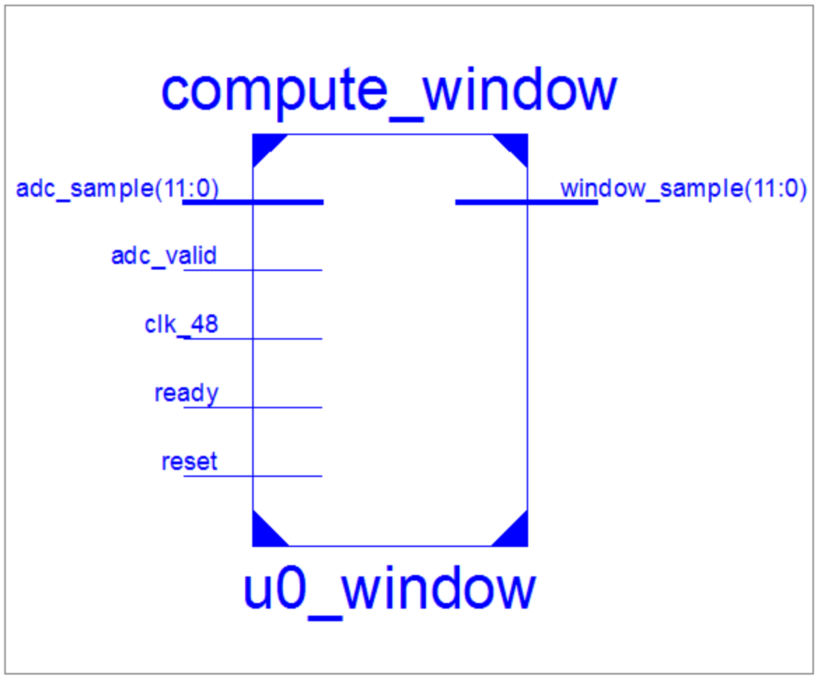


Figure 6.6 RTL view of the Window Processing block

Table 6.3 Pins configurations for the Window block

Pin name	Direction	Description
adc_sample	Input	The 11 bit sampled data from the ADC
adc_valid	Input	The data received are valid
clk_48	Input	The clock frequency for the ADC which is 48 MHz
ready	Input	ADC values are ready with valid data for windowing
reset	Input	Global synchronous reset
window_sample	Output	The 11 bit sampled data after being multiplied by the window function

Fast Fourier Transform processing:

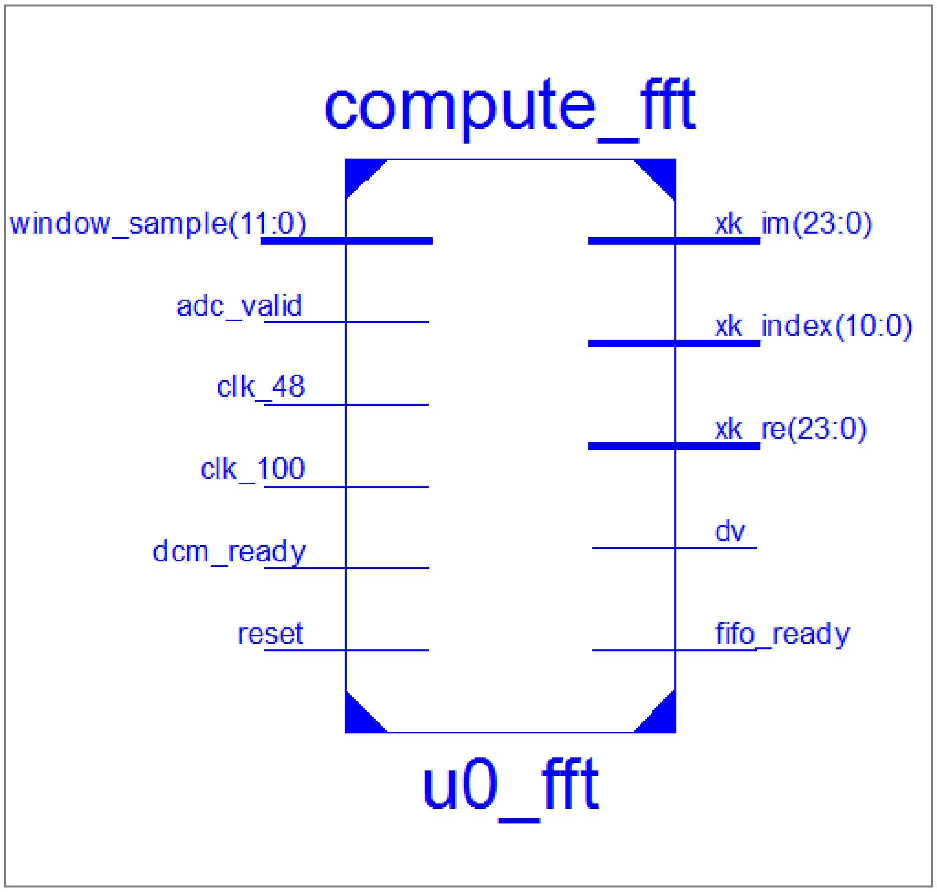


Figure 6.7 RTL view of the FFT process block

Table 6.4 Table showing the pin configurations

HDL port	Direction	Description
window_sample	Input	The 11 bit sampled data after being multiplied by the window function
adc_valid	Input	The data received are valid
clk_48	Input	The clock frequency for the ADC which is 48 MHz
clk_100	Input	Global clock for the system 100 MHz
dcm_ready	Input	Digital clock Manager ready for blocking zero delay
reset	Input	Global synchronous reset
xk_im	Output	Output data bus. Imaginary component represented either in two's complement or single precision floating-point format [40]
xk_re	Output	Output data bus. Real component represented either in two's complement or floating-point format [40]
xk_index	Output	Index of output data [40]
dv	Output	Active high when valid data is present at the output
fifo_ready	Output	The FIFO is full and is ready to output data into the FFT core Generator

The CA-CFAR process block:

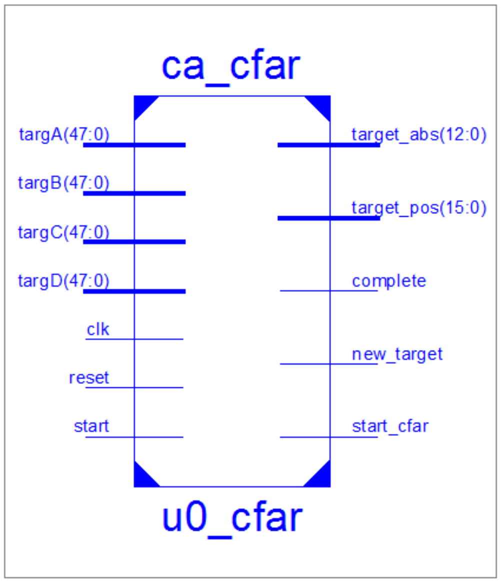


Figure 6.8 The RTL design view of the CA-CFAR process

Table 6.5 HDL port configuration

HDL port	Direction	Description
tarA	Input	Assumed detected target A
tarB	Input	Assumed detected target B
tarC	Input	Assumed detected target C
tarD	Input	Assumed detected target D
clk	Input	Global clock which is 100 MHz
reset	Input	Global synchronous reset
start	Input	A delayed version of the data is set up for using in the multiplier clock enable and the load buffer enable
target_abs	output	Target peak intensity
target_pos	output	New target frequency bin number
complete	Output	When loading of all the squared data is complete
new_target	Output	New target data
start_cfar	Output	CFAR processing is start

Peak-pairing and target information computation:

The data received from the CA-CFAR is paired and the range and velocity value of the target is computed using the process block shown below:

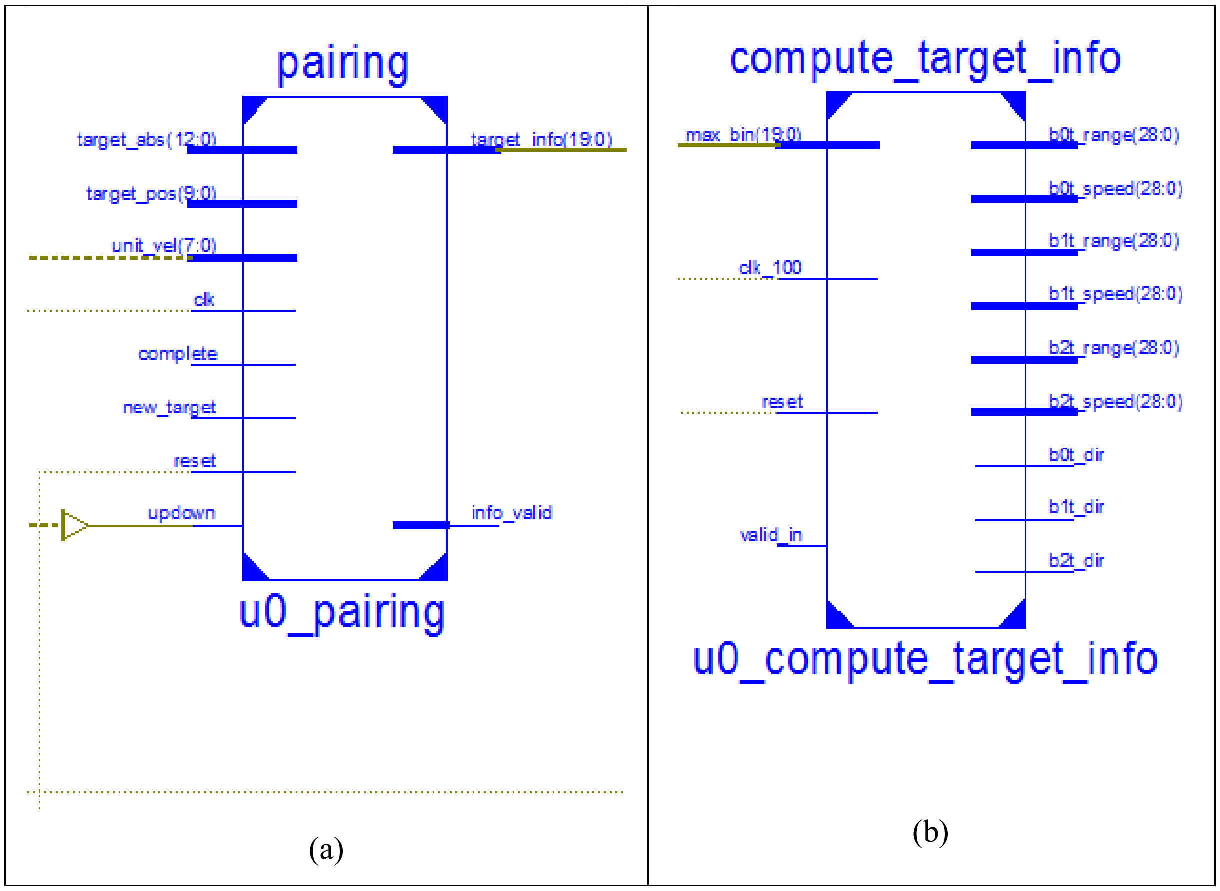


Figure 6.9 Process block showing the (a) peak paring and (b) result computation process

Table 6.6 Pins description for the Process block given above

HDL port name	I/O	description
clk	Input	Global clock which is 100 MHz
reset	Input	Global synchronous reset
unit_vel	Input	Velocity of the host vehicle
target_abs	Input	Target peak intensity
target_pos	Input	New target frequency bin number
complete	Input	When loading of all the squared data is complete
new_target	Input	New target data
updown	Input	Indicated the up and down sweep. Active High indicate up sweep and low refers to a down sweep
valid_in	Input	Valid input received after the info_valid is high in the pairing block
max_bin	Input	The bin of the target received
clk_100	Input	Global clock which is 100 MHz
bot_range/ b1t_range/ b2t_range	Output	The value of range of the target detected
bot_speed/ b1t_speed/ b2t_speed	Output	The value of the velocity of the target detected
b0t_dir/ b1t_dir/ b2t_dir	Output	Holds the sign bit for the velocity

Each process blocks are simulated and the results for range and velocity was calculated and displayed in the LCD in the FPGA platform.

6.7 Hardware Synthesis Results for the Radar design

The resources used in the developing the algorithm is shown in the tables below:

Table 6.7 Resources used for (a) SRR (b MRR) (c) LRR

(a)

Resource	Used	Available	Percentage Usage
Slice registers	2200	32640	6%
Slice LUTs	**3692**	**32640**	**11%**
DSP48E slices	8	288	2%
Fully used LUT-FF pairs	**1397**	**4271**	**32%**
BUFG/BUFGCTRLs	7	32	21%

(b)

Resource	Used	Available	Percentage Usage
Slice registers	1753	32640	5%
Slice LUTs	2126	32640	6%
DSP48E slices	7	288	2%
Fully used LUT-FF pairs	876	3003	29%
BUFG/BUFGCTRLs	7	32	21%

(c)

Resource	Used	Available	Percentage Usage
Slice registers	1634	32640	5%
Slice LUTs	2120	32640	6%
DSP48E slices	7	288	2%
Fully used LUT-FF pairs	875	2879	30%
BUFG/BUFGCTRLs	7	32	21%

Table 6.8 Comparison between the LRR designed in this thesis with a previous version of LRR designed in the University of Windsor [20].

Resource	Percentage Usage	Percentage Usage [21]
Slice registers	5%	4%
Slice LUTs	6%	23%
DSP48E slices	2%	6%
Fully used LUT-FF pairs	30%	9%
BUFG/BUFGCTRLs	21%	3%

It can be seen from the above table that the current design used less Slice LUTs and DSP48E slice than the previous design which indicates that the Tri-mode radar can be implemented using a smaller area compared with the LRR sensor designed previously. In addition to this the results obtained from the current algorithm has greater accuracy than the previous design.

The processing time for each mode of radar are given in the tables below:

Table 6.9 Processing time for each process block for (a) SRR (b) MRR and (c) LRR

(a)

Processing Block	Time (s)
ADC requires:	0.001487
Windowing requires:	0.000583
FFT requires:	0.000266
CA-CFAR requires:	0.00051
Signal Processing Latency	9.60E-09
Total Latency	0.002846

(b)

Processing Block	Time (s)
ADC requires:	0.001582
Windowing requires:	0.000526
FFT requires:	0.000266
CA-CFAR requires:	0.000626
Signal Processing Latency	9.60E-09
Total Latency	0.003

(c)

Processing Block	Time (s)
ADC requires:	0.00146
Windowing requires:	0.000261
FFT requires:	0.000133
CA-CFAR requires:	0.00025
Signal Processing Latency	4.7E-09
Total Latency	0.00210

6.8 Results Comparison between HDL implementation and MATLAB Simulation

The results obtained for different IF generated from the signal generator is obtained from the LCD display after simulating in the HDL was compared against the values acquired from the MATLAB simulations. The percentage error between the values was found to be very low which justifies a successful implementation of the Tri-mode Radar. The cycle time for each mode was found to be 6.144ms where each up and down sweep takes 1.024ms each.

Table 6.10 Results obtained from the MATLAB simulations is compared against calculated values

Target Range* (m)			IF(kHz)	Matlab Values of Range (m)		
SRR	**MRR**	**LRR**		**SRR**	**MRR**	**LRR**
6.14	8.77	15.34	80	6.14	8.77	15.35
9.21	13.15	23.02	120	9.22	13.16	23.03
12.28	17.54	30.69	160	12.29	17.55	30.71
15.34	21.92	38.36	200	15.36	21.94	38.39
18.41	26.31	46.03	240	18.43	26.32	46.07
21.48	30.69	53.70	280	21.50	30.72	53.75
26.08	37.27	65.22	340	26.07	37.24	65.18
30.69	43.84	76.72	400	30.72	43.88	76.79
	54.80	95.90	500		54.79	95.89
	65.76	115.09	600		65.71	114.99
	87.68	153.45	800		87.65	153.39
	98.64	172.63	900		98.68	172.68
	100.29		915		100.28	
		186.06	970			185.98
		192.0	1001			191.59

Table 6.11 Results obtained from the LCD display and Compared with the calculated values

Target Range* (m)			IF(kHz)	HDL values from LCD Display Range (m)		
SRR	**MRR**	**LRR**		**SRR**	**MRR**	**LRR**
6.14	8.77	15.34	80	6.14	8.78	15.36
9.21	13.15	23.02	120	9.21	13.16	23.04
12.28	17.54	30.69	160	12.28	17.55	30.71
15.34	21.92	38.36	200	15.35	21.94	38.39
18.41	26.31	46.03	240	18.43	26.33	46.07
21.48	30.69	53.70	280	21.50	30.72	53.75
26.08	37.27	65.22	340	26.07	37.24	65.18
30.69	43.84	76.72	400	30.72	43.88	76.79
	54.80	95.90	500		54.70	95.80
	65.76	115.09	600		65.70	115.10
	87.68	153.45	800		87.70	153.50
	98.64	172.63	900		98.60	172.70
	100.29		915		100.30	
		186.06	970			186.10
		192.0	1001			191.50

Table 6.12 Error percentage between the MATLAB Simulated values and HDL values

IF(kHz)	Error* between HDL and Matlab (%)		
	SRR	MRR	LRR
80	0.00	0.11	0.07
120	0.11	0.00	0.04
160	0.08	0.00	0.00
200	0.07	0.00	0.00
240	0.00	0.04	0.00
280	0.00	0.00	0.00
340	0.00	0.00	0.00
400	0.00	0.00	0.00
500		0.16	0.09
600		0.02	0.10
800		0.06	0.07
900		0.08	0.01
915		0.02	
970			0.06
1001			0.05

From the table above the average error percentage for the results obtained from MATLAB modeling was found to be 0.015, 0.02 and 0.07 for SRR, MRR and LRR respectively. From HDL it was found to be 0.01, 0.03 and 0.07 percent for SRR, MRR and LRR respectively.

Chapter 7

Conclusion and Future Work

The drastic rise in the number of vehicles on the road every year has made the radar sensor a very popular device parameter for the automotive industry due to the increase number of road accidents. The Tri-mode Radar designed in this thesis has made an important contribution as this sensor can cover three radar ranges, whereas in the current market has individual radars sensors for each range detection. The ability to use one sensor device to cover three modes makes this a very compelling design. In addition the developed Verilog HDL code can be used to fabricate ASCI which will enable to verify the design efficiency and accuracy in a real-time environment. The mathematical modeling and the design approach was based on a previous design [20] where a single mode radar sensor has been designed.

The Tri-mode Radar sensor designed in this thesis shows better performance than the previous design in terms of refresh rate, resource usage and also result accuracy. In addition the ability to use one sensor device for three range coverage make this design more preferable. The refresh rate of 6.144ms accomplished in this thesis has also proved to be eight times faster than the BOSCH LRR3 radar sensor. [61]

The design has been verified by both MATLAB and HDL modeling and the results were found to be in excellent agreement. Also MATLAB modeling was done for multiple target situations ensuring that the sensor can detect targets within the relevant beamwidth.

7.1 Future work

The current design has some scope for improvement. Currently the targets detected are considered to be point sources which means that the IF received from a target is assumed to fall completely within a beam. In reality a large truck may cover two beam width rather than one. Another scope is for the consideration of RCS. At present it is assumed that all the targets give same return echoes which make it difficult to differentiate between a motorcycle and a large cargo truck. The RCS calculation for different targets will also help to detect the threat zone. For this thesis the sweep duration is kept same for the three modes which can be varied to increase result accuracy. The OS-CFAR process can be implemented for better detection in a multiple target environment [14]. The use of a combined LFMCW and FSK modulation technique will improve the measurement time. The clutter detection technique can be improved by generating a road map using the sensor which will help to differentiate between fixed target from a moving one [54].

References

[1] S. Lal, R. Muscedere and S. Chowdhury, "An FPGA-Based signal processing system for a 77GHz MEMS Tri-mode Automative Radar," in *Proceedings of the 2011 22nd IEEE International Symposium on Rapid System Prototyping (RSP)*, Karlsruhe, Germany, 2011, pp 2-8.

[2] J. Hasch, E. Topak, R. Schnabel, T. Zwick, R. Weigel and C. Waldschmidt, "Millimeter-Wave Technology for Automotive Radar Sensors in the 77 GHz Frequency Band," *Microwave Theory and Techniques,* vol. 60, no. 3, pp. 845-860, Mar, 2012.

[3] Infineon Technologies AG, "Infineon targets radar ICs at mid-range cars," INFINEON, 08 November 2007. [Online]. Available: http://www.semiconductor-today.com/news_items/NEWS_2007/NOV_07/INFINEON_081107.htm. [Accessed 15 February 2013].

[4] Group, International Traffic Safety Data and Analysis, "Road Safety Annual Report 2011," OECD/ITF, 2012.

[5] Organization, World Health, "World report on Traffic Injury Prevention," World Health Organization, Geneva, 2004.

[6] A. O. Bauer, "Christian Hülsmeyer and about the early days of radar inventions a survey," Diemen, The Nederlands, 2005.

[7] A. Douglas, "h2g2, The guide to life, the universe and everything," h2g2, 14 July 2003. [Online]. Available: http://www.h2g2.com/approved_entry/A591545. [Accessed 15 Feb 2013].

[8] Altera Corporation, "Implementing Digital Processing for Automotive Radar Using SoC FPGAs," Altera Corporation, San Jose, 2013.

[9] M. Schneider, "Automotive Radar – Status and Trends," in *German Microwave Conference*, South German, 2005, pp 144-147.

[10] D. Kissinger, "Radar Fundamentals," in *Millimeter-Wave Receiver Concepts for 77 GHz Automotive Radar in Silicon-Germanium Technology*, São Paulo, Springer Science+Business Mídia Ltda., 2012, pp. 9-19.

[11] . A. Douglas, "The History of Radar," h2g2, 14 July 2003. [Online]. Available:

http://www.h2g2.com/approved_entry/A591545. [Accessed 20 February 2013].

[12] J. Schneider, "History of Radar," TECH OPS, Arizona, 2003.

[13] R. B. Mahafza, Radar Systems Analysis and Design Using MATLAB, Florida: Chapman & Hall/CRC, 2005.

[14] H. Rohling, "Some Radar Topics: Waveform Design, Range CFAR and Target Recognition," in *Advances in Sensing with Security Applications*, Netherlands, Springer, 2006, p. 293–322.

[15] H. Shibing, W. Xuegang and S. Qiang, "Effects of FM Linearity of Linear FM Signals on Pulse-Compression Performance," in *Radar, 2006. CIE '06. International Conference*, Shanghai, 2006, pp 293-322.

[16] Y. K. Chan and S. Y. Lim, "Synthetic Aperture Radar (SAR) Signal Generation," *Progress In Electromagnetics Research B,* vol. 1, no. 2008, pp. 269-290, 2008.

[17] Devices, Analog, "AD7276/AD7277/AD7278," One Technology Way, Massachusetts, USA, 2005-2009, pp 1-28.

[18] C. Christopher, W. Christopher and F. Timothy, "Computing Performance Benchmarks among CPU, GPU, and FPGA," Mathworks, Worcester, Massachusetts, 2012, pp 1-28.

[19] K. Banovic, M. A. Khalid and A.-R. Esam, "FPGA-Based Rapid Prototyping of Digital SIgnal Processing Systems," in *Circuits and Systems, 2005. 48th Midwest Symposium on*, Covington, KY, 2005, pp 647-650.

[20] S. Lal and S. Chowdhury, "AN FPGA-BASED 77 GHZ MEMS RADAR SIGNAL PROCESSING SYSTEM FOR AUTOMOTIVE COLLISION AVOIDANCE," in *Electrical and Computer Engineering (CCECE), 2011 24th Canadian Conference on*, Niagara Falls, Canada, 2011, pp 1351-1356.

[21] Mercedes-Benz, "Mercedes-Benz Safety," Mercedes-Benz, April 2013. [Online]. Available: http://www.mbusa.com/mercedes/benz/safety. [Accessed 1 March 2013].

[22] Mercedes-Benz, "Mercedes 500SEC.com," The Mercedes-Benz page, 3 Novenber 2013. [Online]. Available: http://500sec.com/distronicdistronic-plus/. [Accessed 1 March 2013].

[23] H. L. Bloecher, M. Andres, C. Fisher, A. Sailer, M. Goppelt and J. Dickmann, "Impact of system parameter selection on radar sensor performance in automotive applications," *Advances in Radio Science,* vol. 10, pp. 33-37, 2012.

[24] G. Oberst, "Automotive Safety Development using radar spectrum," Hogan Lovells, Washington, DC, 2011.

[25] W. Menzel, "Millimeter-Wave Radar for Civil Applications," in *Proceedings of the 7th European Radar Conference*, Paris, France, 2010, pp 89-92.

[26] P. Uhlig, C. Gunner, S. Holzwarth, J. Kassner, R. Kulke, A. Lauer and M. Rittweger, "LTCC Short Range Radar Sensor for Automotive Applications at 24GHz," in *37th IMAPS*, Long Beach, 2004, pp 1-5.

[27] S. Yamano, H. Higashida, M. Shono, S. Matsui, T. Tamaki, H. Yagi and H. Asanuma, "MMIC, 76GHz Millimeter Wave Automobile Radar using Single Chip," FUJITSU TEN TECH, Philippines, 2004, pp 12-19.

[28] LTD., Fujitsu Ten, "Automotive Compact 77GHz 3D Electronic Scan Millimeter Wave Radar," FUJITSU TEN LIMITED, Kobe City, 2012.

[29] W. David and D. William, "FMCW MMW Radar for Automotive Longitudinal Control," California Path Program, California, 1997,pp 1-26.

[30] D. E. Barrick, "FM/CW Radar Signals and Digital Processing," Commerce Publication, Boulder, COLORADO, 1973, pp 1-22.

[31] I. C. Purdle, "VOLTAGE CONTROLLED OSCILLATORS," WWW.ELECTRONICS-TUTORIALS.COM, 8 May 2005. [Online]. Available: http://www.electronics-tutorials.com/oscillators/voltage-controlled-oscillators.htm. [Accessed 5 February 2013].

[32] N. ". Gray, "ABCs of ADCs," National Semiconductor Corporation, Santa Clara, 2006, pp 1-22.

[33] Wikimedia Foundation, Inc., "Wikipedia," 22 April 2013. [Online]. Available: http://en.wikipedia.org/wiki/Window_function. [Accessed 5 March 2013].

[34] M. Integrated, "Tutorial 729: Dynamic Testing of High-Speed ADCs, Part 2," 22 July 2002. [Online]. Available: http://www.maximintegrated.com/app-notes/index.mvp/id/729. [Accessed 5 March 2013].

[35] S. K. Mitra, "Digital Signal Processing Applications," The University of Vermont, BURLINGTON, pp 1143-1243.

[36] W. Kester, "THE DISCRETE FOURIER TRANSFORM," Analog devices.

[37] L. R.G., "WINDOWS," in *Understanding Digital Signal Processing*, Upper Saddle River, New Jersey, Prentice Hall PTR, 2004, pp. 1-688.

[38] E. J. Roger L., "Window Functions," Chester F. Carlson Center for Imaging Science, 2010 2011. [Online]. Available: http://www.cis.rit.edu/resources/software/sig_manual/windows.html. [Accessed 6 March 2013].

[39] Wang Yuke, T. Yiyan (Felix), J. Yingtao, J.-G. Chung, S. Sang-Seob and L. Myoung-Seob, "Novel Memory Reference Reduction Methods for FFT Implementations on DSP Processors," *Digital Object Identifier,* vol. 55, no. 5, pp. 2338-2347, 2007.

[40] Xilinx, "LogiCORE IP Fast Fourier Transform v7.1," Xilinx, 2011, pp 1-59.

[41] M. Mashade, "Analysis of CFAR Detection of Fluctuating Targets," *Progress In Electromagnetics Research C,* vol. 2, pp. 65-94, 2008.

[42] S. R. Thamid, J. K. Ali and Z. T. Yassen, "An FPGA based Implemntation of CA-CFAR Processor," *Asian Journal of Information Technology,* vol. 4, no. 6, pp. 511-514, 2007.

[43] V. Kyovtorov, H. Kabakchiev and G. Kuzmanov, "Parallel FPGA Design of CA CFAR Algorithm," in *Radar Symposium (IRS), 2010 11th International*, Vilnius, Lithuania, 2010, pp 1-4.

[44] J. Marcum, "A Statistical Theory of Target Detection by Pulsed Radar," The Rand Corporation, Santa Monica, 1952, pp 1-81.

[45] P. Swerling, "Probability of Detection For Fluctuating Targets," The Rand Corporation, Santa Monica, 1954, pp 1-45.

[46] A. Drosopoulos and G. Haslam, "PEAK DETECTION OF SWERLING TYPE TARGETS Part 1: DETECTION PROBABILITIES IN WHITE NOISE," DEFENCE RESEARCH ESTABLISHMENT OTTAWA, Ottawa, 1993, pp 1-37.

[47] Wikipedia, "Chi-squared target models," Wikipedia Foundation, Inc, 29 September 2011. [Online]. Available: http://en.wikipedia.org/wiki/Chi-squared_target_models. [Accessed 15 March 2013].

[48] G. Brooker, "Chapter 10: Detection of Signals in Noise," University of Sydney, Sydney, 2002-2013.

[49] C. Wolff, "Noise," Radartutorial .eu, [Online]. Available:

http://www.radartutorial.eu/18.explanations/ex08.en.html. [Accessed 20 March 2013].

[50] F. Merat, "Electrical Noise," Case Western Reserve University, Cleveland, 1988, pp 37-39.

[51] P. Bak, C. Tang and K. Wiesenfeld, "Self-Organized Criticality: An Explanation of 1/f Noise," *Physics Review Letters,* vol. 59, no. 4, pp. 381-384, 1987.

[52] R. Y. Richard Lyons, "Reducing ADC Quantization Noise," Microwaves and RF, 17 June 2005. [Online]. Available: http://mwrf.com/components/reducing-adc-quantization-noise. [Accessed 21 March 2013].

[53] P. Adhikari, "Understanding Millimeter Wave Wireless Communication," Loea Corporation, San Diego, CA, 2008, pp 1-6.

[54] J. Silvious and D. Tahmoush, "Automotive GMTI radar for object and human avoidance," in *Radar Conference (RADAR), 2011 IEEE*, Kansas City, MO, 2011, pp 375-377.

[55] C. Wolf, "Concealment or Masking," Radartutorial.eu, [Online]. Available: http://www.radartutorial.eu/16.eccm/ja09.en.html. [Accessed 22 March 2013].

[56] C. Wolff, "Range Resolution," radartutorial.eu, [Online]. Available: http://www.radartutorial.eu/01.basics/rb18.en.html. [Accessed 23 March 2013].

[57] M. Pichler, . A. Stelzer, P. Gulden and M. Vossiek, "Influence of Systematic Frequency-Sweep Non-Linearity on Object Distance Estimation in FMCW/FSCW Radar Systems," in *33rd European Microwave Conference*, Munich, 2003, pp 1203-1206.

[58] Xilinx, "Virtex-5 Family Overview," Xilinx, 2009, pp 1-13.

[59] Xilinx, "Virtex-5 Platform FPGA Family Technical Backgrounder," Xilinx, 2006, pp 1-10.

[60] Xilinx, "LogiCORE IP Block Memory Generator V6.2," Xilinx, 2011, pp 1-119.

[61] J. Wenger, "RF-Applications in Vehicles – Today and Tomorrow – DaimlerChrysler," DaimlerChrysler AG, Ulm, Germany, 2006, pp 1-36.

APPENDIX-1

(a) MATLAB codes

radar_gen_simdata.m

```
function [ SIM_TIME, SIM_IF, SIM_ACTIVE_BEAM, SIM_ACTIVE_CHIRP ] = ...
    radar_gen_simdata( SYSTEM_TCHIRP, ADC_SAMPLING_FREQ, SIM_FRAMES, ...
    RADAR_BEAMS, RADAR_BEAM_WIDTH, HOST_VELOCITY, TARGETS_INITIAL_POS_X,...
    TARGETS_INITIAL_POS_Y, TARGETS_VELOCITY, TARGETS_NUM, CONSTANT_C,...
    CONSTANT_SIGNAL_ATTEN, SYSTEM_SWEEP_RATE, SYSTEM_MAX_RANGE, ...
    HOST_INITIAL_POS_Y, SYSTEM_FC)
    t_index = 0;
    active_beam = 0;    %0 to RADAR_BEAMS-1, 0 is the rightmost beam
    active_chirp = 0; %0 for the up-chirp, 1 for the down-chirp
     t_period = 1/ADC_SAMPLING_FREQ;
    t_max = SIM_FRAMES*2*SYSTEM_TCHIRP*RADAR_BEAMS - t_period;
    num_samples = (t_max + t_period) / t_period;
     SIM_IF = zeros(1,num_samples);
    SIM_ACTIVE_BEAM = zeros(1,num_samples);
    SIM_ACTIVE_CHIRP = zeros(1,num_samples);
    SIM_TIME = 0:t_period:t_max;
        disp('Generating simulation data...');
    disp('Simulation time is...');
    for t = 0:t_period:t_max
        t_index = t_index + 1; %increment the time index
        if(mod(t,2*SYSTEM_TCHIRP) == 0) %if two chirp periods have elapsed, switch to the next beam
          if( t_index ~=1)  %don't switch the active beam on the very first sample              active_beam = mod(active_beam + 1,
RADAR_BEAMS);
          end
          angle_lo = RADAR_BEAM_WIDTH*active_beam - RADAR_BEAM_WIDTH*RADAR_BEAMS/2;
          angle_hi = RADAR_BEAM_WIDTH*(active_beam+1) - RADAR_BEAM_WIDTH*RADAR_BEAMS/2;
       end
        if(mod(t,SYSTEM_TCHIRP) == 0 && t ~= 0) %keep track of the up/down chirp
           active_chirp = mod(active_chirp+1,2);
       end
        if(mod(t,1e-1) == 0)
           disp(t); %print the simulation time in tenths of second to get an idea of how much longer we need to wait
       end
             SIM_ACTIVE_BEAM(t_index) = active_beam;
       SIM_ACTIVE_CHIRP(t_index) = active_chirp;
        %update the x position of the host and targets
       host_x_pos = HOST_VELOCITY*t;
       targets_x_pos = TARGETS_INITIAL_POS_X + TARGETS_VELOCITY*t;
        %find relative distance between targets and host
       relative_x_pos = targets_x_pos - host_x_pos;
       relative_y_pos = TARGETS_INITIAL_POS_Y - HOST_INITIAL_POS_Y; %assume nobody is making any lane changes;
relative y position always the same
       relative_x_vel = TARGETS_VELOCITY - HOST_VELOCITY;  %this model assumes all targets travel in parallel; no y
velocity
       relative_distance = sqrt(relative_x_pos.^2 + relative_y_pos.^2);
             %figure out what targets are in the active beam's coverage
       f = zeros(TARGETS_NUM,1);
       atten = zeros(TARGETS_NUM,1);
       for i=1:length(relative_distance)
           if(relative_distance(i) <= SYSTEM_MAX_RANGE) %first check to see what targets are 'in range'
              %if the target is in range, compute the vehicle/target angle of             %incidence with respect to our x-axis
              angle = atan(relative_y_pos(i) / relative_x_pos(i) );
              if( angle_lo <= angle && angle < angle_hi  )  %check to see if the angle is in the correct range for the active beam
                 %target is in range and in our active beam
                  %calculate the relative velocity in the radial direction
                  v_radial = relative_x_vel(i) * cos(angle);
at frequency this target will contribute to the IF
                  if(active_chirp == 0) %upsweep
                      f(i) = 2*SYSTEM_SWEEP_RATE*relative_distance(i)/CONSTANT_C +
2*SYSTEM_FC*v_radial/CONSTANT_C;
                  elseif(active_chirp == 1) %downsweep
                      f(i) = 2*SYSTEM_SWEEP_RATE*relative_distance(i)/CONSTANT_C -
2*SYSTEM_FC*v_radial/CONSTANT_C;
                  end
                   %compute the attenuation from this target
                  atten(i) = 10^(-2*CONSTANT_SIGNAL_ATTEN*relative_distance(i)/1000);
              end
          end
      end
       %finally, add together the targets individual contributions to create
      %the instantaneous IF for this ADC sample
      for i=1:TARGETS_NUM
          if(f(i) ~= 0) %only add non-zero frequencies
              SIM_IF(t_index) = SIM_IF(t_index) + atten(i)*sin(2*pi*f(i)*t);
          end
      end
   end
   disp('Simulation data generation complete.');
end
```

radar_compute_adc.m

```
function [ADC_SAMPLES_HDL, ADC_SAMPLES_MATLAB] = radar_compute_adc( SIM_IF, ADC_DYNAMIC_RANGE,
ADC_BITS )
   range = max([abs(max(SIM_IF)) abs(min(SIM_IF))]);
   x = SIM_IF ./ (2*range);
   x = x .* ADC_DYNAMIC_RANGE;
   x = x + 0.5;
     %quantize raw ADC input data
   ADC_SAMPLES_HDL = fi(x, 0, ADC_BITS, ADC_BITS, ...
                         'RoundMode', 'Floor', ...
                         'OverflowMode', 'Saturate', ...
                         'ProductMode', 'KeepMSB', ...
                         'ProductWordLength', ADC_BITS, ...
                         'SumMode', 'KeepMSB', ...
                         'SumWordLength', ADC_BITS, ...
                         'CastBeforeSum', true);

   %now convert to two's complement in range of -1 and 1 so matlab
   %interprets these numbers the way we need
   x = x - 0.5;
   x= x .* 2;
    %get quantized two's complement version of adc input
   ADC_SAMPLES_MATLAB = fi(x, 1, ADC_BITS, ADC_BITS-1, ...
                    'RoundMode', 'Floor', ...
                    'OverflowMode', 'Saturate', ...
                    'ProductMode', 'KeepMSB', ...
                    'ProductWordLength', ADC_BITS, ...
                    'SumMode', 'KeepMSB', ...
                    'SumWordLength', ADC_BITS, ...
                    'CastBeforeSum', true);

   disp('ADC quantization complete.');
end
```

radar_compute_window.m

```
function [WINDOW_SAMPLES, WINDOW_COEFFS] = radar_compute_window( ADC_SAMPLES, WINDOW_TYPE,
WINDOW_OUT_BITS, WINDOW_COEFF_BITS, FFT_POINTS, SIM_FRAMES, RADAR_BEAMS)
    disp('Computing windowed ADC samples...');
   batch_size = length(ADC_SAMPLES)/(RADAR_BEAMS * SIM_FRAMES * 2);
   %setup window
   window_handle = str2func(['@' WINDOW_TYPE]);
   w = window(window_handle, batch_size);
       for i = 1:1:(RADAR_BEAMS * SIM_FRAMES * 2)
      limit1 = batch_size*(i-1)+1;
      limit2 = batch_size*(i);
       %quantized versions of the window coefficients
      WINDOW_COEFFS = fi(w, 0, WINDOW_COEFF_BITS, WINDOW_COEFF_BITS, ...
            'RoundMode', 'Floor', ...
            'OverflowMode', 'Saturate', ...
            'ProductMode', 'KeepMSB', ...
            'ProductWordLength', WINDOW_COEFF_BITS, ...
            'SumMode', 'KeepMSB', ...
            'SumWordLength', WINDOW_COEFF_BITS, ...
            'CastBeforeSum', true);
       %windowed inputs
      WINDOW_SAMPLES(limit1:limit2) = fi(ADC_SAMPLES(limit1:limit2).* WINDOW_COEFFS', 1,
WINDOW_OUT_BITS, WINDOW_OUT_BITS-1, ...
                     'RoundMode', 'Floor', ...
                     'OverflowMode', 'Saturate', ...
                     'ProductMode', 'KeepMSB', ...
                     'ProductWordLength', WINDOW_OUT_BITS, ...
                     'SumMode', 'KeepMSB', ...
                     'SumWordLength', WINDOW_OUT_BITS, ...
                     'CastBeforeSum', true);
    end
end
```

radar_compute_fft.m

```
function [XK_RE, XK_IM, XK_RE_SQ, XK_IM_SQ, FFT_OUTPUT_MAG_SQUARED] = radar_compute_fft(
WINDOW_SAMPLES, FFT_POINTS, XILINX_FFT_GENERICS, XILINX_FFT_SCALING_SCHEDULE, FFT_OUT_BITS,
SYSTEM_TCHIRP, ADC_SAMPLING_FREQ )
      f = ADC_SAMPLING_FREQ/2*linspace(0,1,FFT_POINTS/2+1);
    for i = 1:1:(length(WINDOW_SAMPLES)/(SYSTEM_TCHIRP * ADC_SAMPLING_FREQ))
         limit1 = SYSTEM_TCHIRP * ADC_SAMPLING_FREQ*(i-1)+1;
         limit2 = SYSTEM_TCHIRP * ADC_SAMPLING_FREQ*(i);

         zero_pad_amount = FFT_POINTS - SYSTEM_TCHIRP * ADC_SAMPLING_FREQ;
         samples = [WINDOW_SAMPLES(limit1:limit2) zeros(1,zero_pad_amount)];
          %compute the xilinx fft
         display('Ignore any warnings below regarding the input array being "real-only" numbers.');
         [FFT_OUTPUT(limit1+zero_pad_amount*(i-1):limit2+zero_pad_amount*i), BLKEXP, OVERFLOW_DETECT] = ...
            xfft_v7_1_bitacc_mex(XILINX_FFT_GENERICS, log2(FFT_POINTS), ...
            double(samples), XILINX_FFT_SCALING_SCHEDULE, 1);
          if(OVERFLOW_DETECT)
            disp('Error: FFT overflow detected.');
         end
   end
   %quantize the fft's output
   XK_RE = real(FFT_OUTPUT);
   XK_IM = imag(FFT_OUTPUT);
     XK_RE = fi(XK_RE, 1, FFT_OUT_BITS, (FFT_OUT_BITS/2-1), ...
      'RoundMode', 'Floor', ...
      'OverflowMode', 'Saturate', ...
```

```
        'ProductMode', 'KeepMSB', ...
        'ProductWordLength', (FFT_OUT_BITS*2), ...
        'SumMode', 'KeepLSB', ...
        'SumWordLength', (FFT_OUT_BITS*2), ...
        'CastBeforeSum', true);
     XK_IM = fi(XK_IM, 1, FFT_OUT_BITS, (FFT_OUT_BITS/2-1), ...
        'RoundMode', 'Floor', ...
        'OverflowMode', 'Saturate', ...
        'ProductMode', 'KeepMSB', ...
        'ProductWordLength', (FFT_OUT_BITS*2), ...
        'SumMode', 'KeepLSB', ...
        'SumWordLength', (FFT_OUT_BITS*2), ...
        'CastBeforeSum', true);
    %now compute the squares of XK_RE and XK_IM; only use the first half
    %of each frame (because our fft inputs are real valued, the second half
    %is just a mirror of the first half)
    for i = 1:1:(length(FFT_OUTPUT)/FFT_POINTS)
        limit1 = FFT_POINTS*(i-1)+1;
        limit2 = limit1+FFT_POINTS/2 - 1;
                limit3 = FFT_POINTS/2*(i-1)+1;
        limit4 = FFT_POINTS/2*(i);
                XK_RE_SQ(limit3:limit4) = XK_RE(limit1:limit2) .* XK_RE(limit1:limit2);
        XK_IM_SQ(limit3:limit4) = XK_IM(limit1:limit2) .* XK_IM(limit1:limit2);
    end
         FFT_OUTPUT_MAG_SQUARED = XK_RE_SQ + XK_IM_SQ;
end
```

radar_ca_cfar.m

```
function [cfar] = radar_ca_cfar()
  radar_params;
radar_situation_params;
%UP_SWEEP
 K = CACFAR_PFA^(-1/(2*CACFAR_M)) - 1;  % Cell averaging factor
tmpcfar = [0 0 0 0]';  % Initiate the cfar matrix
countup = 1;
countupfinal = 0;
 for CUT=2:TFFT_POINTS/2
        avgL = 0;  % Average on left side of Cell-Under-Test
    avgR = 0;  % Average on right side of Cell-Under-Test
        % The average of the cells on the right and left of CUT
    if(CUT<=CACFAR_M+CACFAR_GB)
        for i=1:CACFAR_M
            avgR = avgR + abs(Yup(CUT+i+CACFAR_GB));
        end
        avgR = avgR/CACFAR_M;
    elseif(CUT>=TFFT_POINTS/2-CACFAR_M-CACFAR_GB)
        for i=1:CACFAR_M
            avgL = avgL + abs(Yup(CUT-i-CACFAR_GB));
        end
        avgL = avgL/CACFAR_M;
    else
        for i=1:CACFAR_M
            avgL = avgL + abs(Yup(CUT-i-CACFAR_GB));
            avgR = avgR + abs(Yup(CUT+i+CACFAR_GB));
        end
        avgR = avgR/CACFAR_M;
        avgL = avgL/CACFAR_M;
    end
     % Computing threshold
    T = (avgR+avgL)/2 * K;
        % Checking for valid targets
    if(abs(Yup(CUT))>T)
```

```
        countup = countup + 1;
        tmpcfar(1,countup) = abs(Yup(CUT));
        tmpcfar(2,countup) = CUT;
    end
end
tmpcfar(1,countup+1) = 0;
tmpcfar(2,countup+1) = 0;
j = 1;
for i=2:length(tmpcfar(1,:))-1
    if((tmpcfar(2,i)~=tmpcfar(2,i+1)-1)&&(tmpcfar(2,i)~=tmpcfar(2,i+1)))
        if((tmpcfar(2,i)==tmpcfar(2,i-1)+1)||(tmpcfar(2,i)==tmpcfar(2,i-1)))
            tmp1cfar(1,j) = max(tmpcfar(1,i-1),tmpcfar(1,i));
            tmp1cfar(2,j) = tmpcfar(2,i);
            j = j + 1;
        else
            tmp1cfar(1,j) = tmpcfar(1,i);
            tmp1cfar(2,j) = tmpcfar(2,i);
            j = j + 1;
        end
    end
end
 % Eliminating any residual false alarms
ST = 0.6 * mean(tmp1cfar(1,:));
j = 1;
for i=1:length(tmp1cfar(1,:))
    if(tmp1cfar(1,i)>ST)
        cfar(1,j) = tmp1cfar(1,i);
        cfar(2,j) = tmp1cfar(2,i); % * ADC_SAMPLING_FREQ/TFFT_POINTS;
        j = j + 1;
        countupfinal = countupfinal + 1;
    end
end
 % Ploting the targets detected
figure(5)
%subplot(2,1,1)
stem(cfar(2,:),cfar(1,:));
title('CFAR-detected targets for UP SWEEP for SRR')
xlabel('Frequency (Hz)')
ylabel('Peak Intensity')
 % Down_sweep
countdown = 1;
countdownfinal = 0;
 for CUT=2:TFFT_POINTS/2
        avgL = 0;  % Average on left side of Cell-Under-Test
    avgR = 0;  % Average on right side of Cell-Under-Test
        % The average of the cells on the right and left of CUT
    if(CUT<=CACFAR_M+CACFAR_GB)
        for i=1:CACFAR_M
            avgR = avgR + abs(Ydown(CUT+i+CACFAR_GB));
        end
        avgR = avgR/CACFAR_M;
    elseif(CUT>=TFFT_POINTS/2-CACFAR_M-CACFAR_GB)
        for i=1:CACFAR_M
            avgL = avgL + abs(Ydown(CUT-i-CACFAR_GB));
        end
        avgL = avgL/CACFAR_M;
    else
        for i=1:CACFAR_M
            avgL = avgL + abs(Ydown(CUT-i-CACFAR_GB));
            avgR = avgR + abs(Ydown(CUT+i+CACFAR_GB));
        end
        avgR = avgR/CACFAR_M;
        avgL = avgL/CACFAR_M;
    end
     % threshold for down_sweep
    T = (avgR+avgL)/2 * K;
     if(abs(Ydown(CUT))>T)
        countdown = countdown + 1;
        tmpcfar(3,countdown) = abs(Ydown(CUT));
        tmpcfar(4,countdown) = CUT;
    end
end
tmpcfar(3,countdown+1) = 0;
tmpcfar(4,countdown+1) = 0;
 j = 1;
for i=2:length(tmpcfar(1,:))-1
    if((tmpcfar(4,i)~=tmpcfar(4,i+1)-1)&&(tmpcfar(4,i)~=tmpcfar(4,i+1)))
        if((tmpcfar(4,i)==tmpcfar(4,i-1)+1)||(tmpcfar(4,i)==tmpcfar(4,i-1)))
            tmp1cfar(3,j) = max(tmpcfar(3,i-1),tmpcfar(3,i));
            tmp1cfar(4,j) = tmpcfar(4,i);
            j = j + 1;
        else
            tmp1cfar(3,j) = tmpcfar(3,i);
            tmp1cfar(4,j) = tmpcfar(4,i);
            j = j + 1;
        end
    end
end
 ST = 0.6 * mean(tmp1cfar(3,:));
 j = 1;
for i=1:length(tmp1cfar(3,:))
    if(tmp1cfar(3,i)>ST)
        cfar(3,j) = tmp1cfar(3,i);
        cfar(4,j) = tmp1cfar(4,i);% * ADC_SAMPLING_FREQ/TFFT_POINTS;
        j = j + 1;
        countdownfinal = countdownfinal + 1;
    end
end
 % Ploting detected targets
figure(6);
stem(cfar(4,:),cfar(3,:));
title('CFAR-detected targets for DOWN SWEEP for SRR')
xlabel('Frequency (Hz)')
ylabel('Peak Intensity')
```

radar_compute_range_and_velocity.m

```
function [ DETECTED_RANGE, DETECTED_VELOCITY ] = radar_compute_range_and_velocity(
MAX_BIN,RADAR_BEAMS, SIM_FRAMES, CALC_RANGE_FACTOR_DEC, CALC_VELOCITY_FACTOR_DEC,
CALC_FACTOR_BITS )
%UNTITLED15 Summary of this function goes here
%   Detailed explanation goes here
     up_bins = MAX_BIN(1:2:end);
    up_bins_sorted = reshape(up_bins, RADAR_BEAMS, SIM_FRAMES)';
    down_bins = MAX_BIN(2:2:end);
    down_bins_sorted = reshape(down_bins, RADAR_BEAMS,SIM_FRAMES)';
        bins_sum = up_bins_sorted + down_bins_sorted;
    bins_diff = up_bins_sorted - down_bins_sorted;
DETECTED_RANGE = bins_sum .* CALC_RANGE_FACTOR_DEC / 2^CALC_FACTOR_BITS;
DETECTED_VELOCITY = bins_diff .* CALC_VELOCITY_FACTOR_DEC / 2^CALC_FACTOR_BITS;
     end
```

radar_main.m

```
clear all;
close all;
clc

radar_params;
radar_situation_params;

SIM_FRAMES = 2;   %number of 2*SYSTEM_TCHIRP*RADAR_BEAMS*ADC_SAMPLING_FREQ samples to generate
data for
DUMP_TEXT = 0;     %switch to turn text-file generation on or off
 %create simulation data based on our parameters
 [ SIM_TIME, SIM_IF, SIM_ACTIVE_BEAM, SIM_ACTIVE_CHIRP ] = ...
    radar_gen_simdata( SYSTEM_TCHIRP, ADC_SAMPLING_FREQ, SIM_FRAMES, ...
    RADAR_BEAMS, RADAR_BEAM_WIDTH, HOST_VELOCITY, TARGETS_INITIAL_POS_X,...
    TARGETS_INITIAL_POS_Y, TARGETS_VELOCITY, TARGETS_NUM, CONSTANT_C,...
    CONSTANT_SIGNAL_ATTEN, SYSTEM_SWEEP_RATE, SYSTEM_MAX_RANGE, ...
    HOST_INITIAL_POS_Y, SYSTEM_FC);
 %quantize the data
[ADC_SAMPLES, ADC_SAMPLES_MATLAB] = radar_compute_adc( SIM_IF,...
   ADC_DYNAMIC_RANGE, ADC_BITS );
 %window the quantized data
[WINDOW_SAMPLES, WINDOW_COEFFS] = radar_compute_window( ADC_SAMPLES_MATLAB, WINDOW_TYPE,...
   WINDOW_OUT_BITS, WINDOW_COEFF_BITS, FFT_POINTS, SIM_FRAMES, RADAR_BEAMS);
 %compute the fft
[XK_RE, XK_IM, XK_RE_SQ, XK_IM_SQ, FFT_OUTPUT_MAG_SQUARED] = radar_compute_fft( WINDOW_SAMPLES,
FFT_POINTS, XILINX_FFT_GENERICS, XILINX_FFT_SCALING_SCHEDULE, FFT_OUT_BITS, SYSTEM_TCHIRP,
ADC_SAMPLING_FREQ );
 %find the highest intensity peak
[MAX_MAG, MAX_BIN] = radar_compute_max(FFT_OUTPUT_MAG_SQUARED, FFT_POINTS);
 %ca-cfar calculation
[CMAX_BIN] = radar_ca_cfar();
 %compute the detected range and velocity
[ DETECTED_RANGE, DETECTED_VELOCITY ] = radar_compute_range_and_velocity( MAX_BIN, RADAR_BEAMS,
SIM_FRAMES, CALC_RANGE_FACTOR_DEC, CALC_VELOCITY_FACTOR_DEC, CALC_FACTOR_BITS );
 if( DUMP_TEXT == 1)
   %dump quantized data to text files
   radar_print_adc(ADC_SAMPLES);
    %dump the window data
   radar_print_window(WINDOW_SAMPLES, WINDOW_COEFFS);
      %dump the fft results
   radar_print_fft(XK_RE, XK_IM, XK_RE_SQ, XK_IM_SQ, FFT_OUTPUT_MAG_SQUARED);
      %dump the max bin / max magnitude results
   radar_print_max( MAX_MAG, MAX_BIN, FFT_OUTPUT_MAG_SQUARED, FFT_POINTS);
end
```

radar_situation_params.m

```
%this file contains situational data for use in the simulation data
%generator

%Lane parameters
LANE_WIDTH = 3.657;                %simulated lane width in m
LANE_NUMBER = 16;                  %maximum number of lanes used for simulation
 %Radar parameters
RADAR_BEAMS = 3;                   %number of radar beams
RADAR_BEAM_WIDTH = 80 * pi /180;   %radar beam width in radians
 %Host vehicle parameters
HOST_VELOCITY = 70/3.6;        %host vehicle velocity in m/s
HOST_LANE = 2;                 %host vehicle lane 0 to SIM_NUM_LANES-1 (0 for bottom)
 %Target vehicle parameters
```

```
TARGETS_NUM = 1;
TARGETS_LANE = [2];                    %target vehicle lane 0 to SIM_NUM_LANES-1 (0 for bottom)
TARGETS_VELOCITY = [60] ./ 3.6;            %target velocity in m/s
TARGETS_INITIAL_POS_X = [25];              %target initial x position in m (relative to host)
 %Derived parameters
HOST_INITIAL_POS_Y = HOST_LANE*LANE_WIDTH + ...        %host initial y position
   LANE_WIDTH/2;
TARGETS_INITIAL_POS_Y = TARGETS_LANE*LANE_WIDTH + ...   %target initial y position
   LANE_WIDTH/2;

%ERRORS
if( ~all(TARGETS_LANE < LANE_NUMBER))
   disp('Error: Target in lane that does not exist');
end

if(length(TARGETS_LANE) ~= TARGETS_NUM || ...
      length(TARGETS_VELOCITY) ~= TARGETS_NUM || ...
      length(TARGETS_INITIAL_POS_X) ~= TARGETS_NUM )
   disp('Error: Dimensions of SIM_TARGETS_LANE, SIM_TARGETS_VELOCITY, and SIM_TARGETS_INITIAL_POS_X
do not match');
end
```

radar_params.m

```
%This file contains radar system parameters, physical constants and
%derived quantities

%Physical constants
CONSTANT_C = 299704764;        %speed of light in m/s
CONSTANT_SIGNAL_ATTEN = 0.8;    %radar signal attenuation in dB / km

%Radar system parameters
SYSTEM_BW = 2000e6;             %sweep bandwidth in Hz
SYSTEM_TCHIRP = 1.024e-3;       %chirp time (of an up or down sweep, not both) in s
SYSTEM_FC = 77e9;              %centre frequency of radar system in GHz
SYSTEM_MAX_RANGE = 30;         %maximum detection range in m
SYSTEM_MAX_VEL = 300/3.6;       %maximum detectable relative velocity in m/s (around 83m/s)

%Radar system: derived parameters
SYSTEM_SWEEP_RATE = SYSTEM_BW / SYSTEM_TCHIRP;                      %the sweep rate
SYSTEM_MAX_IF = 2*SYSTEM_MAX_RANGE * SYSTEM_SWEEP_RATE/CONSTANT_C + ... %maximum
intermediate frequency
    SYSTEM_FC * SYSTEM_MAX_VEL / CONSTANT_C;
SYSTEM_RANGE_RES = CONSTANT_C / (2 * SYSTEM_BW);                    %best possible range resolution
SYSTEM_VEL_RES = CONSTANT_C / (2 * SYSTEM_TCHIRP * SYSTEM_FC);         %best possible velocity
resolution
SYSTEM_MIN_SAMPLING_FREQ = 2 * SYSTEM_MAX_IF;                      %minimum required sampling
frequency

 %ADC parameters
ADC_DYNAMIC_RANGE = 0.9;       %the fraction of ADC codes in use
ADC_BITS = 12;                %the number of bits that the ADC uses
ADC_SAMPLING_FREQ = 2e6;       %ADC samples per second
ADC_NOISE_AMPLITUDE = 1;       %Amplitude of random noise added to the ADC

%Window parameters
WINDOW_COEFF_BITS = 12;        %number of bits used for the window coefficients
```

```
WINDOW_OUT_BITS = 12;          %number of bits used for the window output
WINDOW_TYPE = 'hamming';       %the window type, see 'help window' for a full list

%FFT parameters
FFT_POINTS = 2048;             %number of FFT points
FFT_OUT_BITS = 24;             %number of output bits used in the FFT

%Xilinx FFT parameters
XILINX_FFT_GENERICS.C_NFFT_MAX = log2(FFT_POINTS);     %number of fft bits
XILINX_FFT_GENERICS.C_ARCH = 4;                        %Xilinx fft architecture to use (see help file)
XILINX_FFT_GENERICS.C_HAS_NFFT = 0;                    %runtime reconfigurable transform length
XILINX_FFT_GENERICS.C_USE_FLT_PT = 0;                  %use floating point number representation
XILINX_FFT_GENERICS.C_INPUT_WIDTH = WINDOW_OUT_BITS;   %number of bits used at the input to
the fft
XILINX_FFT_GENERICS.C_TWIDDLE_WIDTH = 16;              %bits used for twiddle factors
XILINX_FFT_GENERICS.C_HAS_SCALING = 0;                 %type of scaling (0 for no scaling)
XILINX_FFT_GENERICS.C_HAS_BFP = 0;                     %block floating point
XILINX_FFT_GENERICS.C_HAS_ROUNDING = 0;                %enable rounding
XILINX_FFT_SCALING_SCHEDULE = [0 0 0 0 0 0 0 0 0 0 0];

%CA-CFAR parameters
CACFAR_GB = 2;                 %total number of guard bands used in CA-CFAR
CACFAR_M = 8;                  %total number of cells used for averaging
CACFAR_PFA = 10e-6;            %Probability of false alarm threshold paramter

%Range and velocity computation parameters
CALC_FACTOR_BITS = 16;
CALC_RANGE_FACTOR = ADC_SAMPLING_FREQ / FFT_POINTS * CONSTANT_C / 4 /
SYSTEM_SWEEP_RATE;
CALC_RANGE_FACTOR_BIN = dec2bin(CALC_RANGE_FACTOR *
2^CALC_FACTOR_BITS,2*CALC_FACTOR_BITS);
CALC_RANGE_FACTOR_DEC = bin2dec(CALC_RANGE_FACTOR_BIN);
CALC_VELOCITY_FACTOR = ADC_SAMPLING_FREQ / FFT_POINTS * CONSTANT_C / 4 / SYSTEM_FC
* 3.6; %3.6 converts m/s to km/h
CALC_VELOCITY_FACTOR_BIN = dec2bin(CALC_VELOCITY_FACTOR *
2^CALC_FACTOR_BITS,2*CALC_FACTOR_BITS);
CALC_VELOCITY_FACTOR_DEC = bin2dec(CALC_VELOCITY_FACTOR_BIN);

%WARNINGS
if(mod(SYSTEM_TCHIRP*ADC_SAMPLING_FREQ,1) ~= 0)
   disp('Warning, chirp time is not an integer multiple of the sampling frequency');
end

%ERRORS
if(ADC_SAMPLING_FREQ < SYSTEM_MIN_SAMPLING_FREQ)
   disp('Error, ADC sampling frequency too low.');
end

if(FFT_POINTS ~= SYSTEM_TCHIRP*ADC_SAMPLING_FREQ)
   disp('Error, number of samples and FFT size differ.');
end

if(mod(log2(FFT_POINTS),1) ~= 0)
   disp('Error, FFT point size not a power of two.');
```

```
end
```

(b) HDL codes

adc_control.v

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Company: UWindsor
// Engineer: Sabrina
//
// Create Date:    17:24:40 08/18/2011
// Design Name:
// Module Name:    adc_control
// Project Name: Tri-mode Radar Design
// //
////////////////////////////////////////////////////////////////////////////////
module adc_control(
   input clk_48,
   input ADC_MISO,
            input reset,
            input ready,
    output reg ADC_CS,
    output reg [11:0] adc_sample,
    output reg adc_valid
    );

//CS counter -- goes low every 500 ns (2MHz) for 2 MSPS
//to adjust sampling rate, change number of delay cycles from 23 to something else
reg [4:0] cs_counter;
always @(posedge clk_48)
begin
        if(reset)
        begin
                ADC_CS <= 1'b1;
                cs_counter <= 5'd0;
                adc_valid <= 1'b0;
        end
        else if (ready)
        begin
                cs_counter <= cs_counter + 1;
                if(cs_counter < 5'd15)
                begin
                        ADC_CS <= 1'b0;
                        adc_valid <= 1'b0;
                end
                else if(cs_counter == 5'd15)
                begin
                        ADC_CS <= 1'b1;
                        adc_valid <= 1'b0;
                end
                else if(cs_counter == 5'd16)
                begin
                        ADC_CS <= 1'b1;
                        adc_valid <= 1'b1;
                end
                else if(cs_counter < 5'd23) //change this number to adjust sampling rate; 23 for 2 MSPS, probably 47 for 1
MSPS, etc.
                begin
                        ADC_CS <= 1'b1;
                        adc_valid <= 1'b0;
                end
                else if(cs_counter == 5'd23)
                begin
                        ADC_CS <= 1'b1;
                        adc_valid <= 1'b0;
                  cs_counter <= 5'd0;
                end
        end
        else
        begin
                ADC_CS <= 1'b1;
                cs_counter <= 5'd0;
                adc_valid <= 1'b0;
        end
end

//serial-in parallel-out shift register
//positive edge triggered
//first bit is going to be a zero (that's how the AD7276 ADC works; see datasheet for more details)
reg [13:0] sipo;
always @(posedge clk_48)
```

```
begin
		if(!ADC_CS)
			sipo <= {sipo[12:0], ADC_MISO};
		else
			sipo <= sipo;
end


//most significant bit of sipo is a zero
//two least significant bits of sipo are zero
//the rest of the bits make up the 12-bit sample
always @(posedge clk_48)
begin
		if(ADC_CS)
			adc_sample <= sipo[12:1]; //was 13:2; changed to 12:1 to fix a timing problem
																		//note that this problem was
probably caused by a signal integrity issue
																		//and that this will have to
be changed back when we're using proper wiring
end
endmodule
```

compute_window.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: UWindsor
// Engineer: Sabrina
//
// Create Date:    13:06:59 09/02/2011
// Design Name:
// Module Name:    compute_window
// Project Name:
//////////////////////////////////////////////////////////////////////////////////
module compute_window(
    input adc_valid,
	     input clk_48,
    input [11:0] adc_sample,
    input reset,
    input ready,
    output [11:0] window_sample
    );

		//convert ADC input from 12-bit unsigned (range 0-->1) to
		//12-bit two's complement (range -1 --> +1) by inverting
		//the MSBwire [11:0] adc_sample_twos;
		wire [11:0] adc_sample_twos;
		assign adc_sample_twos = {~adc_sample[11],adc_sample[10:0]};

		//using the very slow (2MHz) adc_valid signal as a clock
		//although this was generated using a counter, skew won't be a problem because
		//the clock period is an incredibly generous 500 ns
		wire [9:0] lut_address;
		count10updown u0_window_lut_counter (
		.clk(clk_48),
		.reset(reset),
		.ready(ready),
		.en(adc_valid),
		.dout(lut_address)
		);

		//lookup table ROM holding our window coefficients
```

```
            wire [11:0] lut_out;
            window_lut u0_window_lut (
            .clka(clk_48), // input clka
            .addra(lut_address), // input [9 : 0] addra
            .douta(lut_out) // output [11 : 0] douta
            );

            //fully combinational multiplier
            //should have no problems keeping up with our slow 2MHz clock
            window_mult u0_window_mult (
            .a(lut_out), // input [11 : 0] a (unsigned window coefficient)
            .b(adc_sample_twos), // input [11 : 0] b (signed ADC sample)
            .p(window_sample) // output [11 : 0] p (truncated output)
            );
endmodule
```

compute_fft.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: UWindsor
// Engineer: Sabrina
//
// Create Date:    13:39:17 09/02/2011
// Design Name:
// Module Name:    compute_fft
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module compute_fft(
    input adc_valid,
    input clk_100,
            input clk_48,
    input [11:0] window_sample,
    input reset,
            input dcm_ready,
            output fifo_ready,
            output [10:0] xk_index,
    output [23:0] xk_re,
    output [23:0] xk_im,
            output dv
    );


            wire rfd;
            wire [11:0] fifo_out_fft_in;
            wire prog_empty;
            wire start;
            assign start = ~prog_empty;                //turn every 0 to 1 and every 1 to 0


            //this counter holds the fifo reset signal high for 256 clk_100 clock cycles after DCMs lock
            reg [7:0] fifo_wait;
```

```
		always@(posedge clk_100)
		begin
			if(reset)
				fifo_wait <= 8'd0;
			else if(fifo_wait < 8'd255 && dcm_ready)
				fifo_wait <= fifo_wait + 1;
			else
				fifo_wait <= fifo_wait;
		end
		assign fifo_ready = &fifo_wait;

		window_to_fft_fifo u0_window_to_fft_fifo (
		.rst(reset), // input rst
		.wr_clk(clk_48), // input wr_clk
		.rd_clk(clk_100), // input rd_clk
		.din(window_sample), // input [11 : 0] din
		.wr_en(adc_valid), // input wr_en
		.rd_en(rfd), // input rd_en
		.dout(fifo_out_fft_in), // output [11 : 0] dout
		.full(), // output full
		.empty(), // output empty
		.prog_empty(prog_empty) // output prog_empty
		);

		fft_2048 u0_fft (
		.clk(clk_100), // input clk
		.start(start), // input start
		.unload(1'b1), // input unload
		.xn_re(fifo_out_fft_in), // input [11 : 0] xn_re
		.xn_im(12'b0), // input [11 : 0] xn_im
		.fwd_inv(1'b1), // input fwd_inv
		.fwd_inv_we(1'b0), // input fwd_inv_we
		.rfd(rfd), // ouput rfd
		.xn_index(), // ouput [10 : 0] xn_index
		.busy(), // ouput busy
		.edone(), // ouput edone
		.done(), // ouput done
		.dv(dv), // ouput dv
		.xk_index(xk_index), // ouput [10 : 0] xk_index
		.xk_re(xk_re), // ouput [23 : 0] xk_re
		.xk_im(xk_im)  // ouput [23 : 0] xk_im
		);
endmodule
```

compute_mag_sq.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: UWindsor
// Engineer: Sabrina
//
// Create Date:    21:46:46 09/05/2011
// Design Name:
// Module Name:    compute_mag_sq
// Project Name:
// //////////////////////////////////////////////////////////////////////////////////
module compute_mag_sq(
    input clk_100,
    input [23:0] xk_re,
    input [23:0] xk_im,
    input [10:0] xk_index,
	 input dv,
output [47:0] mag_sq,
output dv_delayed
            );

            //setup a delayed version of the data valid signal for use in multiplier clock enables and
            //for the load buffer enable
            reg [4:0] delay;
            always @(posedge clk_100)
                        delay <= {dv, delay[4:1]};

            assign dv_delayed = delay[0];

           //make a condition that checks to see if data is NOT valid, AND xk_index = 0
           //this keeps our mult_ce and buff_load signals from going high for several
           //cycles after dv goes low
           wire data_invalid;
           assign data_invalid = (!dv) && (xk_index == 0) ? 1'b1 : 1'b0;

           //setup two comparators; one for the multiplier clock enables, one for the buffer load enable
           //using 1029 = (FFT point size / 2) + (pipelined multiplier latency)
           //      1029 = 2048/2 + 5
           //add an extra delay for second comparator (buffer delay is 1)
           wire mult_compare, buff_compare;

           assign mult_compare = xk_index < 1028 ? 1'b1 : 1'b0;
           assign buff_compare = xk_index < 1029 ? 1'b1 : 1'b0;

           wire mult_ce, buff_load;
           assign mult_ce = mult_compare & (dv | (|delay[4:1]) ) & ~data_invalid;
           assign buff_load = buff_compare & delay[0] & ~data_invalid;

           //output of multipliers
           wire [47:0] xk_re_sq, xk_im_sq;

           //fft real output squarer
           mult_48 u0_mult_xk_re_sq (
           .clk(clk_100), // input clk
           .a(xk_re), // input [23 : 0] a
           .b(xk_re), // input [23 : 0] b
           .ce(mult_ce), // input ce
           .p(xk_re_sq) // output [47 : 0] p
           );

           //fft imaginary output squarer
           mult_48 u0_mult_xk_im_sq (
           .clk(clk_100), // input clk
           .a(xk_im), // input [23 : 0] a
           .b(xk_im), // input [23 : 0] b
           .ce(mult_ce), // input ce
           .p(xk_im_sq) // output [47 : 0] p
           );

           //output of adder that gives us (xk^2 + re^2) = |FFT_output|^2
           wire [47:0] mag_sq_add_out_buff_in;
           assign mag_sq_add_out_buff_in = xk_re_sq + xk_im_sq;

           reg [47:0] out_buff;
           always @(posedge clk_100)
           begin
                       if(buff_load)
                                   out_buff <= mag_sq_add_out_buff_in;
                       else
```

```
                                        out_buff <= out_buff;
                end
                //assign the output to the module output
                assign mag_sq = out_buff;
endmodule
```

ca_cfar.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:                     UWindsor
// Engineer:                    Sabrina
//
// Create Date:    16:40:54 03/28/2013
// Design Name:       CA_CFAR Design
// Module Name:    ca_cfar
// Project Name:
//////////////////////////////////////////////////////////////////////////////////
// This module implements the CA-CFAR algorithm to identify valid targets from
// discrete frequency samples with noise and clutter. These samples are obtained
// by computing the peak intensity for every frequency bin as output from the FFT.
//
// - Sabrina Zereen -
//////////////////////////////////////////////////////////////////////////////////

module ca_cfar(
    input clk,
                input reset,
                input [47:0] inA,
                input [47:0] inB,
                input [47:0] inC,
                input [47:0] inD,
                input start,
                output [12:0] target_abs,
                output [15:0] target_pos,
                output new_target,
                output start_cfar,
                output complete
                );

// Internal registers
reg [12:0] target_abs_reg;  //[needs to be changed]
reg [15:0] target_pos_reg;       //[needs to be changed]
reg new_target_reg;
reg start_cfar_reg;
reg complete_reg;

reg [12:0] buffer [31:0]; // store 32 cells for CFAR processing
reg [9:0] indexa; // used in buffering data
reg [4:0] indexb; // used in buffering data
reg [4:0] indexc; // for CFAR routine
(* KEEP = "TRUE"*) reg [14:0] avgL; // cell averaging to left of CUT
(* KEEP = "TRUE"*) reg [14:0] avgR; // cell averaging to right of CUT
reg [12:0] avg; // threshold average
reg cfar_done;
reg [1:0] st; // internal flag to sort CFAR stages
(* KEEP = "TRUE"*) reg [17:0] T; // dynamic threshold result from CFAR processing
reg [4:0] K; // 5-bit decimal constant for CFAR
reg [12:0] CUT;

assign target_abs = target_abs_reg[12:0];             /// [**** linking input with register ]
assign target_pos = target_pos_reg[15:0];             /// [**** linking input with register ]
assign new_target = new_target_reg;                    /// [**** linking input with register ]
assign start_cfar  = start_cfar_reg;              /// [**** linking input with register ]
assign complete = complete_reg;                                 /// [**** linking input with register ]

integer file_hdl_Cfar;

initial begin
        file_hdl_Cfar = $fopen("./test_result/Cfar.txt","w");
end

always @ ( posedge clk )
begin
        if( reset == 1 )
        begin
                indexa <= 10'd0;
                indexb <= 5'd0;
                start_cfar_reg <= 1'b0;
        end

        else if( complete_reg == 1 ) // if all 1024 values have been processed
        begin
                indexa <= 10'd0;
                indexb <= 5'd0;
                start_cfar_reg <= 1'b0;
        end

        else if( start == 0 && start_cfar_reg == 1 ) // if CFAR processing is active
        begin
                if( cfar_done == 1 )
                begin
                        start_cfar_reg <= 1'b0; // clear signal, proceed with buffering
                        indexb <= 5'd0; // reset for next 32 values
                end
                else
                begin
                        start_cfar_reg <= 1'b1;
                        indexb <= 5'd31; // to avoid truncation by Xilinx ISE
                end
        end

        else if( start == 1 && start_cfar_reg == 0 ) // if CFAR processing is not active
        begin
                buffer[indexb] <= inA;
                buffer[indexb+1] <= inB;
                buffer[indexb+2] <= inC;
                buffer[indexb+3] <= inD;

                if( indexa == 1020 ) // 1024 counter
                        indexa <= 10'd1023; // avoid truncation and mark completion of all samples
                else
                        indexa <= indexa + 4;

                if( indexb == 28 )
                begin
                        indexb <= 5'd0; // 32 counter
                        start_cfar_reg <= 1'b1; // start CFAR routine
                end
                else
                        indexb <= indexb + 4;
        end
end
```

```
//--------------
// CFAR process
//--------------
always @ ( posedge clk )
begin
        if( reset == 1 )
        begin
                new_target_reg <= 1'b0;
                target_abs_reg <= 13'd0;
                target_pos_reg <= 16'd0;
                avg <= 13'd0;
                avgR <= 15'd0;
                avgL <= 15'd0;
                indexc <= 5'd0;
                cfar_done <= 1'b0;
                st <= 2'b00;
                K <= 5'b10011; // setting K = (11111) to avoid truncation
                                                // K = Pfa^(-1/(2*M)) - 1 ; e.g. Pfa=10^-6, M=8,
                                                // therefore K=4.63~(10011)
                                                // K has 3 integer bits, 2 fraction bits
                T <= 18'd0;
                CUT <= 13'd0;
                complete_reg <= 1'b0;
        end

        else if( complete_reg == 1 )
                complete_reg <= 1'b0;

        // After every 32 values or valid target detection
        else if( cfar_done == 1 || new_target_reg == 1 )
        begin
                cfar_done <= 1'b0; // reset flag, ready for next batch of 32 cells
                target_abs_reg <= 13'd0;
                target_pos_reg <= 16'd0;
        end

        // Get the averages for M=4
        else if( start_cfar_reg == 1 && cfar_done == 0 && st == 2'b00 )
        begin
                new_target_reg <= 1'b0; // reset new valid target output signal

                if( indexa >= 10'd0 && indexa <= 10'd511 )
                        K <= 5'd20; // Pfa = 10^-6, min. K = 5.00
                else if( indexa >= 10'd512 && indexa <= 10'd851 )
                        K <= 5'd17; // Reduced K = 4.25 for attenuated medium range targets
                else if( indexa >= 10'd852 )
                        K <= 5'd16; // Reduced K = 4.00 for attenuated long range targets

                if( indexc < 6 )
                begin
                        avgR <= buffer[indexc+3] + buffer[indexc+4] + buffer[indexc+5]      + buffer[indexc+6];
                        avgL <= buffer[indexc+3] + buffer[indexc+4] + buffer[indexc+5]      + buffer[indexc+6];
                end
                else if( indexc > 25 )
                begin
                        avgR <= buffer[indexc-3] + buffer[indexc-4] + buffer[indexc-5]      + buffer[indexc-6];
                        avgL <= buffer[indexc-3] + buffer[indexc-4] + buffer[indexc-5] + buffer[indexc-6];
                end
                else
                begin
                        avgR <= buffer[indexc+3] + buffer[indexc+4] + buffer[indexc+5]      + buffer[indexc+6];
                        avgL <= buffer[indexc-3] + buffer[indexc-4] + buffer[indexc-5] + buffer[indexc-6];
                        end
                        st <= 2'b01; // move to next CFAR stage
            end

            // Add the averages
            else if( start_cfar_reg == 1 && cfar_done == 0 && st == 2'b01 )
            begin
                        avg <= avgR[14:3] + avgL[14:3] + 1; // (avgR/4 + avgL/4)/2 + 1 (to avoid zero)
                        st <= 2'b10;
            end

            // Compute the dynamic threshold
            else if( start_cfar_reg == 1 && cfar_done == 0 && st == 2'b10 )
            begin
                        T <= avg * K; // threshold value for current CFAR cells
                        CUT <= buffer[indexc]; // CUT has equal word length as integer part of T
                        st <= 2'b11;
            end

            // Decision to extract valid target from clutter
            else if( start_cfar_reg == 1 && cfar_done == 0 && st == 2'b11 )
            begin //$display("%d %d",CUT,indexa+indexc-32);
                        if( CUT > T[14:2] && CUT > 13'd7 ) // compare integer part and exclude FFT noise
                        begin
                                    new_target_reg <= 1'b1; // assert new valid target signal to pairing module
                                    target_abs_reg <= CUT; // output target peak intensity
                                    target_pos_reg <= indexa + indexc - 30; // output target FFT bin number
                                    K <= 5'b00000; // temporary clear to avoid truncation
                        end
                        if( indexc == 31 ) // mark completion of CFAR processing on current 32 cells
                                    cfar_done <= 1'b1;
                        if( indexc == 31 && indexa == 1023 ) // if all 1024 samples done
                                    complete_reg <= 1'b1; // send completion signal to pairing module
                        indexc <= indexc + 1; // move to next cell for CFAR processing
                        st <= 2'b00;
            end

            if( new_target_reg == 1 )
                        new_target_reg <= 1'b0; // reset new valid target signal

            if(target_abs_reg>0)
            begin
                        $fwrite(file_hdl_Cfar,"target_abs_reg = %b, target_pos_reg = %b\n",target_abs_reg,target_pos_reg);
                        $fwrite(file_hdl_Cfar,"target_abs_reg = %d, target_pos_reg = %d\n",target_abs_reg,target_pos_reg);
            end
end
endmodule
```

pairing.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: UWndisor
// Engineer: Sabrina
//
// Create Date:    17:55:05 04/01/2013
// Design Name:
// Module Name:    pairing
// Project Name:
// //////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////
// This module is responsible for pairing the peaks detected by the CFAR unit
// and producing the target ranges and velocities for all detected targets.
//// - Sabrina Zereen -
////////////////////////////////////////////////////////////////////////////////

module pairing(
    input clk,
                input reset,
                input new_target,
                input [12:0] target_abs,
                input [9:0] target_pos,
                input complete,
                input updown,
                input [7:0] unit_vel,
                output [19:0] target_info,
                output info_valid
                );
/*
// Inputs to module
input clk; // system/global clock
input reset; // synchronous reset
input new_target; // new valid target from CFAR module
input [12:0] target_abs; // target peak intensity
input [9:0] target_pos; // target frequency bin number
input complete; // CFAR completion signal from CFAR module
input updown; ///* sweep direction, 1 for up, 0 for down
                                                         // this signal is used inverted (0 for up, 1 for down) because..
                                                        //..during down sweep sampling, up sweep processing is done and..
                                                        //.. vice versa //
input [7:0] unit_vel; // radar unit's velocity / car's velocity

// Outputs from module
output [19:0] target_info; // 10 bits target velocity, 10 bits target distance
output info_valid; // signal to display module
*/
// Internal registers
reg [19:0] target_info_reg;
reg info_valid_reg;
wire updown_reg;

reg [12:0] abs_bufup [7:0]; // maximum 8 targets in up sweep
reg [9:0] pos_bufup [7:0];
reg upfill; // flag to mark fully filled up sweep buffers
reg [12:0] abs_bufdown [7:0]; // maximum 8 targets in down sweep
reg [9:0] pos_bufdown [7:0];
reg downfill; // flag to mark fully filled down sweep buffers
reg [2:0] count; // index for up sweep and down sweep buffers
reg [2:0] paircount; // final count of records accepted for pairing from CFAR
reg start_pairing; // flag to commence pairing and output process
reg pairing_done; // flag to mark completion of pairing process
reg [2:0] indexup; // counter to count through up sweep records while pairing
reg [2:0] indexdown; // counter to count through down sweep records while pairing
reg [2:0] tmpindex; // used to store the final matching pair index
reg [6:0] vel_fac; // multiplication constant for velocity calculation
(* KEEP = "TRUE"*) reg [17:0] velocity; // computed velocity - (13bits).(6bits)
reg [10:0] range_fac; // multiplication constant for range calculation
(* KEEP = "TRUE"*) reg [21:0] range; // computed range - (11bits).(11bits)
reg [1:0] st; // internal flag
reg stb; // internal flag
reg [9:0] posa, posb; // used to analyse spectral closeness during pairing
reg [13:0] absa, absb, absc; // used to analyse peak intensity closeness during pairing
reg [10:0] sum_pos, diff_pos; // sum for range, diff for velocity
reg faster; // 0 if target is slower, 1 is target is faster
reg updone; // mark up sweep processing done

wire reset_n;
assign reset_n = ~reset;

assign target_info = target_info_reg[19:0];          /// [**** linking input with register ]
assign info_valid = info_valid_reg;                                          /// [**** linking input with register ]
assign updown_reg = 0;

integer file_hdl_Pairing;

initial begin
        file_hdl_Pairing = $fopen("./test_result/Pairing.txt","w");
end


//----------------------------------------------
// Accept data from CFAR module
// - spectral copies are ignored by this module
//----------------------------------------------
always @ ( posedge clk )
begin
$fwrite(file_hdl_Pairing,"Inside first always\n");

        if( reset_n == 1 )
        begin
        $fwrite(file_hdl_Pairing,"Inside first reset_n ==1\n");
                count <= 3'd0;
                paircount <= 3'd0;
                abs_bufup[0] <= 13'd0;
                pos_bufup[0] <= 10'd0;
                abs_bufdown[0] <= 13'd0;
                pos_bufdown[0] <= 10'd0;
                upfill <= 1'b0;
                downfill <= 1'b0;
                start_pairing <= 1'b0;
                updone <= 1'b0;
        end

        // clear pairing process flags
        else if( reset_n == 0 && pairing_done == 1 )
        begin
        $fwrite(file_hdl_Pairing,"Inside reset_n == 0 && pairing_done == 1\n");
                start_pairing <= 1'b0;
                paircount <= 3'd0;
                updone <= 1'b0;
        end

        // if CFAR processing for current sweep direction is complete
        else if( reset_n == 0 && complete == 1 )
        begin
                if( updown_reg == 0 ) // if up sweep is done
                begin
                        paircount <= count; // store the total number of targets for later use
                        updone <= 1'b1;
                end

                count <= 3'd0; // reset_n counter to 0
                upfill <= 1'b0; // clear flags
                downfill <= 1'b0;

                if( updown_reg == 1 && updone == 1 ) // if the down sweep has been completely obtained
```

```
                    begin
                              $fwrite(file_hdl_Pairing,"%d %d %d %d %d %d %d %d
%d",pos_bufup[0],pos_bufup[1],pos_bufup[2],pos_bufup[3],pos_bufup[4],pos_bufup[5],pos_bufup[6],pos_bufup[7],paircount);
                              //$display("%d %d %d %d %d %d %d %d
%d",pos_bufdown[0],pos_bufdown[1],pos_bufdown[2],pos_bufdown[3],pos_bufdown[4],pos_bufdown[5],pos_bufdown[6],pos_
bufdown[7],count);
                              start_pairing <= 1'b1;
                    end
          end

          //----------
          // UP SWEEP
          //----------
          else if( reset_n == 0 && updown_reg == 0 && new_target == 1 && upfill == 0 )
          begin //$display("up %d %d",target_abs,target_pos);
          $fwrite(file_hdl_Pairing,"up-sweep %d %d",target_abs,target_pos);
                    // first valid target detection stored without 'spectral copy' checking
                    if( count == 0 && target_pos > 4 ) // ignore DC values
                    begin
                              abs_bufup[count] <= target_abs;
                              pos_bufup[count] <= target_pos;
                              count <= count + 1;
                    end

                    // 'spectral copy' checking
                    else if( count >= 1 )
                    begin
                              // if new CFAR detection is a 'spectral copy' of previous target
                              if( target_pos == pos_bufup[count-1] + 1 )
                              begin
                                        if( target_abs > abs_bufup[count-1] ) // store larger peak intensity
                                        begin
                                                  abs_bufup[count-1] <= target_abs; // update previous target record
                                                  pos_bufup[count-1] <= target_pos;
                                        end
                              end

                              else
                              begin
                                        abs_bufup[count] <= target_abs; // add new target record
                                        pos_bufup[count] <= target_pos;
                                        count <= count + 1; // increment counter
                                        if( count == 7 )
                                                  upfill <= 1'b1; // mark up sweep buffer filled
                              end
                    end

          end


          //-----------
          // DOWN SWEEP
          //-----------
          else if( reset_n == 0 && updown_reg == 1 && new_target == 1 && downfill == 0 )
          begin //$display("down %d %d",target_abs,target_pos);
          $fwrite(file_hdl_Pairing,"down-sweep %d %d",target_abs,target_pos);
                    // first valid target detection stored without 'spectral copy' checking
                    if( count == 0 && target_pos > 4 ) // ignore DC values
                    begin
                              abs_bufdown[count] <= target_abs;
                              pos_bufdown[count] <= target_pos;
                              count <= count + 1;
```

```
                end

                // 'spectral copy' checking
                else if( count > 0 )
                begin
                        // if new CFAR detection is a 'spectral copy' of previous target
                        if( target_pos == pos_bufdown[count-1] + 1 )
                        begin
                                if( target_abs > abs_bufdown[count-1] ) // store larger peak intensity
                                begin
                                        abs_bufdown[count-1] <= target_abs; // update previous target record
                                        pos_bufdown[count-1] <= target_pos;
                                end
                        end

                        else
                        begin
                                abs_bufdown[count] <= target_abs; // add new target record
                                pos_bufdown[count] <= target_pos;
                                count <= count + 1; // increment counter
                                if( count == 7 )
                                        downfill <= 1'b1; // mark up sweep buffer filled
                        end
                end

        end

        // clear the record from down buffer when a pair has been matched successfully
        if( st == 2'b10 && start_pairing == 1 )
        begin
                abs_bufdown[tmpindex] <= 13'd0;
                pos_bufdown[tmpindex] <= 10'd0;
        end

end

//-----------------------------------------------
// Peak Pairing
// Criteria:
//              (1) +-84 frequency bins
//              (2) compare peak intensity
//-----------------------------------------------
always @ ( posedge clk )
begin

        if( reset == 1 )
        begin
        $fwrite(file_hdl_Pairing,"Peak pairing-1\n");
                target_info_reg <= 20'd0;
                info_valid_reg <= 1'b0;
                pairing_done <= 1'b0;
                indexup <= 3'd0;
                indexdown <= 3'd0;
                tmpindex <= 3'd0;
                vel_fac <= 7'b1101101; // (11.01101)binary = (3.40625)decimal
                range_fac <= 11'b00010111110; // (0.00010111110)binary = (0.0927734375)decimal
                /* these factors have been obtained by converting the equations into
                        constants, saving hardware and making computation quicker:
                        Fr = 4*Fsweep/Tsweep*range/c , Fd = 2*Ft*relative_velocity/c */
                st <= 2'b00;
                stb <= 1'b0;
                posa <= 10'd0;
```

```
		posb <= 10'd0;
		absa <= 13'd0;
		absb <= 13'd0;
		absc <= 13'd0;
		sum_pos <= 11'd0;
		diff_pos <= 11'd0;
		faster <= 1'b0;
		velocity <= 18'd0;
		range <= 22'd0;
	end

	// if pairing is complete
	else if( reset == 0 && pairing_done == 1 )
	begin
	$fwrite(file_hdl_Pairing,"Peak pairing-2\n");
		target_info_reg <= 20'd0;
		info_valid_reg <= 1'b0;
		pairing_done <= 1'b0;
		indexup <= 3'd0;
		indexdown <= 3'd0;
		tmpindex <= 3'd0;
		st <= 2'b00;
		stb <= 1'b0;
		posa <= 10'd0;
		posb <= 10'd0;
		absa <= 13'd0;
		absb <= 13'd0;
		absc <= 13'd0;
		sum_pos <= 11'd0;
		diff_pos <= 11'd0;
		faster <= 1'b0;
		velocity <= 18'd0;
		range <= 22'd0;
	end

	// pair target peaks from up and down sweeps
	else if( reset == 0 && start_pairing == 1 && indexdown <= paircount-1 )
	begin
	$fwrite(file_hdl_Pairing,"Peak pairing-3\n");
		target_info_reg <= 20'd0;
		info_valid_reg <= 1'b0;

		if( st == 2'b00 )
		begin

			// lower limit for criteria (1)
			if( pos_bufup[indexup] > pos_bufdown[indexdown] )
				posa <= pos_bufup[indexup] - pos_bufdown[indexdown]; // limit to +-84 i.e. 300kmph
			else
				posa <= pos_bufdown[indexdown] - pos_bufup[indexup];

			/* calculate peak intensity difference between current up sweep value
				and current down sweep value */
			if( abs_bufup[indexup] > abs_bufdown[indexdown] )
				absa <= abs_bufup[indexup] - abs_bufdown[indexdown];
			else
				absa <= abs_bufdown[indexdown] - abs_bufup[indexup];

			/* calculate peak intensity difference between current up sweep value
				and previously stored best match value */
			if( abs_bufup[indexup] > abs_bufdown[tmpindex] )
				absb <= abs_bufup[indexup] - abs_bufdown[tmpindex];
```

```
                    else
                            absb <= abs_bufdown[tmpindex] - abs_bufup[indexup];

                    /* calculate peak intensity difference between next up sweep value
                            and previously stored best match value for the current target */
                    if( indexup < paircount - 1 )
                    begin
                            if( abs_bufup[indexup+1] > abs_bufdown[tmpindex] )
                                    absc <= abs_bufup[indexup+1] - abs_bufdown[tmpindex];
                            else
                                    absc <= abs_bufdown[tmpindex] - abs_bufup[indexup-1];
                    end
                    else
                            absc <= 13'd8191;

                    // ensure next up sweep sample is within +-84 range of previous best match
                    if( indexup < paircount - 1 )
                    begin
                            if( pos_bufup[indexup+1] > pos_bufdown[indexdown] )
                                    posb <= pos_bufup[indexup+1] - pos_bufdown[indexdown];
                            else
                                    posb <= pos_bufdown[indexdown] - pos_bufup[indexup+1];
                    end
                    else
                            posb <= 10'd1023;

                    st <= 2'b01; // next stage
            end

//////// update best match according to criteria (1,2)
            else if( st == 2'b01 )
            begin
                    // if the peak in the down sweep is spectrally close to peak in up sweep
                    if( posa < 84 && posa <= posb )
                    begin
                            // if current down sweep peak is closer in intensity
                            if( absa <= absb && absa <= absc )
                                    tmpindex <= indexdown; // update best match index
                    end

                    if( indexdown == paircount-1 ) // if all down sweep peaks have been assessed
                            st <= 2'b10; // next stage
                    else
                    begin
                            indexdown <= indexdown + 1; // move to next down sweep peak
                            st <= 2'b00; // return to re-compute new parameters
                    end
            end

//////// obtain sum and difference of matched frequency bin indices
            else if( st == 2'b10 )
            begin
                    indexdown <= 3'd0; // clear index to restart from first record in down sweep
                    sum_pos <= pos_bufup[indexup] + pos_bufdown[tmpindex]; // for target range

                    if( pos_bufdown[tmpindex] > 0 )
                    begin
                            // for target relative velocity
                            if( pos_bufup[indexup] > pos_bufdown[tmpindex] ) // slower target
                            begin
                                    diff_pos <= pos_bufup[indexup] - pos_bufdown[tmpindex];
                                    faster <= 1'b0;
```

```
                                        end
                                        else // faster target
                                        begin
                                                diff_pos <= pos_bufdown[tmpindex] - pos_bufup[indexup];
                                                faster <= 1'b1;
                                        end

                                        st <= 2'b11; // next stage
                                end

                                else
                                begin
                                        if( indexup < paircount - 1 )
                                        begin
                                                indexup <= indexup + 1;
                                                st <= 2'b00;
                                        end
                                        else
                                                pairing_done <= 1'b1;
                                end

                        end

//////// compute the velocity and range and output as single bus
                        else if( st == 2'b11 )
                        begin
                                if( stb == 0 ) // stage to compute velocity and range
                                begin
                                        if( faster == 0 ) // if the target is not faster than own vehicle
                                                velocity <= vel_fac * diff_pos;
                                        else // if the target is faster than own vehicle
                                                velocity <= vel_fac * diff_pos;

                                        range <= range_fac * sum_pos;
                                        stb <= 1'b1;
                                end

                                else // final step: output target_info_reg, update indexup
                                begin
                                        if( faster == 0 ) // extract (9bits).(0bit) velocity
                                                target_info_reg[19:11] <= unit_vel - velocity[13:5];
                                        else
                                                target_info_reg[19:11] <= unit_vel + velocity[13:5];

                                        target_info_reg[10] <= velocity[4]; // attach the fraction bit

                                        target_info_reg[9:0] <= range[18:9]; // extract (8bits).(2bits) range

                                        $fwrite(file_hdl_Pairing,"range = %d, speed =
%d\n",target_info_reg[9:0],target_info_reg[19:10]);

                                        info_valid_reg <= 1'b1; // alert display unit of valid target information
                                        tmpindex <= 3'd0;
                                        posa <= 10'd0;
                                        posb <= 10'd0;
                                        absa <= 13'd0;
                                        absb <= 13'd0;
                                        stb <= 1'b0;
                                        st <= 2'b00; // reset_n to first state
                                        indexup <= indexup + 1; // move to next record in up sweep buffer

                                        if( indexup == paircount ) // if all records have been assessed
                                        pairing_done <= 1'b1;
                                end
                        end
                end
end
endmodule
```

compute_target_info.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: UWindsor
// Engineer: Sabrina
//
// Create Date:    14:20:39 09/07/2011
// Design Name:
// Module Name:    compute_target_info
// Project Name:
//////////////////////////////////////////////////////////////////////////////////
module compute_target_info(
    input clk_100,
    input reset,
    input [15:0] max_bin,/// [**** changed to 16bits from 11 bits ]
    input valid_in,
    output [28:0] b0t_range,/// [**** changed to 29BITS from 16 bits ]
    output [28:0] b1t_range,
    output [28:0] b2t_range,
    output [28:0] b0t_speed,
    output [28:0] b1t_speed,
    output [28:0] b2t_speed,
    output b0t_dir,
    output b1t_dir,
    output b2t_dir
    );

        //3-bit up counter that resets after 5
        reg [2:0] count3;
        always @(posedge clk_100)
        begin
                if(reset)
                        count3 <= 3'd0;
                else
                begin
                        if(valid_in)
                        begin
                                if(count3 < 3'd5)
                                        count3 <= count3 + 1;
                                else
                                        count3 <= 3'd0;
                        end
                        else
                                count3 <= count3;
                end
        end

        //some control signals
        wire up_load, b0_load, b1_load, b2_load;
        assign up_load = (valid_in && (count3[0] == 1'b0)) ? 1'b1 : 1'b0;
        assign b0_load = valid_in & ~count3[2] & ~count3[1] & count3[0];
        assign b1_load = valid_in & ~count3[2] & count3[1] & count3[0];
        assign b2_load = valid_in & count3[2] & ~count3[1] & count3[0];
```

```
//buffer holds up-bin, load enable controlled by up_load
reg [15:0] up_buff;/// [**** changed to 16bits from 11 bits ]
reg [15:0] dn_buff;/// [just testing..... ]
always @(posedge clk_100)
begin
        if(reset)
                up_buff <= 16'd0; /// [**** changed to 16bits from 11 bits ]
        else if(up_load)
                up_buff <= max_bin;
        else
                up_buff <= up_buff;
end

//bin_sum and bin_diff are the sum and difference of the up and down bins
wire [15:0] bin_sum, bin_diff; /// [**** changed to 16bits from 11 bits ]
assign bin_sum = up_buff + max_bin;
assign bin_diff = up_buff - max_bin;

//MSB of bin_diff makes up the sign
wire sign_bit;
assign sign_bit = bin_diff[15]; //[CHANGED TO bin_diif[15] from [9]]

//this block converts the two's complement bin_diff value to its absolute value
reg [15:0] bin_diff_abs;/// [**** changed to 16bits from 10 bits ]
always @(bin_diff)
begin
        if(bin_diff[15] == 1) //[**** changed ]
                bin_diff_abs = ~bin_diff + 1;
        else
                bin_diff_abs = bin_diff;
end

wire [28:0] speed, range; //[**** changed to 29 drom 16bits]
calc_range_mult u0_mult_range_calc (
.a(bin_sum), // input [9 : 0] a
.p(range) // output [15 : 0] p
);

calc_speed_mult u0_mult_speed_calc (
.a(bin_diff_abs), // input [9 : 0] a
.p(speed) // output [15 : 0] p
);


//output registers load enables controlled by previously defined signals
reg [28:0] b0r_buff, b1r_buff, b2r_buff, b0s_buff, b1s_buff, b2s_buff; //[**** changed to 29 drom 16bits]
reg b0d_buff, b1d_buff, b2d_buff;

always @(posedge clk_100)
begin
        if(reset)               //[**** changed to 29 drom 16bits]
        begin
                b0r_buff <= 29'd0;
                b0s_buff <= 29'd0;
                b0d_buff <= 1'b0;
        end
        else if(b0_load)
        begin
                b0r_buff <= range;
                b0s_buff <= speed;
                b0d_buff <= sign_bit;
        end
```

```
		else
		begin
			b0r_buff <= b0r_buff;
			b0s_buff <= b0s_buff;
			b0d_buff <= b0d_buff;
		end
	end

	always @(posedge clk_100)
	begin
		if(reset) //[**** changed to 29 drom 16bits]
		begin
			b1r_buff <= 29'd0;
			b1s_buff <= 29'd0;
			b1d_buff <= 1'b0;
		end
		else if(b1_load)
		begin
			b1r_buff <= range;
			b1s_buff <= speed;
			b1d_buff <= sign_bit;
		end
		else
		begin
			b1r_buff <= b1r_buff;
			b1s_buff <= b1s_buff;
			b1d_buff <= b1d_buff;
		end
	end

	always @(posedge clk_100)
	begin
		if(reset) //[**** changed to 29 drom 16bits]
		begin
			b2r_buff <= 29'd0;
			b2s_buff <= 29'd0;
			b2d_buff <= 1'b0;
		end
		else if(b2_load)
		begin
			b2r_buff <= range;
			b2s_buff <= speed;
			b2d_buff <= sign_bit;
		end
		else
		begin
			b2r_buff <= b2r_buff;
			b2s_buff <= b2s_buff;
			b2d_buff <= b2d_buff;
		end
	end

	assign b0t_range = b0r_buff;
	assign b1t_range = b1r_buff;
	assign b2t_range = b2r_buff;
	assign b0t_speed = b0s_buff;
	assign b1t_speed = b1s_buff;
	assign b2t_speed = b2s_buff;
	assign b0t_dir = b0d_buff;
	assign b1t_dir = b1d_buff;
	assign b2t_dir = b2d_buff;
```

```
endmodule
```

lcd_driver.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: UWindsor
// Engineer: Sabrina
//
// Create Date:    20:12:14 08/15/2011
// Design Name:
// Module Name:    lcd_driver
// Project Name:
// //////////////////////////////////////////////////////////////////////////////////
module lcd_driver(
   input clk_100,
   input reset,
   input ready,
   input [2:0] BEAM_SHOW,                          //now holds BEAM_SWITCH for LCD testing...
            input [28:0] b0t_range,         //[** changed to 29 bits from 16 bit]
            input [28:0] b1t_range,
            input [28:0] b2t_range,
            input [28:0] b0t_speed,
            input b0t_dir,
            input [28:0] b1t_speed,
            input b1t_dir,
            input [28:0] b2t_speed,
            input b2t_dir,
            output reg LCD_E,
   output reg LCD_RS,
   output reg LCD_RW,
   output reg [3:0] LCD_OUT
   );


//inverted reset signal
wire reset_n;
assign reset_n = ~reset;


//divider for 1 kHz clock
reg [16:0] clk_1k_ctr;
reg clk_1k;

always @(posedge clk_100)
begin
        if(reset)
        begin
                clk_1k <= 1'b0;
                clk_1k_ctr <= 17'd0;
        end
        else if(ready)
        begin
                if(clk_1k_ctr < 99998)
                begin
                        clk_1k <= clk_1k;
                        clk_1k_ctr <= clk_1k_ctr + 1;
                end
                else
                begin
                        clk_1k <= ~clk_1k;
                        clk_1k_ctr <= 17'd0;
                end
        end
        else
        begin
                clk_1k <= clk_1k;
                clk_1k_ctr <= clk_1k_ctr;
        end
end

//registers to hold changing characters for LCD
reg [3:0] type, num, r100, r10, r1, rp1, rp01, s100, s10, s1, sp1, sp01, sign;


//latch data for display signals
reg [28:0] disp_rng; //[** changed to 29 bits from 16 bit]
reg [28:0] disp_spd;//[** changed to 29 bits from 16 bit]


//select which data to display
always @(posedge clk_1k or negedge reset_n)
begin
        if(!reset_n)
        begin
                disp_rng <= 29'd0; //[** changed to 29 bits from 16 bit]
                disp_spd <= 29'd0;
                type <= 4'b0010;
                num <= 4'b0000;
                sign <= 4'b0000;
        end
        else if(reset_n && BEAM_SHOW[0])
        begin
                disp_rng <= b0t_range;
                disp_spd <= b0t_speed;
                type <= 4'b0010;
                num <= 4'b0000;
                if(b0t_dir)
                        sign <= 4'b1011;
                else
                        sign <= 4'b1101;
        end
        else if(reset_n && BEAM_SHOW[1])
        begin
                disp_rng <= b1t_range;
                disp_spd <= b1t_speed;
                type <= 4'b0010;
                num <= 4'b0001;

                if(b1t_dir)
                        sign <= 4'b1011;
                else
                        sign <= 4'b1101;
        end
        else if(reset_n && BEAM_SHOW[2])
        begin
                disp_rng <= b2t_range;
                disp_spd <= b2t_speed;
                type <= 4'b0010;
                num <= 4'b0010;
                if(b2t_dir)
                        sign <= 4'b1011;
                else
```

```
                    sign <= 4'b1101;
        end
        else
        begin
                disp_rng <= disp_rng;                           //chamged to b0t_range from disp_rng
                disp_spd <= disp_spd;                           //chamged to b0t_speed from disp_spd
                type <= type;
                num <= num;
                sign <= sign;
        end
end

//format the data for ascii
//upper bits is hard-coded into the lcd display routine
//only the lower bits get changed here
reg [28:0] hundreds, tens, tenths; //[** changed to 29 bits from 16 bit] removed 'ones'
reg [12:0] dbits;
reg [15:0] fbits;

always @(disp_rng, hundreds, tens, tenths)
begin

        dbits <= disp_rng[28:16];
        fbits <= disp_rng[15:0];

        //RANGE
        //first figure out the 100s

        if(dbits>=13'd200)
        begin
                r100<=4'd2;
                hundreds <= 8'd200;
        end

        else if(dbits >= 13'd100 && dbits <13'd200)
        begin
                r100 <= 4'd1;
                hundreds <= 7'd100;
        end
        else
        begin
                r100 <= 4'd0;
                hundreds <= 4'd0;
        end

        //now figure out the 10s
        if( ((dbits-hundreds) >= 13'd90) && ((dbits-hundreds) < 13'd100) ) //90
        begin
                r10 <= 4'd9;
                tens <= 7'd90;
        end
        else if( ((dbits-hundreds) >= 13'd80) && ((dbits-hundreds) < 13'd90) ) //80
        begin
                r10 <= 4'd8;
                tens <= 7'd80;
        end
        else if( ((dbits-hundreds) >= 13'd70) && ((dbits-hundreds) < 13'd80) ) //70
        begin
                r10 <= 4'd7;
                tens <= 7'd70;
        end
        else if( ((dbits-hundreds) >= 13'd60) && ((dbits-hundreds) < 13'd70) ) //60
```

```
begin
        r10 <= 4'd6;
        tens <= 6'd60;
end
else if( ((dbits-hundreds) >= 13'd50) && ((dbits-hundreds) < 13'd60) ) //50
begin
        r10 <= 4'd5;
        tens <= 6'd50;
end
else if( ((dbits-hundreds) >= 13'd40) && ((dbits-hundreds) < 13'd50) ) //40
begin
        r10 <= 4'd4;
        tens <= 6'd40;
end
else if( ((dbits-hundreds) >= 13'd30) && ((dbits-hundreds) < 13'd40) ) //30
begin
        r10 <= 4'd3;
        tens <= 5'd30;
end
else if( ((dbits-hundreds) >= 13'd20) && ((dbits-hundreds) < 13'd30) ) //20
begin
        r10 <= 4'd2;
        tens <= 5'd20;
end
else if( ((dbits-hundreds) >= 13'd10) && ((dbits-hundreds) < 13'd20) ) //10
begin
        r10 <= 4'd1;
        tens <= 4'd10;
end
else
begin
        r10 <= 4'd0;
        tens <= 4'd0;
end

//now the ones
if( ((dbits-hundreds-tens) >= 13'd9) && ((dbits-hundreds-tens) < 13'd10) ) //9
begin
        r1 <= 4'd9;
end
else if( ((dbits-hundreds-tens) >= 13'd8) && ((dbits-hundreds-tens) <  13'd9) ) //8
begin
        r1 <= 4'd8;
end
else if( ((dbits-hundreds-tens) >= 13'd7) && ((dbits-hundreds-tens) < 13'd8) ) //7
begin
        r1 <= 4'd7;
end
else if( ((dbits-hundreds-tens) >= 13'd6) && ((dbits-hundreds-tens) < 13'd7) ) //6
begin
        r1 <= 4'd6;
end
else if( ((dbits-hundreds-tens) >= 13'd5) && ((dbits-hundreds-tens) < 13'd6) ) //5
begin
        r1 <= 4'd5;
end
else if( ((dbits-hundreds-tens) >= 13'd4) && ((dbits-hundreds-tens) < 13'd5) ) //4
begin
        r1 <= 4'd4;
end
else if( ((dbits-hundreds-tens) >= 13'd3) && ((dbits-hundreds-tens) < 13'd4) ) //3
begin
```

```
		r1 <= 4'd3;
	end
	else if( ((dbits-hundreds-tens) >= 13'd2) && ((dbits-hundreds-tens) < 13'd3) ) //2
	begin
		r1 <= 4'd2;
	end
	else if( ((dbits-hundreds-tens) >= 13'd1) && ((dbits-hundreds-tens) < 13'd2) ) //1
	begin
		r1 <= 4'd1;
	end
	else //0
	begin
		r1 <= 4'd0;
	end

	//now the tenths
	if( ((fbits) >= 16'hE666) && ((fbits) < 16'hFFFF) ) //.9
	begin
		rp1 <= 4'd9;
		tenths <= 16'hE666;
	end
	else if( ((fbits) >= 16'hCCCD) && ((fbits) < 16'hE666) ) //.8
	begin
		rp1 <= 4'd8;
		tenths <= 16'hCCCD;
	end
	else if( ((fbits) >= 16'hB333) && ((fbits) < 16'hCCCD) ) //.7
	begin
		rp1 <= 4'd7;
		tenths <= 16'hB333;
	end
	else if( ((fbits) >= 16'h999A) && ((fbits) < 16'hB333) ) //.6
	begin
		rp1 <= 4'd6;
		tenths <= 16'h999A;
	end
	else if( ((fbits) >= 16'h8000) && ((fbits) < 16'h999A) ) //.5
	begin
		rp1 <= 4'd5;
		tenths <= 16'h8000;
	end
	else if( ((fbits) >= 16'h6666) && ((fbits) < 16'h8000) ) //.4
	begin
		rp1 <= 4'd4;
		tenths <= 16'h6666;
	end
	else if( ((fbits) >= 16'h4CCD) && ((fbits) < 16'h6666) ) //.3
	begin
		rp1 <= 4'd3;
		tenths <= 16'h4CCD;
	end
	else if( ((fbits) >= 16'h3333) && ((fbits) < 16'h4CCD) ) //.2
	begin
		rp1 <= 4'd2;
		tenths <= 16'h3333;
	end
	else if( ((fbits) >= 16'h199A) && ((fbits) < 16'h3333) ) //.1
	begin
		rp1 <= 4'd1;
		tenths <= 16'h199A;
	end
	else//.0
```

```
        begin
                rp1 <= 4'd0;
                tenths <= 16'h0000;
        end

        //finally the hundredths
        if( ((fbits-tenths) >= 16'h170A) && ((fbits-tenths) < 16'h199A) ) //.09
        begin
                rp01 <= 4'd9;
        end
        else if( ((fbits-tenths) >= 16'h147B) && ((fbits-tenths) < 16'h170A) ) //.08
        begin
                rp01 <= 4'd8;
        end
        else if( ((fbits-tenths) >= 16'h11EC) && ((fbits-tenths) < 16'h147B) ) //.07
        begin
                rp01 <= 4'd7;
        end
        else if( ((fbits-tenths) >= 16'h0F5C) && ((fbits-tenths) < 16'h11EC) ) //.06
        begin
                rp01 <= 4'd6;
        end
        else if( ((fbits-tenths) >= 16'h0CCD) && ((fbits-tenths) < 16'h0F5C) ) //.05
        begin
                rp01 <= 4'd5;
        end
        else if( ((fbits-tenths) >= 16'h0A3D) && ((fbits-tenths) < 16'h0CCD) ) //.04
        begin
                rp01 <= 4'd4;
        end
        else if( ((fbits-tenths) >= 16'h07AE) && ((fbits-tenths) < 16'h0A3D) ) //.03
        begin
                rp01 <= 4'd3;
        end
        else if( ((fbits-tenths) >= 16'h051F) && ((fbits-tenths) < 16'h07AE) ) //.02
        begin
                rp01 <= 4'd2;
        end
        else if( ((fbits-tenths) >= 16'h028F) && ((fbits-tenths) < 16'h051F) ) //.01
        begin
                rp01 <= 4'd1;
        end
        else //.00
        begin
                rp01 <= 4'd0;
        end
end

//get ascii values for the speed
//format the data for ascii
//upper bits is hard-coded into the lcd display routine
//only the lower bits get changed here
reg [15:0] s_hundreds, s_tens, s_ones, s_tenths;
always @(disp_spd, s_hundreds, s_tens, s_ones, s_tenths)


begin
        //SPEED
        //first figure out the 100s

        if( (disp_spd >= 16'hC800) && (disp_spd <= 16'hFFFF) )
        begin
```

```
		s100 <= 4'd2;
		s_hundreds <= 16'hC800;
	end
	else if( (disp_spd >= 16'h6400) && (disp_spd < 16'hC800) )
	begin
		s100 <= 4'd1;
		s_hundreds <= 16'h6400;
	end
	else
	begin
		s100 <= 4'd0;
		s_hundreds <= 16'h0000;
	end


	//now figure out the 10s
	if( ((disp_spd-s_hundreds) >= 16'h5A00) && ((disp_spd-s_hundreds) < 16'h6400) ) //90
	begin
		s10 <= 4'd9;
		s_tens <= 16'h5A00;
	end
	else if( ((disp_spd-s_hundreds) >= 16'h5000) && ((disp_spd-s_hundreds) < 16'h5A00) ) //80
	begin
		s10 <= 4'd8;
		s_tens <= 16'h5000;
	end
	else if( ((disp_spd-s_hundreds) >= 16'h4600) && ((disp_spd-s_hundreds) < 16'h5000) ) //70
	begin
		s10 <= 4'd7;
		s_tens <= 16'h4600;
	end
	else if( ((disp_spd-s_hundreds) >= 16'h3C00) && ((disp_spd-s_hundreds) < 16'h4600) ) //60
	begin
		s10 <= 4'd6;
		s_tens <= 16'h3C00;
	end
	else if( ((disp_spd-s_hundreds) >= 16'h3200) && ((disp_spd-s_hundreds) < 16'h3C00) ) //50
	begin
		s10 <= 4'd5;
		s_tens <= 16'h3200;
	end
	else if( ((disp_spd-s_hundreds) >= 16'h2800) && ((disp_spd-s_hundreds) < 16'h3200) ) //40
	begin
		s10 <= 4'd4;
		s_tens <= 16'h2800;
	end
	else if( ((disp_spd-s_hundreds) >= 16'h1E00) && ((disp_spd-s_hundreds) < 16'h2800) ) //30
	begin
		s10 <= 4'd3;
		s_tens <= 16'h1E00;
	end
	else if( ((disp_spd-s_hundreds) >= 16'h1400) && ((disp_spd-s_hundreds) < 16'h1E00) ) //20
	begin
		s10 <= 4'd2;
		s_tens <= 16'h1400;
	end
	else if( ((disp_spd-s_hundreds) >= 16'h0A00) && ((disp_spd-s_hundreds) < 16'h1400) ) //10
	begin
		s10 <= 4'd1;
		s_tens <= 16'h0A00;
	end
	else
```

```
begin
        s10 <= 4'd0;
        s_tens <= 16'h0000;
end


//now the s_ones
if( ((disp_spd-s_hundreds-s_tens) >= 16'h0900) && ((disp_spd-s_hundreds-s_tens) < 16'h0A00) ) //9
begin
        s1 <= 4'd9;
        s_ones <= 16'h0900;
end
else if( ((disp_spd-s_hundreds-s_tens) >= 16'h0800) && ((disp_spd-s_hundreds-s_tens) < 16'h0900) ) //8
begin
        s1 <= 4'd8;
        s_ones <= 16'h0800;
end
else if( ((disp_spd-s_hundreds-s_tens) >= 16'h0700) && ((disp_spd-s_hundreds-s_tens) < 16'h0800) ) //7
begin
        s1 <= 4'd7;
        s_ones <= 16'h0700;
end
else if( ((disp_spd-s_hundreds-s_tens) >= 16'h0600) && ((disp_spd-s_hundreds-s_tens) < 16'h0700) ) //6
begin
        s1 <= 4'd6;
        s_ones <= 16'h0600;
end
else if( ((disp_spd-s_hundreds-s_tens) >= 16'h0500) && ((disp_spd-s_hundreds-s_tens) < 16'h0600) ) //5
begin
        s1 <= 4'd5;
        s_ones <= 16'h0500;
end
else if( ((disp_spd-s_hundreds-s_tens) >= 16'h0400) && ((disp_spd-s_hundreds-s_tens) < 16'h0500) ) //4
begin
        s1 <= 4'd4;
        s_ones <= 16'h0400;
end
else if( ((disp_spd-s_hundreds-s_tens) >= 16'h0300) && ((disp_spd-s_hundreds-s_tens) < 16'h0400) ) //3
begin
        s1 <= 4'd3;
        s_ones <= 16'h0300;
end
else if( ((disp_spd-s_hundreds-s_tens) >= 16'h0200) && ((disp_spd-s_hundreds-s_tens) < 16'h0300) ) //2
begin
        s1 <= 4'd2;
        s_ones <= 16'h0200;
end
else if( ((disp_spd-s_hundreds-s_tens) >= 16'h0100) && ((disp_spd-s_hundreds-s_tens) < 16'h0200) ) //1
begin
        s1 <= 4'd1;
        s_ones <= 16'h0100;
end
else //0
begin
        s1 <= 4'd0;
        s_ones <= 16'h0000;
end


//now the s_tenths
if( ((disp_spd-s_hundreds-s_tens-s_ones) >= 16'h00E6) && ((disp_spd-s_hundreds-s_tens-s_ones) < 16'h0100) ) //.9
begin
```

```
			sp1 <= 4'd9;
			s_tenths <= 16'h00E6;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones) >= 16'h00CC) && ((disp_spd-s_hundreds-s_tens-s_ones) < 16'h00E6) )
//.8
		begin
			sp1 <= 4'd8;
			s_tenths <= 16'h00CC;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones) >= 16'h00B3) && ((disp_spd-s_hundreds-s_tens-s_ones) < 16'h00CC) )
//.7
		begin
			sp1 <= 4'd7;
			s_tenths <= 16'h00B3;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones) >= 16'h0099) && ((disp_spd-s_hundreds-s_tens-s_ones) < 16'h00B3) )
//.6
		begin
			sp1 <= 4'd6;
			s_tenths <= 16'h0099;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones) >= 16'h0080) && ((disp_spd-s_hundreds-s_tens-s_ones) < 16'h0099) )
//.5
		begin
			sp1 <= 4'd5;
			s_tenths <= 16'h0080;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones) >= 16'h0066) && ((disp_spd-s_hundreds-s_tens-s_ones) < 16'h0080) )
//.4
		begin
			sp1 <= 4'd4;
			s_tenths <= 16'h0066;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones) >= 16'h004C) && ((disp_spd-s_hundreds-s_tens-s_ones) < 16'h0066) )
//.3
		begin
			sp1 <= 4'd3;
			s_tenths <= 16'h004C;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones) >= 16'h0033) && ((disp_spd-s_hundreds-s_tens-s_ones) < 16'h004C) )
//.2
		begin
			sp1 <= 4'd2;
			s_tenths <= 16'h0033;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones) >= 16'h0019) && ((disp_spd-s_hundreds-s_tens-s_ones) < 16'h0033) )
//.1
		begin
			sp1 <= 4'd1;
			s_tenths <= 16'h0019;
		end
		else//.0
		begin
			sp1 <= 4'd0;
			s_tenths <= 16'h0000;
		end

		//finally the s_hundredths
		if( ((disp_spd-s_hundreds-s_tens-s_ones-s_tenths) >= 16'h0017) && ((disp_spd-s_hundreds-s_tens-s_ones-s_tenths) <
16'h0019) ) //.09
		begin
			sp01 <= 4'd9;
```

```
				//s_hundredths = 16'h0017;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones-s_tenths) >= 16'h0014) && ((disp_spd-s_hundreds-s_tens-s_ones-
s_tenths) < 16'h0017) ) //.08
		begin
				sp01 <= 4'd8;
				//s_hundredths = 16'h0014;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones-s_tenths) >= 16'h0011) && ((disp_spd-s_hundreds-s_tens-s_ones-
s_tenths) < 16'h0014) ) //.07
		begin
				sp01 <= 4'd7;
				//s_hundredths = 16'h0011;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones-s_tenths) >= 16'h000F) && ((disp_spd-s_hundreds-s_tens-s_ones-
s_tenths) < 16'h0011) ) //.06
		begin
				sp01 <= 4'd6;
				//s_hundredths = 16'h000F;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones-s_tenths) >= 16'h000C) && ((disp_spd-s_hundreds-s_tens-s_ones-
s_tenths) < 16'h000F) ) //.05
		begin
				sp01 <= 4'd5;
				//s_hundredths = 16'h000C;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones-s_tenths) >= 16'h000A) && ((disp_spd-s_hundreds-s_tens-s_ones-
s_tenths) < 16'h000C) ) //.04
		begin
				sp01 <= 4'd4;
				//s_hundredths = 16'h000A;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones-s_tenths) >= 16'h0007) && ((disp_spd-s_hundreds-s_tens-s_ones-
s_tenths) < 16'h000A) ) //.03
		begin
				sp01 <= 4'd3;
				//s_hundredths = 16'h0007;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones-s_tenths) >= 16'h0005) && ((disp_spd-s_hundreds-s_tens-s_ones-
s_tenths) < 16'h0007) ) //.02
		begin
				sp01 <= 4'd2;
				//s_hundredths = 16'h0005;
		end
		else if( ((disp_spd-s_hundreds-s_tens-s_ones-s_tenths) >= 16'h0002) && ((disp_spd-s_hundreds-s_tens-s_ones-
s_tenths) < 16'h0005) ) //.01
		begin
				sp01 <= 4'd1;
				//s_hundredths = 16'h0002;
		end
		else //.00
		begin
				sp01 <= 4'd0;
				//s_hundredths = 16'h0000;
		end
end

//setup program counter
reg [8:0] pc;
wire pause;
assign pause = 1'b0;
always @(posedge clk_1k or negedge reset_n)
```

```
begin
        if(!reset_n)
                pc = 9'd0;
        else if(reset_n && pause)
                pc = pc;
        else if(reset_n && !pause && ready)
        begin
                if(pc == 9'd323)
                        pc = 9'd48;
                else
                        pc = pc + 1;
        end
        else
                pc = pc;
end

always @(pc, type, num, r100, r10, r1, rp1, rp01, sign, s100, s10, s1, sp1, sp01)
begin
        case(pc)
                                //start function set
        0:                      begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //load data
                                        LCD_E = 1'b0;
                                end
        1:                      begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //e on
                                        LCD_E = 1'b1;
                                end
        2:                      begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011;
                                        LCD_E = 1'b0; //e-off
                                end
        3:                      begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //hold data
                                        LCD_E = 1'b0;
                                end
        4:                      begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //load data
                                        LCD_E = 1'b0;
                                end
        5:                      begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //e on
                                        LCD_E = 1'b1;
                                end
        6:                      begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011;
                                        LCD_E = 1'b0; //e-off
                                end
```

```
7:                              begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end function set
                                //start display on/off
8:                              begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //load data
                                        LCD_E = 1'b0;
                                end
9:                              begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //e on
                                        LCD_E = 1'b1;
                                end
10:                     begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011;
                                        LCD_E = 1'b0; //e-off
                                end
11:                     begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //hold data
                                        LCD_E = 1'b0;
                                end
12:                     begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //load data
                                        LCD_E = 1'b0;
                                end
13:                     begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //e on
                                        LCD_E = 1'b1;
                                end
14:                     begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010;
                                        LCD_E = 1'b0; //e-off
                                end
15:                     begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end display on/off
                                //start clear screen and return
16:                     begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //load data
                                        LCD_E = 1'b0;
                                end
        17:             begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //e on
                                        LCD_E = 1'b1;
                                end
        18:             begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010;
                                        LCD_E = 1'b0; //e-off
                                end
        19:             begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //hold data
                                        LCD_E = 1'b0;
                                end
        20:             begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1000; //load data
                                        LCD_E = 1'b0;
                                end
        21:             begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1000; //e on
                                        LCD_E = 1'b1;
                                end
        22:             begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1000;
                                        LCD_E = 1'b0; //e-off
                                end
        23:             begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1000; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end clear screen and return
                                //start entry mode set
        24:             begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //load data
                                        LCD_E = 1'b0;
                                end
        25:             begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //e on
                                        LCD_E = 1'b1;
                                end
        26:             begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000;
```

```
                        LCD_E = 1'b0; //e-off
                end
27:     begin
                        LCD_RS = 1'b0;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0000; //hold data
                        LCD_E = 1'b0;
                end
28:     begin
                        LCD_RS = 1'b0;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110; //load data
                        LCD_E = 1'b0;
                end
29:     begin
                        LCD_RS = 1'b0;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110; //e on
                        LCD_E = 1'b1;
                end
30:     begin
                        LCD_RS = 1'b0;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110;
                        LCD_E = 1'b0; //e-off
                end
31:     begin
                        LCD_RS = 1'b0;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110; //hold data
                        LCD_E = 1'b0;
                end
                //end entry mode set


                //a
32:     begin
                        LCD_RS = 1'b0;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0000; //load data
                        LCD_E = 1'b0;
                end
33:     begin
                        LCD_RS = 1'b0;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0000; //e on
                        LCD_E = 1'b1;
                end
34:     begin
                        LCD_RS = 1'b0;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0000;
                        LCD_E = 1'b0; //e-off
                end
35:     begin
                        LCD_RS = 1'b0;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0000; //hold data
                        LCD_E = 1'b0;
                end
36:     begin
                        LCD_RS = 1'b0;
```

```
                                LCD_RW = 1'b0;
                                LCD_OUT = 4'b1100; //load data
                                LCD_E = 1'b0;
                        end
        37:             begin
                                LCD_RS = 1'b0;
                                LCD_RW = 1'b0;
                                LCD_OUT = 4'b1100; //e on
                                LCD_E = 1'b1;
                        end
        38:             begin
                                LCD_RS = 1'b0;
                                LCD_RW = 1'b0;
                                LCD_OUT = 4'b1100;
                                LCD_E = 1'b0; //e-off
                        end
        39:             begin
                                LCD_RS = 1'b0;
                                LCD_RW = 1'b0;
                                LCD_OUT = 4'b1100; //hold data
                                LCD_E = 1'b0;
                        end
                        //end a
                        //start b
        40:             begin
                                LCD_RS = 1'b0;
                                LCD_RW = 1'b0;
                                LCD_OUT = 4'b0000; //load data
                                LCD_E = 1'b0;
                        end
        41:             begin
                                LCD_RS = 1'b0;
                                LCD_RW = 1'b0;
                                LCD_OUT = 4'b0000; //e on
                                LCD_E = 1'b1;
                        end
        42:             begin
                                LCD_RS = 1'b0;
                                LCD_RW = 1'b0;
                                LCD_OUT = 4'b0000;
                                LCD_E = 1'b0; //e-off
                        end
        43:             begin
                                LCD_RS = 1'b0;
                                LCD_RW = 1'b0;
                                LCD_OUT = 4'b0000; //hold data
                                LCD_E = 1'b0;
                        end
        44:             begin
                                LCD_RS = 1'b0;
                                LCD_RW = 1'b0;
                                LCD_OUT = 4'b0001; //load data
                                LCD_E = 1'b0;
                        end
        45:             begin
                                LCD_RS = 1'b0;
                                LCD_RW = 1'b0;
                                LCD_OUT = 4'b0001; //e on
                                LCD_E = 1'b1;
                        end
        46:             begin
                                LCD_RS = 1'b0;
```

```
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0001;
                                        LCD_E = 1'b0; //e-off
                                end
        47:             begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0001; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end b

                                //init done//////////////////////////////////////////

                                //start of line 1//
                                //start char 0
        48:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0100; //load data
                                        LCD_E = 1'b0;
                                end
        49:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0100; //e on
                                        LCD_E = 1'b1;
                                end
        50:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0100;
                                        LCD_E = 1'b0; //e-off
                                end
        51:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0100; //hold data
                                        LCD_E = 1'b0;
                                end
        52:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = type;//4'b0000; //load data
                                        LCD_E = 1'b0;
                                end
        53:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = type; //e on
                                        LCD_E = 1'b1;
                                end
        54:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = type;
                                        LCD_E = 1'b0; //e-off
                                end
        55:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = type; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end char 0
                                //start char 1
        56:     begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //load data
                                        LCD_E = 1'b0;
                                end
        57:     begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //e on
                                        LCD_E = 1'b1;
                                end
        58:     begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011;
                                        LCD_E = 1'b0; //e-off
                                end
        59:     begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //hold data
                                        LCD_E = 1'b0;
                                end
        60:     begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = num; //load data
                                        LCD_E = 1'b0;
                                end
        61:     begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = num; //e on
                                        LCD_E = 1'b1;
                                end
        62:     begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = num;
                                        LCD_E = 1'b0; //e-off
                                end
        63:     begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = num; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end char 1
                                //start char 2
        64:     begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //load data
                                        LCD_E = 1'b0;
                                end
        65:     begin
                                        LCD_RS = 1'b1;
```

```
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //e on
                                        LCD_E = 1'b1;
                                end
        66:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011;
                                        LCD_E = 1'b0; //e-off
                                end
        67:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //hold data
                                        LCD_E = 1'b0;
                                end
        68:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1010; //load data
                                        LCD_E = 1'b0;
                                end
        69:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1010; //e on
                                        LCD_E = 1'b1;
                                end
        70:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1010;
                                        LCD_E = 1'b0; //e-off
                                end
        71:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1010; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end char 2
                                //start char 3
        72:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //load data
                                        LCD_E = 1'b0;
                                end
        73:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //e on
                                        LCD_E = 1'b1;
                                end
        74:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010;
                                        LCD_E = 1'b0; //e-off
                                end
        75:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //hold data
                                        LCD_E = 1'b0;
                                end
        76:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //load data
                                        LCD_E = 1'b0;
                                end
        77:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //e on
                                        LCD_E = 1'b1;
                                end
        78:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000;
                                        LCD_E = 1'b0; //e-off
                                end
        79:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end char 3
                                //start char 4
        80:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //load data
                                        LCD_E = 1'b0;
                                end
        81:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //e on
                                        LCD_E = 1'b1;
                                end
        82:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010;
                                        LCD_E = 1'b0; //e-off
                                end
        83:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //hold data
                                        LCD_E = 1'b0;
                                end
        84:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //load data
                                        LCD_E = 1'b0;
                                end
        85:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //e on
                                        LCD_E = 1'b1;
                                end
        86:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000;
                                        LCD_E = 1'b0; //e-off
                                end
        87:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end char 4
                                //start char 5
        88:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0101; //load data
                                        LCD_E = 1'b0;
                                end
        89:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0101; //e on
                                        LCD_E = 1'b1;
                                end
        90:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0101;
                                        LCD_E = 1'b0; //e-off
                                end
        91:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0101; //hold data
                                        LCD_E = 1'b0;
                                end
        92:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //load data
                                        LCD_E = 1'b0;
                                end
        93:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //e on
                                        LCD_E = 1'b1;
                                end
        94:             begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010;
                                        LCD_E = 1'b0; //e-off
                                end
        95:             begin
                                        LCD_RS = 1'b1;
```

```
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0010; //hold data
                        LCD_E = 1'b0;
                end
                //end char 5
                //start char 6
    96:         begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0100; //load data
                        LCD_E = 1'b0;
                end
    97:         begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0100; //e on
                        LCD_E = 1'b1;
                end
    98:         begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0100;
                        LCD_E = 1'b0; //e-off
                end
    99:         begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0100; //hold data
                        LCD_E = 1'b0;
                end
    100:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1110; //load data
                        LCD_E = 1'b0;
                end
    101:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1110; //e on
                        LCD_E = 1'b1;
                end
    102:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1110;
                        LCD_E = 1'b0; //e-off
                end
    103:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1110; //hold data
                        LCD_E = 1'b0;
                end
                //end char 6
                //start char 7
    104:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0100; //load data
                        LCD_E = 1'b0;
                end
```

```
105:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0100; //e on
                        LCD_E = 1'b1;
                    end
106:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0100;
                        LCD_E = 1'b0; //e-off
                    end
107:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0100; //hold data
                        LCD_E = 1'b0;
                    end
108:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0111; //load data
                        LCD_E = 1'b0;
                    end
109:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0111; //e on
                        LCD_E = 1'b1;
                    end
110:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0111;
                        LCD_E = 1'b0; //e-off
                    end
111:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0111; //hold data
                        LCD_E = 1'b0;
                    end
                    //end char 7
                    //start char 8
112:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //load data
                        LCD_E = 1'b0;
                    end
113:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //e on
                        LCD_E = 1'b1;
                    end
114:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011;
                        LCD_E = 1'b0; //e-off
                    end
```

```
115:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //hold data
                        LCD_E = 1'b0;
                    end
116:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1101; //load data
                        LCD_E = 1'b0;
                    end
117:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1101; //e on
                        LCD_E = 1'b1;
                    end
118:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1101;
                        LCD_E = 1'b0; //e-off
                    end
119:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1101; //hold data
                        LCD_E = 1'b0;
                    end
                    //end char 8
                    //start char 9
120:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //load data
                        LCD_E = 1'b0;
                    end
121:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //e on
                        LCD_E = 1'b1;
                    end
122:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011;
                        LCD_E = 1'b0; //e-off
                    end
123:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //hold data
                        LCD_E = 1'b0;
                    end
124:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = r100; //load data
                        LCD_E = 1'b0;
                    end
```

```
125:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = r100; //e on
                        LCD_E = 1'b1;
                end
126:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = r100;
                        LCD_E = 1'b0; //e-off
                end
127:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = r100; //hold data
                        LCD_E = 1'b0;
                end
                //end char 9
                //start char 10
128:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //load data
                        LCD_E = 1'b0;
                end
129:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //e on
                        LCD_E = 1'b1;
                end
130:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011;
                        LCD_E = 1'b0; //e-off
                end
131:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //hold data
                        LCD_E = 1'b0;
                end
132:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = r10; //load data
                        LCD_E = 1'b0;
                end
133:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = r10; //e on
                        LCD_E = 1'b1;
                end
134:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = r10;
                        LCD_E = 1'b0; //e-off
                end
```

```
135:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = r10; //hold data
                        LCD_E = 1'b0;
                end
                //end char 10
                //start char 11
136:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //load data
                        LCD_E = 1'b0;
                end
137:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //e on
                        LCD_E = 1'b1;
                end
138:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011;
                        LCD_E = 1'b0; //e-off
                end
139:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //hold data
                        LCD_E = 1'b0;
                end
140:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = r1; //load data
                        LCD_E = 1'b0;
                end
141:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = r1; //e on
                        LCD_E = 1'b1;
                end
142:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = r1;
                        LCD_E = 1'b0; //e-off
                end
143:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = r1; //hold data
                        LCD_E = 1'b0;
                end
                //end char 11
                //start char 12
148:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0010; //load data
```

```
                                        LCD_E = 1'b0;
                                end
        149:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //e on
                                        LCD_E = 1'b1;
                                end
        150:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010;
                                        LCD_E = 1'b0; //e-off
                                end
        151:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //hold data
                                        LCD_E = 1'b0;
                                end
        152:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1110; //load data
                                        LCD_E = 1'b0;
                                end
        153:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1110; //e on
                                        LCD_E = 1'b1;
                                end
        154:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1110;
                                        LCD_E = 1'b0; //e-off
                                end
        155:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1110; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end char 12
                                //start char 13
        156:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //load data
                                        LCD_E = 1'b0;
                                end
        157:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //e on
                                        LCD_E = 1'b1;
                                end
        158:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011;
                                        LCD_E = 1'b0; //e-off
                                end
        159:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //hold data
                                        LCD_E = 1'b0;
                                end
        160:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = rp1; //load data
                                        LCD_E = 1'b0;
                                end
        161:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = rp1; //e on
                                        LCD_E = 1'b1;
                                end
        162:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = rp1;
                                        LCD_E = 1'b0; //e-off
                                end
        163:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = rp1; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end char 13
                                //start char 14
        164:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //load data
                                        LCD_E = 1'b0;
                                end
        165:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //e on
                                        LCD_E = 1'b1;
                                end
        166:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011;
                                        LCD_E = 1'b0; //e-off
                                end
        167:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //hold data
                                        LCD_E = 1'b0;
                                end
        168:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = rp01; //load data
                                        LCD_E = 1'b0;
                                end
        169:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = rp01; //e on
                                        LCD_E = 1'b1;
                                end
        170:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = rp01;
                                        LCD_E = 1'b0; //e-off
                                end
        171:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = rp01; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end char 14
                                //start char 15
        172:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0110; //load data
                                        LCD_E = 1'b0;
                                end
        173:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0110; //e on
                                        LCD_E = 1'b1;
                                end
        174:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0110;
                                        LCD_E = 1'b0; //e-off
                                end
        175:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0110; //hold data
                                        LCD_E = 1'b0;
                                end
        176:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1101; //load data
                                        LCD_E = 1'b0;
                                end
        177:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1101; //e on
                                        LCD_E = 1'b1;
                                end
        178:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1101;
                                        LCD_E = 1'b0; //e-off
                                end
        179:            begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1101; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end char 15
                                //end line 1/////////////

                                //start go to line 2
        180:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1100; //load data
                                        LCD_E = 1'b0;
                                end
        181:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1100; //e on
                                        LCD_E = 1'b1;
                                end
        182:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1100;
                                        LCD_E = 1'b0; //e-off
                                end
        183:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1100; //hold data
                                        LCD_E = 1'b0;
                                end
        184:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //load data
                                        LCD_E = 1'b0;
                                end
        185:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //e on
                                        LCD_E = 1'b1;
                                end
        186:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000;
                                        LCD_E = 1'b0; //e-off
                                end
        187:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end go to line 2
```

```
                    //start of line 2////////////////////////////////
                                                            //start char 0
188:        begin
                          LCD_RS = 1'b1;
                          LCD_RW = 1'b0;
                          LCD_OUT = 4'b0101; //load data
                          LCD_E = 1'b0;
                    end
189:        begin
                          LCD_RS = 1'b1;
                          LCD_RW = 1'b0;
                          LCD_OUT = 4'b0101; //e on
                          LCD_E = 1'b1;
                    end
190:        begin
                          LCD_RS = 1'b1;
                          LCD_RW = 1'b0;
                          LCD_OUT = 4'b0101;
                          LCD_E = 1'b0; //e-off
                    end
191:        begin
                          LCD_RS = 1'b1;
                          LCD_RW = 1'b0;
                          LCD_OUT = 4'b0101; //hold data
                          LCD_E = 1'b0;
                    end
192:        begin
                          LCD_RS = 1'b1;
                          LCD_RW = 1'b0;
                          LCD_OUT = 4'b0011; //load data
                          LCD_E = 1'b0;
                    end
193:        begin
                          LCD_RS = 1'b1;
                          LCD_RW = 1'b0;
                          LCD_OUT = 4'b0011; //e on
                          LCD_E = 1'b1;
                    end
194:        begin
                          LCD_RS = 1'b1;
                          LCD_RW = 1'b0;
                          LCD_OUT = 4'b0011;
                          LCD_E = 1'b0; //e-off
                    end
195:        begin
                          LCD_RS = 1'b1;
                          LCD_RW = 1'b0;
                          LCD_OUT = 4'b0011; //hold data
                          LCD_E = 1'b0;
                    end
                    //end char 0
                    //start char 1
196:        begin
                          LCD_RS = 1'b1;
                          LCD_RW = 1'b0;
                          LCD_OUT = 4'b0101; //load data
                          LCD_E = 1'b0;
                    end
197:        begin
                          LCD_RS = 1'b1;
                          LCD_RW = 1'b0;
                          LCD_OUT = 4'b0101; //e on
```

```
                        LCD_E = 1'b1;
                end
198:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0101;
                        LCD_E = 1'b0; //e-off
                end
199:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0101; //hold data
                        LCD_E = 1'b0;
                end
200:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0000; //load data
                        LCD_E = 1'b0;
                end
201:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0000; //e on
                        LCD_E = 1'b1;
                end
202:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0000;
                        LCD_E = 1'b0; //e-off
                end
203:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0000; //hold data
                        LCD_E = 1'b0;
                end
                //end char 1
                //start char 2
204:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0100; //load data
                        LCD_E = 1'b0;
                end
205:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0100; //e on
                        LCD_E = 1'b1;
                end
206:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0100;
                        LCD_E = 1'b0; //e-off
                end
207:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0100; //hold data
```

```
                                        LCD_E = 1'b0;
                                end
        208:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0100; //load data
                                        LCD_E = 1'b0;
                                end
        209:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0100; //e on
                                        LCD_E = 1'b1;
                                end
        210:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0100;
                                        LCD_E = 1'b0; //e-off
                                end
        211:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0100; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end char 2
                                //start char 3
        212:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //load data
                                        LCD_E = 1'b0;
                                end
        213:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //e on
                                        LCD_E = 1'b1;
                                end
        214:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011;
                                        LCD_E = 1'b0; //e-off
                                end
        215:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0011; //hold data
                                        LCD_E = 1'b0;
                                end
        216:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1101; //load data
                                        LCD_E = 1'b0;
                                end
        217:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1101; //e on
                                        LCD_E = 1'b1;
                                end
        218:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1101;
                                        LCD_E = 1'b0; //e-off
                                end
        219:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1101; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end char 3
                                //start char 4
        220:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //load data
                                        LCD_E = 1'b0;
                                end
        221:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //e on
                                        LCD_E = 1'b1;
                                end
        222:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010;
                                        LCD_E = 1'b0; //e-off
                                end
        223:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0010; //hold data
                                        LCD_E = 1'b0;
                                end
        224:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = sign; //load data
                                        LCD_E = 1'b0;
                                end
        225:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = sign; //e on
                                        LCD_E = 1'b1;
                                end
        226:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = sign;
                                        LCD_E = 1'b0; //e-off
                                end
        227:                    begin
                                        LCD_RS = 1'b1;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = sign; //hold data
```

```
                        LCD_E = 1'b0;
                end
                //end char 4
                //start char 5
228:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //load data
                        LCD_E = 1'b0;
                end
229:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //e on
                        LCD_E = 1'b1;
                end
230:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011;
                        LCD_E = 1'b0; //e-off
                end
231:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //hold data
                        LCD_E = 1'b0;
                end
232:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = s100; //load data
                        LCD_E = 1'b0;
                end
233:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = s100; //e on
                        LCD_E = 1'b1;
                end
234:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = s100;
                        LCD_E = 1'b0; //e-off
                end
235:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = s100; //hold data
                        LCD_E = 1'b0;
                end
                //end char 5
                //start char 6
236:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //load data
                        LCD_E = 1'b0;
                end
237:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //e on
                        LCD_E = 1'b1;
                end
238:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011;
                        LCD_E = 1'b0; //e-off
                end
239:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //hold data
                        LCD_E = 1'b0;
                end
240:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = s10; //load data
                        LCD_E = 1'b0;
                end
241:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = s10; //e on
                        LCD_E = 1'b1;
                end
242:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = s10;
                        LCD_E = 1'b0; //e-off
                end
243:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = s10; //hold data
                        LCD_E = 1'b0;
                end
                //end char 6
                //start char 7
244:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //load data
                        LCD_E = 1'b0;
                end
245:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //e on
                        LCD_E = 1'b1;
                end
246:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011;
                        LCD_E = 1'b0; //e-off
                end
247:        begin
                        LCD_RS = 1'b1;
```

```
                    LCD_RW = 1'b0;
                    LCD_OUT = 4'b0011; //hold data
                    LCD_E = 1'b0;
                end
248:        begin
                    LCD_RS = 1'b1;
                    LCD_RW = 1'b0;
                    LCD_OUT = s1; //load data
                    LCD_E = 1'b0;
                end
249:        begin
                    LCD_RS = 1'b1;
                    LCD_RW = 1'b0;
                    LCD_OUT = s1; //e on
                    LCD_E = 1'b1;
                end
250:        begin
                    LCD_RS = 1'b1;
                    LCD_RW = 1'b0;
                    LCD_OUT = s1;
                    LCD_E = 1'b0; //e-off
                end
251:        begin
                    LCD_RS = 1'b1;
                    LCD_RW = 1'b0;
                    LCD_OUT = s1; //hold data
                    LCD_E = 1'b0;
                end
                //end char 7
                //start char 8
252:        begin
                    LCD_RS = 1'b1;
                    LCD_RW = 1'b0;
                    LCD_OUT = 4'b0010; //load data
                    LCD_E = 1'b0;
                end
253:        begin
                    LCD_RS = 1'b1;
                    LCD_RW = 1'b0;
                    LCD_OUT = 4'b0010; //e on
                    LCD_E = 1'b1;
                end
254:        begin
                    LCD_RS = 1'b1;
                    LCD_RW = 1'b0;
                    LCD_OUT = 4'b0010;
                    LCD_E = 1'b0; //e-off
                end
255:        begin
                    LCD_RS = 1'b1;
                    LCD_RW = 1'b0;
                    LCD_OUT = 4'b0010; //hold data
                    LCD_E = 1'b0;
                end
256:        begin
                    LCD_RS = 1'b1;
                    LCD_RW = 1'b0;
                    LCD_OUT = 4'b1110; //load data
                    LCD_E = 1'b0;
                end
257:        begin
                    LCD_RS = 1'b1;
```

```
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1110; //e on
                        LCD_E = 1'b1;
                end
258:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1110;
                        LCD_E = 1'b0; //e-off
                end
259:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1110; //hold data
                        LCD_E = 1'b0;
                end
                //end char 8
                //start char 9
260:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //load data
                        LCD_E = 1'b0;
                end
261:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //e on
                        LCD_E = 1'b1;
                end
262:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011;
                        LCD_E = 1'b0; //e-off
                end
263:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0011; //hold data
                        LCD_E = 1'b0;
                end
264:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = sp1; //load data
                        LCD_E = 1'b0;
                end
265:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = sp1; //e on
                        LCD_E = 1'b1;
                end
266:        begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = sp1;
                        LCD_E = 1'b0; //e-off
                end
267:        begin
                        LCD_RS = 1'b1;
```

```
				LCD_RW = 1'b0;
				LCD_OUT = sp1; //hold data
				LCD_E = 1'b0;
			end
			//end char 9
			//start char 10
	268:		begin
				LCD_RS = 1'b1;
				LCD_RW = 1'b0;
				LCD_OUT = 4'b0011; //load data
				LCD_E = 1'b0;
			end
	269:		begin
				LCD_RS = 1'b1;
				LCD_RW = 1'b0;
				LCD_OUT = 4'b0011; //e on
				LCD_E = 1'b1;
			end
	270:		begin
				LCD_RS = 1'b1;
				LCD_RW = 1'b0;
				LCD_OUT = 4'b0011;
				LCD_E = 1'b0; //e-off
			end
	271:		begin
				LCD_RS = 1'b1;
				LCD_RW = 1'b0;
				LCD_OUT = 4'b0011; //hold data
				LCD_E = 1'b0;
			end
	272:		begin
				LCD_RS = 1'b1;
				LCD_RW = 1'b0;
				LCD_OUT = sp01; //load data
				LCD_E = 1'b0;
			end
	273:		begin
				LCD_RS = 1'b1;
				LCD_RW = 1'b0;
				LCD_OUT = sp01; //e on
				LCD_E = 1'b1;
			end
	274:		begin
				LCD_RS = 1'b1;
				LCD_RW = 1'b0;
				LCD_OUT = sp01;
				LCD_E = 1'b0; //e-off
			end
	275:		begin
				LCD_RS = 1'b1;
				LCD_RW = 1'b0;
				LCD_OUT = sp01; //hold data
				LCD_E = 1'b0;
			end
			//end char 10
			//start char 11
	276:		begin
				LCD_RS = 1'b1;
				LCD_RW = 1'b0;
				LCD_OUT = 4'b0010; //load data
				LCD_E = 1'b0;
			end
```

```
277:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0010; //e on
                        LCD_E = 1'b1;
                    end
278:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0010;
                        LCD_E = 1'b0; //e-off
                    end
279:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0010; //hold data
                        LCD_E = 1'b0;
                    end
280:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0000; //load data
                        LCD_E = 1'b0;
                    end
281:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0000; //e on
                        LCD_E = 1'b1;
                    end
282:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0000;
                        LCD_E = 1'b0; //e-off
                    end
283:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0000; //hold data
                        LCD_E = 1'b0;
                    end
                    //end char 11
                    //start char 12
284:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110; //load data
                        LCD_E = 1'b0;
                    end
285:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110; //e on
                        LCD_E = 1'b1;
                    end
286:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110;
                        LCD_E = 1'b0; //e-off
                    end
```

```
287:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110; //hold data
                        LCD_E = 1'b0;
                end
288:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1011; //load data
                        LCD_E = 1'b0;
                end
289:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1011; //e on
                        LCD_E = 1'b1;
                end
290:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1011;
                        LCD_E = 1'b0; //e-off
                end
291:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1011; //hold data
                        LCD_E = 1'b0;
                end
                //end char 12
                //start char 13
292:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110; //load data
                        LCD_E = 1'b0;
                end
293:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110; //e on
                        LCD_E = 1'b1;
                end
294:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110;
                        LCD_E = 1'b0; //e-off
                end
295:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110; //hold data
                        LCD_E = 1'b0;
                end
296:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1101; //load data
                        LCD_E = 1'b0;
                end
```

```
297:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1101; //e on
                        LCD_E = 1'b1;
                end
298:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1101;
                        LCD_E = 1'b0; //e-off
                end
299:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1101; //hold data
                        LCD_E = 1'b0;
                end
                //end char 13
                //start char 14
300:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0010; //load data
                        LCD_E = 1'b0;
                end
301:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0010; //e on
                        LCD_E = 1'b1;
                end
302:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0010;
                        LCD_E = 1'b0; //e-off
                end
303:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0010; //hold data
                        LCD_E = 1'b0;
                end
304:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1111; //load data
                        LCD_E = 1'b0;
                end
305:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1111; //e on
                        LCD_E = 1'b1;
                end
306:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1111;
                        LCD_E = 1'b0; //e-off
                end
```

```
307:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1111; //hold data
                        LCD_E = 1'b0;
                end
                //end char 14
                //start char 15
308:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110; //load data
                        LCD_E = 1'b0;
                end
309:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110; //e on
                        LCD_E = 1'b1;
                end
310:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110;
                        LCD_E = 1'b0; //e-off
                end
311:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b0110; //hold data
                        LCD_E = 1'b0;
                end
312:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1000; //load data
                        LCD_E = 1'b0;
                end
313:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1000; //e on
                        LCD_E = 1'b1;
                end
314:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1000;
                        LCD_E = 1'b0; //e-off
                end
315:            begin
                        LCD_RS = 1'b1;
                        LCD_RW = 1'b0;
                        LCD_OUT = 4'b1000; //hold data
                        LCD_E = 1'b0;
                end
                //end char 15
                //end of line 2

                //start go to line 1
316:            begin
                        LCD_RS = 1'b0;
```

```
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1000; //load data
                                        LCD_E = 1'b0;
                                end
        317:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1000; //e on
                                        LCD_E = 1'b1;
                                end
        318:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1000;
                                        LCD_E = 1'b0; //e-off
                                end
        319:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b1000; //hold data
                                        LCD_E = 1'b0;
                                end
        320:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //load data
                                        LCD_E = 1'b0;
                                end
        321:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //e on
                                        LCD_E = 1'b1;
                                end
        322:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000;
                                        LCD_E = 1'b0; //e-off
                                end
        323:            begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000; //hold data
                                        LCD_E = 1'b0;
                                end
                                //end go to line 1


        default:    begin
                                        LCD_RS = 1'b0;
                                        LCD_RW = 1'b0;
                                        LCD_OUT = 4'b0000;
                                        LCD_E = 1'b0;
                                end
        endcase
end

endmodule
```

top

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: The University of Windsor
// Engineer (in training): Karl Leboeuf
//
// Create Date:    10:30:19 08/12/2011
// Design Name:
// Module Name:    top
// Project Name: FMCW radar signal processing platform
// Target Devices: ML506 development board, MAX5885 EV DAC board, EVAL-AD7x76 ADC
//               board; essentially the Xilinx Virtex 5, Maxim 5885 DAC, and
//               Analog Devices 7276 ADC.
// Tool versions: Xilinx ISE 13.2
// Description: This design generates the beam switch control signals, DAC control,
//            and captures data serially from the ADC.  ADC data is windowed, run
//            through an FFT, and analyzed to measure target velocity and range.
//            This design supports three beams with exactly one target per beam.

//
// Dependencies: MATLAB code is used to generate a coefficients file for the
//                                                  window ROM; this 'coeffs.coe' file must be used by the core
//             generator to setup the correct coefficients.  Different windows
//                                                  can be used by changing these coefficients (as long as the window
//             is symmetric).
//
// Revision 1.00 - (22/09/2011) Hardware tested and working with ADC board, unable
//                to test with DAC board because the sub-1kHz sweep signal that
//                must be generated cannot be measured on the DAC board's voltage
//                output due to its transformer acting as a low-pass filter.
// Revision 0.90 - (01/09/2011) Design passes post-place-and-route simulation.
// Revision 0.01 - File Created
// Additional Comments: This version uses a 2048 point FFT and the Hamming window.
//                  Each beam has a cycle time of 2.048 ms, making for a total
//                  system cycle time of 3 x 2.048 = 6.144 ms.  Data capture
//                  and data analysis is overlapped.  This design uses a
//                  100 MHz clock, however it may be possible to use a much
//                  slower clock if desired.  Finally, it is important to note
//                  that the FFT does not make use of any scaling; full
//                  precision is preserved from the FFT on out.  If some
//                  information regarding the signal's likely dynamic range is
//                  available this design could be scaled down radically by
//                  shortening many multipliers and registers from 24/48 bits
//                  down to perhaps as little as 8 or 10.
//
//////////////////////////////////////////////////////////////////////////////////
module top(
            input EXT_CLK,
            input EXT_RESET,
            input [2:0] BEAM_SHOW,
            input ADC_MISO,
            output [2:0] BEAM_SWITCH,
            output STATUS_OK, //status red LED; RED is GOOD in this case
            output [7:0] ADC_LED,
            output [3:0] LCD_OUT,
            output LCD_E,
            output LCD_RS,
            output LCD_RW,
            output [15:0] DAC_DATA,
            output DAC_CLK,
            output ADC_SCLK,
            output ADC_CS
            );
```

```
	//clock signals
	wire clk_100;		//100 MHz clock, this is the buffered version of EXT_CLK
	wire clk_64;		//64 MHz clock; output of DCM
	wire clk_64_inv; //inverted 64 MHz clock; output of DCM
	wire clk_48;		//48 MHz clock; output of DCM

	//reset signals
	wire reset;
	wire secondary_reset; //this resets the circuits driven by the 64/48 MHz clocks; must be triggered after dcm lock
established
	wire reset_48; //reset signal synchronized to 48 MHz clock domain
	wire reset_64;

	//status signals
	wire dcm0_lock;
	wire dcm1_lock;
	wire dcm_ready;
	wire fifo_ready; //fifo ready comes on 256 clock cycles after dcm_ready
	wire ready_48; //fifo ready synced to 48 MHz clock
	wire ready_64;

	//other signals
	wire [11:0] adc_sample;
	wire adc_valid;
	wire [11:0] window_sample;
	wire [23:0] xk_re, xk_im;
	wire [10:0] xk_index;
	wire dv;
	wire [47:0] mag_sq;
	wire dv_delayed;
	wire max_valid;
	wire [19:0] max_bin; // [** CHANGED TO 16 BITS FROM 10 BITS]

	wire [12:0] target_abs;
	wire [15:0] target_pos;			//changed to 10 bit from 16
	wire [7:0] unit_vel;
	wire [19:0] target_info;
	wire info_valid;
	wire valid_in;


	// [**** changed to 29bits from 16 bits ]
	wire [28:0] b0t_range, b1t_range, b2t_range, b0t_speed, b1t_speed, b2t_speed;
	wire b0t_dir, b1t_dir, b2t_dir;

	//reset signal assignments
	assign reset = ~EXT_RESET; //my reset button is active-low, remove '~' if this is no longer the case
	assign secondary_reset = dcm_ready & ~fifo_ready; //secondary reset pulse is on while the fifo is resetting

	//ready signal assignemtns
	assign dcm_ready = dcm0_lock & dcm1_lock & ~reset;

	//output assignments
	assign STATUS_OK = fifo_ready;
	assign DAC_CLK = clk_64_inv;
	assign ADC_SCLK = clk_48;
	assign ADC_LED = adc_sample[11:4];


	//buffer the external clock input before sending it to the DCMs
	//this is also the source of our 100 MHz system clock
```

```
//note that this clock is not reset when the DCMs are reset
IBUFG IBUFG_CLKIN (
.O(clk_100),  // 1-bit output Clock buffer output
.I(EXT_CLK) // 1-bit input Clock buffer input
);

//setup the 48 MHz clock for the ADC serial link
clock_100_48 u0_clock_100_48 (
.CLKIN_IN(clk_100),
.RST_IN(reset),
.CLKFX_OUT(clk_48),
.CLK0_OUT(),
.LOCKED_OUT(dcm0_lock)
);

//setup the 64 MHz clock for the DAC interface
clock_100_64 u0_clock_100_64 (
.CLKIN_IN(clk_100),
.RST_IN(reset),
.CLKFX_OUT(clk_64),
.CLKFX180_OUT(clk_64_inv),
.CLK0_OUT(),
.LOCKED_OUT(dcm1_lock)
);

//syncrhonizer circuit takes the reset signal from the 100-MHz clock domain, and
//synchronizes it to the 48 MHz domain; this is to avoid any metastability problems
synchronizer u0_reset_48_synchronizer (
.clk_in(clk_100),
.clk_out(clk_48),
.di(secondary_reset),
.dout(reset_48)
);

//synchronizer for the 48 MHz ready signal
synchronizer u0_fifo_ready_48_synchronizer (
.clk_in(clk_100),
.clk_out(clk_48),
.di(fifo_ready),
.dout(ready_48)
);

//syncrhonizer circuit takes the reset signal from the 100-MHz clock domain, and
//synchronizes it to the 64 MHz domain; this is to avoid any metastability problems
synchronizer u0_reset_64_synchronizer (
.clk_in(clk_100),
.clk_out(clk_64_inv),
.di(secondary_reset),
.dout(reset_64)
);

//synchronizer for the 64 MHz ready signal
synchronizer u0_fifo_ready_64_synchronizer (
.clk_in(clk_100),
.clk_out(clk_64_inv),
.di(fifo_ready),
.dout(ready_64)
);

//instantiate the fft
compute_fft u0_fft (
.adc_valid(adc_valid),
		.clk_100(clk_100),
		.clk_48(clk_48),
		.window_sample(window_sample),
		.reset(reset),
		.dcm_ready(dcm_ready),
		.fifo_ready(fifo_ready),
		.xk_index(xk_index),
		.xk_re(xk_re),
		.xk_im(xk_im),
		.dv(dv)
		);

		//this module computes the magnitude squared output of the fft
		compute_mag_sq u0_compute_mag_sq(
		.clk_100(clk_100),
		.xk_re(xk_re),
		.xk_im(xk_im),
		.xk_index(xk_index),
		.dv(dv),
		.mag_sq(mag_sq),
		.dv_delayed(dv_delayed)
		);


		//instantiate the CA_CFAR
		 ca_cfar u0_cfar (
 .clk(clk_100),
		 .reset(reset),
		 .targA(mag_sq), // inA,inB, inC, inD are obtained from 4 different sqrt modules
		 .targB(mag_sq),
		 .targC(mag_sq),
		 .targD(mag_sq),
		 .start(dv_delayed),
		 .target_abs(target_abs),			// new target peak intensity
		 .target_pos(target_pos),			// new target frequency bin number
		 .new_target(new_target),
		 .start_cfar(start_cfar),
		 .complete(complete)
		 );

		 //instantiate the Peak_pairing procedure
		 pairing u0_pairing(
 .clk(clk_100),
		 .reset(reset),
		 .new_target(new_target), // new target detected signal from CACFAR_32 module
		 .target_abs(target_abs), // new target peak intensity
		 .target_pos(target_pos), // new target frequency bin number
		 .complete(complete), // CFAR completion signal
		 .updown(updown), // updown = 1(0) during up(down) sweep sampling i.e. down(up) sweep processing
		 .unit_vel(unit_vel), // vehicle velocity
		 .target_info(target_info), // MSB -> 10 bits velocity, 10 bits range <- LSB
		 .info_valid(info_valid) // target information valid signal to display unit
		 );

		//this module computes the range and velocity information for each beam
		compute_target_info u0_compute_target_info (
		.clk_100(clk_100),
		.reset(reset),
		.max_bin(target_info),
		.valid_in(info_valid),
		.b0t_range(b0t_range),
		.b1t_range(b1t_range),
.b2t_range(b2t_range),
.b0t_speed(b0t_speed),
.b1t_speed(b1t_speed),
.b2t_speed(b2t_speed),
.b0t_dir(b0t_dir),
.b1t_dir(b1t_dir),
.b2t_dir(b2t_dir)
);

//instantiate the adc controller
adc_control u0_adc (
.clk_48(clk_48),
.ADC_MISO(ADC_MISO),
.reset(reset_48), //circuit has a synchronizer to bring the reset signal into its clock domain
.ready(ready_48),
.ADC_CS(ADC_CS),
.adc_sample(adc_sample),
.adc_valid(adc_valid)
);

//instantiate the dac controller
dac_control u0_dac (
.clk_64(clk_64_inv), //inverted clock because the dac latches data on the rising edge
.ready(ready_64),
.reset(reset_64),
.DAC_DATA(DAC_DATA)
);

//instantiate the windowing circuit
compute_window u0_window (
.adc_valid(adc_valid),
.clk_48(clk_48),
.adc_sample(adc_sample),
.reset(reset_48),  //circuit has a synchronizer to bring the reset signal into its clock domain
.ready(ready_48),
.window_sample(window_sample)
);

//instantiate the beam select controller
beam_switch          u0_beam_switch (
.clk_48(clk_48),
.reset(reset_48),
.ready(ready_48),
.BEAM_SWITCH(BEAM_SWITCH)
);

//instantiate lcd driver
lcd_driver u0_lcd (
.clk_100(clk_100),
.reset(reset),
.ready(fifo_ready),
.BEAM_SHOW(BEAM_SHOW),
.b0t_range(b0t_range),
.b1t_range(b1t_range),
.b2t_range(b2t_range),
.b0t_speed(b0t_speed),
.b0t_dir(b0t_dir),
.b1t_speed(b1t_speed),
.b1t_dir(b1t_dir),
.b2t_speed(b2t_speed),
.b2t_dir(b2t_dir),
.LCD_E(LCD_E),
        .LCD_RS(LCD_RS),
        .LCD_RW(LCD_RW),
        .LCD_OUT(LCD_OUT)
        );

endmodule
```

VITA AUCTORIS

Sabrina Zereen was born in 1985 in Dhaka, Bangladesh. She passed her O' Levels and A' Levels from Maple Leaf International School and worked there for a year as a teacher. She completed her Bachelors of Science in Electrical and Electronic Engineering from American International University-Bangladesh (AIUB) in 2008. She was employed as a lecturer in AIUB for two years before she applied for her Masters in Applied Science at the University of Windsor. Her research interests include Field programmable logic, Electronics, MicroelctromechanicalSystems (MEMS) and Digital signal processing. Sabrina is a member of the MEMS Research Group, and a candidate for the degree of M. A. Sc. in Electrical and Computer Engineering, under the supervision of Dr. Sazzadur Chowdhury, at the University of Windsor (Ontario, Canada).