Developing a Radar Signal Processor for the Detection of Targets

A Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Aswin Raj K
(121801008)

under the guidance of

Dr. Swaroop Sahoo & Dr. Subrahmanyam Mula

INDIAN INSTITUTE
OF TECHNOLOGY
PALAKKAD

DEPARTMENT OF ELECTRICAL ENGINEERING

CERTIFICATE

*This is to certify that the work contained in this report entitled "**Developing a Radar Signal Processor for the Detection of Targets**" is a bonafide work of **Aswin Raj K (Roll No. 121801008)**, carried out in the Department of Electrical Engineering, Indian Institute of Technology Palakkad.*

Dr. Swaroop Sahoo & Dr. Subrahmanyam Mula

Assistant Professor

Department of Electrical Engineering

Indian Institute of Technology Palakkad

Acknowledgements

I would like to express my sincere thanks and gratitude to my mentor Dr. Swaroop Sahoo for letting me work on this project. I am very grateful to him for his support and guidance throughout the project. I also thank my co-mentor Dr. Subrahmanyam Mula for clarifying my doubts.

I was able to successfully complete this project with the help of their guidance and support. Finally, I want to thank all my dear friends as well.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Radars are an important component of any aerospace surveillance system. The working of radar is essentially based on the electromagnetic principle of backscattering at radio frequencies, which occurs when an object is illuminated by the radar beam. The processors and algorithms should not give a false alarm for birds and clutter. Considering these the signal processors and algorithms have to be designed for detecting the changes due to the micro-Doppler effect.

FPGA is extensively used for high computing applications. It provides us with the flexibility of massive parallel data processing, better performance, efficiency, and faster prototyping. It can be configured to run several digital signal processing algorithms within in short period. FPGA, when compared to equivalent discrete circuits, occupies less space. All these capabilities of FPGA make it the perfect platform for the implementation of a radar signal processor. In traditional radar signal processors, slow processing and low resolution have limited it's working. The implementation of a radar signal processor on an FPGA has increased the computation speed together with the improvement in the accuracy and precision of the results.

The signal processing part is designed to be implemented on an FPGA (ZC702) board. It is designed to produce pulses with monotonous frequency as well as chirp[1, p.420]. The required RF pulse is generated at baseband that will be upconverted to X-band on the SDR and transmitted out. The FPGA is programmed to store the sampled data coming from the SDR in the form of a data matrix. The data stored in the data matrix is used to extract the doppler [2, p.92] and range information. Before its implementation onto an actual FPGA board the radar is simulated using MATLAB to find the best parameters required for the radar to operate.The implementation of Hamming window [4], FFT algorithm , Double delay-line canceler [1, p.107], and Matched filtering [3, p.161] is thoroughly explained in this report. Finally the FPGA simulation result is compared with the reference done in MATLAB for verification.

1.1 Objectives

Project Objectives

① **Simulating the radar using Matlab.**

- Before implementing the radar onto an actual FPGA, we need to find the right parameters to ensure it is working as expected.
- A matlab simulation of the radar model is done to finalize the required parameters which meets our need.

② **Implementing the transmitter on the FPGA.**

- Chirp signal generator is to be implemented on the FPGA in digital domain.
- This generated digital signal needs to be sent to the SDR for transmitting it through the antenna.

③ **Ensuring that the chirp pulses transmitted from the FPGA is as expected.**

- This can be done using another receiver and observing the transmitted signal transmitted from the FPGA.

④ **Implementing the receiver on the FPGA.**

- Storing the echo data received by the SDR onto the memory block of FPGA.
- Processing the digitized data on the FPGA itself.
- Blocks for Hamming window, 3 pulse canceler, Matched filtering, Threshold detection, and FFT is to be designed for processing the data.

(5) **Exporting the processed data for visualization.**

- Export the final processed data from the FPGA to matlab for easy visualization.
- Plotting the result in matlab to ensure that the radar is working as expected.

1.2 Organization of The Report

You can write the about organization of your report in the following manner.

This chapter provides a background for the topics covered in this report. We provided a description of radar and their applications. This chapter gave a brief overview of the objectives. The rest of the chapters are organised as follows: next chapter we provide a overview about the working of radar. In Chapter 3 we discuss about the how the radar is simulated in matlab prior to its implementation on hardware. In Chapter 4 we discuss about the how the radar processor is implemented on the hardware. Chapter 5 discusses about configuring the micro processor on the board to send data to the designed FPGA module. In chapter 6 discusses the comparison of the results obtained through matlab simulation and hardware implementation. And finally in chapter 7, we conclude with some future works.

Chapter 2

Radar Working

2.1 Radar Working

The radar sends RF pulses at regular intervals and these pulses get reflected by the target. After transmitting the pulse, the radar switches to a listening mode where it listens for the reflected signal from the target. The radar will down-convert the received signal to baseband, sample it and store the IQ data in the form of a data matrix. Each row in the data matrix corresponds to radar returns from each pulse and each column corresponds to the sampled data (or Range Bin). A 4×10 Data Matrix is as shown in figure 2.1.

In figure 2.1 the orange and green bin refers to two different targets. From figure 2.1 it is clear that the orange target lies in the same range bin for each pulse interval whereas the green target is changing the range bin which indicates that the later is moving whereas the former one is stationary. Each range bin corresponds to a particular distance from the radar which depends on the sampling frequency of the received pulses. If the magnitude of the sampled data goes above a particular threshold we can conclude that the target is located at that range from the radar which corresponds to that particular range bin. Thus we can easily calculate the range information of the target from the data matrix and now to get the velocity of the target we calculate the doppler shift using fast fourier transform. We take FFT along the slow time samples so that we can get the frequencies corresponding to every range bin. The newly created matrix after taking FFT is as shown below in figure 2.2.

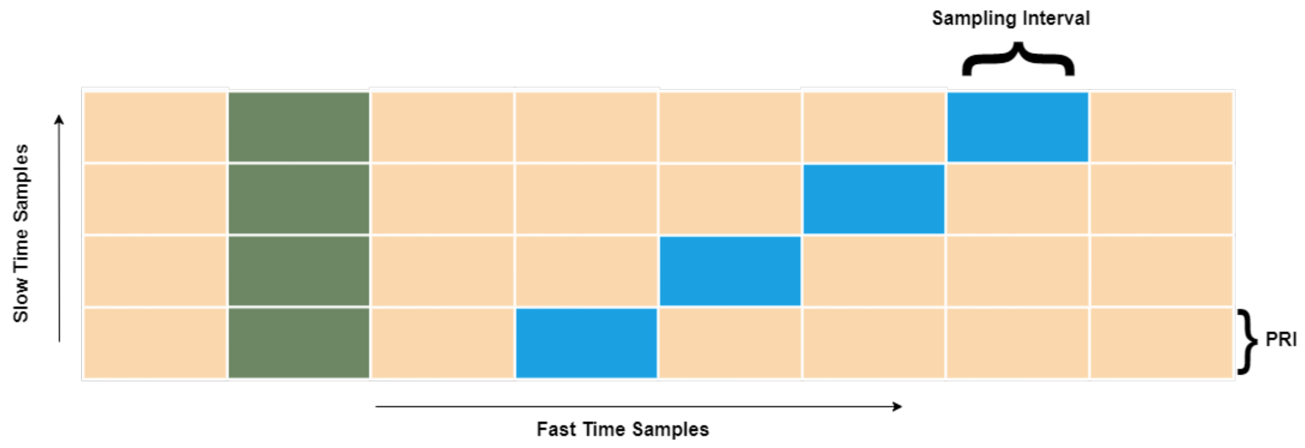


Fig. 2.1 Data matrix with fast time samples along the row and slow time samples along the columns.

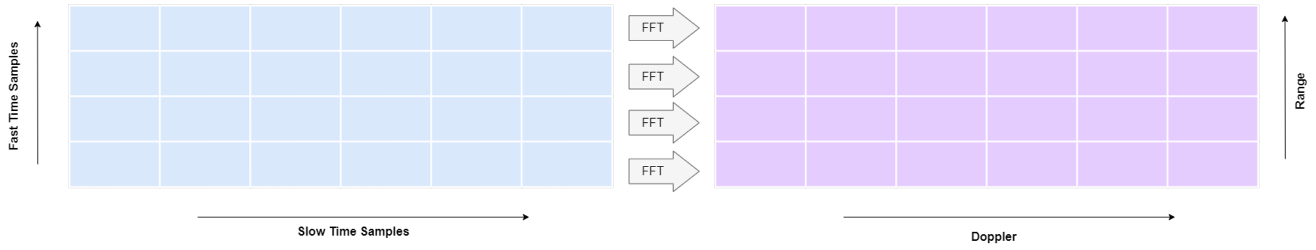


Fig. 2.2 Taking FFT along Slow Time samples to produce another matrix which has both Range and Doppler Information

2.2 Radar Designing

The first step in designing the radar system is determining the pulse width and pulse repetition frequency (PRF) of the pulse to be transmitted. The next step is that pulse width determines the resolution [2, p. 21] of the radar while pulse repetition frequency determines the range of detection. The resolution of radar is its ability to distinguish between targets that are in very close proximity. It is given by the equation

$$\rho = \frac{cT}{2} \tag{2.1}$$

where c is the speed of light and T is the pulse width.

A typical radar pulse is as shown in figure 2.3. Each pulse contains only a single frequency component. Such a pulse won't give us the required performance. Thus, instead of sending normal pulses at regular intervals, chirp signal [1, p. 420] is used.

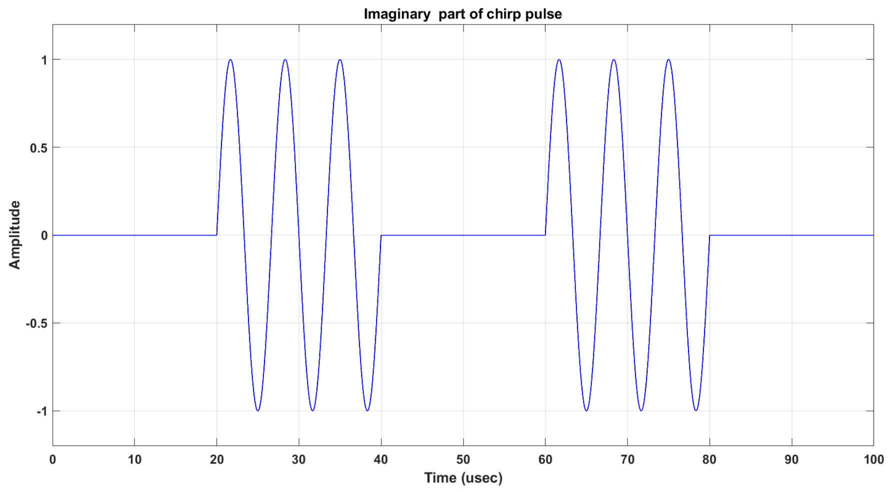


Fig. 2.3 Normal pulse

Chirp is a signal in which the frequency is varied. It is a pulse compression [1, p. 420] technique which allows a radar to radiate a large amount of energy but can simultaneously obtain the range resolution [2, p. 21] of a small pulse. Long pulse gives more range whereas the chirp signal within the pulse allows achieving range resolution of a small pulse. Equation 2.2 gives the value of the chirp signal for the nth sample.

$$S(n) = e^{\frac{j\pi\omega}{T}\left(nT_s \frac{(N-1)f_s}{2}\right)^2} \tag{2.2}$$

Chapter 3

Matlab Simulation

The entire radar is first simulated using matlab. Matlab simulation can be split into mainly two parts which are,

- Creating the received echo signal with some targets positioned at a particular range and velocity relative to the radar.
- Extraction of the target range and velocity from this unprocessed (echo) data.

The parameters used in the matlab simulation are given in table 3.1.

Parameter	**Value**
Pulse width (T)	10μs
Pulse repetition frequency	25 kHz
Pulse repetition interval	40μs
R_{min}	1.8 km
R_{max}	5.7 km
Unambiguous range	3.9 km
Unambiguous velocity	375 m/s

Table 3.1 Radar parameters used

3.1 Creating the test data

Sample data is created using matlab. Chirp signal is used as the pulse. The equation 2.2 for generating a chirp signal. The real and imaginary part of signal is as shown in figure 3.2. The

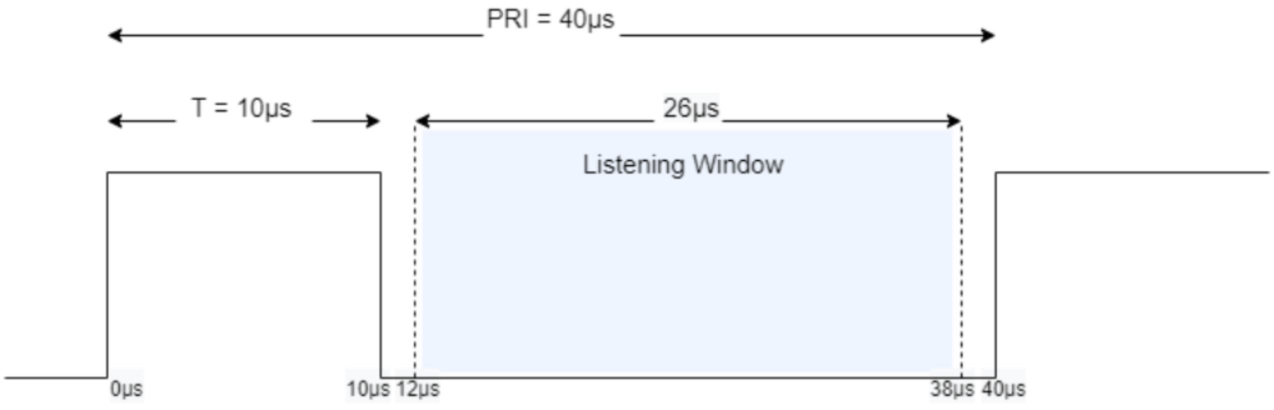
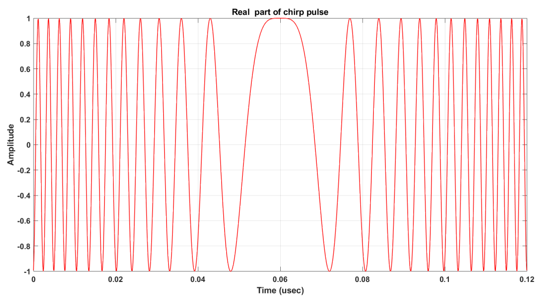
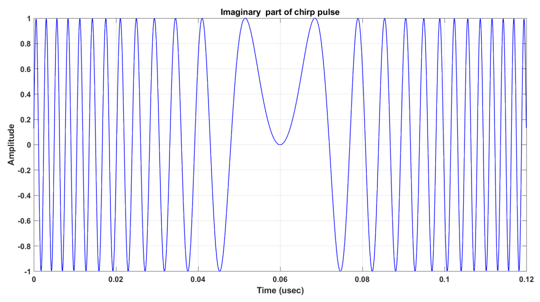


Fig. 3.1 Pulse used for simulation

signal is sampled at f_s =12MHz.



(a) Real part of chirp pulse



(b) Imaginary part of chirp pulse

Fig. 3.2 Chirp Pulse

4 targets were specified with position and velocity as given in the table (3.2). The echo signal is calculated depending on the position and velocity of the target. The task is to extract the target information from the received echo.

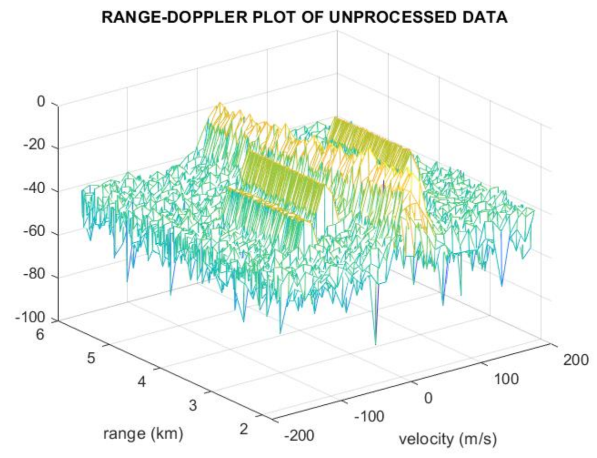
Target No	Range (km)	SNR (dB)	Velocity (m/s)
1	2.3	10	-112.5
2	2.7	20	-56.5
3	3.1	10	37.5
4	3.1	20	93.75

Table 3.2 Targets Specified

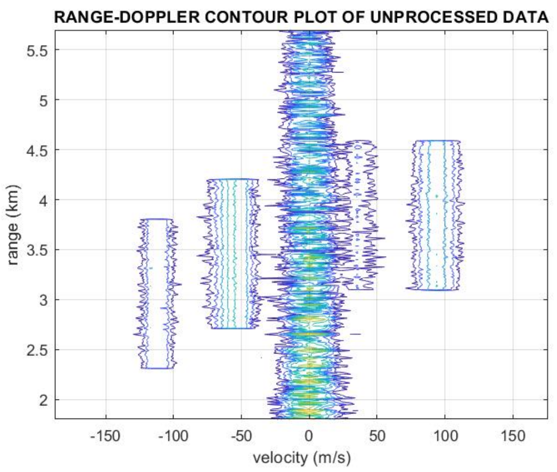
Target information onto the echo signal Gaussian noise [2] and clutter [1, p. 470] is added to the signal. The signal is then sampled at the desired sampling frequency of 12MHz. A total of 40 pulses and its received echo is simulated. The received echo for a period of 26μs has a total of 313 samples. Since 40 pulses are used we will have a total of 313 $\times$ 40 samples. These samples can be represented in the form of a data matrix of size 40 $\times$ 313 with slow time samples along the columns and fast time samples along the rows.

3.2 Processing the test data

The test data is now processed to extract the position and velocity of the target. The signal at each stage is plotted for clarity.



(a) 3D Plot

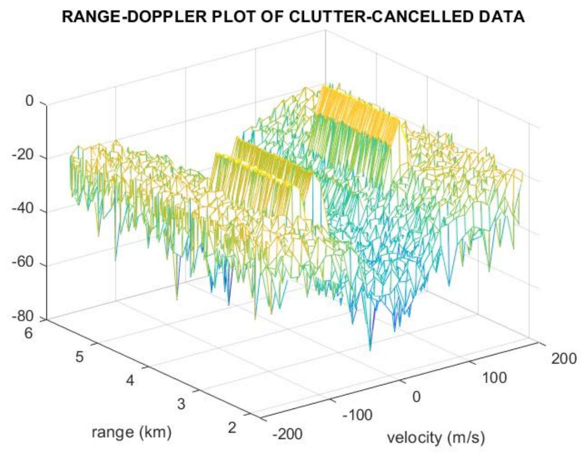


(b) Contour Plot

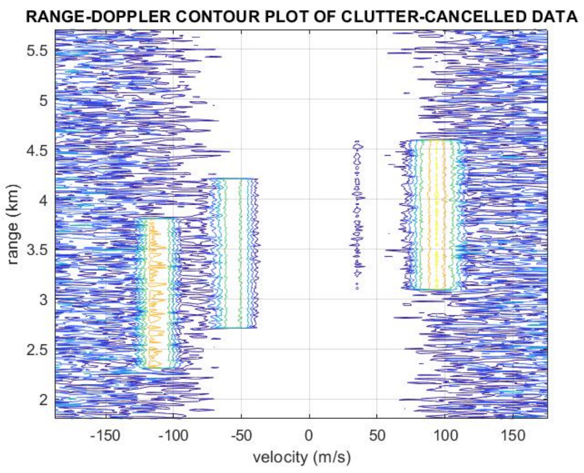
Fig. 3.3 Range and contour plot of unprocessed data

It is clear from figure (3.3) that there is clutter[1, p. 470] (with velocity = 0) and noise present in the echoes. Several signal processing techniques are employed to get rid of the clutter and noise to extract the target information.

Looking at the figure (3.4) it is clear that the clutter is successfully removed but it adds up high frequency noise. Now to remove the noise, matched filtering [3, p. 161] range is employed. Matched filtering can significantly improve the SNR ratio. The figure 3.5 shows the data after matched filtering. Looking at figure 3.5(b) it is seen that the noise is significantly reduced and the peaks is clearly seen which represents the target. There is a total of 4 peaks which represents the 4 target. The velocity and position of the target is obtained from the data and is tabulated in the table (3.3).

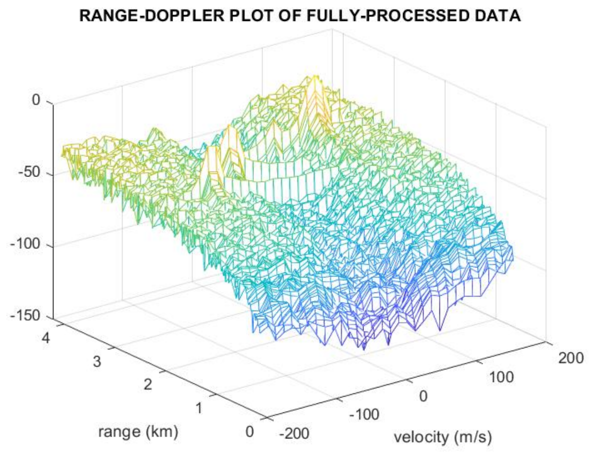


(a) 3D Plot

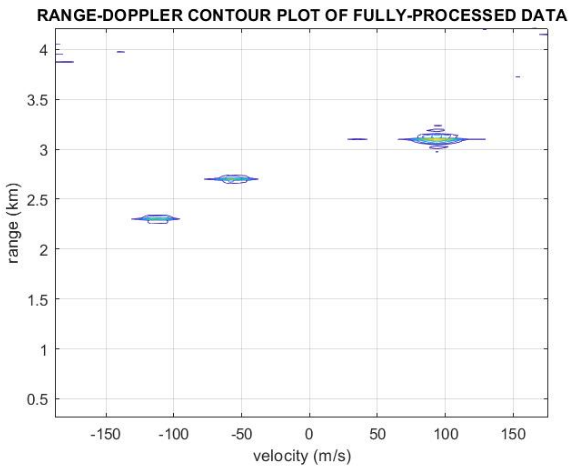


(b) Contour Plot

Fig. 3.4 Range and contour plot after clutter cancellation



(a) 3D Plot



(b) Contour Plot

Fig. 3.5 Range and contour plot after clutter cancellation and matched filtering

Target Number	Rel Amp (dB)	Range (km)	Velocity (m/s)
1	-11	2.3	-113.3
2	-0.482	2.7	-57
3	0	3.1	93.76

Table 3.3 Estimated Target Parameters

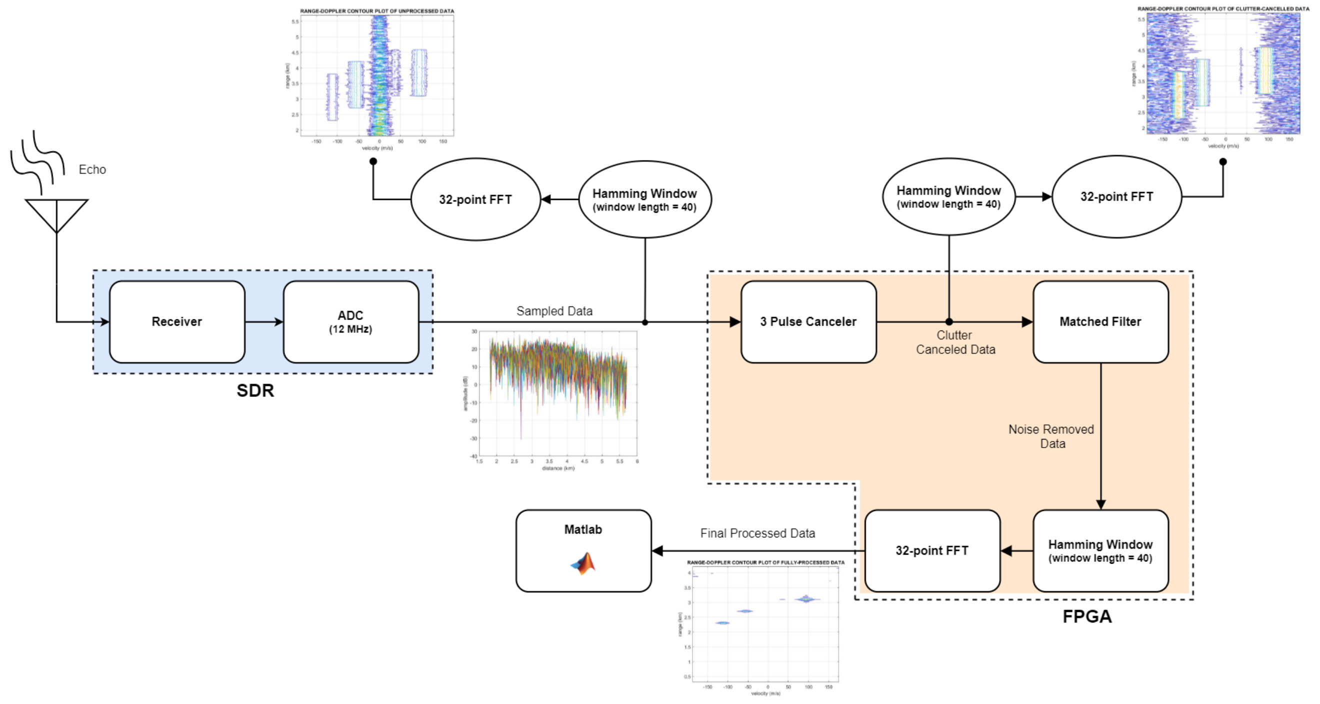

Echo
Receiver
ADC (12 MHz)
SDR
Sampled Data
Hamming Window (window length = 40)
32-point FFT
3 Pulse Canceler
Clutter Canceled Data
Matched Filter
Noise Removed Data
Hamming Window (window length = 40)
32-point FFT
FPGA
Final Processed Data
Matlab
RANGE-DOPPLER CONTOUR PLOT OF UNPROCESSED DATA
RANGE-DOPPLER CONTOUR PLOT OF CLUTTER-CANCELLED DATA
RANGE-DOPPLER CONTOUR PLOT OF FULLY-PROCESSED DATA

Fig. 3.6 Block Diagram

In figure 3.6, the doppler [2, p. 92] and range information at each stage of processing is plotted. The raw data coming from SDR contains noise as well as clutter [1, p. 470], which is removed in later steps. It is understood from figure 3.6 that after clutter cancellation the clutter is removed, adding up the high frequency noise. Matched filtering [2, p. 161] is then employed to reduce the effect of noise. The final processed data only contains the information of the targets. Chirp Signal Generator, Matched filter, 3 Pulse Clutter Canceler [1, p. 107], 32-point FFT and Hamming

Window [4] is designed using Vivado and to be implemented onto an actual hardware.

Chapter 4

Simulation and FPGA Implementation

The board contains both processor and FPGA. The entire signal processor part is implemented on the FPGA. The data flow on the hardware is summarized in the figure 3.6. We assume the data is coming from the SDR. The data to be processed is actually stored as a text file inside a SD card. It is the work of the processor to fetch the data from the SD card and supply it to the implemented hardware on the FPGA. It also writes back the final output from the processor to the SD card as a text file. The data flow is as shown in figure 4.1.

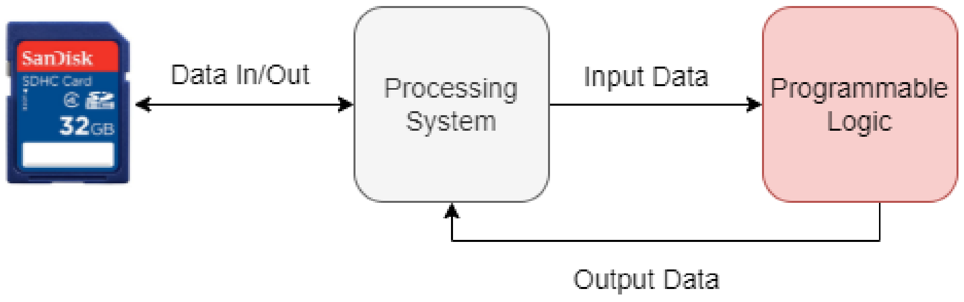


Fig. 4.1 Block Diagram

4.1 Work Flow

The entire algorithm is created using Vitis HLS [5], where we can write our algorithm in C++ and then convert them into Verilog. This can be then exported as an IP to Vivado. It is in Vivado where the configuration file (.xsa file) for the FPGA is created. After creating the hardware configuration file in Vivado it is exported to Xilinx Vitis SDK for developing the software part. C programming is used to write instructions for the processor. It is the processor which supplies the data to the hardware implemented on the FPGA.

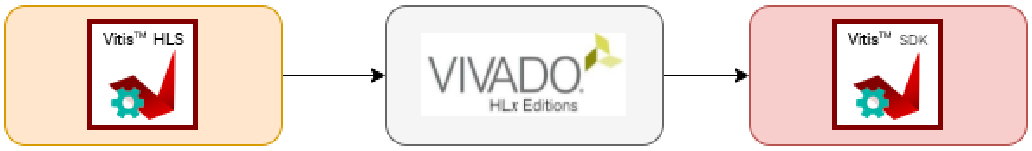


Fig. 4.2 Work flow

4.2 Data Format

Fixed point representation is used for processing the data as it consumes less resources and is easier to implement. 32-bit is used for both imaginary and real part of the data. The bit allocation is as shown in figure 4.3. The fixed point representation is denoted by Qm.n where m is the number of integer bits including the sign bit and n is the number of fractional bits. Therefore the fixed point representation shown in the figure can be denoted as Q24.8.

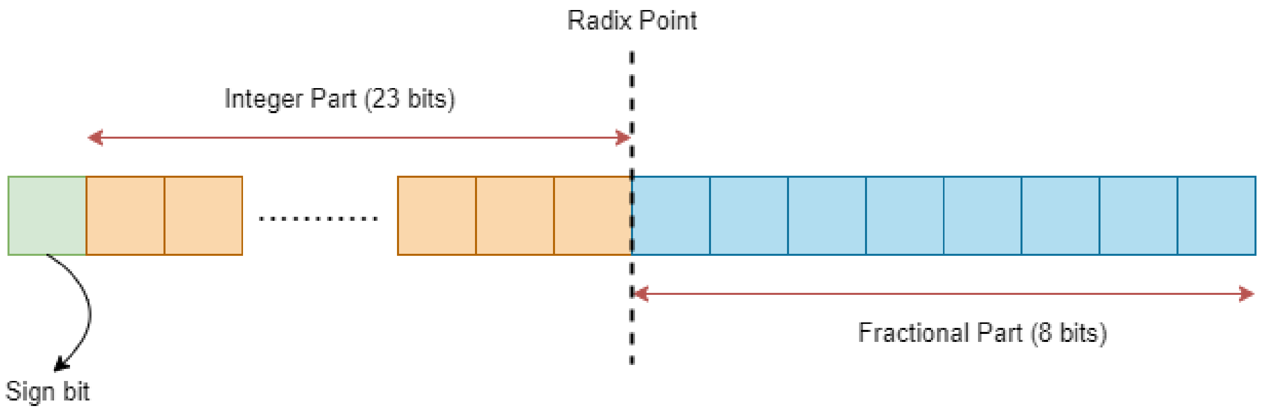


Fig. 4.3 Data Number Format

With 8-bit as the fractional part we can get a resolution of $2^{-8} = 3.9 \times 10^{-}3$. The dynamic range of the fixed point representation is $[-2^{23},\ 2^{23} - 2^{-8}]$. The data is transferred in row major format of 313×40 data matrix. The real part of the data is first transferred and then the imaginary part. The data goes to the designed hardware as a single data stream with a total data of 25040.

4.3 Modules

Since the data is coming to the module as a stream we need to convert the stream data in the form of a matrix so that further processing will be much easier. After the final processing the data is converted again back to stream and then sent out of the module. The following subsections discusses the implementation of each of the modules in detail.

4.3.1 Pulse Generation

As discussed in section 2.2 we make use of the chirp signal. The sampled data shown in figure 3.2 is stored inside the BRAM of the FPGA which is then send to the DAC on the SDR which is then upconverted to desired passband signal and is transmitted at regular intervals through the antenna. The digitized data is to be sent to the SDR attached to the FPGA.

The SDR captures the incoming echoes and samples it. The sampled data is then stored inside a BRAM inside of FPGA which is later send to the designed radar processor on the FPGA for further processing.

4.3.2 3 Pulse Canceler (Double delay-line canceler)

The delay-line canceler [1, p. 107] acts as a filter which rejects the d-c component of clutter [1, p. 470]. Because of its periodic nature it rejects energy in the vicinity of the pulse repetition frequency and its harmonics. The frequency response of a single-delay-line canceler does not always have a broad clutter rejection property. The clutter rejection notch can be widened by passing the output of the single delay-line canceler through one more delay-line canceler. This configuration is called double-delay line canceler.

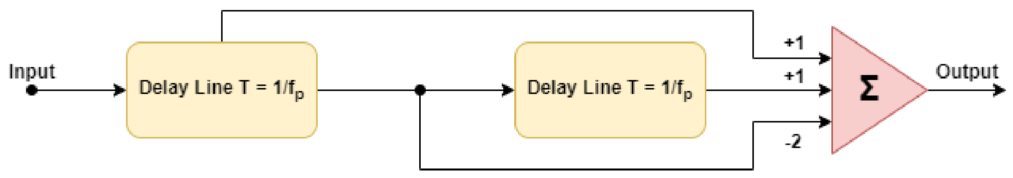


Fig. 4.4 Double delay-line canceler block diagram

The data matrix is multiplied with the coefficient matrix to get the output. The dimension of the data matrix is 313 x 40 and that of the coefficient matrix is 40 x 39. Therefore, the output matrix would be 313 x 39. The matrix multiplication algorithm is designed using Xilinx Vitis [5]. The coefficient matrix is as shown in figure 4.1

$$\begin{bmatrix} 1 & 0 & 0 & . & . & 0 & 0 & 0 \\ -2 & 1 & 0 & . & . & 0 & 0 & 0 \\ 1 & -2 & 1 & . & . & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . \\ 0 & 0 & 0 & . & . & 1 & -2 & 1 \\ 0 & 0 & 0 & . & . & 0 & 1 & -2 \end{bmatrix}_{40\times39} \tag{4.1}$$

4.3.3 Matched Filtering

After clutter cancellation, the SNR ratio is improved using matched filtering [2, p. 161]. Matched filtering is performed on the data matrix along the fast time samples.The filter is time delayed function of the chirp signal. The chirp signal is delayed by N/2 samples. The filter coefficient is generated using Matlab and is stored as an array. The coefficients is then convoluted with data in data matrix along the fast time samples to get the resultant data matrix matrix. The convolution operation is given by the equation 4.2.

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \tag{4.2}$$

where x[k] is the fast time samples and h[k] is the matched filtering coefficients.

4.3.4 Hamming Window

If a radar signal has a constant frequency that persists for a long time, the FFT of that signal would result in a single narrow frequency line (referred to as an impulse response) on the FFT plot. However, if we pulse that signal or we capture a limited portion of the signal in time, the FFT of that signal will spread into adjacent frequency bins. This spreading of frequency around the transmitted center frequency is known as spectral leakage. Ignoring spectral leakage can result in the non detection of the targets. Windowing functions are the best way to reduce the effects of spectral leakage. There are many windowing functions which can be used to pass the signal before FFT is taken. Here we use Hamming windowing function which is as shown in figure 4.5

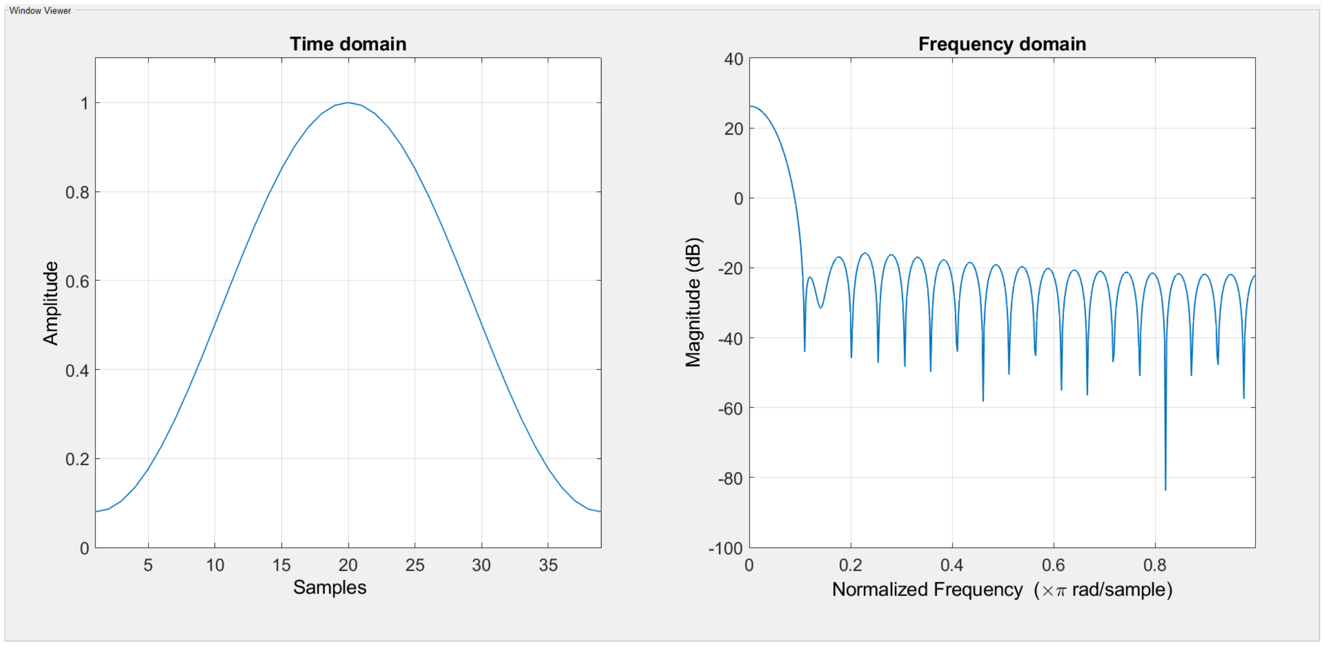


Fig. 4.5 Hamming Window

The hamming window coefficients are generated in matlab using the function 'hamming'. The coefficients are then stored as an array and is multiplied with the data in the data matrix along the slow time samples.

4.3.5 Fast Fourier Transform

After windowing 32-point FFT is performed along the slow time samples in the data matrix. FFT along the slow time samples can be easily performed via simple matrix multiplication. The equation for finding the FFT samples is as given in equation 4.3.

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ . \\ . \\ X_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & .\ . & 1 \\ 1 & W_N & W_N^2 & W_N^3 & .\ . & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & W_N^6 & .\ . & W_N^{2(N-1)} \\ . & . & . & . & .\ . & . \\ . & . & . & . & .\ . & . \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & W_N^{3(N-1)} & .\ . & W_N^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ . \\ . \\ x_{N-1} \end{bmatrix} \tag{4.3}$$

where $x_0, x_1..x_{N-1}$ are the slow time samples and $X_0, X_1..X_{N-1}$ are the FFT result.

Since we are taking 32-point FFT, $N = 32$. The coefficient matrix is generated using matlab and is stored are an 2d array which is then multiplied with each of the slow time sample set. The resultant data matrix is then stored inside BRAM.

The algorithm for the modules discussed in the section 4.3.1 - 4.3.5 is written in C++ and then converted to Verilog and exported as an IP to Vivado Design Suite. The imported IP block is as shown in figure 4.6. The IP block uses AXI Stream interface for transferring the data to and from the module.

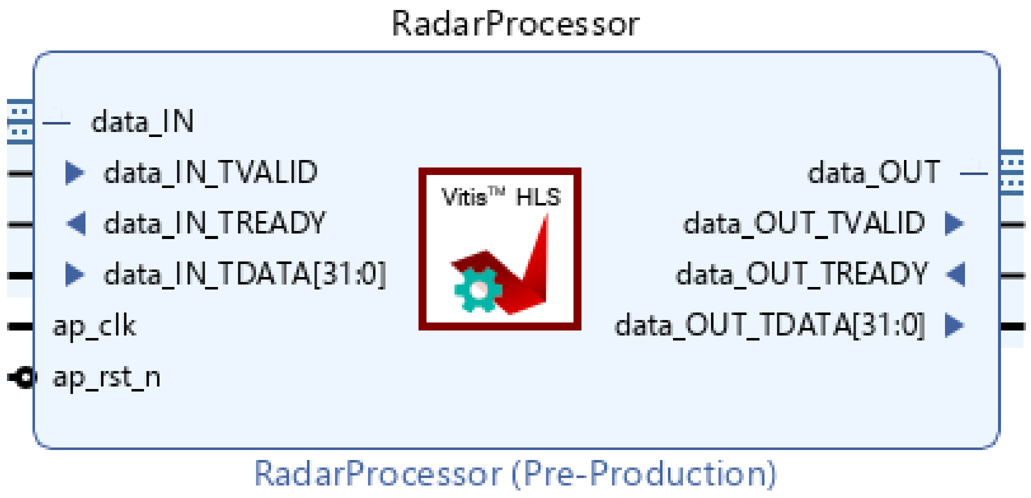


Fig. 4.6 Work flow

4.4 AXI Stream Interface

As discussed in the previous section the data is transferred to and fro from the module via an AXI Stream interface.The AXI4-Stream protocol is used as a standard interface to connect components that wish to exchange data. The interface can be used to send data between a slave and master interfaces. The IP shown in figure 4.6 has 'data_IN' port as the slave interface, it takes in the data from a master interface. Whereas the 'data_OUT' port act as the master interface, it supplies data to a slave interface. The AXI stream interface have mainly tvalid,tready as the control signals. The tready is issued by the slave module and tvalid by the master module. tvalid indicates whether the data transferred from the master module is valid and tready state whether the slave module is ready to accept the data.

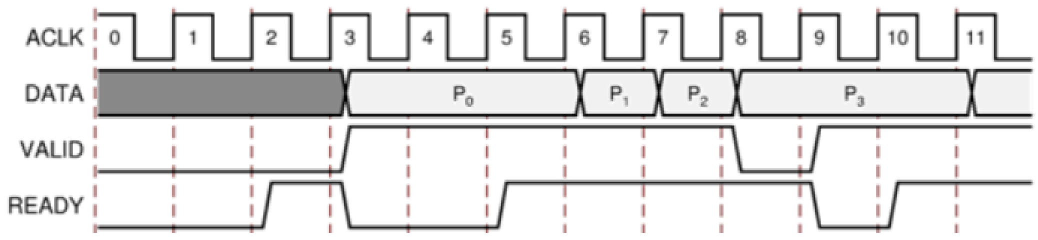


Fig. 4.7 Example of ready/valid handshake

4.5 Vivado Simulation

The block in figure 4.6 is simulated in Vivado. The waveform output at the time when data is ready at the output port is as shown in figure 4.8.

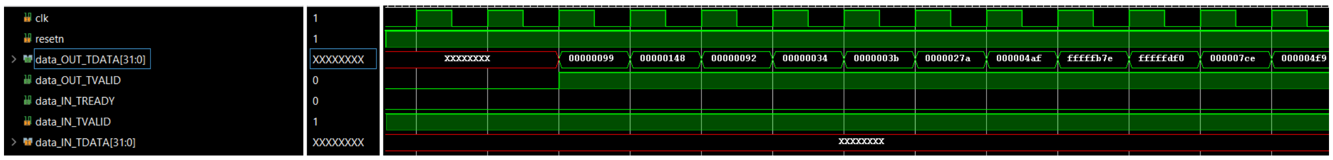


Fig. 4.8 Vivado Simulation of the Designed Radar Processor IP

The 'data_OUT_TDATA' is the port through which the final result is obtained. 'data_OUT_TVALID' and 'data_OUT_TREADY' is the handshaking signals. It is clear from the simulation waveform that when both 'data_OUT_TVALID' and 'data_OUT_TREADY' is high data starts coming out from 'data_OUT_TDATA' port.

Chapter 5

Micro Processor

In the previous chapter we discussed the implementation of the algorithm on FPGA. In this chapter the hardware configuration and programming of the processing system is discussed. Zynq-7000 devices are equipped with dual-core ARM Cortex-A9 processors. The sole purpose of the processor in this project is to help in the transferring of data from the SD card to the designed module on the FPGA.

By default Vitis SDK [5] uses 1KB for stack as well as Heap memory for running the application on the processing system. Both the memory is manually changed to around 100KB to account for the large data.

5.1 Hardware Configuration of the Processor

The designed radar processing module is exported to Vivado. Since the processing system does not support AXI Stream interface, Direct Memory Access is used.

5.1.1 Direct Memory Access

Direct memory access (DMA) is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations.

DMA channels are used to communicate data between the peripheral device and the system memory.A DMA channel enables a device to transfer data without exposing the CPU to a work overload. Without the DMA channels, the CPU copies every piece of data using a peripheral bus from the I/O device. Using a peripheral bus occupies the CPU during the read/write process and does not allow other work to be performed until the operation is completed. With DMA, the CPU can process other tasks while data transfer is being performed. The transfer of data is first initiated by the CPU.

5.1.2 Vivado Block Diagram

Using DMA the large amount of data can be transferred from the processing system to the programmable logic without overloading the processor. Xilinx provides an inbuilt IP named 'AXI Direct Memory Access' [6] for configuring the DMA. The entire block diagram connection is as shown in figure 5.1.

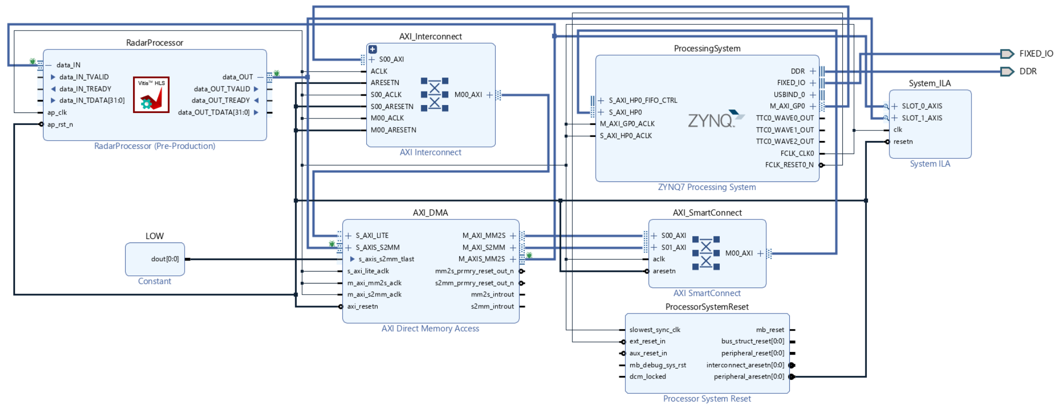


Fig. 5.1 Work flow

'RadarProcessor' is the module which performs all the algorithms for the target detection. The 'ProcessingSystem' IP block is the processing part of the board. The rest are supporting blocks which help in the smooth communication between these modules.

5.2 Programming the Processor

After configuring the hardware in Vivado, it is exported as .xsa file. This file is used to create an application project in Vitis SDK. SDK is used to write instructions in C/C++ for the processor to perform.

5.2.1 Reading and Writing Data to and from SD card

The data is made into a text file with first 313×40 data as the real part and the last 313×40 as the imaginary part. FatFs module is used to read data from the text file contained in the SD card. FatFs [7] is a generic FAT/exFAT filesystem module for small embedded systems and is written in C language. Functions such as 'f_read' and 'f_write' in the 'ff.h' are used for reading and writing to and from the SD card. The text file contains data in the form of bits line by line. Each line is read one by one and stored in 'u32' array and is later send to the radar processor. The processor then waits for the data to be processed. Once its done the processed data is written back to the SD card as a text file which can be plotted in matlab.

5.2.2 Processing System and Programmable Logic interaction

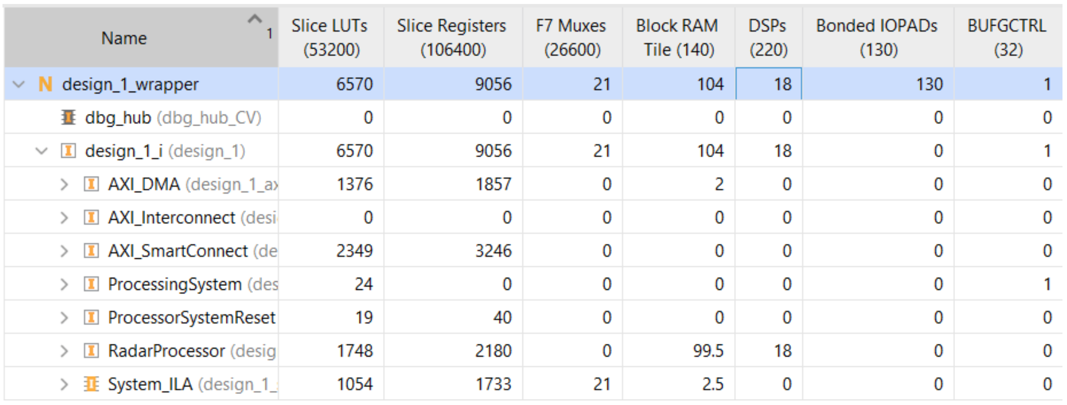
Xilinx provides drivers for configuring the 'AXI Direct Memory Access' [6] IP. It contains functions for configuring the DMA. It also has functions for managing data flow between the PL and PS parts. Since there is a lot of data to be transferred between the PS and PL part, it is send in small packets. We use interrupt to manage the data transfer. Whenever a packet is successfully transferred the processor issues an interrupt which is handled by an interrupt handler. The interrupt handler then send the next set of data. This is continued till all the data is completely transferred.

Chapter 6

Final Results

6.0.1 Resource Utilization and Latency

The resource utilization table is as shown in figure 6.1. From the figure it is clear that 'RadarProcessor' IP has utilized 1748 Slice LUTs, 2180 Slice Registers, 99.5 Block RAM and 18 DSPs. The 'RadarProcessor' takes almost 7.241ns for processing the entire algorithm.

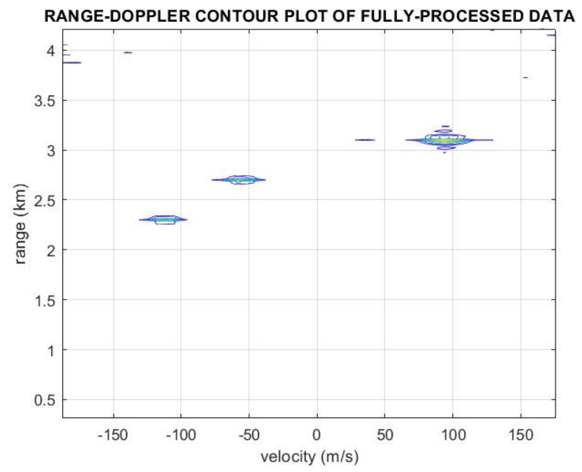


Name	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	Block RAM Tile (140)	DSPs (220)	Bonded IOPADs (130)	BUFGCTRL (32)
design_1_wrapper	6570	9056	21	104	18	130	1
dbg_hub (dbg_hub_CV)	0	0	0	0	0	0	0
design_1_i (design_1)	6570	9056	21	104	18	0	1
AXI_DMA (design_1_a	1376	1857	0	2	0	0	0
AXI_Interconnect (desi	0	0	0	0	0	0	0
AXI_SmartConnect (de	2349	3246	0	0	0	0	0
ProcessingSystem (des	24	0	0	0	0	0	1
ProcessorSystemReset	19	40	0	0	0	0	0
RadarProcessor (desig	1748	2180	0	99.5	18	0	0
System_ILA (design_1_	1054	1733	21	2.5	0	0	0

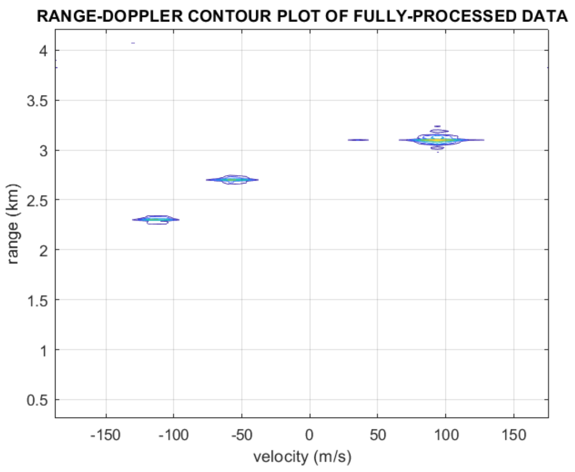
Fig. 6.1 Resource Utilization

6.0.2 Final Output

The final data obtained after computation is plotted in matlab for easy visualization and is compared with the matlab simulation output. The results are as shown in figure 6.2



(a) Matlab Output



(b) Output from 'RadarProcessor' module

Fig. 6.2 Final Output comaprison with matlab output

It is clear that both the output matches with one another.

Chapter 7

Conclusion and Future Work

Matlab simulation of the radar model is successfully completed. The target parameters were successfully extracted and it matches with the actual target parameters.The entire radar signal processor is implemented. The working of radar processor is verified by comparing with the reference done in Matlab.

In future, the algorithms used in processing the data can be made faster by efficient coding styles. Also, instead of taking data from the SD card it can be directly taken from the SDR attached to the SDR. Also the entire transmitter module can be built which along with the receiver part. The final processed output can be shown on a LCD display using an HDMI cable. Further this can be made to work as an standalone system, like a complete radar system used to detect targets.

References

[1] Merrill I. Skolnik, “Introduction To Radar Systems (2nd ed.)”, Tata McGraw-Hill Edition, 1980.

[2] Mark A Richards, “Fundamentals of Radar Signal Processing (1st ed.)”, Tata McGraw-Hill Edition, 2005.

[3] Robert Fisher; Simon Perkins; Ashley Walker; Erik Wolfart. ”Image Synthesis — Noise Generation”. Retrieved 11 October 2013.

[4] Smith, Julius O., “Pectral Audio Signal Processing”, W3K Publishing, December 2011.

[5] Vitis High-Level Synthesis User Guide

[6] AXI DMA v7.1 LogiCORE IP Product Guide

[7] FatFs - Generic FAT Filesystem Module

[8] AXI DMA Xilinx Vitis Drivers API Documentation

Note: Link to the github repo is attached here.