Indian Institute of Technology Palakkad
भारतीय प्रौद्योगिकी संस्थान पालक्काड
Under Ministry of Human Resource Development, Govt. of India
मानव संसाधन विकास मंत्रालय के अधीन, भारत सरकार

BTP 1 Report

Electrical Engineering

IIT Palakkad

December 12, 2021

Developing a Radar Signal Processor for the Detection of Drones

Mentors: Dr. Swaroop Sahoo & Dr. Subrahmanyam Mula

Name	: Aswin Raj K
Roll Number	: 121801008
Department	: Electrical Engineering

Developing a Radar Signal Processor for the Detection of Drones

Abstract

In this report, the work done as a part of developing the radar signal processor for the detection of drones is discussed. The signal processing part is designed to be implemented on an FPGA (ZC702[2]) board. It is designed to produce pulses with monotonous frequency as well as chirp[1]. The required RF pulse is generated at baseband that will be upconverted to X-band on the SDR[3] and is transmitted out. The signal transmitted is scattered and received by the radar. The FPGA is also 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[1] 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 Matlab implementation of the radar is clearly explained in the report.

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. This approach works very well when dealing with large drones made of metal pieces, carbon fiber, and composite material. The main difficulty arises when the size of the drones to be detected is small, where the probability of detection is reduced because of the smaller radar cross-section of drone made of plastic or styrofoam. The detection of such drones is possible only if the signal processing system is sensitive enough to smaller changes produced by such targets. Thus, designing sensitive signal processors and post-processing algorithms is very important for the detection of such drones. These 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.

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 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

$$\rho = \frac{cT}{2} \tag{1}$$

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

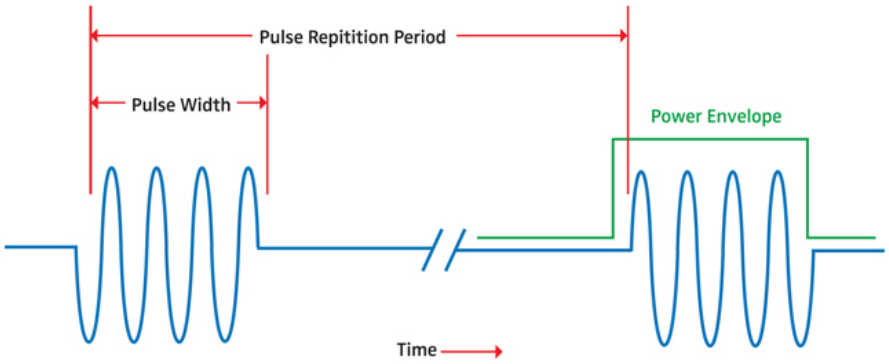


Figure 1: Normal pulse

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.

In figure 2 the orange and green bin refers to two different targets. From figure 2 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 3.

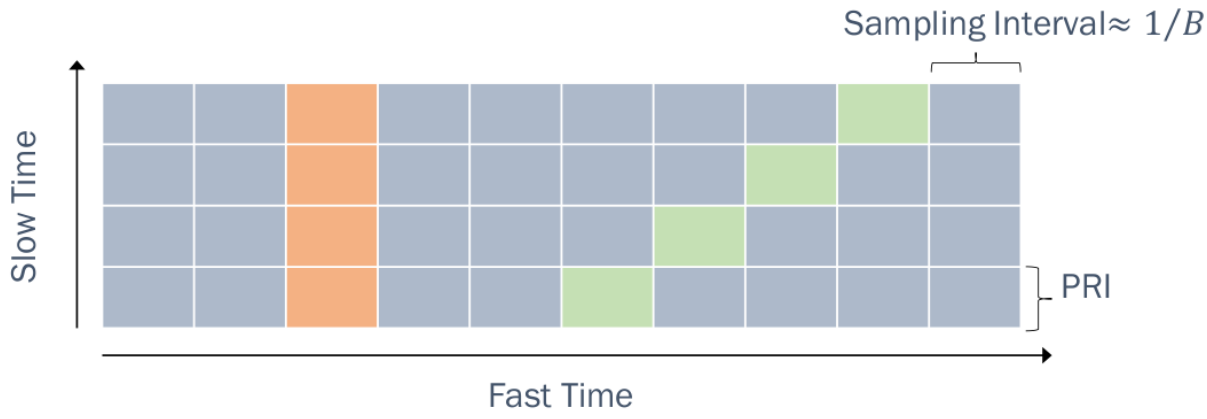


Figure 2: Data matrix with fast time samples along the row and slow time samples along the columns.

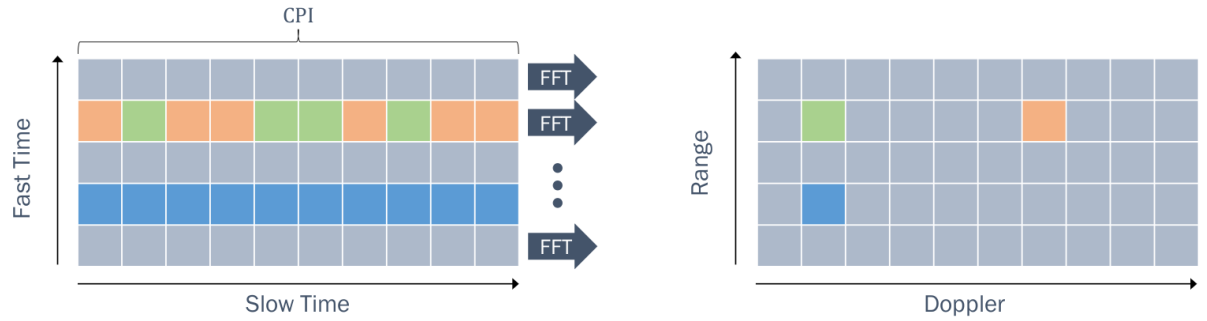


Figure 3: Taking FFT along Slow Time samples to produce another matrix which has both Range and Doppler Information

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.

Objective 1 is completed in this till now and some part of Objective 2 & 3 is also completed as the part of OELP the previous semester. Slight changes need to be made to the work done during the OELP. Section 2 describes how the radar is simulated in Matlab to ensure its feasibility before implementing onto an actual FPGA board.

2 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.

2.1 Creating the test data

Sample data is created using matlab. Chirp signal is used as the pulse. The equation for generating a chirp signal is given below,

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

where T is the pulse width, f_s is the sampling frequency.

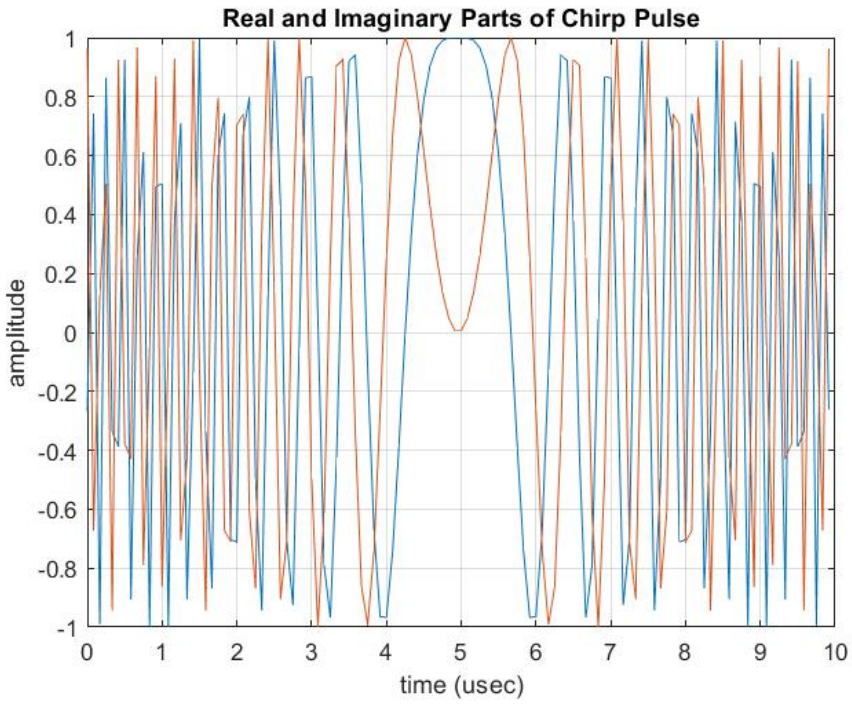


Figure 4: Chirp signal to be used

The pulse parameters used is as follows,

- $T = 10\mu s$ (Pulse Width)
- PRF = 25 kHz
- $PRI = 40\mu s$
- R.F Frequency = 10 GHz
- $R_{min} = 1.8$km & $R_{max} = 5.7$km
- Unambiguous Range = 3.9 km
- Unambiguous Velocity = 375 m/s

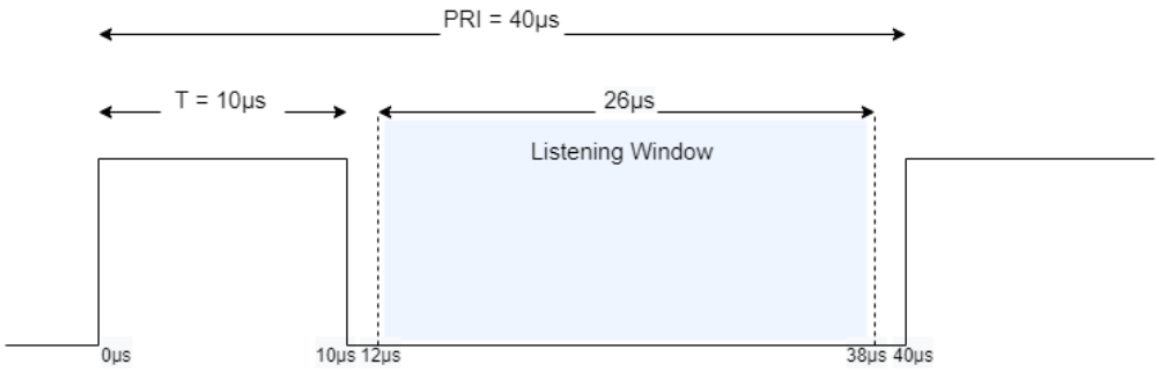


Figure 5: Pulse used for simulation

Also 4 targets were specified with position and velocity as given in the table 1. The echo signal is calculated depending on the position and velocity of the target.

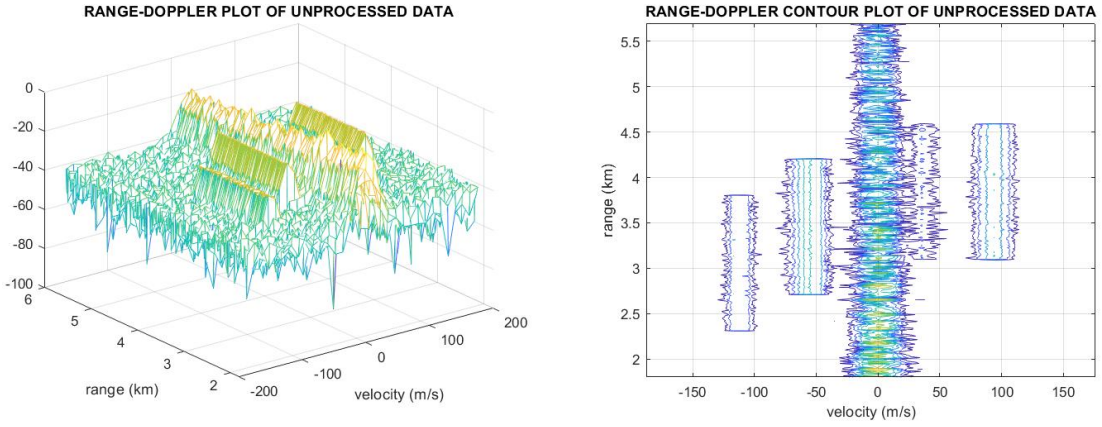
Target No	Range (km)	SNR (dB)	Velocity (m/s)
1	2.3	10	-112.5
2	2.7	20	-56.5
3	3.1	10	37.5
4	3.1	20	93.75

Table 1: Targets Specified

After adding the target information onto the echo signal Gaussian noise and clutter 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.

2.2 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.



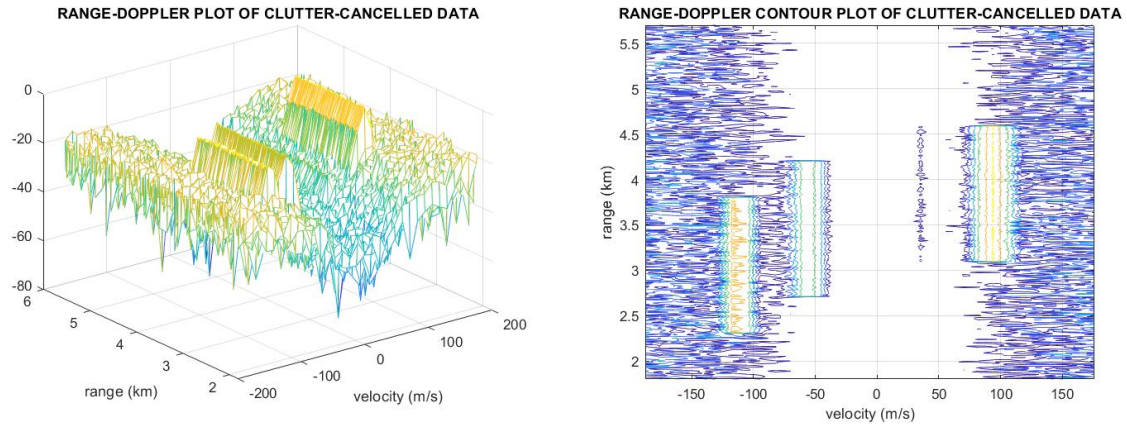
(a) 3D Plot

(b) Contour Plot

Figure 6: Range and contour plot of unprocessed data

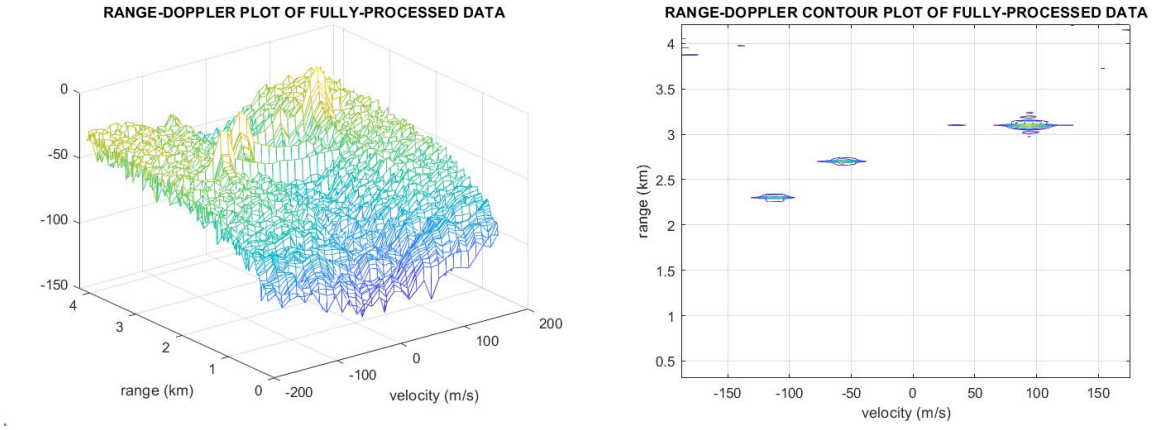
It is clear from figure (6) that there is clutter[4] (with velocity = 0) and noise present in the echoes. Several signal processing techniqes are employed to get rid of the clutter[4] and noise to extract the target information.

Looking at the figure (7) it is clear that the clutter[4] is successfully removed but it adds up high frequency noise. Now to remove the noise, matched filtering[1] in range is employed. Matched filtering[1] can signigicantly improve the SNR ratio. The figure 8 shows the data after matched fitering. Looking at figure 8b it is seen that the noise is significanlty 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 (2).



(a) 3D Plot (b) Contour Plot

Figure 7: Range and contour plot after clutter cancellation



(a) 3D Plot (b) Contour Plot

Figure 8: Range and contour plot after clutter cancellation and matched filtering

a**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 2: Estimated Target Parameters

3 Hardware Implementation

The matlab simulation is successfully compeleted and we can use the parameters used in the simulation for our actual design. The entire signal processor is to be implemented onto an FPGA board with an SDR attached to it. Chirp Signal Generator, Matched filter[1], Clutter Canceler[4], 32-point FFT[4] and Hamming Window are to be designed using Vivado and later implemented onto actual hardware. The entire process is summarised in the figure 9.

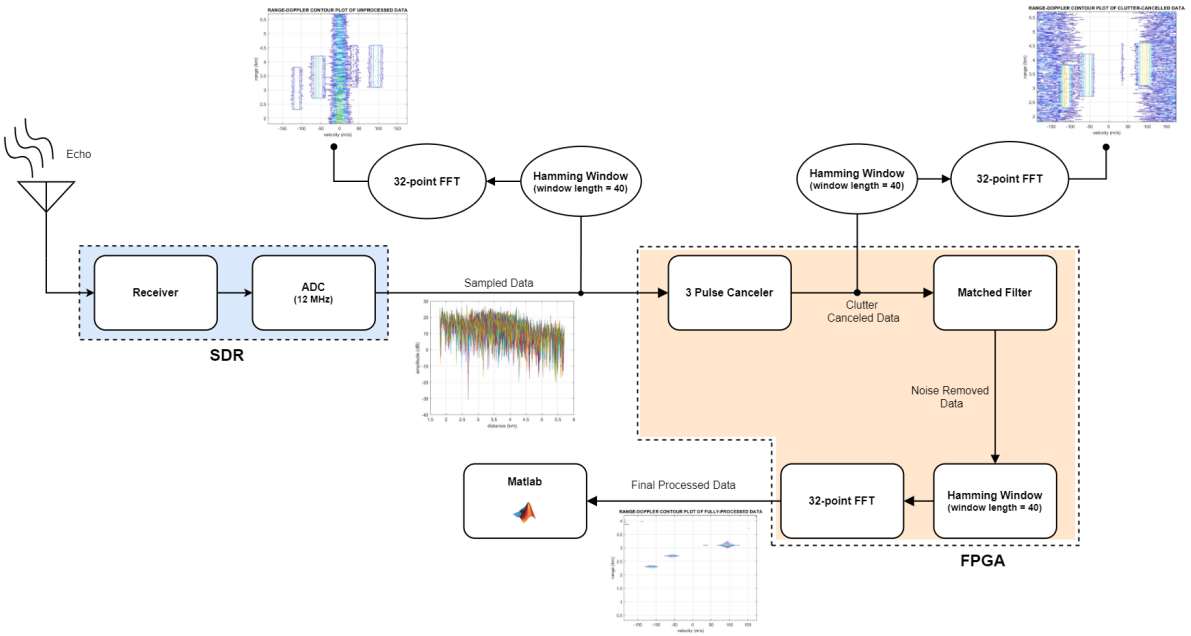


Figure 9: Block Diagram

3.1 Memory Type and Number Format

There are two types of memory inside FPGAs (Block RAM and Distributed RAM). Distributed RAM is built up with LUT whereas block RAM is the dedicated RAM that does not consume any additional LUT in your design. The distributed RAM is much faster than the block RAM, but the downside of using distributed RAM is that it consumes FPGA resources. Block Rams are used if the amount of data stored is much larger. In our case the amount of data is much larger and thus we use block RAM for storing the data.

Fixed point representation is used for processing the data. As it consumes less resources and is easier to implement. 16 data is used for both imaginary and real part of the data. The bit allocation is as shown in figure (10).

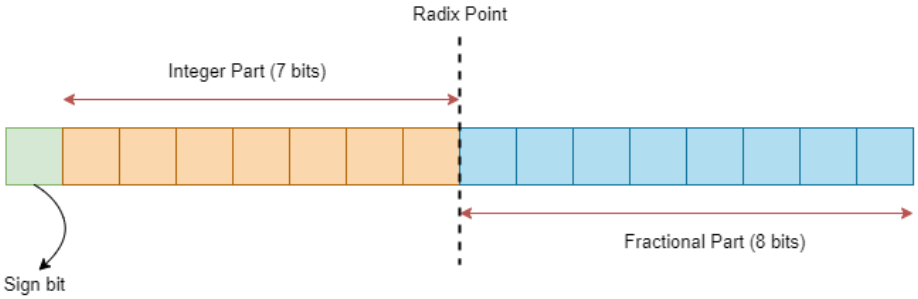


Figure 10: Fixed Point Number Representation

The data to be processed is stored inside BRAM. For allocating block memory we use block memory generator[5] IP. The data which has both real and imaginary part is converted to fixed point (Q8.8) hex format using matlab. The 16 bit real and imaginary part is joined to form 32-bit long data in which the first 16 bits represent the imaginary part and the last 16 bits represent its real part. The data in hex format is then imported as a .mem file which can be used to initialize the BRAM via the property window of the block memory generator IP. Since the memory are continuous locations the data is stored in column major format. Each element can be fetched from the BRAM by supplying the corresponding address of the element to the block memory generator IP. The read latency of the IP is 2 clock cycles, which means the data will be available after 2 clock cycles of address input. The data matrix to be stored

inside the BRAM has a dimension of 313×40. The representation of how the data is stored inside the BRAM is as shown in figure (11).

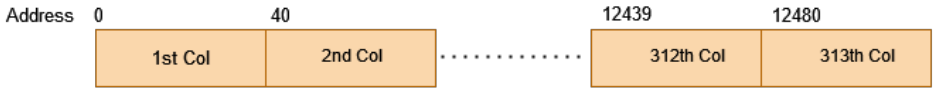


Figure 11: Visual representation of how the data is stored inside BRAM. The starting address of each col is given

3.2 Hamming Window

Hamming window coefficients are obtained using hamming function in matlab. The coefficients is then converted to Q8.8 and is exported as .mem file which is then stored inside a BRAM. The block diagram created inside the IP Integrator of Vivado is as shown in figure (12)

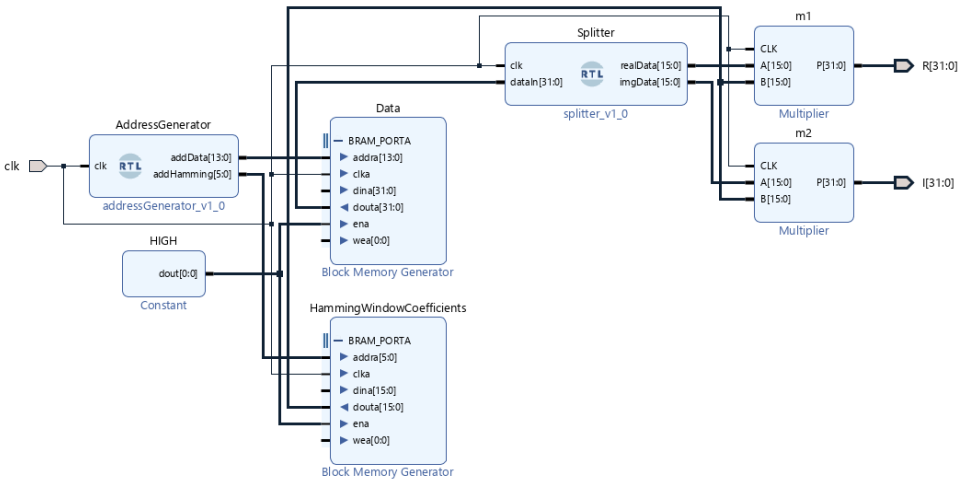


Figure 12: Hamming window implementation ip block diagram

In figure (12) AddressGenerator, Multiplier and the Splitter are the IPs created using verilog. The AddressGenerator IP supplies the address to each of the block memory generators. The AddressGenerator IP has two counters, one running from 0 to 39 (for hamming coefficients) and another running from 0 to 12519 (for the data to be processed). The data is splitted into its real and imaginary part which is then multiplied with the corresponding hamming window coefficient.

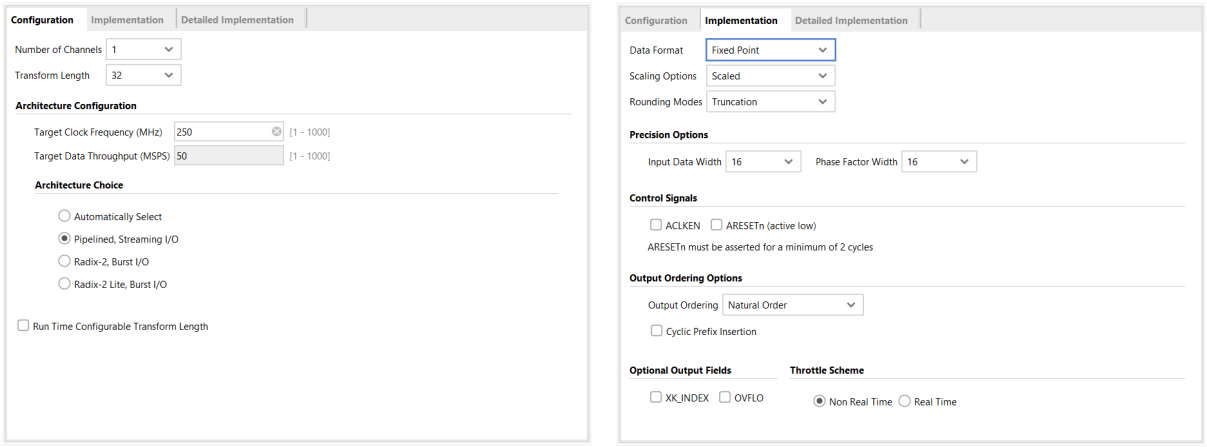
Port Name	Description
clka	The clock signal
addra	Memory address for port read and write operations.
dina	Data input to the BRAM for write operation
douta	Data output from the BRAM for read operation
ena	Enables Read, Write, and reset operations for the port
wea	Enables write operation through the port

Table 3: Block Memory Generator Port Descriptions (Single Port)

3.3 FFT

The Fast Fourier Transform[5] IP created by Xilinx is used for computing the FFT.This block uses AXI Stream interface for exchanging data. We can specify the length of the FFT to in the property window

of the IP. The configuration used is as shown in figure (13).



(a) Configuration Tab

(b) Implementation Tab

Figure 13: Fast Fourier Transform IP Property Window

Number of channel is 1 and the max length of fft is specified as 32. Pipelined, Streaming I/O is choosen as the architecture. Output of the fft is configured to come in the natural order. The input data width and phase factor width is taken 16 bit. Throttle scheme is set to non real time. Apart from these configurations we need to specify fft direction, fft length and scaling schedule through the S_AXIS_CONFIG port. Since in the property window we have unchecked 'Run Time Configurable Transform Length' there is not need to specify the transform length and scaling schedule. The fft length is always taken as 32. The direction of the fft must be specifed to the S_AXIS_CONFIG for each channel. Since we are using single channel s_axis_tdata should be 8'b00000001. The last one bit indicates the direction of the channel 1 and the rest are padded zeroes. When Pipelined Streaming I/O is selected and no cyclic prefix is used, the fft ip takes in data continuously and output the processed data continuously after an initial delay.

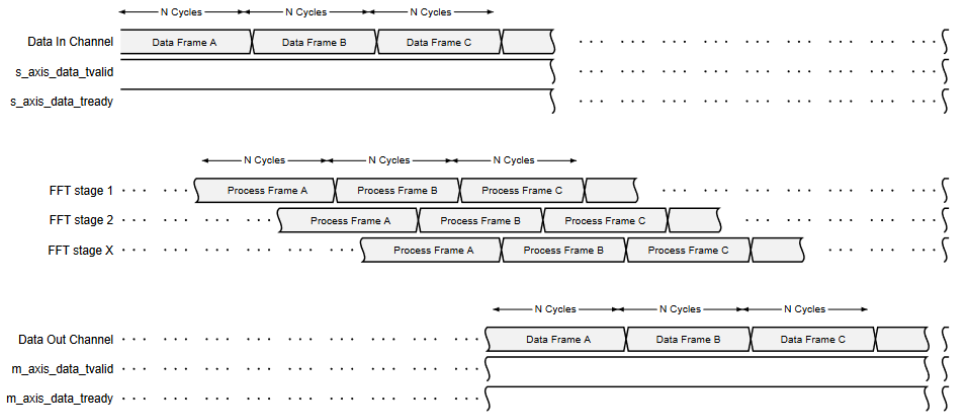


Figure 14: shows the general timing for back-to-back frames in the Pipelined Streaming architecture.

The block diagram created for testing the IP is as shown in figure (15). FFT_CONFIG is a constant ip block supply the value 8'b00000001 to the s_axis_tdata. DataSrc supplies the fft block with data from a text file.The sample data for testing is created using matlab. The function and sampling rate used is as follows,

$$f(x) = sin(2\pi 2000t)$$
$$fs = 15kHz$$

DataWrite takes the processed data from fft block and prints them in the console when ready. This data is then copied and made into a text file which is imported into matlab for verification.

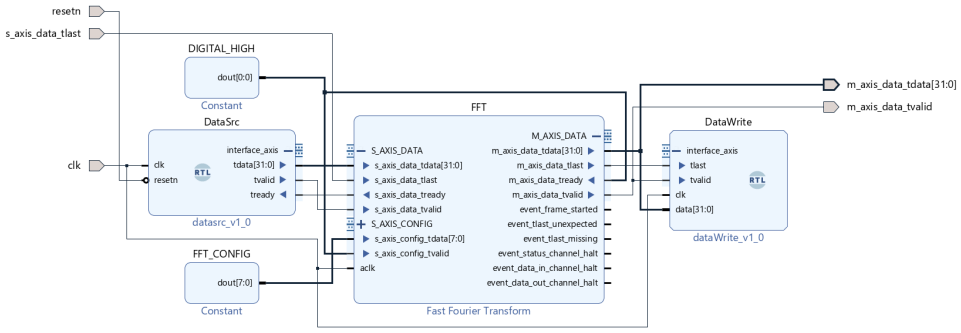


Figure 15: Block diagram for testing Fast Fourier Transform IP

Port Name	Description
clka	The clock signal
s_axis_data_tdata	Data for input channel.Imaginary (1st 16 bit) and the real part (last 16 bit) of the data to be processed.
s_axis_data_tready	Asserted by the fft to signal that it is ready to accept data
s_axis_data_tvalid	Asserted by the external master to signal that the data provided is valid
s_axis_data_tlast	Asserted by the master on the last sample of the frame
s_axis_config_tdata	Carries te configuration informations such as fft direction, fft length, and scaling schedule.
s_axis_config_tready	Asserted by the fft block to signal that it is ready to accept the configuration
s_axis_config_tvalid	Asserted by the master to signal the config data provided is valid.
m_axis_data_tdata	Carries the processed data. First 16 bit si the imaginary part and the last 16 bit is the real part
m_axis_data_tready	Asserted by the slave accepting the data to signal that it is ready to accept data
m_axis_data_tvalid	Asserted by the fft block to signal that the output is valid
m_axis_data_tlast	Asserted by the fft block on the last sample of each frame.

Table 4: Fast Fourier Transform IP Port Descriptions (Single Channel)

3.4 Integration of FFT with Hamming Window

Hamming window and FFT need to be integrated together. The data from the BRAM is passed to for taking fft after windowing. The block diagram in figure (12) is combined to form a single IP named 'Controller'. This is designed using verilog. The controller provides the address to both the BRAM containing the hamming coefficients and the data to be processed. It takes these data and multiply them together and provide it to the fft via an AXI Stream interface. The block diagram is as shown in figure (16).

Port Name	Description
aclk	Clock
nreset	Active-Low synchronous clear
hamming_data	Hamming coefficients output from the BRAM

data	Data to be processed
s_axis_tdata	Data output after windowing
s_axis_tlast	Asserted by the controller to indicate the last sampe of each frame
s_axis_tvalid	Asserted by the controller to indicate that the data provided is valid
s_axis_tready	Asserted by the fft to indicate that the it is ready to accept data
hamming_addr	Address corresponding to hamming coefficient stored in BRAM
data_addr	Address corresponding to each data to be processed, stored in the BRAM

Table 5: Controller IP Port Description

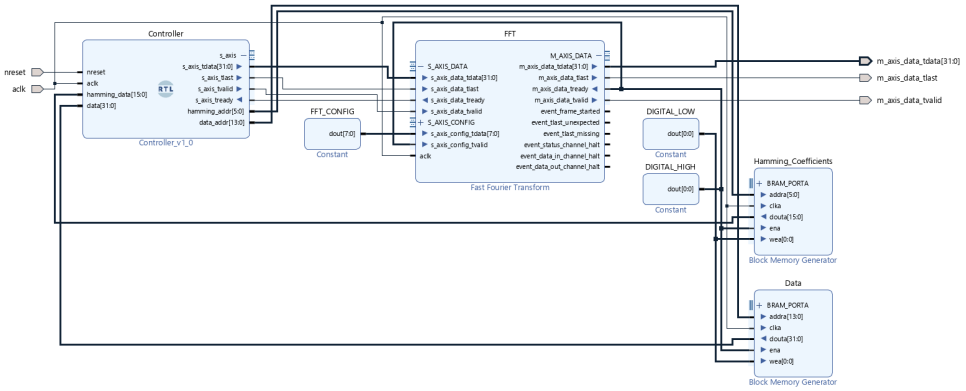


Figure 16: Hamming Window and FFT integrated together

4 Results

Each of the block diagrams is verified via simulation in Vivado. For this an HDL wrapper is created for each of these designs, which can be easily done in Vivado. After creating the HDL wrapper the designs are simulated in Vivado.

Note: You may have to zoom in to see the results clearly

- **Hamming window**

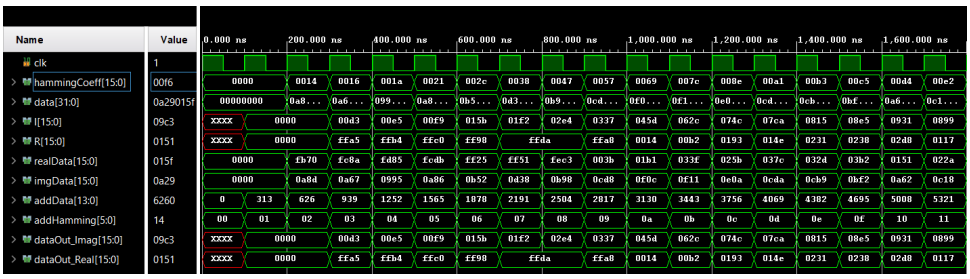
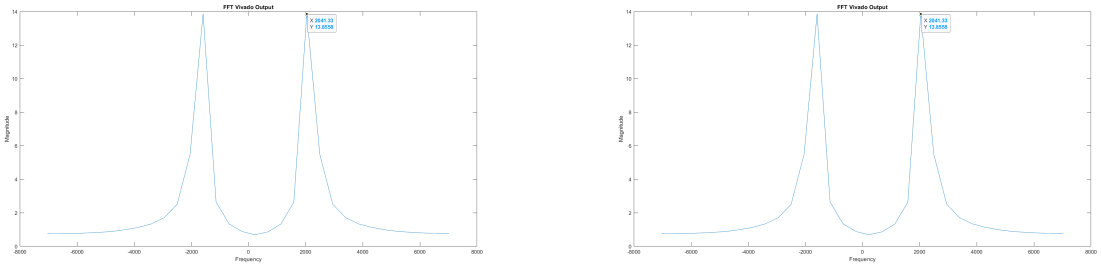


Figure 17: Hamming window simulation

From the simulation it is clear that the data is taken from the BRAM (data) and is splitted into its real (realData) and imaginary part (imgData), then multiplied with the hamming window coefficient (hammingCoeff) to get the windowed output (dataOut_Imag and dataOut_real).

- **Fast Fourier Transform**
The data output of the fft core is imported into matlab as text file for plotting. The result obtained is as shown in figure (18).



(a) Output obtained using FFT IP core

(b) FFT computed usign matlab

Figure 18: Fast Fourier Transform simulation result.

The fft comupted using the fft core and matlab exactly matches with each other.

- **FFT integrated with hamming window**

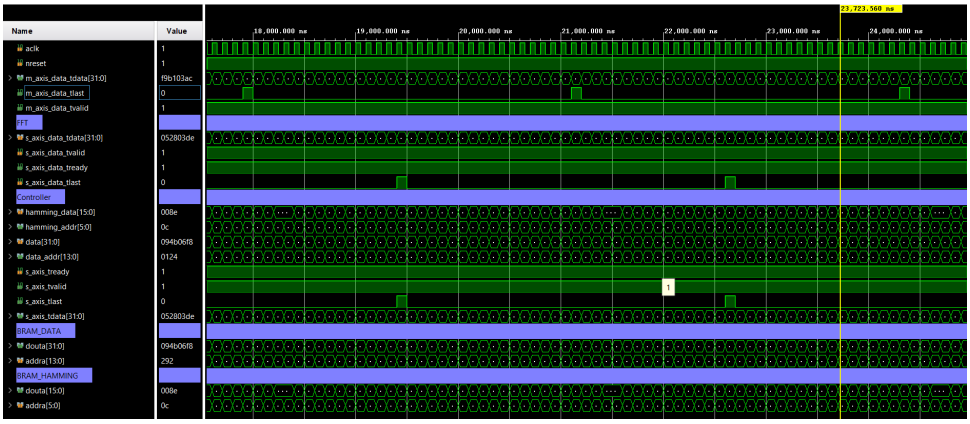


Figure 19: FFT and hamming window single frame output

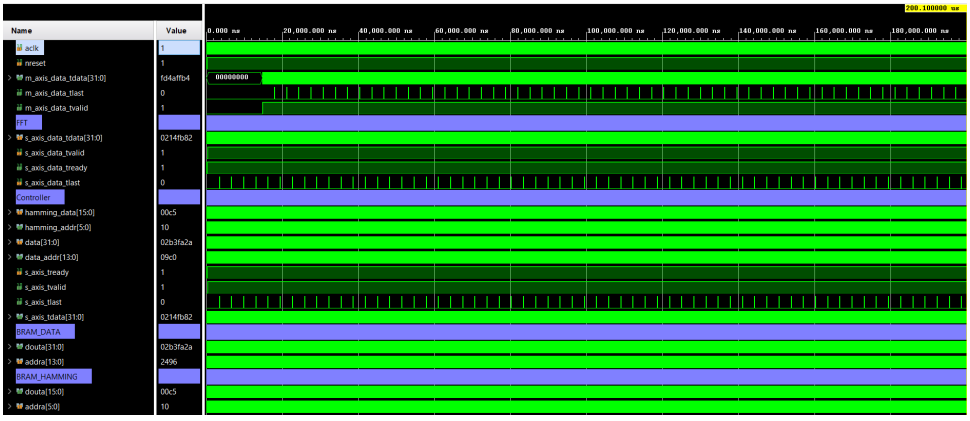


Figure 20: FFT and hamming window Fultiple frames

The dimension of the data matrix used is 40×313. FFT of the 313 slow time samples is successfully calculated. m_axis_data_tlast indicates last sample of each output frame. For 313 frames the m_axis_data_tlast will be high 313 times.

5 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.FFT of the slow time samples is taken

using Fast Fourier Transform IP core and is verified via simulation. Other components such as Matched Filtering, 3-pulse canceler are to be implemented in the next stage.

6 References

[1] Merrill I. Skolnik, (2001). Introduction To Radar Systems (3nd ed.). Tata McGraw-Hill Edition 2001.

[2] Xilinx User Guide for ZC702 Evaluation Board for the Zynq-7000 XC7Z020 SoC.

[3] Analog Devices AD-FMCOMMS3-EBZ User Guide.

[4] Fundamentals of Radar Signal Processing, Mark A Richards.

[5] Block Memory Generator v8.3 LogiCORE IP Product Guide.

[6] Fast Fourier Transform v9.1 LogiCORE IP Product Guide

Note: Link to the github repo is attached here.