MASTER THESIS 31 August 2016

DESIGN AND IMPLEMENTATION OF AN FMCW RADAR SIGNAL PROCESSING MODULE FOR AUTOMOTIVE APPLICATIONS

Suleyman Suleymanov

Faculty of Electrical Engineering, Mathematics and Computer Science
Computer Architecture for Embedded Systems

EXAMINATION COMMITTEE

Prof.dr.ir. M.J.G. Bekooij
Prof.dr.ir. G.J.M. Smit
Ir. J.Scholten
V.S. El Hakim, M.Sc.

UNIVERSITY OF TWENTE.

Abstract

In the recent years, the radar technology, once used predominantly in the military, has started to emerge in numerous civilian applications. One of the areas that this technology appeared is the automotive industry. Nowadays, we can find various radars in modern cars that are used to assist a driver to ensure a safe drive and increase the quality of the driving experience. The future of the automotive industry promises to offer a fully autonomous car which is able to drive itself without any driver assistance. These vehicles will require powerful radar sensors that can provide precise information about the surrounding of the vehicle. These sensors will also need a computing platform that can ensure real-time processing of the received signals.

The subject of this thesis is to investigate the processing platforms for the real-time signal processing of the automotive FMCW radar developed at the NXP Semiconductors. The radar sensor is designed to be used in the self-driving vehicles.

The thesis first investigates the signal processing algorithm for the MIMO FMCW radar. It is found that the signal processing consists of the three-dimensional FFT processing. Taking into account the algorithm and the real-time requirements of the application, the processing capability of the Starburst MPSoC, 32 core real-time multiprocessor system developed at the University of Twente, has been evaluated as a base-band processor for the signal processing. It was found that the multiprocessor system is not capable to meet the real-time constraints of the application.

As an alternative processing platform, an FPGA implementation of the algorithm was proposed and implemented in the Virtex-6 FPGA. The implementations uses pre-built Xilinx IP cores as hardware components to build the architecture. The architecture also includes a MicroBlaze core which is used to generate the artificial input data for the algorithm and manage the operation of hardware components through software.

The results of the implementation show that the architecture can provide reliable outputs regarding the range, velocity and bearing information. The accuracy of the results are limited by the range, velocity and angular resolu-

tion which are determined by the specific parameters of the RF front-end and the designed waveform pattern. However, the real-time performance on the architecture cannot be achieved due to the high latencies introduced by the memory transpose operations. A few techniques have been tested to decrease the latency bottleneck caused by the SDRAM transpose processes, however none of them have shown any significant improvements.

Contents

List of Figures

List of Tables

List of Acronyms

ADC Analog to Digital Converter

AXI Advanced eXtensible Interface

CAES Computer Architecture for Embedded Systems

CLB Configurable Logic Block

CPU Central Processing Unit

CW Continuous Wave

DDR Double Data Rate

DFT Discrete Fourier Transform

DMA Direct Memory Access

DSP Digital Signal Processing

DVI Digital Visual Interface

FFT Fast Fourier Transform

FIFO First-In First-Out

FMCW Frequency Modulated Continuous Wave

FPU Floating Point Unit

FPGA Field-Programmable Gate Array

LMB Local Memory Bus

MIMO Multiple Input Multiple Output

MPSoC Multiprocessor System-on-Chip

NoC Network on Chip

RF Radio Frequency

SDRAM Synchronous Dynamic Random-Access Memory

SODIMM Small Outline Dual In-line Memory Module

TDM Time-Division Multiplexing

UART Universal Asynchronous Receiver/Transmitter

WCET Worst Case Execution Time

Chapter 1

Introduction

1.1 Context

For a long time radars have been used in multiple military and commercial applications. The development of the ideas that lead to the radar systems emerged in the late nineteenth and early twentieth centuries. However, the main developments of the system have been seen during the Second World War. During that period radars were extensively used for air defence purposes such as long-range air surveillance and short-range detection of low altitude targets. In the post-war period, improvements had been made in the development of the radar technology for both the military and civilian applications. Major civilian applications of the radar that emerged during that period were the weather radar and the air-traffic control radar that used to ensure the safety of the air traffic in the airports [2].

Recently, applications of radars in the automotive industry have started to emerge. High-end automobiles already have radars that provide parking assistance and lane departure warning to the driver [3]. Currently, there is a growing interest in the self-driving cars and some people consider it to be the main driving force of the automotive industry in the coming years. With the start of the Google's self-driving car project, the progress in this area has got a new acceleration.

Self-driving cars offer a totally new perspective on the application of the radar technology in the automobiles. Instead of only assisting the driver, the new automotive radars should be capable of taking an active role in the control of the vehicle. As a matter of fact, they will be a key sensor of the autonomous control system of a car.

Radar is preferred over the other alternatives such as sonar or lidar as it is less affected by the weather conditions and can be made very small to

decrease the effect of the deployed sensor to the vehicle's aerodynamics and appearance. The Frequency Modulated Continuous Wave (FMCW) radar is a type of radar that offers more advantages compared to the others. It ensures the range and velocity information of the surrounded objects to be detected simultaneously. This information is very crucial for the control system of the self-driving vehicle to provide a safe and collision-free cruise control.

A radar system installed in a car should be able to provide the necessary information to the control system in real-time. It requires to have a base-band processing system which is capable of providing enough computing power to meet the real-time system requirements. The processing system performs digital signal processing on the received signal to extract the useful information such as range and velocity of the surrounded objects. One of the platforms that can achieve this task is a multiprocessor system-on-chip (MPSoC) which uses multiple processors to increase the computational power.

The Starburst multiprocessor system has been developed at the Computer Architecture for Embedded Systems (CAES) group of the University of Twente. This system is used to carry out research on real-time design and analysis. It is prototyped on a Xilinx ML605 development board which hosts a Virtex-6 FPGA and several peripheral devices such as DDR3 SDRAM, Ethernet and UART interface. The main processing element of the Starbust is Xilinx's soft processor core - MicroBlaze. A number of MicroBlaze cores are connected through Network-on-Chip (NoC) with a ring topology which provides arbitration for all the processing elements connected to it. The platform also supports hardware accelerator integration to improve it's computing capabilities [4].

The aim of this thesis is to analyze the Starburst platform from the perspective of the requirements of the FMCW radar signal processing and propose an alternative architecture if it fails to meet the real-time requirements. First, a theoretical study on the MIMO FMCW radar signal processing will be performed, second, computational requirements of the algorithm will be analyzed and based on the requirements a platform for the implementation will be chosen, third, a signal processing architecture will be designed and implemented, finally, the tests will be performed and the results will be analyzed.

1.2 FMCW Radar Fundamentals

This section introduces the basics of radar systems and gives a brief introduction to the FMCW type radar. In addition, the basic working principle

of the FMCW radar is discussed and some application examples are given.

Radar which stands for Radio Detection and Ranging, is a system that uses electromagnetic waves to detect and locate objects. A typical radar system consists of a transmitter, receiver and a signal processing module. Initially, the transmitter antenna radiates electromagnetic energy in space. If there is an object within the range of the antenna, it will intercept some of the radiated energy and reflect it in multiple directions. Some of the reflected electromagnetic waves will be returned and received by the receiver antenna. After amplification and some signal processing operations, target information such as distance, velocity and direction can be acquired [2].

Nowadays, radars are used for many different purposes. The applications of radars include but are not limited to surveillance, object detection and tracking, area imaging and weather observation. Each type of radar requires the radar sensor to have specific features which can deliver useful information to the user [2]. In case of automotive radars, the radar sensor should provide the range and the relative velocity information of the surrounded objects to the driver with a high accuracy and resolution. In addition, the sensor is desirable to be smaller in size and lower in cost. Currently, FMCW radar is the most common radar type used for this purpose [5].

FMCW radar is a type of Continuous Wave (CW) radars in which frequency modulation is used. The first practical application of this type of radar emerged in 1928, when it was patented by J.O.Bentley to be used on airplane altitude indicating system. Industrial applications of this radar started to appear at the end of the 1930s, after exploitation of the ultra-high frequency band. In the following years, FMCW radar had been applied in the number of civilian and military applications in which estimation of the range with a very high accuracy was crucial. Few examples of these systems are vehicle collision avoidance systems, radio altimeters and the systems to measure the small motion changes caused by vibrations of various components of machines and mechanisms [6].

The theory of operation of FMCW radar is simple. FMCW radar sends a continuous wave with an increasing frequency. A transmitted wave after being reflected by an object is received by a receiver. Transmitted and received signals are mixed (multiplied) to generate the signal to be processed by a signal processing unit. The multiplication process will generate two signals; one with a phase equal to the difference of the multiplied signals, and the other one with a phase equal to the sum of the phases. The sum signal will be filtered out and the difference signal will be processed by the signal processing unit [7]. The block diagram of the radar sensor can be seen in the Figure 1.1.

FMCW radar offers a lot of advantages compared to the other types of

radars. These are [6]:

- Ability to measure small ranges with high accuracy
- Ability to measure simultaneously the target range and its relative velocity
- Signal processing is performed at relatively low frequency ranges, considerably simplifying the realization of the processing circuit
- Functions well in many types of weather and atmospheric conditions as rain, snow, humidity, fog and dusty conditions
- FMCW modulation is compatible with solid-state transmitters, and moreover represents the best use of output power available from these devices
- Small weight and small energy consumption due to absence of high circuit voltages

The FMCW radar signal processing requires Fast Fourier Transform (FFT) algorithm to be implemented. More detailed coverage of this topic will be presented in Chapter 2.

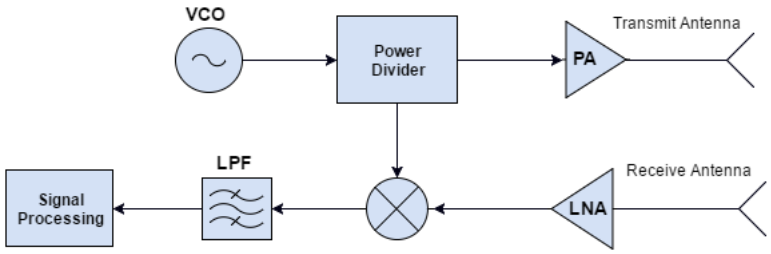

Figure 1.1: FMCW radar block diagram

1.3 Research Platform

This section introduces the Starburst MPSoC and the hardware platform on which the radar application will be implemented.

The hardware platform ton which the application will be implemented is Xilinx's ML605 development board (Figure 1.2). The board is equipped with a Virtex-6 FPGA which contains 241,152 logic cells, 37,680 configurable logic blocks (CLBs) and 416 36 Kb block RAM (BRAM) blocks. Additionally, the board contains several peripherals such as 512 MB DDR3 SODIMM SDRAM, an 8-lane PCI Express interface, a tri-mode Ethernet PHY, general purpose I/O, DVI output and a UART interface [8]. Currently, the platform is used for the development and testing of the Starburst MPSoC.

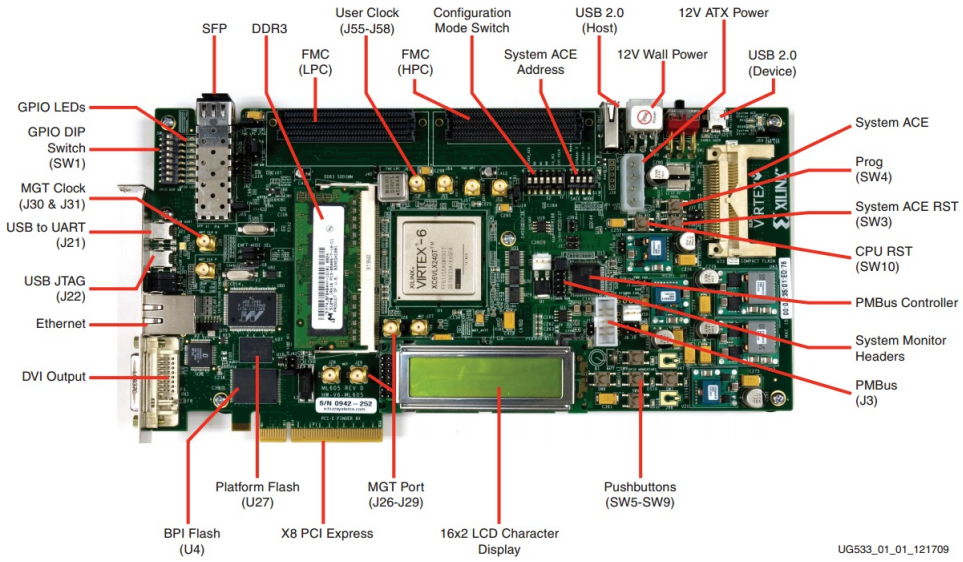

Figure 1.2: Xilinx ML605 development board

The Starburst MPSoC consists of number of processing tiles connected through Network on Chip. Currently, the platform supports up to 32 processing cores and a Linux core to provide an easy interaction with a host PC. In addition, the platform also supports hardware accelerator integration.

The main processing tile of the Starburst is a MicroBlaze, the soft processor core developed by Xilinx. The MicroBlaze is highly configurable soft-core processor that can be implemented using FPGA logic. It is based on Harvard CPU architecture and has a 5 stage single issue instruction pipeline. It has additional hardware support for number of operations such as floating point processing, division, multiplication and bit shifting. In addition, MicroBlaze

has a local memory and a scratchpad memory which sizes are reconfigurable at design time. Both memories are connected to MicroBlaze through Local Memory Bus (LMB), and can be accessed from local MicroBlaze core, although, the scratchpad memory is also connected to the ring interconnect and can accept data from it. All the processors run a real-time POSIX compatible micro-kernel called Helix which supports the newlib C library and implements the Pthread standard.

The communication network of Starburst consists of two parts. The first one is the Nebula ring interconnect which supports all to all communication between processing tiles and hardware accelerators. The ring is unidirectional and has an arbitration policy based on ring slotting which prevents the occurrence of starvation. Each processing tile is connected to a router via a Network Interface and each router is connected to its two neighbouring routers which makes a ring-like structure. The processors are processing the stream of data and can transfer their computation results to other processors connected to the ring. The communication between processors is achieved through C-FIFO algorithm which allows arbitrary number of simultaneous streams between processor tiles. The second communication network is the Warpfield arbitration tree which provides a communication to the shared resources such as UART, DVI and SDRAM. The access to the resources is given on a first-come-first-served basis.

The Starburst MPSoC allows a number of CPUs to run in parallel to achieve a high computation power. Additional support of hardware accelerators allows to improve the performance for the applications which are limited by the computational power of MicroBlaze cores. The resulting heterogeneous MPSoC is an important research and development platform for the stream processing applications which also allows real-time multiprocessor system analysis [4].

1.4 Problem Description

Recent developments in the digital electronics has led to the major improvements in number of areas. Novel microwave transmitters are capable of generating extremely high frequency signals in real time which allows the usage of these high frequency signals in numerous applications. Recently, number of automotive radar chips have emerged which take advantage of the mm-Wave band such as 77 Ghz and 79 Ghz [3].

Earlier this year, NXP Semiconductors introduced it's 77 GHz single-chip radar transceiver (Figure 1.3) which is based on multiple-input multiple-output (MIMO) FMCW principle. The chip is planned to be used in self-

driving vehicles such as self-driving cars. Currently, researchers at NXP Semiconductors are working on the development of the base-band processor for the above mentioned chip.

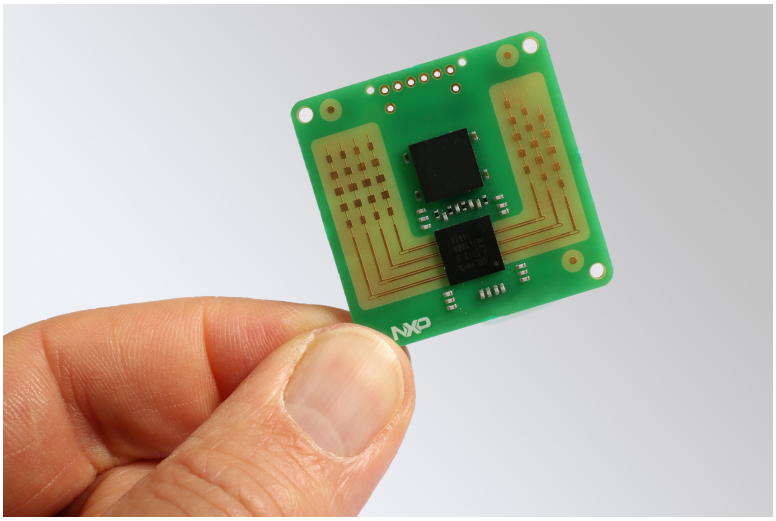

Figure 1.3: NXP Semiconductor's automotive radar chip

This thesis works as a supportive research to test concepts of the Starburst MPSoC to be used in a base-band processor. The main aim of this research is to analyse the computational and real-time requirements of the FMCW radar application and extend the Starbust MPSoC platform accordingly to support the MIMO FMCW radar signal processing.

The main research objectives for the thesis are:

- Research the theory of the MIMO FMCW radar signal processing and evaluate the proposed signal processing architectures.
- Propose the efficient architecture for the Starburst platform to support the FMCW radar application.
- Propose and implement a new architecture in case Starburst cannot achieve the real-time computational requirements for the application.

Chapter 2

FMCW Signal Processing

This section consists of two main parts; the first part explains the FMCW signal processing scheme and the second part introduces the MIMO radar concept.

2.1 FMCW Signal Analysis

There are several different modulations that are used in FMCW signals such as sawtooth, triangle and sinusoidal. In our case, we will consider a sawtooth model of the FMCW signal, seen in the Figure 2.1;

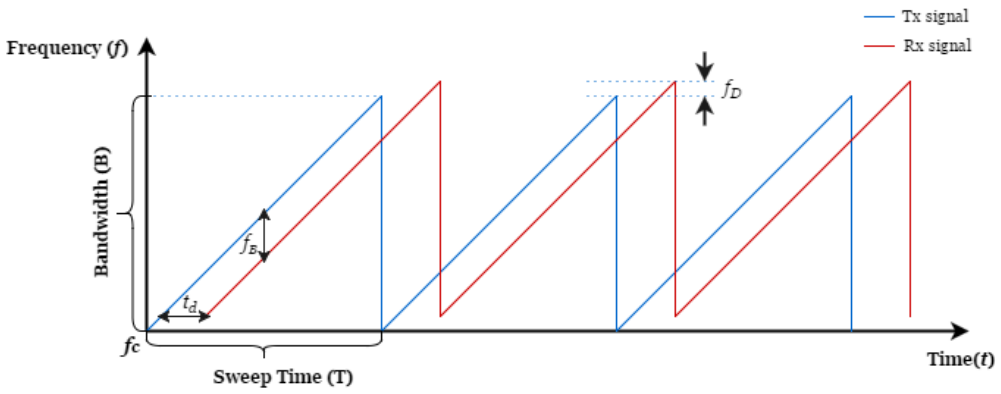

Figure 2.1: FMCW sawtooth signal model

As it can be seen, transmitted frequency increases linearly as a function of time during Sweep Repetition Period or Sweep Time (T). Starting frequency is f_c, which is 79 GHz in our calculations. Frequency at any given time t can

be found by:

$$f(t) = f_c + \frac{B}{T}t \tag{2.1}$$

Here, $\frac{B}{T}$ is a chirp rate and can be thought as a "speed" of the frequency change. We can substitute it with α:

$$\alpha = \frac{B}{T} \tag{2.2}$$

By using frequency change over time, we can find the instantaneous phase:

$$\mu(t) = 2\pi \int_0^t f(t)dt + \mu_0 = 2\pi(f_c t + \frac{\alpha t^2}{2}) + \varphi_0 \tag{2.3}$$

Therefore, the transmitted signal in the first sweep, considering φ_0 to be the initial phase of the signal, can be written as:

$$x_{tx}(t) = A\cos(\mu(t)) = A\cos(2\pi(f_c t + \frac{\alpha t^2}{2}) + \varphi_0) \tag{2.4}$$

The equation above only describes the transmitted signal in the first sweep. If we want to describe the transmitted signal in the n^{th} sweep, a modification should be made. We can consider t_s as a time from the start of n^{th} sweep and define t as:

$$t = nT + t_s \quad \text{where} \quad 0 < t_s < T \tag{2.5}$$

Therefore, our signal form for the transmitted signal in the n^{th} sweep becomes:

$$x_{tx}(t) = A\cos(\mu(t)) = A\cos(2\pi(f_c(nT + t_s) + \frac{\alpha t_s^2}{2}) + \varphi_0) \tag{2.6}$$

Let's consider an object located at an initial distance of R which is moving with a relative velocity of v. The returned signal from the object will have the same form, but with some delay τ which can be defined as:

$$\tau = \frac{2(R + vt)}{c} = \frac{2(R + v(nT + t_s))}{c} \tag{2.7}$$

Considering the delay τ, we can describe the returned signal as:

$$x_{rx}(t) = B\cos(\mu(t-\tau)) = B\cos(2\pi(f_c(nT+t_s-\tau)+\frac{\alpha(t_s-\tau)^2}{2})+\varphi_0) \tag{2.8}$$

According to the FMCW radar principle, the returned signal is mixed with the transmitted signal:

$$x_m(t) = x_{tx}(t)x_{rx}(t) \tag{2.9}$$

The equation above will include cosine multiplication which can be transformed using the trigonometric formula below:

$$\cos(\alpha)\cos(\beta) = (\cos(\alpha+\beta) + \cos(\alpha-\beta))/2 \tag{2.10}$$

The sum term in our case will have a very high frequency ($2 \cdot f_c = 158GHz$) which will be filtered out. Therefore, the resulting signal will only include the subtraction term:

$$x_m(t) = \frac{AB}{2}\cos(2\pi(f_c(nT+t_s) + \frac{\alpha t_s^2}{2} - f_c(nT+t_s-\tau) - \frac{\alpha(t_s-\tau)^2}{2}) \tag{2.11}$$

After simplification we get:

$$x_m(t) = \frac{AB}{2}\cos(2\pi(f_c\tau + \alpha\tau t_s - \frac{\alpha\tau^2}{2})) \tag{2.12}$$

If we replace τ with its equivalent from Equation 2.7, we will get:

$$\begin{aligned} x_m(t) = \frac{AB}{2}\cos(2\pi(f_c\frac{2(R+v(nT+t_s))}{c} + \alpha t_s\frac{2(R+v(nT+t_s))}{c} \\ -\alpha\frac{4(R+v(nT+t_s))^2}{2c^2})) \end{aligned} \tag{2.13}$$

We can simplify and write the equation as:

$$\begin{aligned} x_m(t) = \frac{AB}{2}\cos(2\pi((\frac{2\alpha R}{c} + \frac{2f_cv}{c} + \frac{2\alpha vnT}{c} - \frac{4\alpha Rv}{c^2} - \frac{4\alpha nTv^2}{c^2})t_s \\ +(\frac{2f_cv}{c} - \frac{4\alpha Rv}{c^2})nT + \frac{2f_cR}{c} + \frac{2\alpha vt_s^2}{c} - \frac{2\alpha R^2}{c^2} \\ -\frac{2\alpha v^2n^2T^2}{c^2} - \frac{2\alpha v^2t_s^2}{c^2})) \end{aligned} \tag{2.14}$$

If we look at the Equation 2.14, we see that there is a frequency and a phase that influences how the signal changes over time. In the literature, the frequency is usually named as a "beat frequency". The difference in frequency between the transmitted and the received signals is denoted by f_B in the Figure 2.1. The above equation shows that the "beat frequency" is affected by number of terms such as initial range to the object, object's velocity and the chirp number.

According to the Matlab model provided, the following values are used for the parameters:

Parameter	Value
B	1GHz
T	35.6 μs
f_c	79 GHz
c	$3 \cdot 10^8$ m/s
Number of chirps	96
Number of samples per chirp	1024
Number of Tx antennas	3
Number of Rx antennas	4

Table 2.1: Parameter table

If we assume an object at a distance of 15 m ($R = 15$) which is moving with a velocity of 10 m/s ($v = 10$), and assuming t_s equal to T and n to be 50, we can find how the individual expressions in the equation affect the final value of $x_m(t)$:

$$\begin{aligned} x_m(t) = \frac{AB}{2}\cos(2\pi((2.81 \cdot 10^6 + 5.26 \cdot 10^3 + 3.33 \cdot 10^3 - 0.1873 \\ -2.22 \cdot 10^{-4})t_s + (5260 - 0.19)nT + 7.9 \cdot 10^3 \\ +0.0024 - 0.1404 - 1.97 \cdot 10^{-7} - 7.9 \cdot 10^{-11})) \end{aligned} \tag{2.15}$$

Few observations can be made based on the equation above; first, we see that the values of the expressions $\frac{4\alpha Rv}{c^2}$ and $\frac{4\alpha nTv^2}{c^2}$ are very small and can easily be neglected. Apart from that, the terms $\frac{2f_cv}{c}$ and $\frac{2\alpha vnT}{c}$ are relatively small and their effect to the main frequency component $\frac{2\alpha R}{c}$ can be considered negligible. Second, other terms which have c^2 in their denominators are also very small and can be neglected too. Third, the term with t_s^2, $\frac{2\alpha vt_s^2}{c}$ is also very small (0.0024) and can be neglected as well.

Consequently, $x_m(t)$ equation can be approximated as:

$$x_m(t_s, n) = \frac{AB}{2}\cos(2\pi(\frac{2\alpha R}{c}t_s + \frac{2f_cvn}{c}T) + \frac{4\pi f_cR}{c}) \tag{2.16}$$

where the term $\frac{4\pi f_cR}{c}$ is a constant phase term, since R is an initial distance at which the object is located.

The frequency spectrum of the signal computed over one modulation period will give us $\frac{2\alpha R}{c}$ as a main frequency component which is the beat frequency. The derivation of the beat frequency is usually based on the Fast Fourier Transform (FFT) algorithm which efficiently computes the Discrete Fourier Transform (DFT) of the digital sequence. Consequently, by applying the FFT algorithm over one signal period, we can easily find the beat

frequency (2.17) and thus the range to the target:

$$f_b = \frac{2\alpha R}{c} \quad \text{and} \quad R = \frac{f_b c}{2\alpha} \tag{2.17}$$

Range resolution of a radar is the minimum range that the radar can distinguish two targets on the same bearing [9]. Based on the above equation and substituting α with Equation 2.2, we can find the range resolution of a radar. It is based on the fact that the frequency resolution Δf_b of the mixed signal is bounded by the chirp frequency ($\Delta f_b \geq \frac{1}{T}$) which means that in order to be able to detect two different objects, the frequency difference of the mixed signal returned from that objects cannot be smaller than the chirp frequency. This intuition gives the range resolution which can be found as:

$$\Delta f_b = \frac{2B\Delta R}{c} \cdot \frac{1}{T} \quad \text{and} \quad \Delta R = \frac{c}{2B} \tag{2.18}$$

On the other hand, there is also a phase ($\frac{2f_c v}{c} \cdot nT$) associated with the beat frequency which changes linearly with the number of sweeps. The change of the phase indicates how the frequency of the signal changes over consequent number of periods. This change is based on the Doppler frequency shift which is the shift in frequency that appears as a result of the relative motion of two objects. The Doppler shift can be used to find the velocity of the moving object:

$$f_d = \frac{2f_c v}{c} \quad \text{and} \quad v = \frac{f_d c}{2f_c} \tag{2.19}$$

The Doppler shift of the signal can be found by looking at the frequency spectrum of the signal over n consecutive periods ($n \cdot T$). In this case, the FFT algorithm is applied on the outputs of the first FFT. Figure 2.2 describes this process; first, the row-wise FFT is taken on the time samples, second, the column-wise FFT is taken on the output of the first FFT. After two dimensional FFT processing, we have a range-Doppler map which contains range and velocity information of the target.

Velocity resolution of a radar is the minimum velocity difference between two targets travelling at the same range of which the radar can distinguish. It can be found in a similar way as the range resolution. Here, the Doppler frequency change over n chirp durations is bounded by the frequency resolution ($\Delta f_d \geq \frac{1}{nT}$). Thus, the velocity resolution can be expressed as:

$$\Delta v = \frac{c}{2f_c} \cdot \frac{1}{nT} \tag{2.20}$$

Another conclusion that can be drawn from the equation is that if we have multiple antennas which are separated by some distance, each of them will

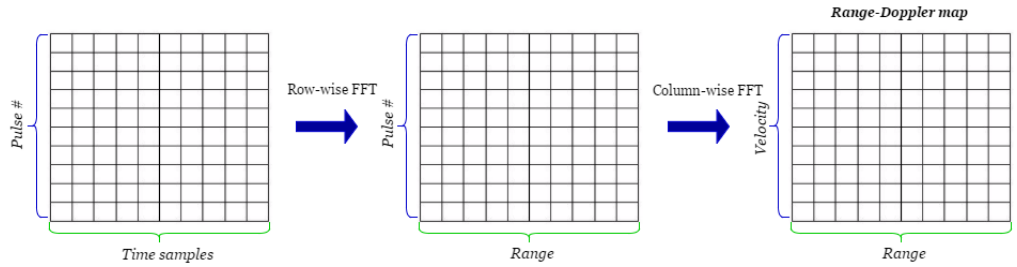

Figure 2.2: FMCW signal 2D FFT processing

have a different phase shift based on the distance. This information can be used to find the angle of arrival of the wave and thus angular position of the target. To achieve that a third FFT can be taken over processed signals from different antennas. Using a phase comparison mono-pulse technique, see Figure 2.3, we can find the phase shift between two array antennas.

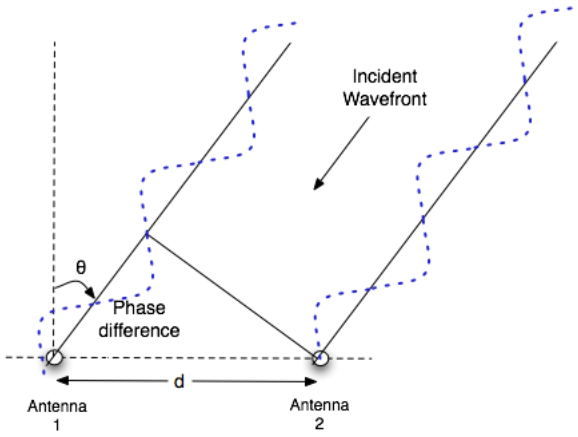

Figure 2.3: Principle of phase interferometry [1]

If antennas are located in distance d from each other, and the angle of arrival of waves is θ, we can find the phase difference through Equation 2.21, where λ is the wavelength of the signal:

$$\Delta\varphi = \frac{2\pi d \sin(\theta)}{\lambda} \tag{2.21}$$

Since 2π phase shift equals to λ and the wave that reaches to antenna 1 travels $d \sin\theta$ more distance, we can find the phase shift associated with that additional travel distance which will give us the equation above. If we consider having K number of equally spaced antennas with distance d, we

can rewrite 2.16 as:

$$x_m(t_s, n, k) = \frac{AB}{2}\cos(2\pi(\frac{2\alpha R}{c}\cdot t_s + \frac{2f_c vn}{c}\cdot T + \frac{dk\sin\theta}{\lambda}) + \frac{4\pi f_c R}{c}) \quad (2.22)$$

where $0 \leq k \leq K-1$ and $1 \leq n \leq N$, and N is the total number of chirps per frame.

2.2 MIMO Radar Concept

Multiple input multiple output (MIMO) radar is a type of radar which uses multiple TX and RX antennas to transmit and receive signals. Each transmitting antenna in the array independently radiates a waveform signal which is different than the signals radiated from the other antennas. The reflected signals belonging to each transmitter antenna can be easily separated in the receiver antennas since orthogonal waveforms are used in the transmission. This will allow to create a virtual array that contains information from each transmitting antenna to each receive antenna. Thus, if we have M number of transmit antennas and K number of receive antennas, we will have $M \cdot K$ independent transmit and receive antenna pairs in the virtual array by using only $M + K$ number physical antennas. This characteristic of the MIMO radar systems results in number of advantages such as increased spatial resolution, increased antenna aperture, higher sensitivity to detect slowly moving objects [10, 11].

2.2.1 MIMO Signal Model

As stated above, signals transmitted from different TX antennas should be orthogonal. Orthogonality of the transmitted waveforms can be obtained by using time-division multiplexing (TDM), frequency-division multiplexing and spatial coding. In the presented case, TDM method is used which allows only a single transmitter to transmit at each time. Considering M number of transmitting antennas and K number of receiving antennas (Figure 2.4), the transmitting signal from i^{th} antenna towards target can be defined as:

$$x_{tx}(t, m) = A\cos(\mu(t) + \frac{2\pi d_t m\sin\theta}{\lambda}) \quad (2.23)$$

where $0 \leq k \leq K-1$ and $0 \leq m \leq M-1$.

The corresponding received signal at j^{th} antenna can be expressed by:

$$x_{rx}(t, m, k) = B\cos(\mu(t-\tau) + \frac{2\pi d_t m\sin\theta}{\lambda} + \frac{2\pi d_r k\sin\theta}{\lambda}) \quad (2.24)$$

and consequently the difference signal can be written as:

$$x_m(t_s, n, m, k) = \cos(2\pi(\frac{2\alpha R}{c} \cdot t_s + \frac{2f_c vn}{c} \cdot T + \frac{d_t m \sin\theta}{\lambda} + \frac{d_r k \sin\theta}{\lambda})) \quad (2.25)$$

The steering vector represents the set of phase delays experienced by a plane wave as it reaches each element in an array of sensors. By using the equations above, we can describe the steering vector of transmitting array as:

$$a_t(\theta) = [1, e^{\frac{-j2\pi d_t sin\theta}{\lambda}}, e^{\frac{-j2\pi d_t 2\sin\theta}{\lambda}}, ..., e^{\frac{-j2\pi d_t (M-1)\sin\theta}{\lambda}}]^T \quad (2.26)$$

and the steering vector of receiving array as:

$$a_r(\theta) = [1, e^{\frac{-j2\pi d_r sin\theta}{\lambda}}, e^{\frac{-j2\pi d_r 2\sin\theta}{\lambda}}, ..., e^{\frac{-j2\pi d_r (K-1)\sin\theta}{\lambda}}]^T \quad (2.27)$$

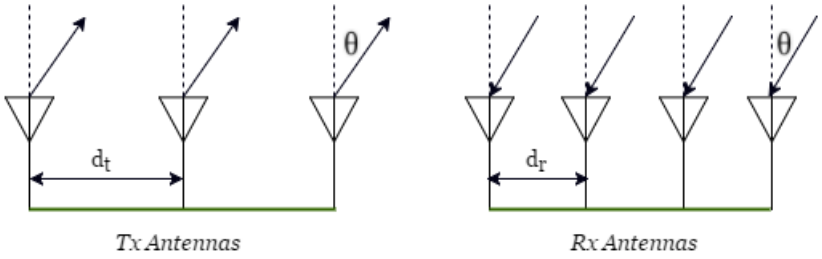

Figure 2.4: TX and RX antennas of MIMO radar

The steering vector of the virtual array (Figure 2.5) can be found by the Kronecker product of the steering vector of transmitting array and the steering vector of receiving array. Kronecker product can be thought as multiplying each element of the first vector with all the elements of the second vector and concatenate all the multiplication results together to form one vector. Kronecker product of two vectors sized $M \times 1$ and $K \times 1$, will result in an $M \times [K \times 1]$ size vector. Thus, steering vector of the virtual array can be expressed by:

$$a_v(\theta) = a_t(\theta) \otimes a_r(\theta) = [1, e^{\frac{-j2\pi d_r sin\theta}{\lambda}}, ..., e^{\frac{-j2\pi d_t \sin\theta}{\lambda}}, e^{\frac{-j2\pi (d_t + d_r)\sin\theta}{\lambda}}, ..., e^{\frac{-j2\pi (d_t (M-1) + d_r (K-1))\sin\theta}{\lambda}}]^T \quad (2.28)$$

The vector above contains phase delays that waveform experiences in its path from each transmitting antenna to each receiving antenna. It can be used to find the angular position of the object which can be expressed as:

$$P(\theta) = \sum_{l=0}^{L-1} X_l(f) \cdot a_v^l(\theta) = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} X_{m,n}(f) \cdot e^{\frac{-j2\pi (d_t m + d_r k)\sin\theta}{\lambda}} \quad (2.29)$$

where L is the number of elements in the virtual array and $X_l(f)$ refers to the spectrum of the signal in the l^{th} virtual array element and $a_v^l(\theta)$ refers to the l^{th} element of the steering vector. Intuitively, the formula above finds the amplitudes (gains) associated with the angle of arrivals (AOA) in the whole imaging area. It can be thought as finding a frequency spectrum of a time-domain signal where frequency corresponds to direction and time samples correspond to space samples:

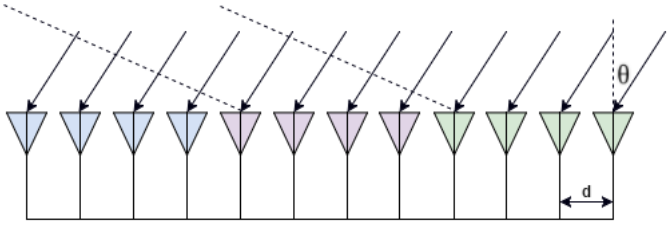

Figure 2.5: Virtual antenna array

Consequently, assuming antennas in the virtual array uniformly spaced and distance between two antennas is d, we can find the relation between θ and virtual array as:

$$\sum_{l=0}^{L-1} X_l(f) \cdot e^{\frac{-j2\pi sl}{L}} = \sum_{l=0}^{L-1} X_l(f) \cdot e^{\frac{-j2\pi dl \sin\theta}{\lambda}} \tag{2.30}$$

where the range of s is $1 \leq s \leq L$.

The left side of the Equation 2.30 is the Discrete Fourier Transform and the right side is the Equation 2.29 modified for virtual array representation. The equation above will help us to describe the relation of a virtual antenna number or FFT bin s with AOA (θ):

$$-\frac{j2\pi sl}{L} = -\frac{j2\pi dl \sin\theta}{\lambda} \tag{2.31}$$

which gives us θ expressed as:

$$\theta = \arcsin\frac{s\lambda}{dL} \tag{2.32}$$

Since we want 180^o view, the angle of arrival θ will range from -90^o to 90^o.

Angular resolution of a radar is the minimum angular separation that the radar can distinguish two objects located at the same range. It is determined by the antenna beam width; the smaller the beam width is, the better the angular resolution becomes [9]. The beam width of the antenna is directly

proportional to the the wavelength of the transmitted signal and inversely proportional to the effective antenna aperture. Hence, considering a constant wavelength, increasing the effective antenna aperture will decrease the antenna beam width and increase the angular resolution [12]. This is one of the reasons why the MIMO radar is preferred over other alternatives such as the phased-array radar. It allows us to increase the antenna aperture, thus the angular resolution by using the same number of antennas.

Chapter 3

Requirements

This chapter describes the analysis of the algorithm that the signal processing is based on. The first section describes the Matlab model of the radar signal processing. The second section provides the computational analysis on the FFT algorithm which is the main functional block of the signal processing and gives the requirements for the architecture to be implemented. The next section discusses the architectures proposed in the recent literature. Finally, the last section provides a signal-flow analysis of the radar processing.

3.1 Matlab Model

The Matlab model of the reception part of the radar application was provided by the NXP Semiconductors. The essential part of the code is 3D FFT module which is used to get the frequency domain representation of the received signals from their time and space domain equivalents. Later, the frequency domain representation is used to plot the Range-Doppler spectrum and the bearing information. Provided code had no measurement file that could be used to test the model. To be able to test the radar Matlab function was implemented which generates an input signal based on the MIMO FMCW model presented in Chapter 2. The function implements the Equation 2.25 from Chapter 2 with three transmitting and four receiving antennas. The output of the tested Matlab model can be seen in the Figure 3.1 and 3.2. The input signal was generated considering an object located at 4 m initial distance with 1 rad counter-clockwise angular position and moving with a relative velocity of 4 m/s (14.4 km/h). Figure 3.1 shows the range-Doppler spectrum of the radar. It can be seen that the range of the the target is 4 m and its relative velocity is around 15 km/h. Figure 3.2 shows the relative position of the object with respect to the radar transceiver.

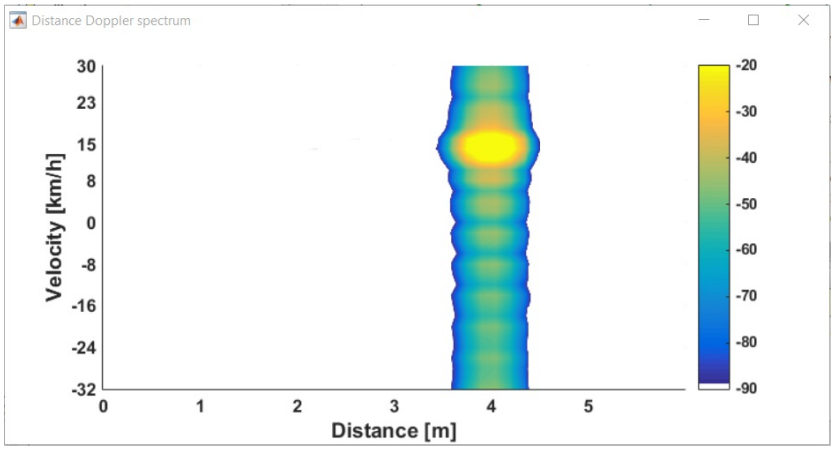

Figure 3.1: Range-Doppler Spectrum

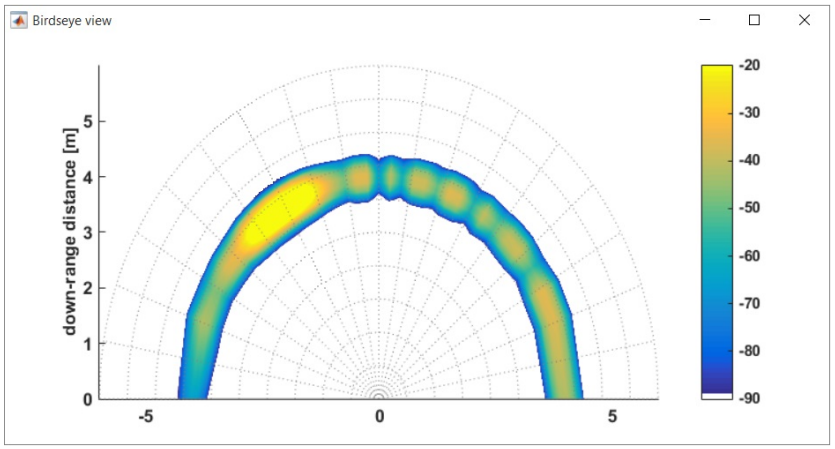

Figure 3.2: Birdseye view

3.2 Computational Analysis

We have seen in Chapter 2 that the main processing block of the radar application is the FFT block. The FFT is a fast algorithm that computes the discrete Fourier transform of the time domain samples x_n:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} \tag{3.1}$$

The algorithm allows to reduce the complexity of the DFT computation from $O(n^2)$ to $O(n \log n)$.

Straightforward Starburst MPSoC implementation of the signal processing would be to use MicroBlaze cores to perform Fast Fourier Transforms.

The platform has enough processing cores to support simultaneous processing of signals coming from multiple receiving antennas. Hence, it should be studied if MicroBlaze cores could provide enough computing power for the FFT processes used in the signal processing while taking into account the real-time constraints of the application.

The computational requirements for the FFT process can provide us with an overview of the required computational power. The analysis of the algorithm shows that N point FFT requires $\frac{N}{2}(\log_2 N)$ number of complex multiplications and $N \log_2 N$ number of complex additions. Taking into account the fact that multiplications in the last stage of the FFT are simply multiplications by 1, we can exclude the multiplication operations in that stage. Therefore, the number of complex multiplications required will be $\frac{N}{2}(\log_2 N - 1)$. Additionally, each complex multiplication contains four real multiplications and two real additions. By combining these two we can express the number of real multiplications (RM) required as:

$$RM = 2N(\log_2 N - 1) \tag{3.2}$$

Similarly, each complex addition contains two real additions. As a result, the number of real additions (RA) can be expressed as:

$$RA = N(\log_2 N - 1) + 2N \log_2 N \tag{3.3}$$

According to the MicroBlaze Reference Guide, the core has a Floating Point Unit (FPU) which supports single-precision floating point arithmetic. As stated in the reference, floating-point addition and multiplication requires 4 clock cycles in non-area optimized mode and 6 clock cycles in area optimized mode. Considering using the single-precision floating point numbers and configuring the MicroBlaze core in non-area optimized mode, we can find the number of clock cycles (NCC) required for the FFT processing as:

$$NCC = 4 \cdot (RM + RA) \tag{3.4}$$

The first FFT stage is very crucial from the perspective that it has a real-time requirement to finish the 1024 point FFT processing in 35.6 μs, since in every 35.6 μs new 1024 samples will be available. By using the Equations 3.2 and 3.3, we can find that the number of real multiplications and additions needed for this FFT; which are 18432 and 29696 respectively. By substituting the values in Equation 3.4, we can find that the number of clock cycles required to finish the FFT equals to 192512. Since the MicroBlaze core in Starburst runs at 100 MHz clock frequency, the time needed to finish the FFT process will be 1925.12 μs. The resulting value gives us the lower bound

for the computation since it only takes into account the actual computation required by the FFT algorithm and excludes the overheads such as variable initializations, function calls, loops and memory accesses. It can be concluded that the result is 54 times larger than the provided chirp time which is 35.6 μs. Consequently, we can conclude that it is not possible to meet the real time requirements by using one MicroBlaze core as an FFT processor.

The calculations show that even if we are able to use fixed-point arithmetic for the FFT process, we are not able to reach the real-time requirement needed. The MicroBlaze reference guide [13] specifies that the integer addition and multiplication take 1 clock cycle to finish. By following the same procedure as above, we can calculate and find that the fixed-point FFT process will take at least 48128 clock cycles (481.28 μs) to finish which is 13.5 times bigger than the requirement.

The analysis above shows that using only the Microblaze processors in the Starburst architecture for base-band processing will not allow to achieve the real-time requirements demanded by the application. Although, the Starbust platform also supports a hardware accelerator integration, the current application does not benefit from it. Therefore, we should consider alternative architectures that can provide better performance characteristics. In the next section we discuss the architecture considerations that can lead to the higher performance.

3.3 Architecture Considerations

We have seen in Chapter 2 that the three dimensional FFT processing can give us the range, velocity and the relative position information of the target. In the previous section, we discussed the computational requirements of the FFT and found out that using MicroBlaze soft cores for FFT processing does not allow us to meet the real-time requirements. Consequently, we concluded that the Starbust architecture is not very useful in terms of meeting the real-time demands of the radar application. This section discusses the architecture that can be used to achieve the real-time performance in the Virtex-6 FPGA.

Implementation of the FMCW signal processing on hardware has been investigated by number of previous works. In [14], the authors provide an FPGA based real-time implementation of range-Doppler image processing. The architecture performs 2D FFT processing by storing the intermediate data in a DDR SDRAM. The authors propose using two DDR SDRAM controllers which control the access to two different SDRAM modules. This prevents any lose of data and allows the data from the second frame to be

written to the second SDRAM while the processed data from the first frame is read from the first SDRAM for the second FFT processing. However, the authors provide no details about the resource usage and the performance of the proposed implementation.

In [15], the authors propose an architecture for range-Doppler processing which supports sampling rates up to 250 MSPS and a maximum of 16 parallel receiving channels. The architecture uses digital down-sampling to enable various sampling frequencies to be used and a low pass FIR filter to suppress the aliasing effects arising from the down-sampling process. Similar to [14], the data after the first FFT processing is stored in the SDRAM. The authors propose to interleave the usage of multiple banks of the SDRAM to improve the data throughput. That is, the outputs of the first FFT block should be distributed over multiple banks. The paper describes an example addressing scheme based on that idea which reduces processor stall cycles. In spite of the fact that the detailed resource usage of the implementation on Virtex-7 FPGA is given, no information on the performance is provided in the paper.

The architecture described in [16] allows a pipelined and parallel hardware implementation of signal processing for an FMCW multichannel radar. The architecture supports a 3D FFT based signal processing algorithm which has been described in Chapter 2. It consists of the FFT processing blocks for range, Doppler and beamforming calculations and the dual-port memory blocks inserted between them to store the intermediate data. In contrast to the architectures described above, this implementation does not use the SDRAM and takes advantage of the FPGA on-chip memory blocks instead. In addition, the authors provide the hardware resource usage of the architecture and the processing time of the algorithm implemented on Virtex-5 FPGA.

Another architecture for the radar signal processing is described in [17]. The RF front end of the design has four transmit and four receive antennas and applies the TDM technique for the transmit signals. This allows sixteen virtual antennas to be synthesized. Consequently, the processing of the received signal is based on the MIMO virtual array concept. The architecture uses an 1D FFT processing to extract the range information from the "beat" signal and a digital beamformer to find the angular information. The implementation of the architecture is based on combined FPGA and DSP pipeline approach. The FFT processing is done on the FPGA side, on the other hand the beamforming algorithm runs on the DSP side. After the processing, the radar image is displayed on the LCD panel which is actuated by the FPGA at a frame rate of 50 Hz. According to the authors, the implementation can achieve a real-time imaging rate of 1.5625 Hz.

To summarize the architectures presented, we can see that there are two

types of architectures for the hardware implementation of the algorithm. The first type uses the off-chip SDRAM to store the intermediate results of the processing. The architectures presented in [14] and [15] are based on this type. In this type it is important to minimize the time required to open and close a page of the SDRAM when accessing the data for the second and the third FFT processings. The second type of architecture uses on-chip FPGA memory blocks to store the intermediate results. This type of architecture is more efficient and can achieve faster processing due to the fact that there is much less overhead in accessing the intermediate data of on-chip FPGA memory blocks rather than the SDRAM. However, it should be noted that this architecture is limited by the amount of available on-chip memory and will bound the number of points used for the FFT processing.

3.4 Signal-flow Analysis

In the Matlab model described in Section 3.1, we have considered only a single radar scanning. From the provided model it is not clear if the radar will start the consecutive scanning immediately after finishing the previous scanning or there will be a time interval between them. Here we consider a model in which the consecutive scannings happen without any time interval (see Figure 3.3). Therefore, we will consider the performance of the implementation to be real-time if it provides enough computational power to process the consecutive radar scannings without any delays.

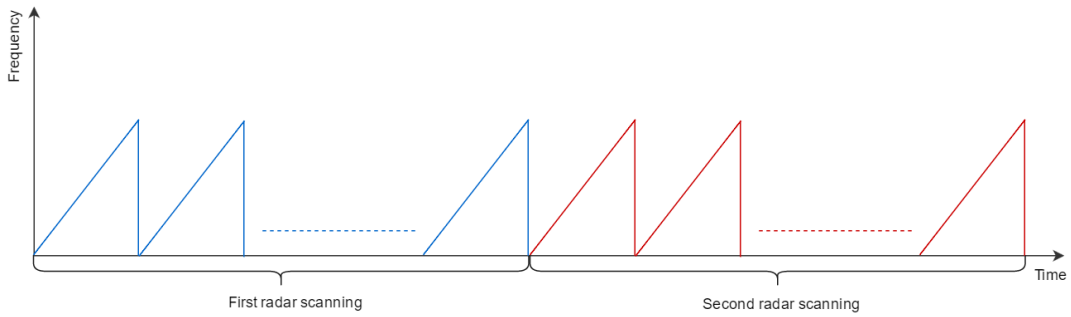

Figure 3.3: Radar scannings

Based on the signal processing architectures described in the previous section and our requirements, we can construct the signal-flow graph of the algorithm. The Figure 3.4 shows the signal-flow graph of the signal processing algorithm for one receiving antenna. After sampling by the 40 MHz ADC and decimation by a factor of 2, the first FFT can be performed. The first

FFT is performed on 1024 time samples from one chirp period. To achieve a real-time performance, the worst case computation time of the first FFT block should be equal to the chirp time which is 35.6 μs based on our model. It means that we can process a frame as soon as it is available, thus avoiding any time delays. In addition, the outputs of the FFT block should be stored in a memory for further Doppler processing. Given the worst-case execution time (WCET) of the FFT block and the number of samples required to be stored in the memory, we can calculate the required minimum bandwidth from the first FFT block to the memory

$$B_1 = \frac{1024}{T_c} = \frac{1024}{35.6 \cdot 10^{-6}} = 28.76\ MS/s \tag{3.5}$$

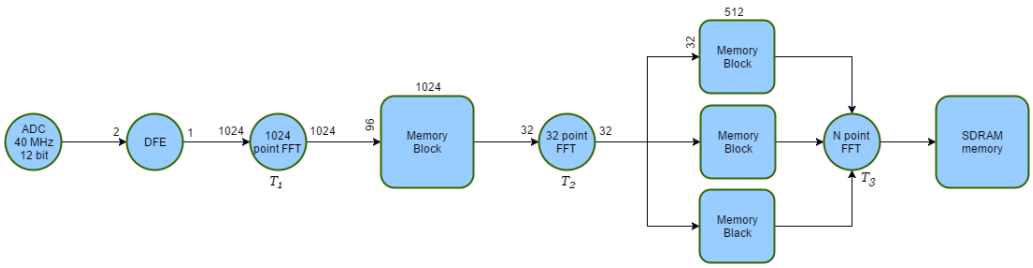

Figure 3.4: Signal Flow Graph of 3D FFT Procesing

The ADC used for the sampling of the received signal has 12 bits of resolution. Knowing that the output of the FFT is a complex-valued number, we can easily calculate the minimal memory required to store the FFT data. Our model uses 96 chirps per frame, thus our memory requirement equals to $96 \cdot 1024 \cdot 2 \cdot 12 = 2359296$ bits $= 294912$ bytes.

The second FFT block computes the column-wise FFT for each transmitting antenna from the stored data. To illustrate, the first row contains the samples from the first transmit antenna, the second row contains the samples from the second one and the third row contains the samples from the third transmit antenna. Similarly, the fourth row will contain the samples from the first transmit antenna too. So, FFT will be performed on samples from the rows 1, 4, 7...91, 94 for the first transmit antenna, 2, 5, 8...92, 95 for the second transmit antenna and 3, 6, 9...93, 96 for the third one. Out of the 1024 columns of the matrix, it is sufficient to process the first 512 columns since the output of the real valued FFT is always symmetric and the second half of the columns will not provide any additional information. Given that we have 512 columns, in total 1536 ($512 \cdot 3$) 32 point FFTs should be performed for the single radar antenna image processing. We know that

the total time for that processing is $n \cdot Tc$, where n is the number of chirps and T_c is the chirp time. Therefore, given the parameters we can find the worst-case computation time for the second FFT:

$$T_2 = \frac{n \cdot T_c}{1536} = \frac{96 \cdot 35.6 \cdot 10^{-6}}{1536} = 2.22\ \mu s \tag{3.6}$$

All the outputs from the FFT block should be stored for the third FFT processing. As it can be seen in Figure 3.4, there are three 2D arrays for the single receiving antenna each of them containing 16384 ($512 \cdot 32$) complex values. We can easily calculate the memory required for each of the arrays which equals to $32 \cdot 512 \cdot 2 \cdot 12 = 393216$ bits = 49152 bytes.

Given WCET of the second FFT block we can find the required minimum bandwidth from block to the memory:

$$B_2 = \frac{32}{T_2} = \frac{32}{2.22 \cdot 10^{-6}} = 14.4\ MS/s \tag{3.7}$$

The third FFT is performed on samples from all Range-Doppler spectrum's. Considering the real-time constraints, the required time to complete all the FFTs equals to $n \cdot Tc$. The number of points that the third FFT performs is based on the equation provided on Matlab model:

$$N = 2^{\lceil \log_2 A*K \rceil} \tag{3.8}$$

where A is the interpolation factor for the Angle of Arrival spectrum and K is the number of virtual antennas. Considering only a single FFT block, worst-case execution time of the block will be:

$$T_3 = \frac{96 \cdot 35.6 \cdot 10^{-6}}{16384} = 0.21\ \mu s \tag{3.9}$$

Consequently, the bandwidth can be found as:

$$B_3 = \frac{N}{0.21 \cdot 10^{-6}} \tag{3.10}$$

Additionally, we can find the memory requirements for the processing. One thing to note is that we need double buffering to prevent the overwriting of the data that is already in the memory. The reason is that while the second FFT stage will be busy performing the column-wise memory reads, writing the received new data to the same memory will cause the previous data to be lost. Therefore, if the calculated latency and bandwidth constraints are met, the double buffering should be sufficient for the real-time performance.

Based on the Figure 3.4, we can calculate that the memory requirement for a single antenna as:

$$MEM = 2 \cdot (294912 + 49152 \cdot 3) = 884736\ bytes \tag{3.11}$$

The application requires to have 4 receiver antennas. We can find that the memory requirement for the receiver with four antennas is around 3.5 MByte (4·884736 bytes). According to the Xilinx Virtex-6 FPGA family documentation, the Virtex-6 FPGA deployed on the ML605 board - XC6VLX240T - has maximum 1.872 MByte block ram capability which is considerably less than the required memory for our application. This requirement adds a constraint of using the off-chip SDRAM to store the intermediate results of the FFT processing.

Furthermore, it should be noted that the above mentioned requirement can change based on the design decisions. To illustrate, if we consider having enough time between consecutive radar scannings and consider using an in-place computation, then the actual minimum memory requirement will be equal to $4 \cdot 96 \cdot 1024 \cdot 2 \cdot 12 = 9437184$ bits = 1.125 MByte. We can see that it is considerably less than the memory available in the FPGA.

However, representing a 12 bit value with 12 bit fixed-point format will not be very reliable as it does not allow any bit growth and might result in serious errors in the calculations. Instead, a common 16 bit fixed-point format can be used for that purpose. We can find that the memory requirement in this case will be equal to $4 \cdot 96 \cdot 1024 \cdot 2 \cdot 16 = 12582912$ bits = 1.5 MByte. It is still less than the available on-chip FPGA memory and can fit in it if the other hardware components require less than 0.372 MByte of on-chip memory.

In the current case, a single-precision floating-point format was used for the implementation. It requires each value to be represented by 32 bits thus, the total memory requirement in this case will be equal to $4 \cdot 96 \cdot 1024 \cdot 2 \cdot 32 = 25165824$ bits = 3 MByte. It is clear that these amount of data cannot fit on on-chip FPGA memory blocks. Therefore, the implementation will require to store the data on off-chip SDRAM.

Chapter 4

System Implementation

The previous chapter presented the analysis of the algorithm and the architectures found in the literature to implement it. This chapter will describe the architecture that is used to implement the algorithm on the Virtex-6 FPGA based on the given requirements. The first section describes the implemented algorithm based on the signal processing scheme described in Chapter 2 and the requirements found in Chapter 3. The second section presents the components or hardware blocks required to implement the processes found in the algorithm. Finally, the last section describes the hardware architecture that has been used to implement the algorithm.

4.1 The algorithm

This section describes the three dimensional FFT processing algorithm on which the signal processing is based on.

The first process in the algorithm is performing 1024 point FFT on the time samples. In Chapter 3 we found that the storage of the intermediate results of the FFT processing should be stored in the off-chip SDRAM. Therefore, the output of the first FFT process must be written to the SDRAM. The second FFT process will read the data from the SDRAM and perform the transform. It was mentioned in Chapter 2 that this process can be thought as a column-wise FFT of a matrix. Thus, all the 512 data samples from the 32 different chirps (rows) will be read at a different time slices. To illustrate, first the first column of data samples will be read from 32 different chirp outputs, second the second column of data samples will be read from the 32 different chirps and so on. This process will continue till all the 512 data samples have been read. Knowing how the modern DRAM memories function, we observe that this is not an efficient way of addressing the SDRAM.

Modern SDRAM memories are usually organized in multiple banks. Each bank has a matrix structure and consists of rows and columns. To access a memory address for reading or writing requires to activate a row which will read the data stored in the row to the row buffer. After activating the row the data can be read or written based on the column addresses. After reading or writing the data, the row will be closed and the data will be written back to the bank. Thus, accessing the memory address requires three operations; activating the row, doing a read or write operation and closing the row. It is clear that it will introduce a huge overhead if the memory is addressed in an arbitrary order.

The ML-605 board contains 512 MB DDR3 SDRAM from Micron Technology (MT4JSF6464HY-1G1B) [8]. The module has 4 chips placed on the board each having 16 bits data output. In addition, the module is organized in 8 internal device banks. Each bank has 8K rows and 1K columns. It is easy to find that each row of the bank can store 8 KByte of data. If we use single-precision floating point representation, each row of a bank will contain a processed FFT data from a single chirp, since each complex-valued number contains 8 Bytes and having 1024 numbers will make 8 KByte. Therefore, the second FFT will require to open and close a row for reading of each sample which will make in total 16384 ($32 \cdot 512$) requests per virtual antenna. This process can add significant delays to the FFT processing time.

One way to overcome this overhead is to transpose the data matrix. We can transpose the data stored in 32x1024 matrix to 1024x32 matrix form. In this way the memory addressing will be in sequential order resulting in less overhead in reading the data from the SDRAM. Thus, we need to have a memory transpose process after finishing the first FFT processing of all chirps from a given frame. After completing the transpose operation, the second FFT can be performed on the data.

According to the requirements, we have 3 transmitting and 4 receiving antennas making in total 12 virtual antennas. After the transpose operation and the second FFT processing, the data will be stored in the memory as in 12x512x32 3D matrix. The third FFT requires the data samples from all virtual antennas. As it can be seen, these data are not located in the consecutive memory locations and will need to open and close a row for each read operation. As it was discussed above, this can add a big overhead. Thus, we need to transpose the memory again make it suitable for the third FFT processing. In this case, the transpose operation will take the 12x512x32 3D matrix and output the 512x32x12 3D matrix. Now, the third FFT can be performed on the data. After finishing the third FFT, the data can be stored in the SDRAM for further processing. At this moment, the range, velocity and the angle information can be extracted from the data.

To summarize, we have seen that the algorithm consists of multiple FFT and transpose operations. The whole process can be described with the following algorithmic flowchart.

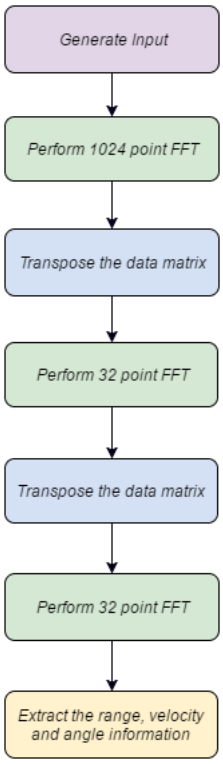

Figure 4.1: Signal processing algorithm flowchart

4.2 The hardware components

This section describes the hardware components used in the architecture implementation. A brief description and function of all the components have been provided.

4.2.1 FFT Core

It can be seen from the algorithm that the FFT is the major process in the realization of the algorithm. In order to reduce the implementation time, the FFT algorithm is implemented using Xilinx LogiCORE IP Fast Fourier Transform v8.0 [18]. The IP core implements the Cooley-Tukey FFT algorithm for the transform sizes of $N = 2^m$ where m ranges between 3 and

16. The core supports processing with fixed-point data ranging from 8 to 34 bits as well as single-precision floating point data. In the latter case, the input data is a vector of N complex values represented as dual 32-bit floating-point numbers with a phase factors represented as 24 or 25-bit fixed point numbers.

The FFT core provides four architecture options;

- Pipelined Streaming I/O
- Radix-4 Burst I/O
- Radix-2 Burst I/O
- Radix-2 Lite Burst I/O

The pipelined streaming architecture pipelines several Radix-2 butterfly processing engines to allow continuous data processing. Each processing engine has its own dedicated memory banks which are used to store the input and intermediate data. This allows the core to simultaneously perform a transform on the current frame of data, load input data for the next frame of data and unload the results of the previous frame of data.

For the current implementation, the pipelined streaming architecture was chosen for the two main reasons. First, the pipelining allows the FFT block to receive the data while it is processing the data from the previous frame. This is convenient for the first FFT processing in our application, since it eliminates the need for buffering of the incoming data and allows the data immediately to be received by the FFT block. Second, the processing latency of the pipelined streaming architecture is much less than the latency of the burst-architectures and meets the latency constraints found in Section 3.4.

The FFT IP core is compliant with the AXI4-Stream interface. All inputs and outputs to the FFT core use the AXI4-Stream protocol. Since the FFT core needs to access to the main memory to read a data, we need an additional hardware block which can access the memory and translate the AXI4-Memory Mapped (AXI4-MM) transactions to AXI4-Stream (AXI4-S) transfers and vice versa. This is achieved by using LogiCORE IP AXI DMA core [19] of Xilinx.

4.2.2 AXI DMA Core

The AXI DMA engine supports high-bandwidth direct memory access between memory and AXI-Stream peripherals. The data movement is achieved through two data channels; Memory-Map to Stream (MM2S) channel and

Stream to Memory-Map (S2MM) channel. Reading a data from the memory is accomplished by AXI4 Memory Map Read Master interface and AXI MM2S Stream Master interface. On the other hand, writing a data to the memory is achieved through AXI S2MM Stream Slave interface and AXI4 Memory Map Write Master interface. The core also has an AXI4-Lite slave interface which is used to access the registers and control the DMA engine.

The DMA core allows maximum 8 MByte of data to be transferred between a memory and a stream peripheral per transaction. According to the documentation [19], the core can achieve high throughput in transfers, namely; 399.04 MByte/s in MM2S channel and 298.59 MByte/s in S2MM channel.

4.2.3 Memory Interface Core

To access an off-chip memory from an FPGA a memory controller is required. Xilinx provides a memory interface core [20] to interface the FPGA designs to DDR3 SDRAM devices. The core handles the memory requests from hardware blocks such as AXI DMA and translates them to SDRAM commands. It allows the data movement between FPGA user designs and the external memory. In addition, the core also manages the refresh operation of the memory.

4.2.4 Microblaze Core

The information about the Microblaze core was provided in Chapter 1. The design uses a single Microblaze core to generate the input data for the algorithm, to configure the AXI DMA blocks for data transfers, to transpose the memory, to measure the time required for each process and to extract the range, velocity and the angle information form the frequency spectrum data.

4.3 The architecture and operation

The hardware components of the architecture were described in the previous section. This section describes how the components are interconnected to each other and how the architecture functions.

It was mentioned in the previous chapters that the RF front of the design has four receiving antennas. By using an FPGA for a signal processing we can achieve a parallel processing of the received signals from all receiving antennas. However, since the RF front end of the design is not yet ready, the architecture also includes an input signal generation as part of it.

The architecture of the implementation can be seen in the Figure 4.2. The figure shows the hardware blocks implemented in the FPGA and the communication channels between that blocks and SDRAM. The widths of the data buses between FFT block, AXI DMA block and Memory Interface Core are 64 bit. The Microblaze core and AXI-Lite channels of the AXI DMA blocks are connected to the 100 MHz clock source. The channels of the FFT cores and the other channels of the AXI DMA blocks run at 200 MHz clock frequency and the main memory runs at 400 MHz clock frequency.

Based on the provided input data, such as range of the target, its velocity and angular information, the Microblaze core generates an input data in single-precision floating-point arithmetic and stores it in the SDRAM. Following that, the Microblaze initializes the AXI DMA block to read the data stored in the SDRAM, transfer it to the first FFT block and writes back the output data from the block. In the design, a single AXI DMA and single FFT block are used for the processing of the whole 3D array. With the current design, having multiple DMA and FFT blocks will not accelerate the processing since all the instructions of the Microblaze run sequentially.

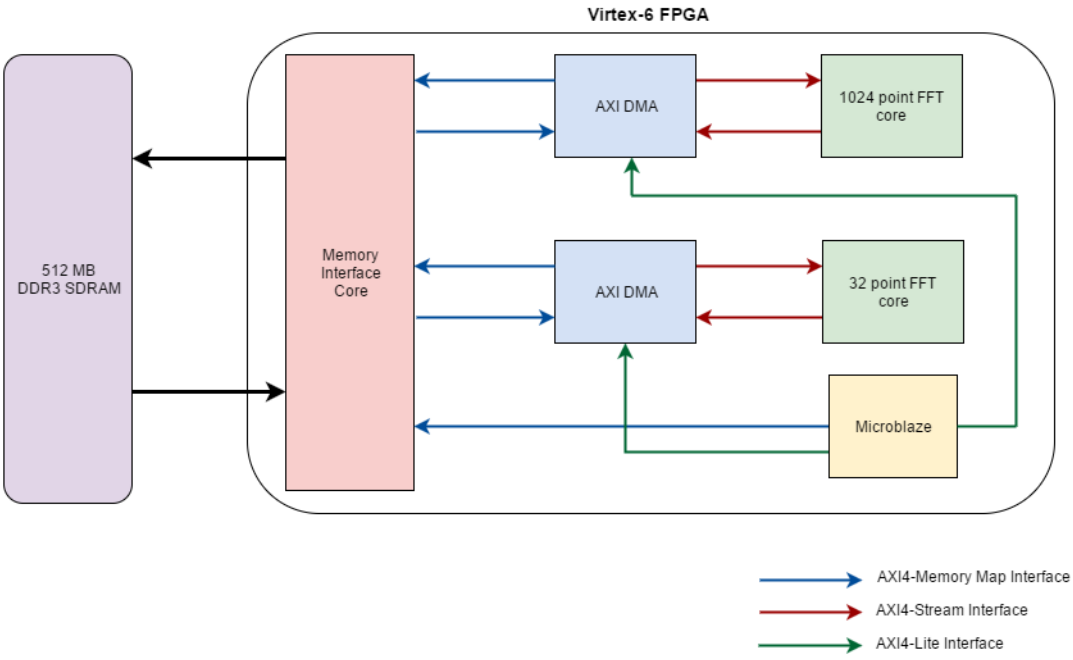

Figure 4.2: The architecture of the implementation

After finishing the first FFT processing, the Microblaze will perform a transpose operation (see Figure 4.3) on the data stored in the SDRAM. As it was mentioned in the previous section, this operation is done using

the Microblaze core by doing the column-wise reads from the SDRAM and row-wise writes to the SDRAM. The output of this operation is a matrix in 12x512x32 3D format. Following that, the Microblaze will instruct the second AXI DMA block to start fetching the data from the SDRAM, transfer it to the second FFT block and write back the results from it. After completion of the second FFT operation, the Microblaze will transpose the data for the last FFT processing. The output of this transpose operation will have 512x32x12 3D matrix format where the third dimension contains the data from all virtual antennas. The Microblaze now can instruct the DMA block to transfer the data for the third FFT process. Although we have 12 data samples available, based on the Equation 3.8 in Chapter 3, 32 point FFT will be performed on the data. This means that the fetched data vector will be padded with zeros before processing. Consequently, the third FFT process can use the same hardware blocks used in the second FFT since they both need the same number of point FFT core. In a similar way, the output of the third FFT will be stored in the SDRAM.

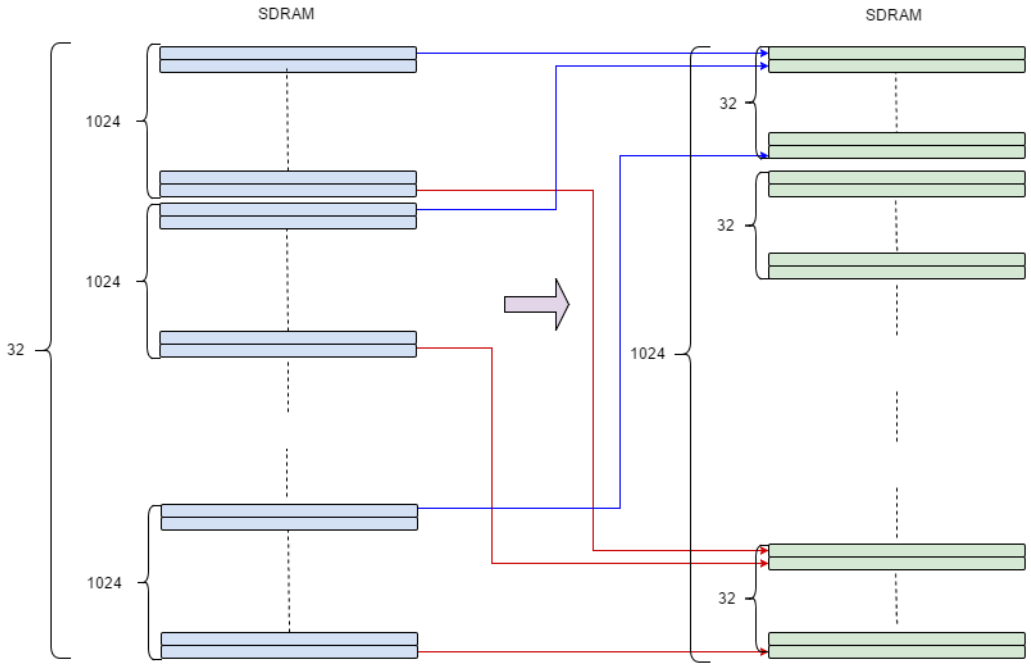

Figure 4.3: An example transpose operation

In this phase, we have a 3 dimensional matrix that contains a frequency spectrum of all the signals from all virtual antennas. The Microblaze now can read the data and process it accordingly to calculate the range, velocity and angle information. This is achieved by using the range, velocity and

angle equations derived in Chapter 2.

The range of a target and it's velocity are found as following; first, 2D FFT processed signal data from the first virtual antenna is taken and it's absolute value is calculated, then, a maximum value in the 2D matrix and it's location were found, following that, based on the row (range bin - r_b) and column (velocity bin - v_b) location information, the range and the velocity of the target were calculated. Based on the Equation 2.17 in Chapter 2, the range can be found as:

$$R = \frac{f_b c}{2\alpha} = \frac{r_b f_s c}{2\alpha N} \tag{4.1}$$

where f_s is the sampling frequency of the mixed signal and equals to:

$$f_s = N/T \tag{4.2}$$

In a similar way, based on the Equation 2.19 in Chapter 2, the velocity of the target can be found as:

$$v = \frac{f_d c}{2f_c} = \frac{v_b f_{ch} c}{2f_c n} \tag{4.3}$$

where f_{ch} is the chirp frequency and equals to:

$$f_{ch} = 1/T \tag{4.4}$$

The angle information is found as following; first, data samples were taken from all virtual antennas based on the row and column information from the previous calculation making a snapshot vector, second, absolute values of the vector elements were calculated and stored in a vector, following that, angle bin values were calculated using the Equation 2.32 in Chapter 2, finally, based on the bin location of the maximum value of the vector, the angle value is found. The distance between virtual antenna elements was taken to be equal to the distance between receiving antennas. It should be noted that the first half of the snapshot vector represents the positive angles which range between 0^o and 90^o, while the second half of the vector will contain the magnitude of the angle bin values in -90^o - 0^o range. Consequently, by combining them together we will have a bird's-eye view in -90^o - 90^o range.

In Chapter 3 we found the real-time requirements for the FFT processes. These values were 35.6 μs, 2.22 μs and 0.21 μs for the first, second and the third FFT respectively. According to Xilinx's Core Generator tool, generated 1024 point single-precision floating-point FFT with natural ordered output has 4237 clock cycle latency, whereas the 32 point FFT under the same conditions has 228 clock cycle latency. For 200 MHz clock frequency,

the time required to finish these FFT processes will be 21.185 μs and 1.14 μs respectively. We can see that the first and the second FFT processes meet the timing constraints, however the third FFT process fails to meet the requirement. It requires more concurrent functioning FFT hardware blocks to achieve the real-time performance for the third FFT. In this case, we need at least 6 32 point FFT cores to meet the real-time constraints. However, considering the sequential processing of FFTs, it is not mandatory for our implementation.

Chapter 5

Results and Analysis

In the previous chapter, we discussed the algorithm and the implemented architecture. This chapter describes the results of the implemented architecture tested on Virtex-6 FPGA.

5.1 Results

Multiple tests have been carried out with the implemented architecture. As specified in Chapter 4, the Microblaze core generates input data based on the provided parameters such as range, velocity and angle information of the target, number of transmitting and receiving antennas of the RF front end and number of range and velocity bins in the FFT process. The test results have been compared with the Matlab implementation. In addition, the measurements have been taken to find out execution time required for each process and to determine the bottleneck of the implementation.

5.1.1 Hardware Resource Usage

As it was mentioned in Chapter 1, the Xilinx ML605 board is used as a hardware platform for the implementation. The boards contains Virtex-6 XC6VLX240T-1FFG1156 FPGA which is the main processing unit. The hardware resources of this FPGA are: 241152 Logic Cells or LUTs, 37680 Logic Slices, 768 Digital Signal Processing Slices (DSP48E1) and 416 36 Kbit Block RAM blocks. Single look-up-table (LUT) has 6 input ports and 1 output port which can optionally be registered in a flip-flop. Each logic slice consists of four LUTs, eight flip-flops and additional multiplexer and arithmetic carry logic.

The Table 5.1 shows the hardware resource usage and utilization of the

implemented architecture. We can see that the memory interface core and the AXI4 interconnect together take almost half of the total resources. Another thing to note is that the FFT cores use considerable amount of resources. In fact, the hardware resources required for 1024 point FFT core is much more than the resources needed for the Microblaze processor core. One explanation of this can be that both FFT cores have been configured to perform a single-precision floating-point processing with a pipelined-streaming architecture which require usage of significantly more hardware resources.

HW Component	Logic Slices	LUTs	BRAM	DSP48E1
Memory Interface Core	3044	6567	-	-
AXI DMA (x2)	995	2028	14	-
FFT Core - 1024 pt.	1795	5066	16	36
FFT Core - 32 pt.	1023	2869	2	16
Microblaze	1065	2565	6	5
AXI4 Interconnect	3102	7735	18	-
AXI4-Lite Interconnect	139	326	-	-
Total	12158	30184	70	57
Utilization (%)	32.3	12.5	16.8	7.4

Table 5.1: Resource usage of the architecture

5.1.2 Tests

The architecture was tested by generating input data with varying values of range, velocity and angle parameters. The Table 5.2 contains the performed test cases and their outputs.

We can observe that the range output values have 0.15 m difference between consequent ranges. This is due to the fact that range resolution of the radar is equal to 0.15 m. It was derived in Chapter 2 and can be found by Equation 2.18;

$$\Delta R = \frac{c}{2B} = \frac{3 \cdot 10^8 \; m/s}{10^9 \; Hz} = 0.15 \; m \tag{5.1}$$

The data associated with each FFT bin represents a different range where FFT bins range from 0 to 511. Consequently, we can find the maximum range that the radar can detect which will be equal to 76.65 m ($511 \cdot 0.15$).

On the other hand, the velocity of the moving target has 1.66 m/s difference between two consequent bins. The value represents the velocity resolution of the radar which was derived in Chapter 2 and can be found by using

Equation 2.20;

$$\Delta v = \frac{c}{2f_c nT} = \frac{3 \cdot 10^8}{2 \cdot 79 \cdot 10^9 \cdot 32 \cdot 35.6 \cdot 10^{-}6} = 1.6668\ m/s \qquad (5.2)$$

In a similar way we can find that the maximum velocity that the radar can detect. It will be equal to 25.002 m/s ($15 \cdot 1.6668$) due to the fact that the FFT bins that represent positive frequencies range from 0 to 15. Therefore, the radar can be used to detect velocities in -25 - 25 m/s range. Here the negative velocities depict a target that is approaching to the radar and the positive a one that is moving away.

We can see from the table that the angular resolution of the radar is not constant over the range. It is higher in the angles closer to zero (boresight) and smaller in the angles that are far away. This is due to the fact that the angular resolution is dependent on the beam width of the antenna which is smaller at angles closer to zero. We can also observe it in the shape of $arcsin()$ function which is used to calculate the angle FFT bin values. It has a sharper change in the higher angles in 0^o - 90^o range as well as in the lower angles in 0^o - -90^o range which results in having smaller resolution in that angles.

	Input			Output		
Test #	Distance	Velocity	Angle	Distance	Velocity	Angle
1	4 m	4 m/s	60.2^o	4.05 m	3.33 m/s	61.0^o
2	4.16 m	5 m/s	57.3^o	4.20 m	5.00 m/s	54.3^o
3	4.33 m	6 m/s	50.0^o	4.35 m	6.66 m/s	48.6^o
4	4.55 m	7 m/s	45.0^o	4.50 m	6.66 m/s	43.4^o
5	4.68 m	-8 m/s	-5.7^o	4.65 m	-8.33 m/s	-7.2^o
6	5.07 m	-9 m/s	-11.5^o	5.10 m	-8.33 m/s	-10.8^o
7	5.81 m	-10 m/s	-15.0^o	5.85 m	-10.00 m/s	-14.5^o
8	6 m	-11 m/s	-19.5^o	6.00 m	-11.66 m/s	-18.2^o

Table 5.2: Radar test results

5.1.3 Performance

The performance of the implementation was measured by finding the execution times of the processes. For this purpose, Xilinx's LogiCORE IP AXI TIMER core [21] was used. The results of the measurements can be found in the Table 5.3. It should be noted that the FFT processes also include the time spent on DMA transfers. We can see that the time needed for 1024

point FFT and 32 point FFT are 24.14 μs and 3.02 μs, respectively. From Section 4.3 we know that 1024 point FFT process takes 21.185 μs and 32 point FFT process takes 1.14 μs. Consequently, we can find the time spent on DMA transfers for these processes. It will be equal to 2.955 μs (24.14 - 21.185) for the AXI DMA block connected to 1024 point FFT and 1.88 μs (3.02 - 1.14) for the AXI DMA block connected to 32 point FFT.

Process	Time (clock cycles)	Time (μs)
FFT - 1024 pt.	2414	24.14
FFT - 32 pt.	302	3.02
First FFT - Total (x384)	925164	9251.64
Second FFT - Total (x6144)	1825811	18258.11
Third FFT - Total (x16384)	4868601	48686.01
First transpose	22935952	229359.52
Second transpose	23207150	232071.50

Table 5.3: Timing results of the implementation

It is clear that the second FFT does not meet the requirement found in Chapter 3. Based on the DMA transfer time we can find a constraint on the processing time required for this FFT process. It can be calculated by finding the difference between the required time for the second FFT process and the DMA transfer which will be equal to 0.34 μs (2.22 - 1.88). Hence, the processing time for this FFT should be less than or equal to 0.34 μs.

However, according to the Core Generator tool, latency of the pipelined 32 point floating-point FFT with the maximum achievable frequency equals to 0.415 μs. Thus, we can conclude that it is not possible to achieve the desired processing time by using a single hardware block. Consequently, we should add another 32 point FFT hardware block in order to meet the requirements with the current implementation conditions.

Another conclusion that can be drawn based on the table is that the memory transpose times are very high and will add huge delays to the signal processing time. We can see that the transpose processes together take around 0.46 s which is a lot higher compared to the FFT processing times. This will hinder the real time performance and will add a minimum time constraint of 0.46 s between two consequent radar scannings.

To summarize, we see that due to the DMA transfer times the number of FFT hardware blocks for the second and the third FFT should be increased in order to meet the real-time requirements. In addition, the transpose time of the memory is very high and adds a time constraint between two consequent radar scannings and therefore prevents to meet the real-time requirements.

5.2 Analysis

In the previous section we found out the bottleneck of the implementation. This section describes the evaluation of the model and presents the techniques that can be applied to reduce the effect of the bottleneck.

5.2.1 Evaluation

Based on the results from the previous section we can conclude that complete real-time performance of the architecture is not possible. The main reason behind it is the time required for the memory transpose operations which is the main bottleneck of the implementation (see Figure 5.1). Due to this bottleneck, the processing time requirements for the second and the third FFT processes can be loosened. In fact, no additional hardware blocks are required for the current implementation of the architecture. However, consequent radar scannings must happen with a specified interval which is 0.46 s based on the Table 5.3.

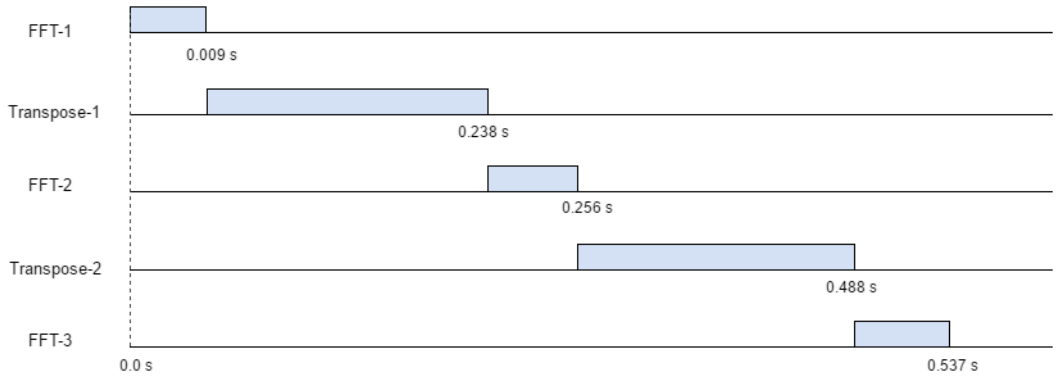

Figure 5.1: Processes and their performance

Moreover, it should be noted that precise real-time guarantees cannot be given on the architecture. This is due to the fact that the implementation requires frequent access to the SDRAM to which access latency is not constant and can vary based on number of reasons such as cache page misses and refresh operations of inside the SDRAM. This uncertainty can be reduced by knowing the exact scheduling scheme of the SDRAM controller. However, Xilinx documentation of the Memory Interface Controller does not provide any information related with the scheduling scheme of the provided core.

A few methods have been tested to reduce the time needed for the memory transpose operations. The first method was to use different memory banks for the transposed data and for the data to be transposed. The intuition behind

it was to separate the SDRAM read and write operations, hence reducing the time spent on page openings and closings. This was expected to allow less time for the memory write operations, since one column of the transposed data could be written to the memory without any more page openings. The result of the performed test proved it to be true; the first transpose operation in this case takes 22849682 clock cycles which is 86270 clock cycles less than the former one.

The second method that was tested on the architecture was to add a block RAM module to the architecture and use it as a base location for memory transpose operations. The main intuition behind the idea was that the access latency to the block RAM is considerably less than the SDRAM and no page opening and closing are required for accessing the data in a random order. Thus, the memory transpose operation would include fetching the data to the block RAM, performing transpose operation on the data and writing it back to the SDRAM. Consequently, the architecture was extended with AXI BRAM Controller [22] module to provide the AXI transfers to be read and written to the block RAM. However, the performed test showed that this method is not very effective as the first memory transpose time became 43408097 clock cycles which is almost twice more than the former one.

Another method that can be used and was mentioned in the paper [15] is to store the FFT outputs in multiple banks so that while reading the data back wait cycles for opening and closing a row could be hidden. That is, by interleaving the banks for memory accesses the data can be immediately accessed without any time spent on waiting. This would reduce the amount of time needed for read operations and additionally would require no transpose operation to be performed. However, this method has a major limitation due to the DMA transfer characteristic. It will require to use a AXI DMA core to transfer a single data sample which is not very efficient in terms of bandwidth since it does not take advantage of the burst transfer. We can observe it on the timing results Table 5.3 which shows that transferring 16 KByte (8 KByte on MM2S channel and 8 KByte on S2MM channel) of data using AXI DMA block takes 2.955 μs, whereas transferring 512 Byte (256 Byte on MM2S channel and 256 Byte on S2MM channel) of data takes 1.88 μs. We can conclude that the higher the amount of data for the transfer, the higher the bandwidth of the transfer.

To sum up, we have described few methods that have been tested to reduce the memory transpose time which is the main bottleneck of the implementation. Application of these methods did not result in getting considerable improvements in terms of operation time. Two bank implementation of the transpose operation can decrease the amount of time required, however this change is not significant compared to the complete transpose time.

The other two methods described will result in the increase of the transpose time and does not provide any value. Therefore, we can conclude here that transpose operation using MicroBlaze is the major limitation of the implementation and has a little room for further improvement.

Chapter 6

Conclusion

The initial aim of this thesis was to analyze the MIMO FMCW signal processing scheme and extend the Starburst platform based on the real-time requirements of the automotive radar application. However, it was found that Starburst platform is not suitable for this application and does not provide any means to meet the requirements needed for the real-time performance. Therefore, an alternative architecture was proposed based on the signal processing algorithm and implemented on Virtex-6 FPGA. This chapter describes the conclusions drawn during this process and future work that can be carried out to improve the architecture.

6.1 Conclusions

There were three main findings after analysis of the MIMO FMCW signal processing algorithm. First, it was found that the Fast Fourier Transform is the core process of the algorithm. Second, the algorithm requires huge amount of intermediate data to be stored in a memory. Third, memory transpose operation might be required in order to have an efficient memory access. In addition, the algorithm had to meet certain constraints to provide the real-time performance. These constraints were based on the parameters used in the Matlab model which was provided by the NXP Semiconductors.

The initial design idea for the implementation was based on the Starburst platform. The idea was to use a MicroBlaze core which is the main processing element of the Starbust platform to run the FFT algorithm. However, it was theoretically found that FFT algorithm running on the MicroBlaze core could not meet the processing time constraints and the real-time performance could not be achieved with this implementation.

To meet the constrains an FPGA implementation of the algorithm was

proposed. The MicroBlaze core was used as a main unit in the architecture to generate the input signal to be processed and to control all the other operations such as transposing the memory and configuring AXI DMA cores. For the FFT operations, Xilinx's FFT IP was found to be suitable enough to be included in the architecture. It provides a number of architecture options for the FFT implementation and meets the real-time constraints required by the application.

Moreover, on-chip memory provided by Virtex-6 FPGA was found to be insufficient for the storage of the intermediate data. Thus, it has been decided to use the off-chip SDRAM memory for this purpose. This constraint would require transpose operation to be performed in the memory. As it was discovered later, this operation would be the main bottleneck of the implementation.

The memory transpose process is the main area in which major improvements need to be made. A few methods have already been evaluated, however no solution was found that meets the real-time requirements.

It was also found that the accuracy of the range, velocity and bearing results are limited by the resolution. The accuracy of the output can be found by finding the half of the resolution. Range resolution of the radar is 0.15 m which means that the accuracy of the range output will be ≤ 0.075 m. The velocity resolution of the radar is 1.6668 m/s which will means the accuracy of the output will be ≤ 0.8334 m/s. As it was mentioned in Chapter 5, the angular resolution of the radar is not constant and changes based on the beam width.

In addition, it should be noted that the current implementation uses single-precision floating-point representation for the signal processing. Main reason for that is the MicroBlaze core can generate single-precision floating-point input data which offers more precision than fixed-point. Furthermore, the FFT IP core supports single-precision floating-point processing as well which makes it easier to implement in the architecture. However, the specific advantages of using floating-point over fixed-point are not clear and further research should be carried out to find out the trade-offs between floating-point and fixed-point implementation.

6.2 Future Work

Further research and work should be carried out on the FPGA implementation of the algorithm to improve the performance. As it was mentioned in the previous section, the main work should concentrate on reducing the time needed for the transpose operations. It should be noted that the current

analysis and implementation assumed that the consecutive radar scannings would happen without any time interval between them so that at any time period the driver of the vehicle would have an information about its surrounding environment. However, the necessity of this condition is not very clear according to the provided model. Therefore, it should be investigated whether the SDRAM storage of the intermediate data is really a requirement if there is enough time between consecutive radar processing. According to the calculations, if we consider having enough time after the first radar scanning and use 16 bit fixed-point arithmetic, the memory inside the Virtex-6 might be sufficient to store the intermediate data of the first FFT processing. In addition, in-place computation can be used to store the data of the second FFT processing without using any additional memory storage. This would require a design of a new DMA core since the current one used in the implementation has very low bandwidth for single transfers.

Moreover, the current design uses pre-existing IP blocks developed by Xilinx which provides no information about the internal design of the blocks. Therefore, it is difficult to perform the data-flow analysis on the model and provide the real-time guarantees. For the future, some of the blocks such as the DDR controller could be redesigned based on the specific memory access patterns of the application. This would allow to perform data-flow analysis on the model and give estimates about the performance of the implementation beforehand.

Additionally, we saw in Section 5.1 that FFT blocks take a lot of hardware resources. An improvement can be made in this part by redesigning the FFT core to achieve less hardware usage at the same time to provide a sufficient latency and bandwidth.

Bibliography

[1] "Phase-comparison monopulse." `https://en.wikipedia.org/wiki/Phase-comparison_monopulse`. Accessed: 2016-07-23.

[2] M. Skolnik, *Radar-Handbook*. McGraw-Hill Professional, 2008.

[3] 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," *IEEE Transactions on Microwave Theory and Techniques*, vol. 60, pp. 845 – 860, 2012.

[4] B. Dekens, *Low-Cost Heterogeneous Embedded Multiprocessor Architecture for Real-Time Stream Processing Applications*. PhD thesis, University of Twente, 10 2015.

[5] V. Issakov, *Microwave Circuits for 24 GHz Automotive Radar in Silicon-based Technologies*. Springer, 2010.

[6] I. V. Komarov and S. M. Smolskiy, *Fundamentals of Short-Range FM Radar*. McGraw-Hill Professional, 2005.

[7] V. Winkler, "Range Doppler Detection for Automotive FMCW Radars," *Proceedings of the 4th European Radar Conference*, 2007.

[8] Xilinx, "ML605 Hardware User Guide," 2012. UG534(v1.8).

[9] W. Wiesbeck, *Radar Systems Engineering*. Karlsruhe Institute of Technology, 2009. Lecture Script.

[10] R. Feger, C. Wagner, S. Schuster, S. Scheiblhofer, H. Jäger, and A. Stelzer, "A 77-GHz FMCW MIMO Radar Based on an SiGe Single-Chip Transceiver," *IEEE Transactions on Microwave Theory and Techniques*, vol. 57, pp. 1020 – 1035, April 2009.

[11] J. Li and P. Stoica, "MIMO Radar with Collocated Antennas," *IEEE Signal Processing Magazine*, vol. 24, pp. 106 – 114, September 2007.

[12] Y. Qu, G. S. Liao, S. Q. Zhu, X. Y. Liu, and H. Jiang, "Performance Analysis of Beamforming for MIMO Radar," *Progress In Electromagnetics Research*, pp. 123–134, 2008.

[13] Xilinx, "MicroBlaze Processor Reference Guide," 2013. UG081(v14.7).

[14] W. Zong-bo, J. C. Moya, A. B. del Campo, J. G. Menoyo, and G. Meiguo, "Range-Doppler Image Processing in Linear FMCW Radar and FPGA Based Implementation," *Journal of Communication and Computer*, vol. 6, no. 53, pp. 55–62, 2009.

[15] F. Meinl, E. Schubert, M. Kunert, and H. Blume, "Realtime FPGA-based Processing Unit for a High-Resolution Automotive MIMO Radar Platform," *European Radar Conference*, pp. 213–216, 2015.

[16] E. Hyun, S.-D. Kim, D.-J. Yeom, and J.-H. Lee, "Parallel and Pipelined Hardware Implementation of Radar Signal Processing for an FMCW Multi-channel Radar," *Elektronika IR Elektrotechnika*, vol. 21, no. 2, 2015.

[17] W. Wang, D. Liang, Z. Wang, H. Yu, and Q. Liu, "Design and Implementation of a FPGA and DSP Based MIMO Radar Imaging System," *Radioengineering*, vol. 24, pp. 518–527, 2015.

[18] Xilinx, "LogiCORE IP Fast Fourier Transform v8.0," 2012.

[19] Xilinx, "LogiCORE IP AXI DMA v6.03," 2012.

[20] Xilinx, "Virtex-6 FPGA Memory Interface Solutions," 2013. UG406.

[21] Xilinx, "LogiCORE IP AXI TIMER v1.03," 2012.

[22] Xilinx, "LogiCORE IP AXI Block RAM (BRAM) Controller v1.03," 2012.