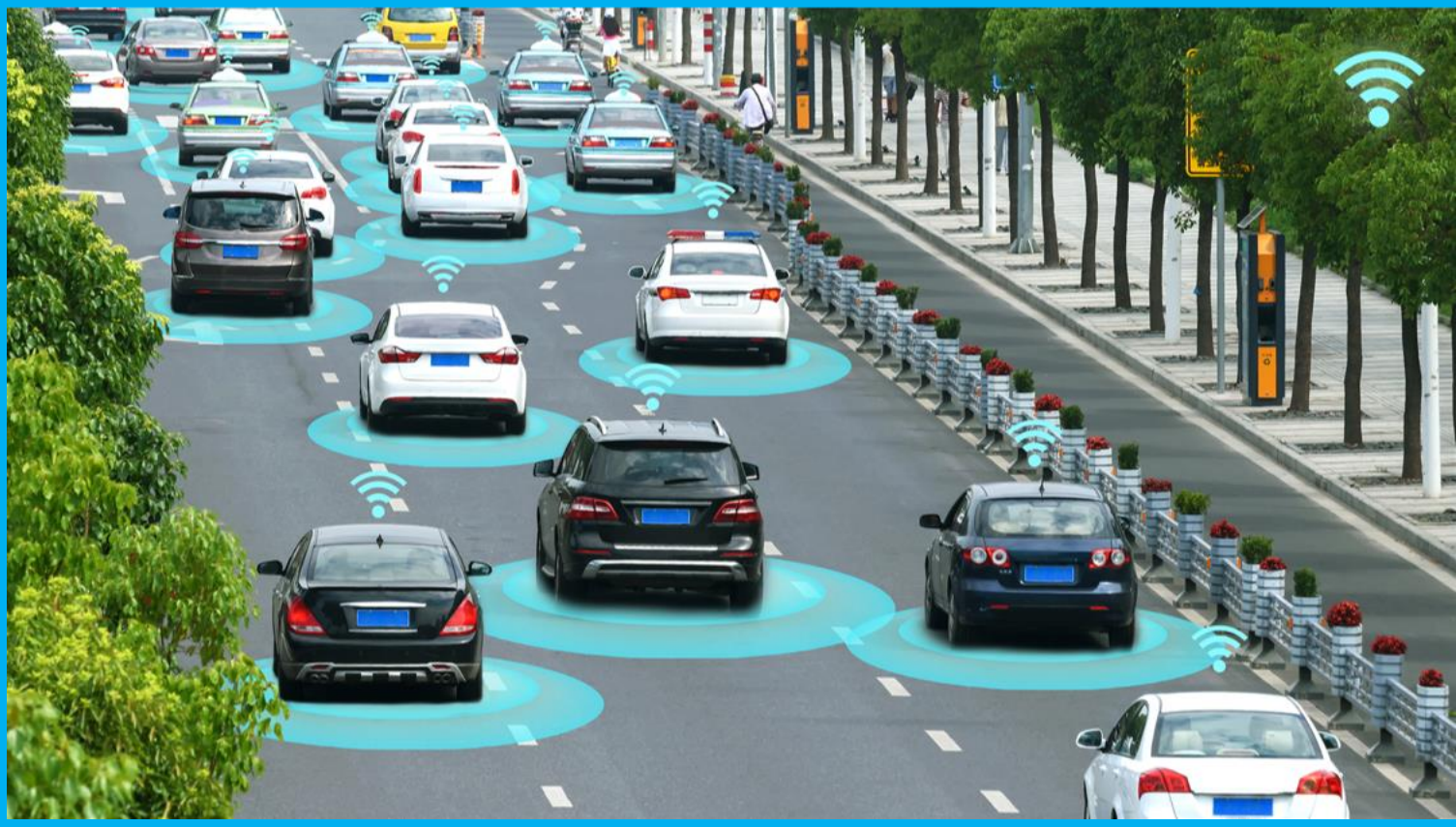
Automotive radar: Real-time implementation of a joint sensing and communication waveform on a microcontroller

Tasneem Rahaman Khan



TU Delft

Challenge the future

Automotive RADAR: Real-time implementation of joint sensing and communication waveform on a microcontroller

by

Tasneem Rahaman khan

to obtain the degree of Master of Science

in

Computer Engineering

at the Delft University of Technology,
to be defended publicly on Friday November 30, 2018 at 14:30 PM.

Student number: 4513673
Project duration: January 1, 2018 – November 30, 2018
Thesis committee: Prof. DSc. Alexander Yarovoy, Technische Universiteit Delft
Dr. Ir. Arjan J. van Genderen, Technische Universiteit Delft
Dr. Ir. Faruk Uysal, Technische Universiteit Delft

TU Delft

Acknowledgements

Eleven months ago I embarked onto conducting this thesis work with a motivation to contribute my knowledge towards the future of the cars. It has been a commendable journey ever since then. This thesis document is the fulfillment of the thesis project concerning my master study. The underlying study and work would have been impossible to complete without receiving support and help in many different ways. I would like to start with conveying my sincerest gratitude towards my advisor **Prof. DSc. Alexander Yarovoy** for giving me the opportunity to be a part of such a potential thesis topic and give me a place in the Microwave Sensing, Signals and Systems Group. I want to extend sincerest credits and gratitude to my daily supervisor **Dr. Faruk Uysal** for his scientific and technical guidance and for showing continued confidence in my skills along with being a good mentor. I also want to thank my second supervisor **Dr. Arjan van Genderen** for his constant feedbacks relating to computer engineering concepts. I would also like to acknowledge **Ir. Pascal Aubry** for supporting me extensively with issues relating with MATLAB. Besides, I also thank **Mr.René Geraets** from NXP for providing me with all the technical information throughout the thesis work.

It was delightful to be able to work in the friendly, supportive, and productive atmosphere of the Microwave Sensing, Signals and Systems Group. I warmly thank all of my fellow department students from whom I learned many concepts concerning my thesis, receive important feed-backs and shared countless cups of coffee and laughs.

I thank all my well wishers for being there for me time to time. I am hugely indebted to the Bechan family and friends for all their love, affection and homely warmth.

Studying at the Technical University of Delft has been truly a pleasure. The institution has offered me of all the help and support required in times of crisis and will never be forgotten.

In the end, I express all my sublime gratitude to **Abbu, Maa and Tahmiah** for all the encouragement, friendly support, philosophy, wishes, and belief. Above all, I thank Almighty for making all this possible for me.

Tasneem Rahaman khan
Delft, November 2018

Contents

Summary

The automotive radars are significantly drifting towards the 77GHz electromagnetic spectrum band and by 2020, they will no longer operate in the 24GHz band as per the regulations given by the European Telecommunications Standards Institute (ETSI) and Federal Communications Commission (FCC) in Europe and the U.S. With the communication systems also heaping towards the magnitude of radar functionality, the 77GHz spectrum has opened a wide scope and possibility to combine both communication and radar waveforms and offer joint radar sensing-communication functionality.

The true aim of this thesis work is to implement one of the widely used digital modulation techniques namely Binary Phase Shift Keying (BPSK) to embed information bits in the Frequency Modulated Continuous Waveform (FMCW) radar waveform via a Microcontroller (MCU) to facilitate real time radar processing and information exchange.

The report starts with presenting a theoretical overview on FMCW automotive radar, waveforms for both sensing and communication and a brief review on the MCU's in the automotive radars. The next chapter of report gives a detailed explanation on the hardware and software architectures on which the thesis implementation is carried out. The following chapter of the report focuses on the main adopted implementation to embed information bits and all the underlying methodologies related to it. Finally, the results of this thesis work is diversified into obtaining real time 2D Fast Fourier Transform (FFT) spectrum, beat signals, reconstructing the message embedded in the FMCW waveform and validate the correctness of the phase coding.

List of Acronyms

WHO	World Health Organization
ETSI	European Telecommunications Standards Institute
FCC	Federal Communications Commission
OFDM	Orthogonal Frequency Division Multiplexing
BPSK	Binary Phase Shift Keying
PSK	Phase Shift Keying
QPSK	Quadrature Phase Shift Keying
EM	Electromagnetic
BPSK-LFM	Binary Phase Coded Linear Frequency Modulated Waveform
MCU	Microcontroller
DSRC	Dedicated Short Range Communication
V2V	Vehicle-to-Vehicle
mm-Wave	Millimeter-Wave
LFM	Linear Frequency Modulated
DSSS	Direct Spread Sequence Spectrum
CPU	Central Processing Unit
ALU	Arithmetic Logic Unit
RISC	Reduced Instruction Set Computing
MIMO	Multiple Input Multiple Output
FMCW	Frequency Modulated Continuous Waveform
CW	continuous Waveform
FFT	Fast Fourier Transform
IF	Intermediate Frequency
NRZ	Non-Return to Zero

TCP/IP Transmission Control Protocol/Internet Protocol

UDP User Diagram Protocol

DCC Dual Credit Card

SRAM Shared Random Access Memory

GUI Graphical User Interface

ADC Analog Digital Converter

AWGN Additive White Gaussian Noise

BER Bit Error Rate

ENIAC Electronic Numerical Integrator And Computer

EDVAC Electronic Discrete Variable Automatic Computer

API Application Programming Interface

SPT Signal Processing Toolbox

ISA Instruction Set Architecture

SPI Serial peripheral Interface

EMAC Ethernet Media Access Control Address

VCO Voltage Control Oscillator

BSD Berkeley Socket Distribution

ASCII American Standard Code for Information Interchange

1

Introduction

In today's world, cars are increasingly becoming the center of human's lives. According to the recent survey conducted by the World Health Organization (WHO) on road safety [20], it was deduced that around 1.25 million people die due to road accidents. It was also predicted that by the year 2030, this situation will be the 7th leading cause of human death. As the number of vehicles on roads are dramatically increasing, the need for driver assistance is widely a subject of research interest to the automotive industries. To address this, the automotive industry has found its interest in deploying radars in the vehicles to avoid road fatalities. Today's most desirable vehicles have automotive radars that provide parking and adaptive cruise control assistance to the driver, while also warning the driver of any fore-coming collisions to avoid accidents. The main function of radar is to provide detection of any presence of one or more targets and provide information on the range, angle and motions relative to the radar. Conventionally, radars are used because of its immunity to the environmental influences such as high temperatures, weather conditions or variation in lighting making radar to be used as an appropriate technology in autonomous automobiles. Different applications provided by an automotive radar including the scope of information exchange in vehicles is shown in the figure 1.1.

Automotive radars operate at the bands in 24 Giga Hertz (GHz) and 77 GHz of the electromagnetic spectrum which is referred to as mmWave band [6]. Such high operational frequencies are used to obtain higher range resolution. Since the inception of automotive radars, the 24 GHz were initially developed for short range radar applications such as blind spot detection and collision avoidance. However, this band shares some limitations as it may cause interference in the radio astronomy and satellite services. Due to the spectrum regulations given by the European Telecommunications Standards Institute (ETSI) and Federal Communications Commission (FCC), these radars will no longer be available in vehicles by 2020 in Europe and the U.S. The 24 GHz radars will be replaced by 77 GHz in the future. Particularly, the 77 GHz band(77-81 GHz) provides various advantages such as smaller an-

tennas, a sweep bandwidth of 4 GHz which is much larger than what 24 GHz offers, greater accuracy, high transmit power and a better object resolution.

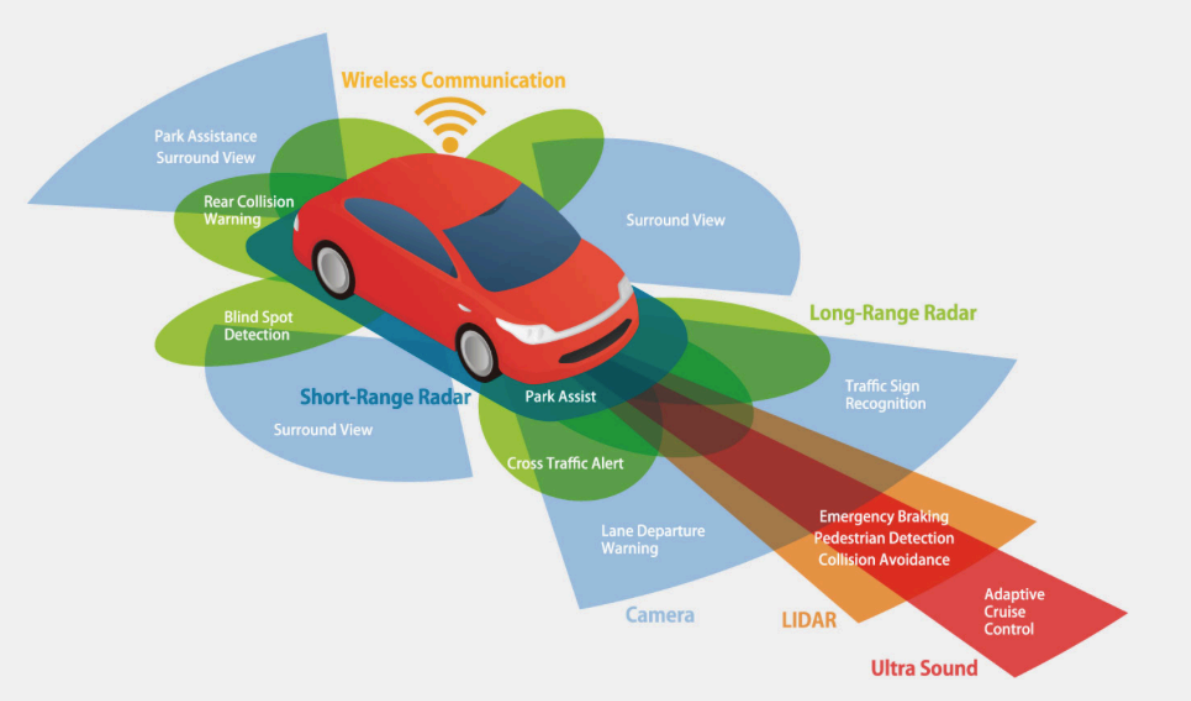


Figure 1.1: Automotive radar applications [9]

In addition to the driver assistance feature that is fulfilled by automotive radars by sensing the vehicles around them, it is also essential for the vehicles on road to be able to share their information like speed and updates on road situations with the vehicles around them in order to avoid accidents and ensure safety. To address this, it is essential to establish a Vehicle-to-Vehicle (V2V) communication medium that will allow the vehicles to safely exchange raw sensor data to deliver applications such as adaptive cruise control and forward collision warning [14]. The current v2v communication is built on Dedicated Short Range Communication (DSRC) standard which functions in the 5.9 GHz band [4]. However, this standard provides much lesser data rates of 27 Mbps that is difficult to support the higher end applications for automated driving. To overcome this limitation and taking advantage of the very recent developments in this aspect of higher frequency spectrum allocation for automotive radars, it is been observed that with the digital wireless communication systems have their carrier frequencies diverging towards the mmWave standards, which means that it shared the same order of magnitude compared to the radars. The inception of the idea concerning the joint functionality of both sensing and communication is the true motive behind conducting this thesis. Such a joint functionality system would unquestionably offer various possibilities for more autonomous applications.

1.1. Related work

In order to achieve the integration of sensing and communication in radar system, there are currently three techniques as stated in [5] namely: time-sharing, sub beams and signal sharing. Time sharing technique involves in reusing the same antenna, transmitter and receiver by using a strobe switch for switching between communication and radar modes. The advantage of such technique is its simple design with respect to communication and radar waveform. However, at any given time, it can only perform one functionality and that affects the system performance. The second technique is based on sub-beams which is utilized in phased array radars,wherein the array is broken down with respect to the radar and communication functionalities and the third approach is based on signal sharing. The key idea of this approach is to use a single waveform to perform both radar sensing and communication without degrading the performance of sensing functionality. Two different waveform design concepts were proposed namely Orthogonal Frequency Division Multiplexing (OFDM) and Direct Spread Sequence Spectrum (DSSS) to achieve the fusion of the functionalities. Most of the current digital communication systems work on the principle of OFDM mentioned in [13]. In the paper by [7], OFDM was also introduced as a potentially suitable radar waveform and in [19], it was also shown that the OFDM offers Doppler estimation that depends only on the OFDM symbols instead of the base-band signal. However, from the recent research on OFDM waveform, very advanced signal processing methods are required since OFDM has multicarriers. Another waveform studied by [18] describes spread spectrum related concepts that uses single carrier. This approach provided high data rates and good radar performance. However, it increased the complexity of the Rad-Com implementation. Therefore, in order to employ a joint radar waveform for both sensing and communication to facilitate ease in implementation and reduce the complexities, we choose to make use of one of the simplest digital modulation technique's that can used to embed information bits in the Linear Frequency Modulated (LFM) chirp. Digital communication waveforms such Binary Phase Shift Keying (BPSK), Quadrature Phase Shift Keying (QPSK) and others can be modulated into the linear frequency modulated waveform which is typically used by current automotive radars. The approach of signal sharing is highlighted in the work proposed by [18] for joint sensing and communication system (Rad-Com).

1.2. Problem Statement

The aim of this thesis is to implement a joint waveform on a Microcontroller (MCU) that can simultaneously be employed for radar sensing as well as for information transmission in the Electromagnetic (EM) spectrum (77-81GHz) allotted for automotive vehicles without degrading the performance of radar sensing. This will enable intelligent interaction among the vehicles on the road and improve road safety. This implementation is based on the signal sharing approach for integration of radar and communication system as

stated in [5].

To achieve this goal, we consider one of the most widely known radar waveforms used in automotive radars, namely, LFM waveform, also known as chirps for the purpose of sensing. In order to attain information exchange among vehicles, we encode the data by phase coding the chirp with 0 or π, which means only one bit can be embedded per chirp. We take the advantage of the NXP's 77 GHz transceiver's inbuilt feature of binary phase shift keying to encode the data as it is the simplest digital communication waveform. Further, we investigate the correctness of the message reconstructed.

The main goals of the thesis is to

- Propose software architecture for interface between NXP's S32R274 MCU and TEF810X transceiver to achieve joint sensing-communication functionalities via radar transceiver for automotive radar applications
- Implement the Binary Phase Coded Linear Frequency Modulated Waveform (BPSK-LFM) on the MCU to facilitate real time processing by controlling the radar settings and signal processing in real-time
- Verify the correctness of phase coding by establishing various different calibration test scenarios

1.3. Organization

This thesis document is structured in the following chapters:

- Chapter 2: We present all the underlying theory relating to the Frequency Modulated Continuous Waveform (FMCW) radar, discuss the BPSK communication waveform and show the motivation behind using BPSK communication scheme in the radar waveform. In addition to this, we review the basic concepts of MCU and discuss the idea behind the trend of deploying MCU in automotive radars.
- Chapter 3 : In this chapter, we explain in detail of all the required hardware and software architecture concepts supporting our implementation.
- Chapter 4 : Here, we present our system Implementation model and strategies employed to achieve joint sensing-communication waveform via a MCU to facilitate real time signal processing.
- Chapter 5 : We discuss the results that we achieved as a result of our system implementation and present the validation of the results.
- Chapter 6: Finally, we summarize the thesis work with a conclusion along with discussions regarding possible future work that can be immediately investigated as a result of this thesis.

2

Theory and Background

To explain the implementation of a joint waveform on an MCU to achieve both radar sensing and to enable the information exchange, we start with presenting the basic concept and working principle of a radar in the first section. Following this, we explain the basic concept of an automotive radar that works on the FMCW principle in the second section. After this, to achieve communication functionality, we give a brief explanation on the BPSK waveform in the next section. Then, we show the motivation behind using these two waveforms to achieve the goal. The last sections describe the basic concepts of MCU for the reader to get familiarized with.

2.1. Radar

Radar which stands for Radio Detection and Ranging is a device that enables certain objects to be seen- that is, detected and located at distances beyond the sight of an unaided eye. Furthermore, radar measures the range of the object it detects and also measures the speed of such a moving object. Radars traditionally were invented during the second world war to serve its purpose in the defence and military applications for long-range air surveillance and short-range detection of low altitude incoming targets. Later on, it was improvised to be used as weather radars and air-traffic control radar in the airports to guarantee the safety of air congestion, meteorology, radio astronomy, and medicine.

Conventionally, a radar system comprises of a transmitter which creates the energy pulse, an antenna to send these pulses into the atmosphere and hit the target object and receive the reflected pulse back and a receiver which detects, amplifies and the transforms the received echo signals.
The radars are broadly classified into pulsed radars and continuous wave radar. Pulsed radars are used to transmit short and powerful pulses at regular time intervals. These radars provide range estimations from the time delay between the transmitted and received echo pulse. In other words, the target range can be obtained from recording the round trip travel time of a pulse, τ. Therefore, the distance R to the target is given by the following equation.

$$R = c\tau/2 \tag{2.1}$$

where c is the velocity of Electromagnetic (EM) wave in free space. In contrast to the pulsed radars, unmodulated continuous Waveform (CW) radar systems transmit a fixed frequency signal, and the receiver gets superimposed reflections from multiple targets.

In the case of the target is in motion, the received signal will be shifted in frequency due to the doppler effect. Such a phenomenon can be used to estimate the velocity of the target. The limitation of unmodulated CW radars is its incapability of estimating the range of the target as it operates continuously at the single frequency by which it is not possible to calculate the time delay between the transmitted and the received signal. In order to overcome this limitation, a modulation technique was proposed. In this technique, the carrier wave will be modulated with respect to amplitude, frequency or phase in order to measure both the time delay between the transmitting and receiving signal and the doppler frequency shift. Most widely used form of CW radars used in the automotive industry is the Frequency Modulated Continuous Waveform (FMCW) waveform.

2.2. FMCW Radar

FMCW radars are most widely used in the automotive industry today. They work on the principle of transmitting a radar signal referred to as a ”chirp” as seen in the figure 2.3.

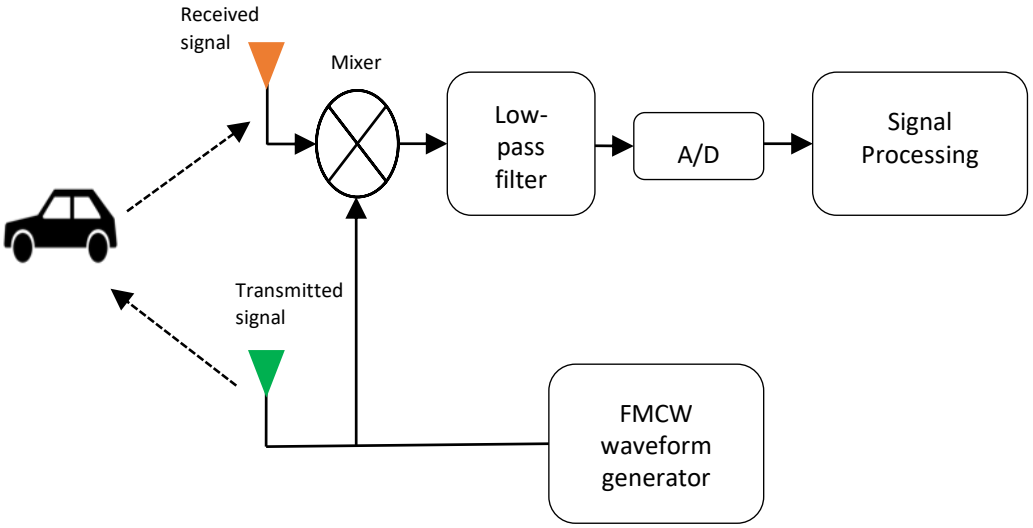


Figure 2.1: FMCW radar block diagram

This is a sinusoidal wave whose frequency modulation is linearly increased with respect to the time; hence, the name linear frequency modulated(LFM)[2].

In other words, the chirp is linearly swept over a range of frequencies with bandwidth B for a duration of chirp, T. This phenomenon causes the frequency $f(t)$ to vary linearly at any given time, and is given by the equation:

$$f(t) = f_c + St \tag{2.2}$$

where f_c is the carrier frequency, t is the instantaneous time and S is the slope of the chirp shown in the figure 2.3 which is expressed as:

$$S = \frac{B}{T} \tag{2.3}$$

When S is greater than 0, the chirp is referred to as up-chirp, and when S is less than 0, it is referred to as down chirp. The depiction of an up-chirp centered at a 100 Hz center frequency is as shown in the figure 2.2.

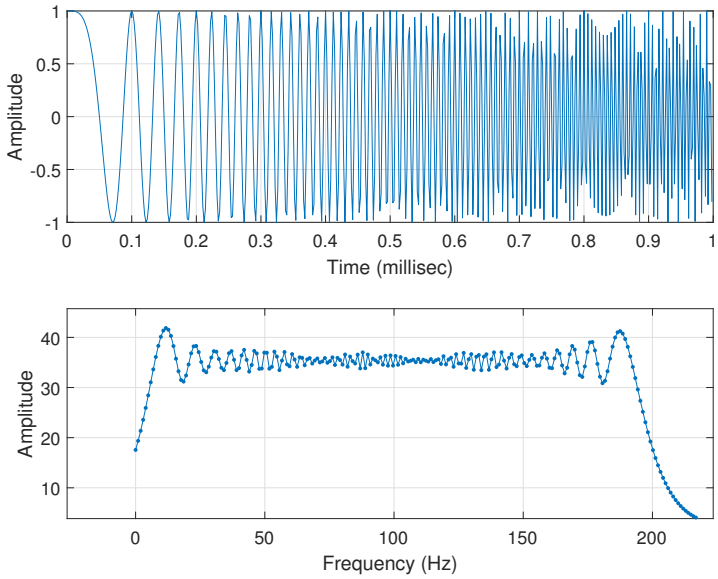


Figure 2.2: (a) LFM signal in Amplitude-time domain, (b) Chirp signal in frequency-time domain

As frequency changes over the time, we can obtain instantaneous phase as:

$$\psi(t) = 2\pi \int_0^t f(t)\, dt + \Psi_0 \tag{2.4}$$

$$\psi(t) = 2\pi \left(f_c t + \frac{S}{2} t^2 \right) + \Psi_0 \tag{2.5}$$

Therefore, a transmitting LFM chirp signal is represented as,

$$X_{tx}(t) = A_{tx} cos(\psi(t)) \tag{2.6}$$

$$X_{tx}(t) = A_{tx}cos\left(2\pi\left(f_c t + \frac{S}{2}t^2\right) + \psi_0\right) \tag{2.7}$$

where A is the amplitude of the transmitted LFM chirp, ψ_0 is the initial phase, f_c is the carrier frequency, S is the slope of the chirp, and t is the instantaneous time.

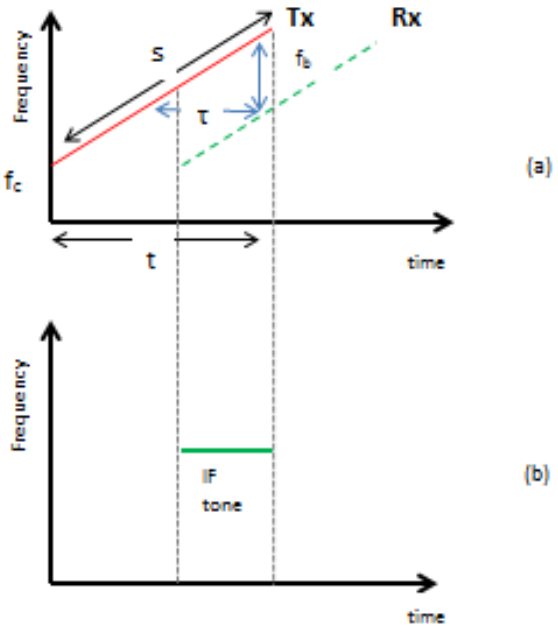


Figure 2.3: (a) Single target frequency-time domain, (b) IF signal[12]

According to the FMCW radar principle, the LFM signal is transmitted and hits the target which is located at an initial distance R with a radial velocity v. The echo signal referred to as returned signal from the target will be the replica of the transmitted signal but with some time delay τ. The τ is expressed as,

$$\tau = \frac{2(R + vt)}{c} \tag{2.8}$$

Due to the time delay τ, the received signal is represented as,

$$X_{rx}(t - \tau) = B_{tx}cos(\psi(t - \tau)) \tag{2.9}$$

$$X_{rx}(t - \tau) = B_{tx}cos\left(2\pi\left(f_c(t - \tau) + \frac{S}{2}(t - \tau)^2\right) + \psi_0\right) \tag{2.10}$$

From the figure 2.1, it can be seen that the received signal is mixed with the transmitted signal. This results in an Intermediate Frequency (IF) signal $X_{IF}(t)$ that is given by,

$$X_{IF}(t) = X_{tx}(t) \cdot X_{rx}(t) \tag{2.11}$$

The IF signal is obtained as,

$$X_{IF}(t) = \frac{A_{tx}B_{rx}}{2} cos\left(2\pi\left(f_c t + \frac{S}{2}t^2 - f_c(t-\tau) - \frac{S}{2}(t-\tau)^2\right)\right) \tag{2.12}$$

$$X_{IF}(t) = \frac{A_{tx}B_{rx}}{2} cos\left(2\pi\left(f_c \tau + St\tau - \frac{S}{2}\tau^2\right)\right) \tag{2.13}$$

By substituting equation 2.8 in equation 2.13, we obtain,

$$X_{IF}(t) = \frac{A_{tx}B_{rx}}{2} cos\left(2\pi\left(f_c \frac{2(R+vt)}{c} + St\frac{2(R+vt)}{c} - \frac{S}{2}\left(\frac{2(R+vt)}{c}\right)^2\right)\right) \tag{2.14}$$

By simplifying the equation 2.14, we get,

$$X_{IF}(t) = \frac{A_{tx}B_{rx}}{2} cos\left(2\pi\left(\left(\frac{2SR}{c} + \frac{2f_c v}{c} - \frac{4SRv}{c^2}\right)t + \frac{2f_c R}{c} - \frac{2SR^2}{c^2} + \frac{2Sv^2t^2}{c^2}\right)\right) \tag{2.15}$$

From the equation 2.15, we can observe a frequency and a phase term that has an influence on the how the signal is varying over the time. This frequency component is usually referred to as ”beat frequency”. The frequency difference between the transmitted and the received signal is expressed as f_b.

Further, the equation 2.15 can be simplified into,

$$X_{IF}(t) = \frac{A_{tx}B_{rx}}{2} cos\left(2\pi\left(\frac{2SR}{c}t + \frac{2f_c v}{c}t\right) + \frac{4\pi f_c R}{c}\right) \tag{2.16}$$

From equation 2.16, we can see that phase term $\frac{4\pi f_c R}{c}$ is constant since R is an initial distance from the target, and the main frequency component of the signal over one chirp period is given by $S \cdot \frac{2R}{c}$. To obtain the beat signals, it is required to apply the FFT algorithm over the digital sequence. For one chirp period, the beat frequency can be expressed as,

$$f_b = S \cdot \frac{2R}{c} \tag{2.17}$$

From the equation 2.17, we can obtain the range R as,

$$R = \frac{f_b c}{2S} \tag{2.18}$$

From the equation 2.18, it is seen that the frequency of IF signal is directly proportional to the range estimated to each target. This means that for both single and multiple target scenarios, certain limits have to established on how far the targets can be detected by the radar and how close the targets in the case of multiple targets can be to each other for the radar to be able to distinguish the targets. To set such limits, the main characteristic is range resolution. By estimating the range resolution, it is possible to distinguish closely spaced targets. The range resolution ΔR is given by:

$$\Delta f_b = \frac{2B\Delta R}{cT} \tag{2.19}$$

$$\Delta R = \frac{c}{2B} \tag{2.20}$$

where B is the swept bandwidth of the chirp, and in the multi-target scenarios, the range resolution can be estimated when the two IF signals can be resolved when the frequency difference between them is greater than $\frac{1}{\tau}$.

From equation 2.16, we can also observe a phase term $\frac{2f_c v}{c}$ that corresponds to the beat frequency that changes linearly with the number of chirps. As the phase changes, the frequency of the signal also changes over the number of chirp. The reason for this is because of the doppler shift in the frequency due to the relative motion of the target. To obtain the velocity of the target, we use the doppler shift f_d and is given by,

$$f_d = \frac{2f_c v}{c} \tag{2.21}$$

$$v = \frac{f_d c}{2f_c} \tag{2.22}$$

We can obtain velocity measurements by looking at the doppler shift of the signal. This is done by observing the frequency spectrum of the signal over n consecutive chirp periods (nT). This means that the doppler processing is applied on consecutive chirps. In the first approximation, range and velocity estimations are obtained independently by processing each chirp for range measurements and processing phase over consecutive chirps for velocity measurements. A detailed structure of processing range-doppler FFT is explained in the system implementation chapter.

Similar to the concept of range resolution, the radar has to know minimum velocity between two targets for it to be able to distinguish the targets in order to detect them. Hence, it is essential to have the information on the velocity resolution.

The radar can separate two or more peaks of the IF signals only if the doppler frequency which changes over n chirp periods is bounded by the frequency resolution.

$$f_d \geq \frac{1}{nT} \tag{2.23}$$

Therefore, the velocity resolution can be expressed as,

$$\Delta v = \frac{c}{2f_c}\frac{1}{nT} \tag{2.24}$$

The benefit of such the LFM waveform is its improved target detection feature. It also improved the range resolution of the radar due to reduced chirp durations. Due to its advantage of canceling the interference and its low doppler sensitivity, the LFM gained its usage in the area of communication [21].

More advanced automotive applications call for an advanced FMCW radar concept namely fast chirp radar. These radars transmit identical chirps with much shorter duration than the traditional FMCW radars. The fast chirp

radars provide independent doppler and range measurements. Due to transmitting chirps with shorter durations and higher slopes, these radars are capable of increasing the range frequency shift independent of doppler frequency shift.

Concept of MIMO Radar

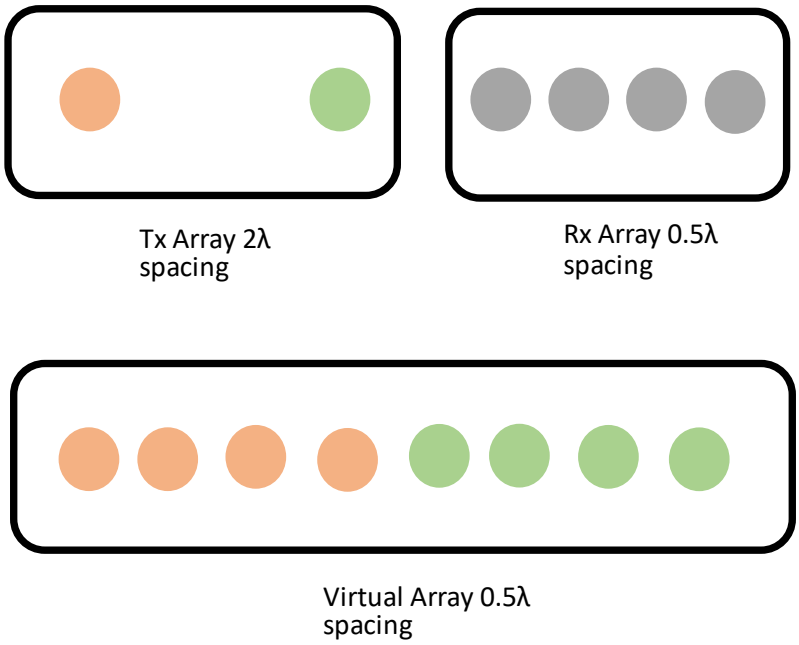


Figure 2.4: MIMO radar with the transmitter, receiver, and virtual Arrays

MIMO refers to a type of radar system that employs multiple transmitter and receiver waveforms. MIMO radar uses multiple antennas at both transmitter and receiver ends. With these type of radars, it is possible to measure the target reflections due to the presence of multiple paths between the radar and the target object. This means that all the transmitting antennas transmit a waveform which is not dependent and different from other transmitting antennas creating a virtual array (spatial channel) as shown in the figure 2.4 containing information from each transmitting antenna. This means that if there are M number of transmit antennas and N number of receive antennas, then we achieve a $M * N$ number of spatial channels.

It is undeniable that by using such an arrangement of multiple antennas both at the transmit and receive sides which results in fewer physical antennas, therefore, constructing an efficient virtual array. In addition to this, MIMO radar offers other advantages for automotive applications such as increased spatial resolution and higher sensitivity in order to detect very slow moving targets[8].

2.3. Communication waveform

For digital modulation techniques used for transmission of digitally represented data, we study and investigate one of the widely known modulation techniques for communication in a radar system.

2.3.1. Binary-phase coded modulation

Binary-phase shift keying is the simplest form of Phase Shift Keying (PSK). The BPSK digital modulation is a technique which modulates the phase of a constant sinusoidal carrier(frequency reference signal). Ideally, it uses two phases 0° and 180° degree phase shifts. In the case of modulating the LFM waveform, binary data is transmitted by mapping the bits into up-chirps or down-chirps [3].

Consider two time-limited energy signals $s_1(t)$ and $s_2(t)$ and a basis function $\phi_1 t$.

Mathematically,

$$s_1(t) = \sqrt{\frac{2E_b}{T_b}}.cos(2\pi f_c t) = \sqrt{E_b}\phi_1(t) \tag{2.25}$$

$$s_2(t) = \sqrt{\frac{2E_b}{T_b}}.cos(2\pi f_c t + \pi) = -\sqrt{E_b}\phi_1(t), and \tag{2.26}$$

$$\phi_1(t) = \sqrt{\frac{2}{T_b}}.cos(2\pi f_c t), 0 \leq t \leq T_b \tag{2.27}$$

where E_b is the bit energy, T_b is the bit duration and $\frac{N_0}{2}$ is the power spectral density of Additive White Gaussian Noise (AWGN). The constellation diagram of the BPSK modulation is shown in the figure 2.5.

The Bit Error Rate (BER) of BPSK in AWGN is given by,

$$BER = Q\sqrt{\frac{2E_b}{N_0}} \tag{2.28}$$

In digital communications, base-band signals are serially transmitted and represented in 0's and 1's logic. In hardware circuitry, a logic 0 relates to low voltage and logic 1 represents high voltage. However, in order to transmit information over any significant distance, this representation is not used. All the circuits in hardware carrying the signal with information must have a frequency response that extends to DC. This is technically not achievable in the case of communication circuits as they have transformers. In order to overcome this problem, the digital modulation takes their inputs in the form of Non-Return to Zero (NRZ) form. In this case, the encoded bits 1's and 0's are now transmitted as 1 and -1 state values. A basic signal representation of BPSK modulated LFM is shown in figure 2.29.

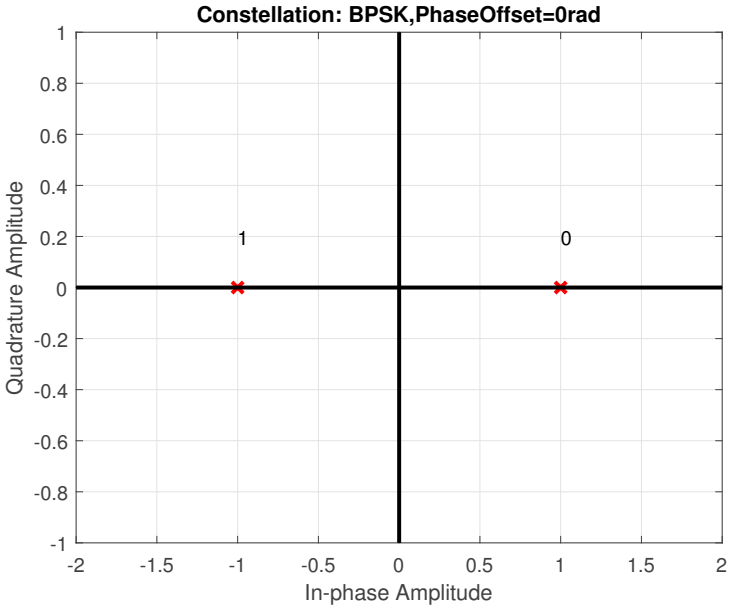


Figure 2.5: Constellation diagram of BPSK modulation

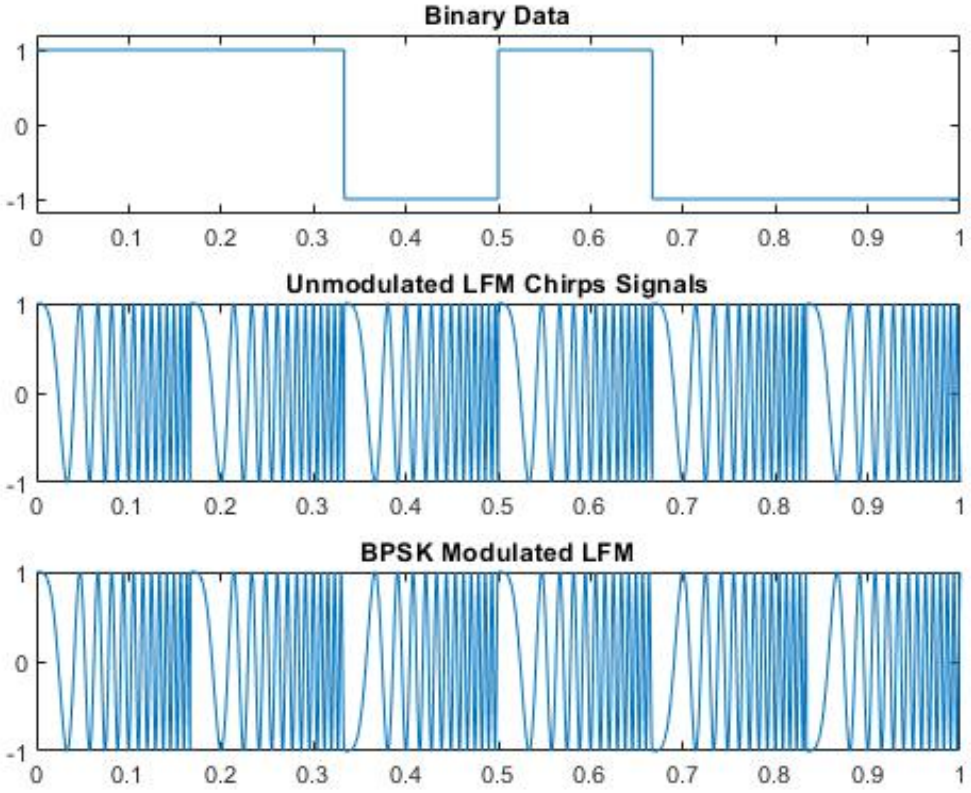


Figure 2.6: Signal representation of BPSK modulated LFM waveform

2.4. Joint BPSK-LFM waveform

According to [21], the BPSK-LFM signal can be represented as:

$$Y(t) = A(t).X(t) \tag{2.29}$$

where $A(t)$ is the BPSK signal and $X(t)$ is the LFM signal.

$$Y(t) = \sqrt{\frac{2}{T_b}} A_{tx}.cos(2\pi(f_c + St)t + \pi(n)) \tag{2.30}$$

where n is the data bits corresponding to either 0 or 1 that serve as a message signal.

In BPSK modulation, the phase shift occurs whenever there are two bits with different polarities adjacent to each other. This makes extremely easy for an observer to see the message encoding in the LFM.

2.5. Microcontroller Architecture

After looking into the automotive radar concepts, it is important to understand the architecture of an MCU in terms of memory and the processors. The general idea behind the utilization of multicore architecture for automotive radars is presented in this section.

An MCU is an intersect between a microprocessor and a microcomputer. The microprocessors, commonly referred to as the heart of a much larger system such as a desktop computer, constitute in what is called as the Central Processing Unit (CPU) of the computer. They consist of a Arithmetic Logic Unit (ALU), an instruction decoder, a number of registers, digital input/output lines, memory spaces such as a cache or stack which can be used for more fast temporary storage and retrieval of data than having to access system memory. There is a data bus provided to access the memory and input/output that are externally connected to the processor. Every processor has a memory architecture and based on that; the processor may have a few registers such as a program counter that keeps track of the address of the next instruction register that loads and stores the next incoming instruction. Additionally, there are general purpose registers available that stores data. A microcomputer, on the other hand, is a device that contains all the components of a computer in a small circuit, but not on a single chip. A typical microcomputer is an assembly of a microprocessor, memory storage devices, and input/output lines. They are commonly used for laptops and desktops but are not in use any longer.

A Microcontroller, on the other hand, refers to a single device containing the entire microcomputer on a single chip. That being said, an MCU is an assembly of the microprocessor, onboard memory, timers, and various other input/output devices. MCU has replaced microprocessors due to its flexible, scalable and straightforward use. The critical advantages behind integrating all the peripherals on a single chip are the reduced space consumption, lower manufacturing cost, low power consumption, higher reliability and shorter

development time cycle. All these advantages constitute in an ideal design of the embedded systems. Typically, an MCU can be configured to have specific functionality as per the applications dictated by the user.

Today, the embedded systems are based on the MCU's. Essentially, embedded systems find their applications in the consumer, cooking, industrial, automotive, medical, commercial and military fields. The use of MCU in the automotive industry is expected to lead to significant growth from 2016 to 2024 due MCU used for enhancing the control of autonomous vehicles range from 8-bit to 32-bit Harvard architecture that offers high performance, low-cost processors and efficient memory storage capability.

Due to the need for fast throughput alongside the occupation of minimal chip space, the Reduced Instruction Set Computing (RISC) is preferred as industrial CPU standard. Current CPUs, such as the ones used in the S32R274 MCU by NXP Semiconductors, have an efficient clock speed and processing power [11].

2.5.1. Memory architectures

In this section, we review the details on two architectures relating to the memory arrangement in an MCU and discuss the memory architecture used in the NXP's S32R274 MCU. We attempt to provide details within the scope of this thesis.

Von Neumann architecture

The first generation computers like Electronic Numerical Integrator And Computer (ENIAC) and other computational devices had fixed programs that were inbuilt with the machines. In order to change the program, the scientists have to rebuild the entire machine from scratch that would take several weeks and the computational time increased, and so did the machine complexity. During this time, the von Neumann architecture was named after a scientist who was involved in the Manhattan project which was dedicated to overcoming the complexity issue by developing the next computer called Electronic Discrete Variable Automatic Computer (EDVAC) that would store programs on the machine. This problem was fixed by using the von Neumann memory architecture that stored the program in memory, hence called as stored-program. This memory block is shared between the store-program and data storage, which allows data to be treated as code and vice-versa. This also leads to using self-modifying code that would reduce memory use and improve the machine performance. It was necessary for the CPU to now communicate with external memory, and this constituted as the bottleneck in such a memory model due to the less throughput and memory access time by different components on the machine.

Harvard and modern Harvard architectures

To overcome the Von Neumann bottleneck, a solution was proposed to allot different memory blocks for the program and the data. This immediately caused parallel and simultaneous access to the program memory and data memory. Another bottleneck to the Von Neumann architecture was using two

different read/write operations on the same path making it singular in operation. This increased the complexity of the machine. In the Harvard memory architecture the, write to the data memory and read from the program memory for the next incoming operation can be duly performed simultaneously. This reduces the time required for any instruction that accesses data memory. The Harvard architecture shares one major bottleneck of not being able to modify the program memory which hugely affects its usage for general purpose systems such as desktops. However, this disadvantage does not apply to embedded systems. Modern MCU's are built on a slightly modified Harvard architecture that allows both read/write operations to program memory used majorly for the boot-loaders.

Another variant is the modified Harvard architecture which is a cross of von Neumann and Harvard architectures which inherits the benefits of each model. Specifically, the data and program again share a memory space similar to how the von Neumann architecture works, however, data and instructions do not share cache memories or paths between the CPU and memory. This allows for less restricted memory access than von Neumann, and yet the ability to treat code and data as each other that Harvard lacks. One of the most common examples of processors that use this architecture is the x86 processor found in most personal computers [1].

2.5.2. Need for Multi-core architecture in automotive radars

The automotive industry demands for a more real-time based computational capability from the MCU so that it can fulfill different application needs. This leads to increasing complexity and power usage in case of using an MCU with a single core that is handling and executing different instructions at a given time. The single core processor work at a clock frequency that does not support high computational tasks. Apart from the architecture issues, it also becomes difficult at the hardware level such as heating issues and difficulty in testing and verifying the platform. Today's applications work on the principle of multithreading, and this trend is shifting towards parallelism. Even though the single-core systems provide an illusion of parallelism, it is far from achievement. Let us assume a scenario wherein a single core is performing 4 tasks of highest priority. These tasks are executed in an interleaved fashion, where switching between 4 different tasks could be impulsed at every fixed interval of time. This means that if the task with the highest priority is running, it will receive 25% of the processor time. This also introduces context switching overheads which leads to over-consumption of processor utilization. If we now consider increasing the number of tasks, it means that each task is receiving only some amount of the processor time. Even though the processor is fast, it still affects the application in terms of performance and resource utilization. In order to overcome this problem, we consider using a multicore processor system. Assuming the same 4 tasks now given to the multicore processor system, true parallelism is achieved by scheduling each task on one of the cores, which now means that each task now gets 100% of the processor's utilization. However, besides the advantages that multicore

platforms provide, there still are issues with synchronizing the cores that access the same shared resource which could cause computational complexity.

The autonomous cars today are increasingly becoming software driven. The car today contains more software with 100 million lines of code, and it will only exponentially increase. The automotive industry due to its advanced driving features is introducing complex challenges, and this will only need more scalability and high-performance processing speeds to meet the increasing demands of secure and safe driving. To achieve this demand, various industries have been manufacturing computationally efficient MCU as a means to provide signal processing acceleration and to meet the requirements of high-performance computations and fast chirp-modulation radar systems.

2.6. Conclusions

In this chapter, the brief concepts supporting the FMCW radar has been explained. One of the widely known modulation technique namely, BPSK, corresponding to encoding information in the FMCW radar waveform has been introduced. The motivation behind using an BPSK-LFM waveform was also presented. In the end, we attempt to provide a basic concept of an MCU and discuss the need for MCU in an automotive radar.

3

Hardware and Software Architecture

This section introduces the hardware platform namely the DCC Radar sensor development board used for automotive radar applications. The DCC Radar sensor kit is the integration of an S32R274 automotive MCU and a TEF810X 77GHz radar transceiver chip-set. The goal of this thesis, which is achieving joint waveform implementation to facilitate real-time processing, is realized on this hardware set-up. The first section presents architectures of the NXP's transceiver board and the S32R274 MCU platform that is based on multi-core architecture which meets high-performance computation demands required by the fast chirp modulation radar systems by offering unique signal processing acceleration modules.

The second section is intended to provide detailed information on the structure of a chirp and how it is configured by means of software tools such as an Application Programming Interface (API) along with an illustration on the structure of the extracted results of the raw ADC samples and the FFT from the DCC.

3.1. Hardware Architecture

This section is broadly classified into two parts which correspond to the hardware architecture on which the work presented in this thesis is carried out: the first part describes the radar front-end and the second section illustrates the MCU that is used for controlling the radar front-end to facilitate real-time signal processing.

3.1.1. TEF810x Radar front-end

The TEF810x radar front-end is a single-chip FMCW radar transceiver for short-, medium- and long-range automotive applications operating in the frequency band of 76-81 GHz[10].

Some key applications of TEF810X are:

- Adaptive Cruise Control (ACC)

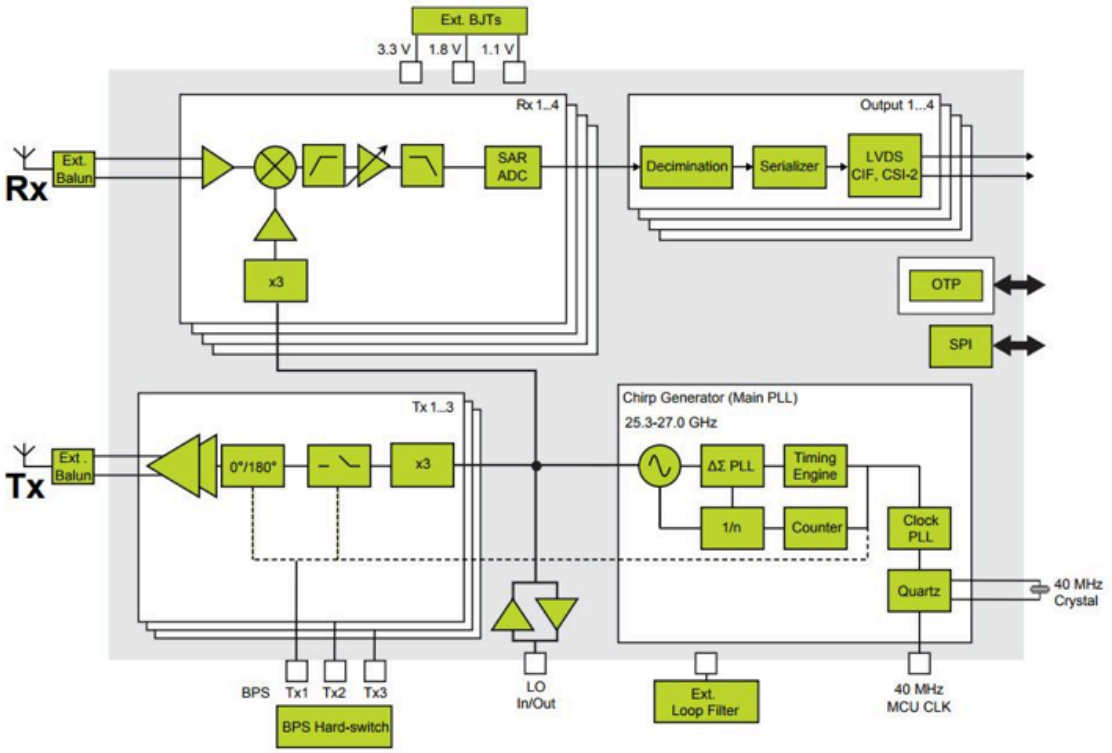


Figure 3.1: TEF810X Radar waveform generator module block diagram [10]

- Autonomous Emergency Braking (AEB)
- Blind-spot monitoring
- Collision avoidance
- Parking assistance

The figure 3.1 shows the flow diagram of various functionalities within the radar. The radar has three transmitters and four-receiver channels working on MIMO configuration. It consists of ADC's at each receiver path, a waveform generator module (WGM) with a timing engine that is connected to the MCU using Serial peripheral Interface (SPI) and is used to control the timing of each chirp and binary phase shifters at each transmitter chain. At the receiver side, there are high pass filters used for suppression of strong low-frequency signals and low pass filters for suppressing of signals in the ADC aliasing band. Each receiver has a SAR ADC that is responsible for digitizing the signals for processing and has a sampling rate up to 40MS/s and also consists of a programmable decimation filter with decimation factors of 2,4 and 8.

The ADC digitized output from the receiver end are serialized. To output this data to the MCU, there are 3 different drivers on the radar front-end: either CSI-2, CIF, or LVDS. The setup in this thesis work in the DCC uses the CSI-2 interface. The control of the TEF810x from the MCU is via SPI, and the adc data is transported from TEF810x to S32R274 via the CSI-2 interface. The CSI-2 interface can operate using 1, 2 or 4 lanes, and can have a sustained

data throughput rate of up to 2 Gbps.We attempt to explain one main module of our interest in this thesis in the following section which is the waveform generator.

Waveform Generator Module

The WGM is an integration of three main modules: the timing engine, sweep control block and a CW generator block. The timing engine is responsible for controlling the functional state (power up/down) of the transmitter and receiver are controlled by means of the enable control lines. It controls the phase settings of the chirp in cases where the chirp is encoded with data bits via the binary phase shifters. It also provides a fast switch (On/Off control) in the transmitter module that is responsible for changing the phase of the chirp by toggling the phase shifter on and off in quick succession during transmission of chirp.

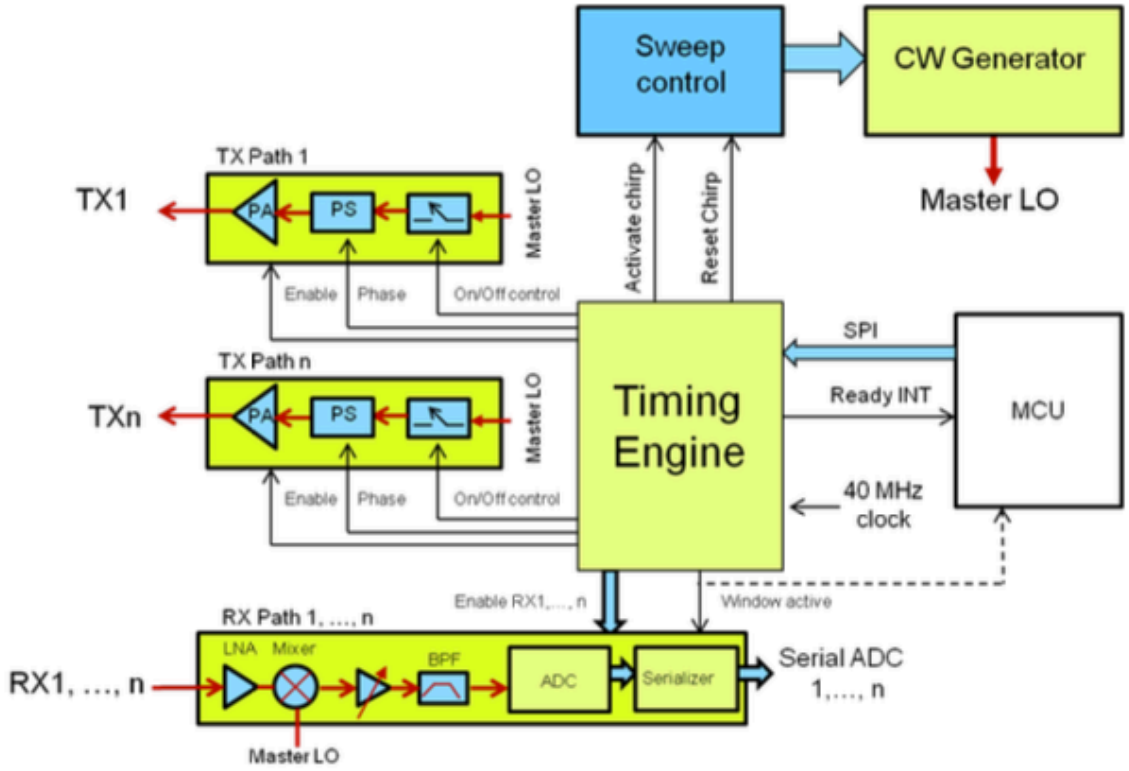


Figure 3.2: TEF810X Radar Front-end block diagram [10]

3.1.2. S32R274 Radar Microcontroller

S32R274 is a heterogeneous multi-core MCU platform. To support the radar and process the data for radars applications, the system is partitioned to make use of the multiple cores efficiently and available hardware accelerators to enhance the overall performance [11].

- Core_0 is an e200z4 processor running at 120 MHz clock frequency and executes the main entry point function and controls TEF810X. It is responsible for the triggering acquisition cycles (transceiver RADAR frame cycles) and managing the operation of the SPI as it processes the ADC data received from the RADAR front-end.
- Core_1 is e200z7 processor running at 240 MHz clock frequency which is responsible for handling the FNET networking stack. An interrupt request to core_1 is raised by core_0 either when the acquisition has completed (all ADC data is available in MCU Shared Random Access Memory (SRAM)) or when the Signal Processing Toolbox (SPT) has finished performing FFT operations. The interrupt service routine uses the FNET stack to transfer data according to the chosen output mode.
- Core_2 is not used in this application and remains disabled throughout the project.

The following sections magnify the specifications of each partition and also gives a brief detail on the communication module that is tailoring all the functionalities within an MCU and radar.

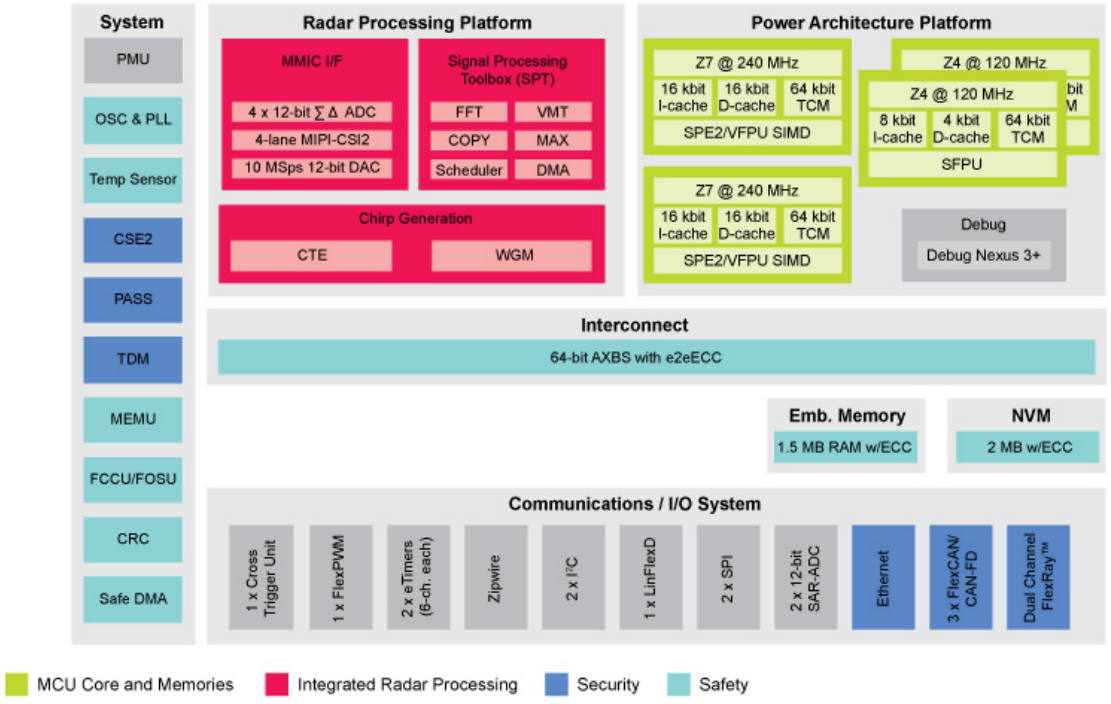


Figure 3.3: S32R274 Radar MCU block diagram [11]

Power architecture: e200z7 and e200z4

S32R274 is a multicore MCU with a safety core which has Power Architecture e200Z4 which is a 32-bit CPU and a checker core with a dual-issue computation which is a 32-bit processor and has power architecture e200Z7. The e200 family of processors were specifically designed for real-time applications that

require high computation performance. The e200z4 processor implements a low-cost version of power architecture technology (PC). To obtain high performances and low-cost MCU, three industries: Apple, IBM and Motorola collaborated on a common RISC architecture to design a new architecture which was named as Power architecture(PC). This new architecture resulted in improved clock rates, a more simplified instruction set, and a higher degree of super-scalar executions which supports 64-bit. An Instruction Set Architecture (ISA) by its definition intends to provide all the required information when someone wants to write a program in machine language or translate from a high-level language to machine language[15]. In other words, it specifies the functionalities that the processor has to provide to the designers. The e200z4 and e200z7 processor cores are compatible with the PowerPC ISA which makes the PowerPC industry-friendly as it offers a common ISA. One of the striking features of e200 processor family is that each core is a 32-bit dual-issue processor with 64-bit general purpose registers(GPR's) and independent instruction and data bus interface units.

Major features of e200z7 processor core include:

- 32-bit dual-issue compliant with PowerPC ISA
- 16KB, four-way set-associative Harvard instruction, and data caches
- signal processing extension unit (SPE) which supports SIMD fixed-point single-precision floating point operations using the 64-bit GPR's that include operations such as add, subtract, diff, multiply-add, compare, conversion and so on
- low power design-extensive clock gating
- In-order execution, interrupt and exception handling with nested interrupt capability and extensive interrupt vector programmability
- AMBA (advanced microcontroller bus architecture) AHB-Lite (advanced high-performance bus) 64-bit system bus
- Supports load, store and load multiple/store multiple instructions with a dedicated 64-bit interface to memory that offers storing and restoring up to 2 registers per cycle for multiple word instructions; two cycle load latency and supports Big- and little-endian

The e200z4 processor core differs from the e200z7 processor in the following features:

- 4KB, 2/4 way set-associative instruction and data caches
- Dual advanced high-performance (AHB) 2.v6 64-bit system buses

Communication modules

S32R274 offers various types of communication modes such as Zipwire, Serial inter-processor interface (SIPI), FlexRay, Integrated circuit (I2C), serial peripheral interface (SPI) and Ethernet MAC (ENET). The aim of this thesis is achieved by utilizing two communication protocols, namely SPI for radar to MCU communication and ENET for transferring the data over the internet. A few advantages of using SPI are that it has a faster data rate of 10s of Mbit-s/s as compared to I2C communication and is easy to implement in software. For transporting the processed data from the MCU such as raw ADC samples, range and doppler FFT values to any host for post-processing, we use ENET module.

Serial peripheral Interface (SPI)

The Serial peripheral Interface (SPI) communication module is primarily used for short distance communication in embedded systems and is responsible for exchanging serial data between the radar and MCU and is synchronous. The SPI works on the full-duplex principle which means that the communication between the radar and the MCU is bidirectional and can occur simultaneously. The SPI configuration makes use of a master-slave architecture and has four channels namely:

- SCK: Serial clock generated by the master to initialize the data transfer
- Master SOUT/ Slave SIN: Master Output Slave Input, or Master Out Slave In. This is the data output from the master peripheral
- Master SIN/ Slave SOUT: Master Input Slave Output, or Master In Slave Out. This is the data output from the slave peripheral
- PCS[0]/SS: Slave Select

The master peripheral initiates the transfer and generates clock signal SCK to the slave peripheral it selects. Since it is a full duplex system, during each SPI clock cycle, the master sends first data bit on the Master SOUT/ Slave SIN pin and lets the slave read it, and then the slave sends the bit on the Master SIN/ Slave SOUT pin for the master to read it. The CPOL and CPHA attribute to the Clock and Transfer Attributes Registers (CTARn) are responsible for selecting the polarity and phase of the serial clock SCK in the master mode. CPOL establishes the polarity of the SCK signal, and the CPHA determines the validity of the data from the slave output before or on the first edge of the SCK signal. Different configuration modes can be determined with respect to CPHA and CPOL. Further configuration modes with respect to CPOL and CPHA are out of scope to describe in this document. For ease of understanding a basic SPI master-slave transfer, we explained a basic SPI transfer format with CPHA = 1 as shown in the figure 3.4.

After the transfer of the first bit, the tCSC delay which is the delay between the PCS and SCK signals is elapsed during which the master and slave sample the first data bit on their serial data input signals. After this transfer is

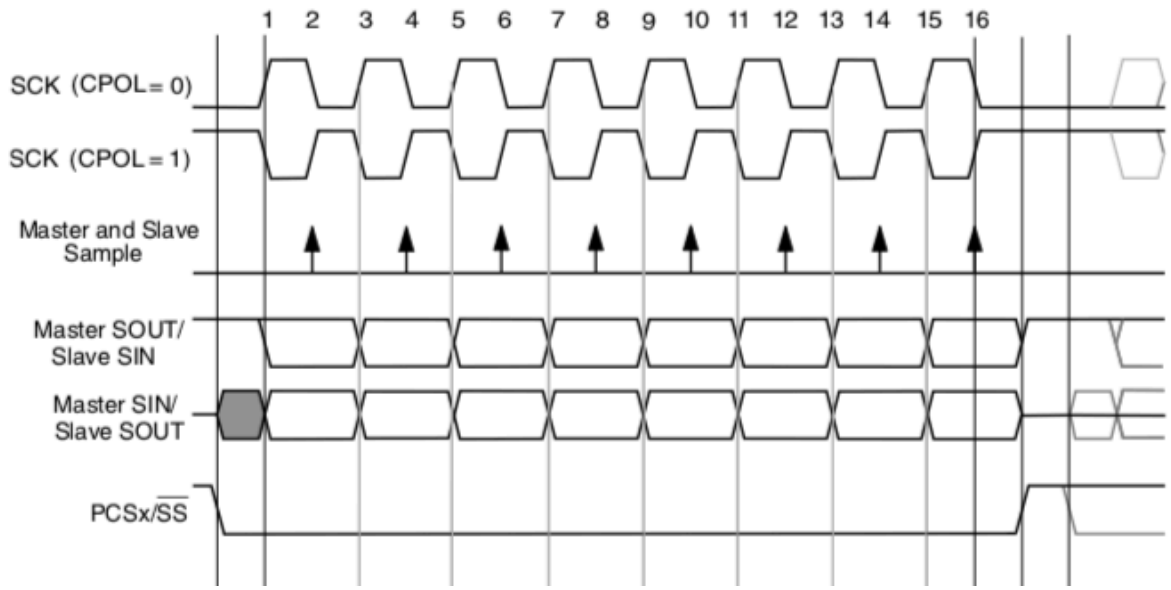


Figure 3.4: SPI transfer timing diagram [10]

complete, the master and slave peripherals now transfer the second data bit by placing them on the serial data output signals and sample their SIN pins. For the case with CPHA = 1, the master and the slave peripherals change the data on their SOUT pins on the odd-numbered clock edges and sample their SIN pins on the even-numbered clock edges [11].

Ethernet Media Access Control Address (EMAC) module

The EMAC module is mainly used to transfer data between the device and the host device that are connected on the same network based on the Ethernet protocol. This module is capable of implementing 10/100/10001-Mbit/s synchronous operations. The ENET is compliant with the IEEE802.3-2002 standard and provides half 10/100-Mbit/s and full-duplex Gigabit Ethernet Local area network (LAN)[11]. The purpose of describing this module in this document is its purpose of providing various hardware acceleration blocks to optimize the performance of network controllers that provide TCP/IP and UDP protocol services. We use these protocols in this thesis for establishing communication between the host device to the MCU to send a message that is to be encoded in the chirp and between the MCU to the host to send the raw data that is processed for various other radar applications. The use of these protocols is described in the system implementation chapter.

Signal Processing Toolbox (SPT)

The SPT contains all the hardware modules required for processing of the sampled radar signals. It is a powerful processing engine containing high-performance signal processing operations, driven by a specific use-oriented instruction set. The programmability ensures flexibility for modifications of signal processing. The Central Processing Unit (CPU) is removed from frequent scheduling of hardware operations, but still controls and interacts with the processing flow. SPT has an interface to the MIPI-CSI2 receiver module to

transport received ADC samples into the MCU Shared Random Access Memory (SRAM). The FFT engine is used to perform window and FFT operations on the received data.

3.2. Software Architecture

3.2.1. FMCW chirp and frame configurations

The TEF810x radar transmits a sequence of chirps in frames. It is possible to transmit up to 65534 chirps per frame. The TEF810x allows the user to control the parameters of chirps in a frame by essentially defining what is called as profiles. A profile is a template for the timing parameters of a single chirp. It is possible to define four profiles, and each chirp in a frame is associated with one profile. The chirp timing parameters that are set within timing engine registers per profile include:

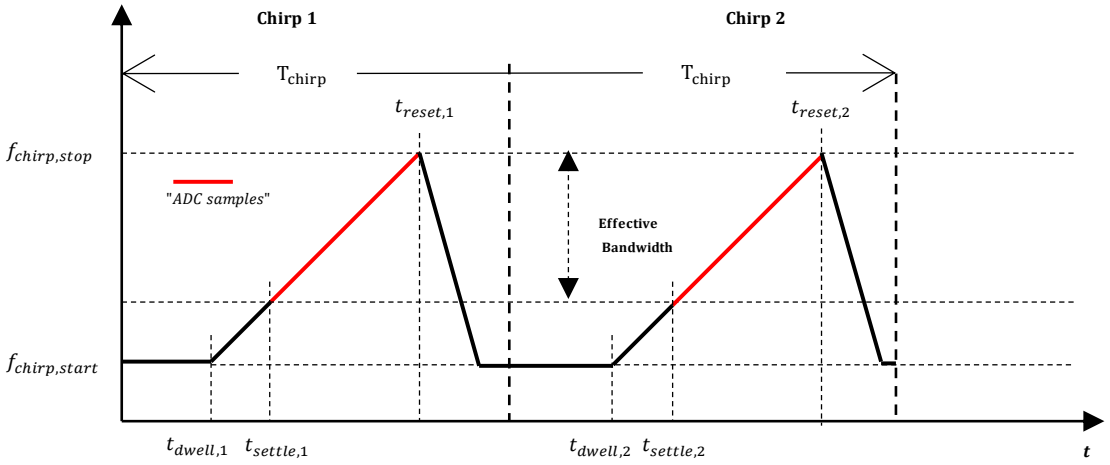


Figure 3.5: FMCW Chirp timing parameters [10]

- chirp period, T_{chirp}
- Dwell time, t_{dwell}
- Settle time, t_{settle}
- reset time, t_{reset}
- Number of ADC samples
- On/off control and the phase state of the TX sections.

3.2.2. Application Programming Interface (API)

In computer programming, an API is a set of routines, functions and data structures that are used to interface with the software components and make them communicate with each other. The NXP's radar API provides convenient access to the radar functions and offers full flexibility on configuring the radar and its chirp parameters depending on the application requirements. The following API calls are used in this thesis [16]:

- chip_ISM_Init
- chip_CC_IPFuncReset
- chip_CAFC_VCOFreqCalibrate
- chip_TE_ChirpTrigMode
- chip_TE_StaticConfig
- chip_Chirp_Program
- chip_LO_Control
- chip_TX_ProfileConfig
- chip_TX_Control
- chip_RX_ProfileConfig
- chip_CAFC_EnableVCO
- chip_CC_IPFuncReset
- chip_CC_SerializerInterfaceSet
- chip_CSI2_Config
- chip_SC_CWMode
- chip_MCLK_Recalibrate
- chip_TE_ChirpStart

The radar is initialized by calling the chip_ISM_Init() function. After the radar is initialized, it needs to reset in order to configure the radar registers by calling chip_CAFC_VCOFreqCalibrate() that is responsible for performing the loading of Voltage Control Oscillator (VCO) calibration table and chip_CC_IPFuncReset(). The next step is to program the parameters of the chirp by defining different profiles both at transmitter and receiver end. This is done by calling the chirp program, profileConfig functions for both transmitter and receiver. After this the radar has to reset again; therefore the chip_CC_IPFuncReset() is used, but this is different from the first reset function, after which the MIPI CSI-2 interface of the radar is configured. In the

end, the master clock is recalibrated with the chip_MCLK_Recalibrate() function. After the entire radar configuration is loaded, the radar now is now ready to start the transmission of the sequence of chirps in one radar frame by initializing the chip_TE_ChirpStart() call function. More API calls will be introduced in the phase coding section of chapter 4.

3.2.3. DCC data output modes

The MCU supports three output modes based on the acquired digitized samples of the received signal that it receives from the radar namely: ADC samples output mode, range FFT and doppler FFT. To fulfill the goals of this thesis, it is essential to extract the raw ADC samples and FFT results and store them on the host PC to further process them in MATLAB. For this purpose, it is essential to study the packet structures of these output modes to build plots that support our requirements in MATLAB. Further, to obtain the phase information with respect to the communication message, we follow the flow diagram shown in the figure 3.6 where we consider both the output modes. The most relevant results will be shown in the chapter 5.

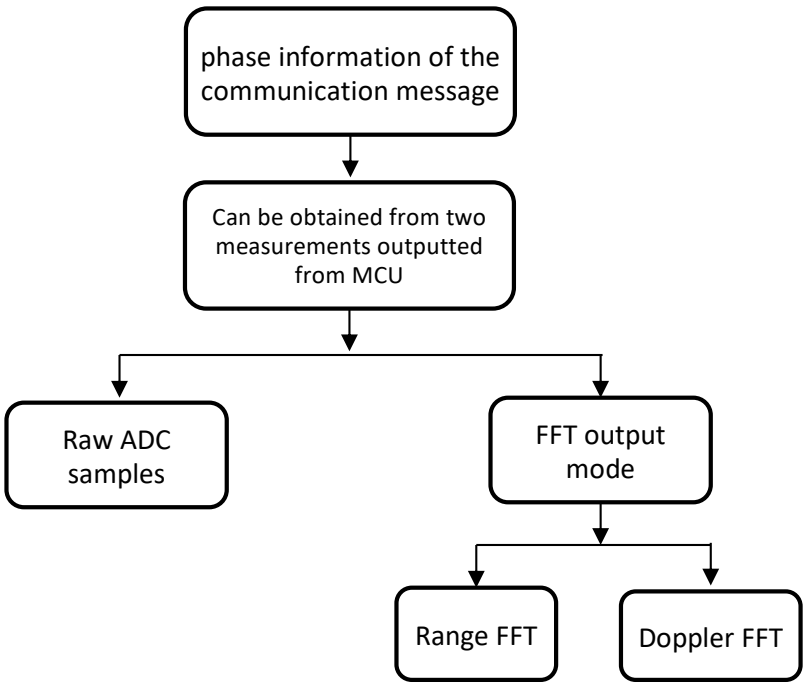


Figure 3.6: Flow diagram of obtaining phase information

ADC sample output mode

The ADC sample output mode acquires the ADC samples for a full RADAR frame (M samples * P channels * N chirps). After the acquisition, the radar

transports the ADC data to the MCU in the form of UDP packets via the SPI communication channel between the radar front-end and the MCU. This data is buffered in MCU SRAM, and the output transfer begins when the frame acquisition is completed. The ADC samples that are output over UDP are a signed 16-bit value. 1024 KB of MCU SRAM is allocated for buffering the ADC sample data, and each ADC sample occupies 2 bytes in the memory [11].

ADC Output mode	
UDP Payload Byte Offset	Field Description
0	Data type identifier (constant) 0x2016 = ADC data output
2	RADAR frame number (scan cycle counter) 0 - 65535 (rolls over)
4	RADAR chirp sequence number 0 to N (number of programmed chirps)
6	Packet number in sequence for this chirp
7	Total packets expected for this chirp
8 End of packet	ADC sample data

Table 3.1: ADC Output mode [11]

The ADC samples output is in the form of UDP packets. For each chirp in the radar frame, a sequence of UDP packets is transferred with a data payload that contains the ADC samples for the channels that are enabled. Each packet has a custom header before the start of UDP payload data. A typical UDP packet has a size of 1472 bytes of payload data. The structure of the ADC payload data header structure is shown in table 3.1. Generally, the ADC sample data for a chirp cannot be fully contained in the payload of a single UDP packet so it will be transported by a number of full-size packets followed by a smaller final packet containing the remaining data. Each of these packets has an ADC data payload header so that the sample data can be reconstructed at the receiver[11].

FFT data output mode

After the raw ADC samples are acquired in the SRAM, the FFT hardware accelerator in the SPT performs FFT operations. To obtain range FFT values, the FFT is performed on the real-valued ADC samples. Since the ADC samples are real-valued and applied as the input to the FFT to get range FFT, half the bins in the range FFT can be eliminated. This results in M/2 range bin outputs. With this, second FFT is performed for these range bin outputs across all the chirps to get the doppler FFT. This means that an M- Point FFT is performed using all samples of each chirp and N-point FFT is performed on the M/2 times. The output of the FFT is either a 24-bit or 16-bit signed

complex number. The host device receives the FFT results in the form of an array of unsigned 8 bytes. To further plot 2D FFT(range/doppler), the host device has to convert the unsigned 8 bytes integer to 32-bit signed complex double.

FFT Output mode	
UDP Payload Byte Offset	Field Description
0	Data type identifier (constant) 0x2017 = Doppler FFT Output (24-bit complex number format) 0x2018 = Range FFT Output (24-bit complex number format) 0x2019 = Doppler FFT Output (16-bit complex number format) 0x2020 = Range FFT Output (16-bit complex number format)
2	RADAR frame number (scan cycle counter) 0 - 65535 (rolls over)
4	Range bin index (0 – M/2)
6	Receiver channel (4 bits), 0 = Channel A, 1 = Channel B, 2 = Channel C, 3 = Channel D
6.5	Doppler FFT Output mode - Starting Doppler bin index (12 bits) 0–N, Range FFT Output mode – Starting chirp index (12 bits) 0–N
8 to End of packet	FFT result data

Table 3.2: FFT Output mode [11]

To explain this clearly, let us consider an example with the radar configured to transmit 256 chirps(N) in one frame. Each chirp consists of 256 samples(M). This means that for each receiver channel, we receive 128(M/2) UDP packets, one for each range bin which will give 128 range bins for 256 samples. These UDP packets contain 256 complex data points corresponding to one chirp each in the frame. These complex data points are fixed point 24-bit complex numbers. It can be possible that the FFT result data for a particular range bin is too large exceeding the usual 1472 bytes of payload size in a UDP packet. Therefore, it is sent in another packet. For the first packet with FFT result, the UDP byte offset field at 6.5 has the index of 0, and the other packets that follow the first packet are given an index of 183 which is clearly mentioned in the table 3.3. In our case, we configured 256 chirps and obtain 24-bit complex values. Two UDP packets with index 0 and index 183 are required.

Starting Doppler bin Index						
Chirps (N)	Complex Number Format	Memory size:Single Range Bin	Total UDP Packets Needed	Packet 0 Index	Packet 1 Index	Packet 2 Index
128	24-bit	1024 bytes	1	0	N/A	N/A
128	16-bit	512 bytes	1	0	N/A	N/A
256	24-bit	2048 bytes	2	0	183	N/A
256	16-bit	1024 bytes	1	0	N/A	N/A
512	24-bit	4096 bytes	3	0	183	366
512	16-bit	2048 bytes	2	0	367	N/A

Table 3.3: FFT Output mode [11]

3.3. Conclusions

This chapter reflects on the hardware and the software platforms that support our thesis work. The details on the architectures of both the radar front end and the MCU that relates to this work have been presented. In terms of the radar front end, of all the other modules on the radar, information on the timing engine module in the waveform generator module was explained.

With respect to the MCU, we discussed the functionality of different on-board cores(processors), main features of the processors, communication module responsible for establishing the communication between the three devices (host PC, MCU, and the radar front-end) and signal processing toolbox that is responsible for computing algorithms for various radar applications. Two different MCU output modes namely: raw ADC sample values and FFT which is the result of the SPT performing algorithms on the raw ADC data have been discussed.

4

System Implementation

In the previous chapters, all the necessary theory on the FMCW radars, different waveforms considered to achieve joint sensing-communication functionality along with supporting the theory on MCU's have been explained. Along with this, related software and hardware architectures on which the implementation is carried have been explained in detail.

The purpose of this chapter is to describe and clarify various aspects of the implementation conducted to achieve joint-sensing and communication functionalities. The chapter is sectioned into four parts. The first part explains various communication protocol schemes to build a reliable communication channel between the host PC to MCU to send and receive information back and forth and address the need of intercore MCU communication.

The second part explains the methodology to implement the phase modulation on the LFM chirps to encode the message received from MCU by the host PC. The third part illustrates the MATLAB post-processing scenarios to validate the radar and communication functionalities and the fourth part presents the test scenario considered for analysis of the data.

4.1. Microcontroller programming

The communication functionalities needed to fulfill the goal which is modulating the LFM chirps generated by the waveform generator on the radar front-end by the information that is present in the MCU fed by the host machine is broken down into three stages:

1. Radar to MCU communication via SPI
2. MCU to host communication via Giga Ethernet using TCP/IP protocol for exchanging the communication message
3. MCU to host communication via Giga Ethernet using UDP protocol to output raw ADC samples and plot range and doppler FFT acquired by the MCU in order to observe and test the phase coding

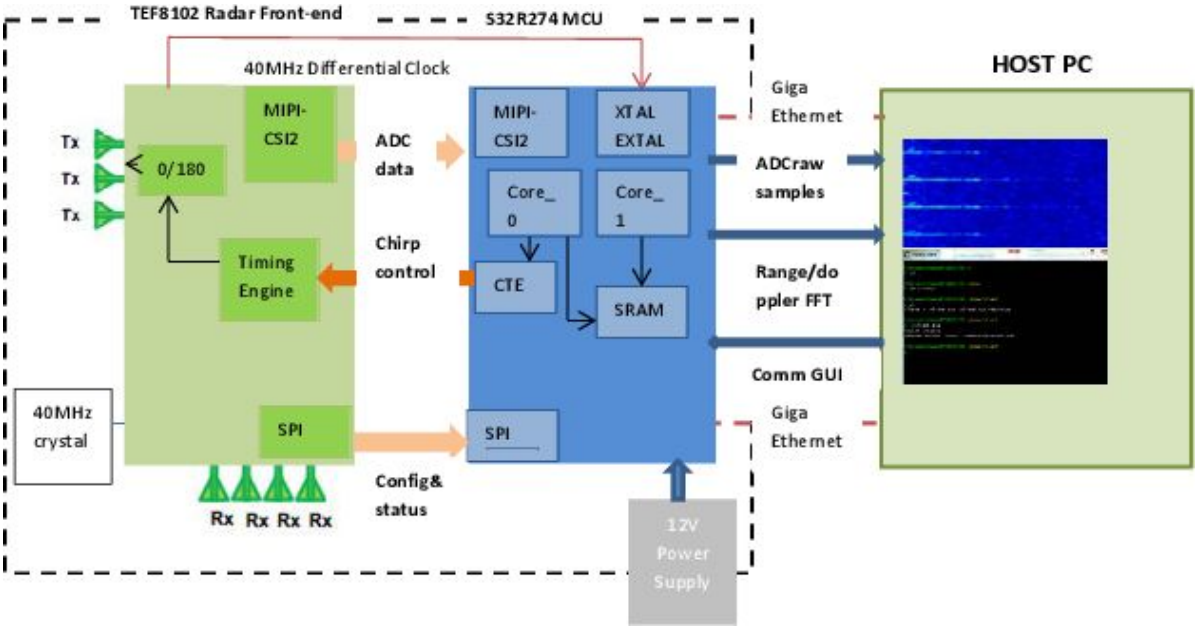


Figure 4.1: System implementation block diagram

4.1.1. host PC to MCU Communication

To be able to achieve an application that would send a message across the MCU, the host has to establish a communication channel with the MCU. Essentially, these two devices will be communicating over the internet. The most commonly used internet communication protocols are TCP/IP and UDP to send bits of data in packets to an IP address. TCP/IP belongs to the transport-layer protocol in terms of the Open Systems Interconnection model(OSI) network model. It provides a reliable virtual-circuit connection between two devices connected over a network, that is, a connection is first established before data transmission begins. The information is transferred without errors or duplication and is received in the same order as it is sent. TCP/IP treats the data as a stream of bytes. The S32R274 MCU is configured to support Ethernet stack for communication purposes on core_1 which is based on the FNET stack. The FNET is a free, open source dual IP stack to attain embedded communication software for 32-bit MCU and is primarily used to support data exchange over UDP. However, we also extended the possibility of it to support additional network services such as TCP/IP.

UDP is also a transport-layer protocol and is an alternative to TCP/IP. UDP is not preferred for this task due to its unreliable datagram connection between the devices. Data is transmitted link by link, and there is no end-to-end connection. The service provides no guarantees. Data can be lost or duplicated, and datagrams can arrive out of order. Therefore we choose to implement the TCP/IP protocol due to its data integrity for this task as no loss of data is desired. For this purpose, we built a listening echo server on the DCC board. Moreover, we implemented a client connection on the PC which makes communication possible between MCU and any PC. This setup

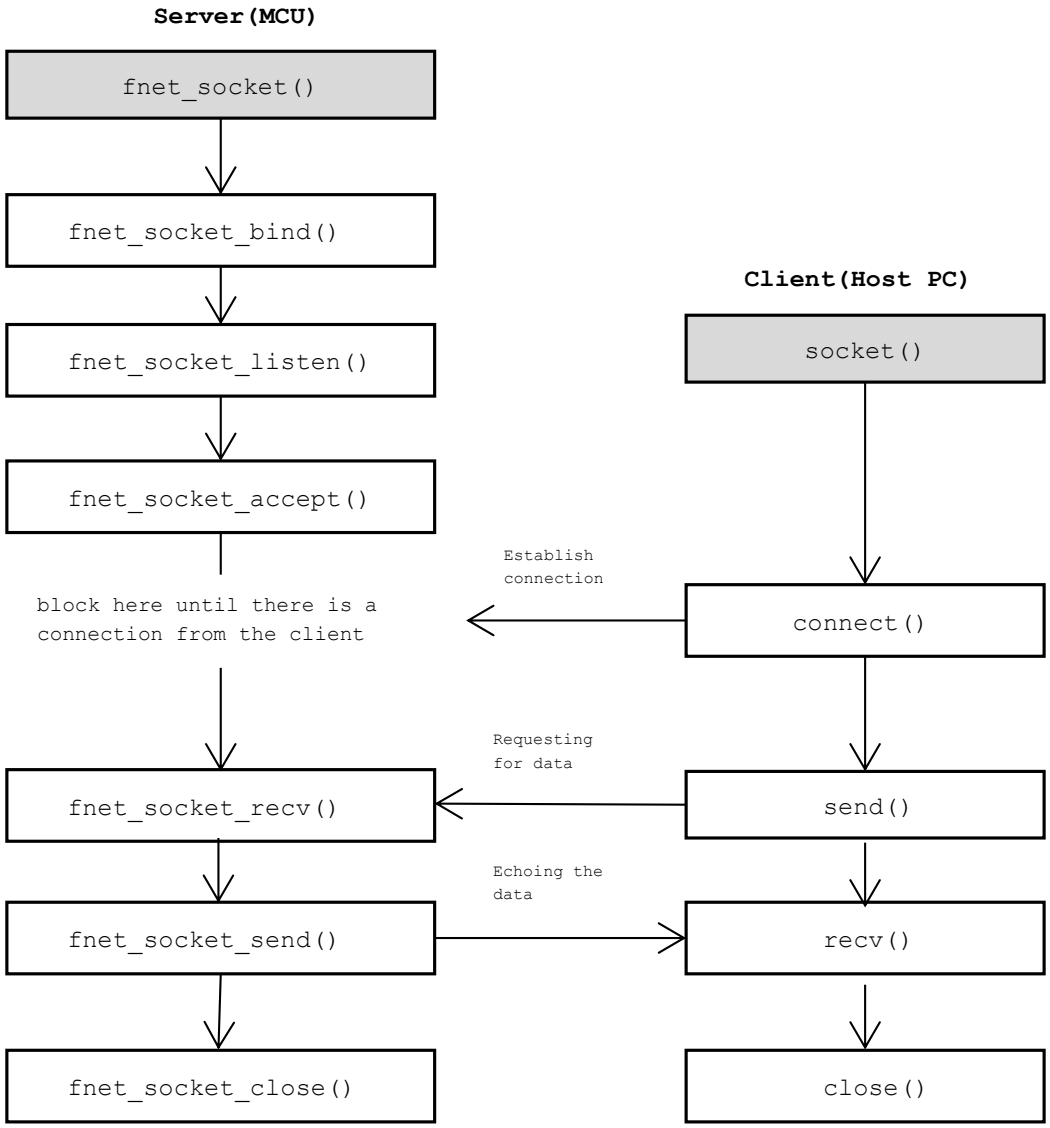


Figure 4.2: Flow diagram for establishing TCP/IP communication protocol between client and server

is bidirectional. The TCP/IP protocol works on the functionality of creating sockets at both the client and server ends. We use one of the widely known socket programming API namely the Berkeley Socket Distribution (BSD) sockets that facilitate an easy flow of creating various distributed applications that exchange the data between programs located on two different or same computer devices without the user having much understanding on the various levels of the OSI network model. In order to use this software tool, a certain set of system calls are utilized to establish communication endpoints known as sockets and then exchange information between the two devices each configured with a socket. To utilize BSD sockets, we will make two different applications, each associated with what is called as a client/server model

where one process (the client) makes a request for a connection and the other process (the server) accepts that connection request and initializes the data transfer. The method in which the task of sending data from the host PC to the MCU is performed is shown in the flowchart of figure 4.2.

The server, in our case for this task, is the MCU. The server first creates a socket to initialize the data transfer, binds it to an address and sets up queues to listen to the incoming connection requests. On the client (host PC) side, a socket is created as well and bonded with the same address as of the server. After creating the socket, it requests for establishing a connection with the server to whom it has to send the data. Soon after the server accepts the client's request for connection, it grants, and a bi-directional communication takes place. Once the connection is established between the host PC and the MCU, the client sends a message, and the server acknowledges it by echoing the message which proves that the MCU has received the message sent by the client. After the communication process is finished, the close call terminates all the sockets.

4.1.2. MCU to host PC communication

The received signals from the radar front-end are processed into three output modes: raw ADC samples, range, and doppler FFT. The visualizer application on the host PC to create the plot of 2D FFT (range/doppler) is achieved by establishing a UDP communication channel with the MCU. Unlike the communication application using TCP/IP protocol, this task uses a UDP connection due to the following advantages. One of the attractive features of UDP is since it does not need to retransmit the lost packets nor does it do any connection setup, sending the data comes with less delay. This lower delay makes UDP an appealing choice for delay-sensitive applications like audio, video and real-time applications. Also, UDP has a small packet header overhead of only 8 bytes whereas TCP/IP has 20 bytes of header.

The Radar software programmed on MCU supports three output modes: raw ADC samples, range FFT or doppler FFT. The default output mode is doppler FFT. In order to capture raw ADC samples and plot range FFT, the default configuration on radar and MCU is modified. The default output mode performs 2D FFT on the acquired ADC samples and outputs the FFT results as User datagram protocol (UDP) packets over the Ethernet interface. To further process the FFT values in order to obtain various results corresponding to the testing of phase coding is achieved by making a UDP listener application on the host PC that will receive all the data of the FFT in real time. The application is built on the same principle such as the TCP/IP. However, since UDP is a connection-less protocol, the calls such as listening and accepting from the server side is not needed.

The server, in our case for this task, is the host PC. The server first creates a socket to initialize the data transfer. On the client (MCU) side, a socket is created as well and bonded with the same address as of the server with the system call fnet_Socket(). Once the connection is established, the client sends

the data to the server using the call fnet_socket_send(). After the communication process is finished, the close call terminates all the sockets with close() call. The method in which the task of sending SPT processed ADC samples and FFT values from the MCU to the host PC is shown in the flowchart of figure 4.3.

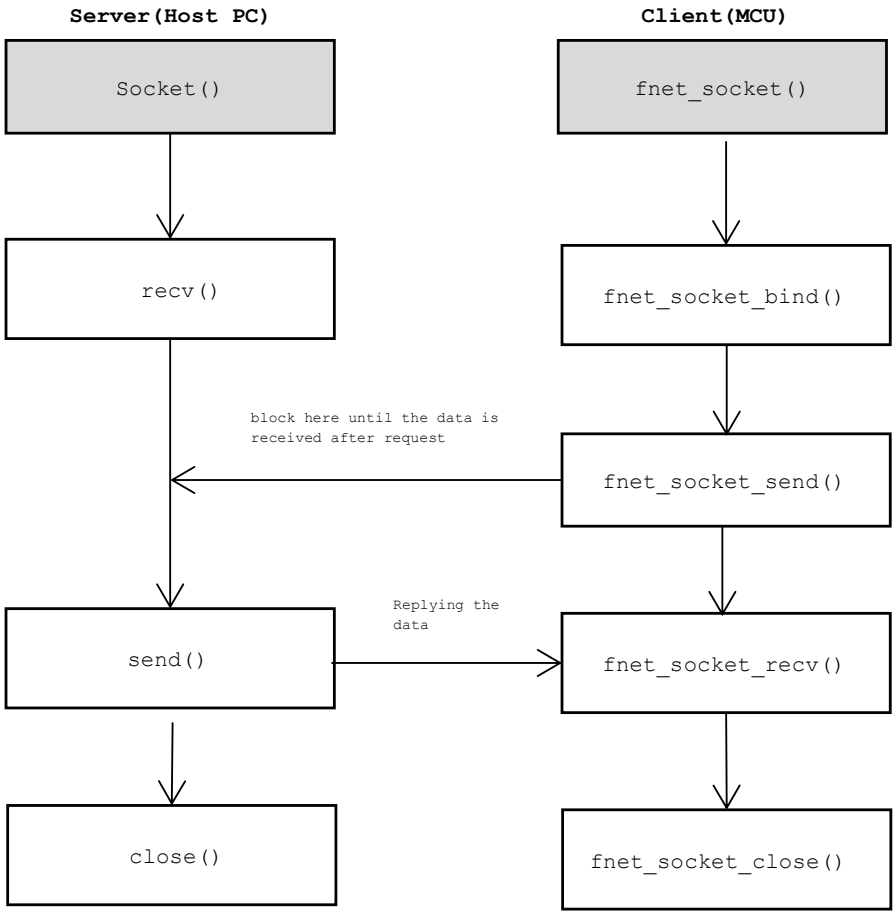


Figure 4.3: Flow diagram for establishing UDP communication protocol between client and server

4.1.3. Inter-core communication

The Core_0 on the MCU handles the radar processing. The Ethernet stack responsible for all the communications is located on core_1. The incoming data from the host to the MCU is handled by the core_1, that being said, this core stores the data it received from the host into the local memory. This data has to be applied to the chirps generated from the waveform generator situated on the radar front-end. The radar front-end is controlled by the core_0 on the MCU. This means that the core_0 now should be able to communicate

with core_1 to apply the data to the chirps. For this purpose, we use a special module namely hardware semaphores present on MCU [17].

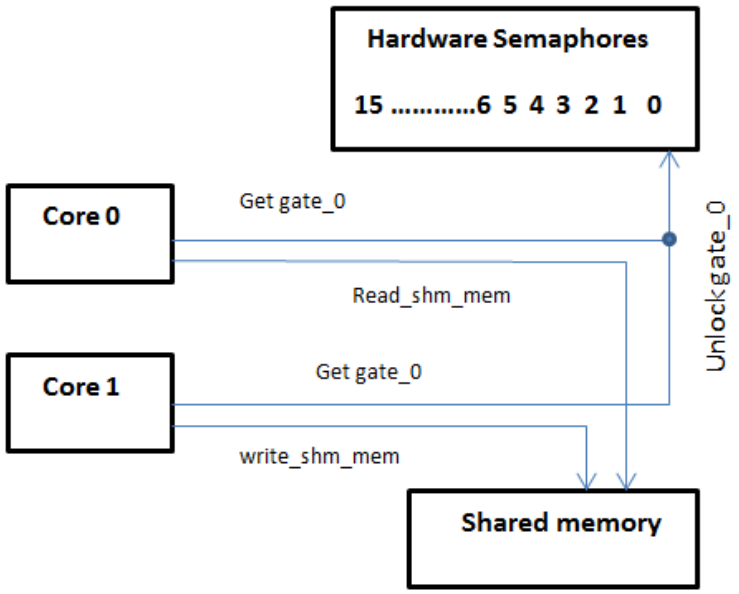


Figure 4.4: Hardware Semaphores mechanism for Inter-core communication

In a dual-processor chip, semaphores are used to let each processor know who has control of shared memory. In our case, core_0 and core_1 share the same memory to read and write respectively. Before a core can update or read memory coherently, it has to check the semaphore to see if the other core is not already updating the memory. If the semaphore is clear, it can write common memory, but if it is set, it has to wait for the other core to finish and clear the semaphore. During initialization of the process of reading and writing, the shared memory is assigned to semaphore gate 0.

- Core 1 locks the semaphore gate 0. Updates the memory, unlocks the gate, and generates a signal to the core 0 to read.
- Core 0 waits for core_1 to finish writing into the shared memory and after receiving signal to read, it locks the gate 0, reads and resets the gate.
- Before writing in the shared memory again, core_1 has to check the semaphore to see if the other core has finished reading, in order to prevent overwriting. If the semaphore is clear, it can write into the memory, but if it is set, it has to wait for the other core to finish reading and clear the semaphore.

One of the major reasons why we implemented semaphores for inter-core communications is its mutual exclusion principle. The shared memory is accessed by both the cores in the critical section of their semaphore code

```
if (!GetLock(SEMA42_GATE0))

// lock shared memory for current core
     Lock(SEMA42_GATE0,CORE_ID);
          buf[i] =
px1->ms3_radar_data[ms3_rdata_index];
     core_z4_0_counter++;
         // we are done - lets unlock
shared memory
     Unlock(SEMA42_GATE0);
          INTC.SSCIR[1].B.SET = 1;
```

Figure 4.5: Hardware Semaphores for Inter-core communication in core_0

```
if (!GetLock(SEMA42_GATE0))
// lock shared memory for current core
           Lock(SEMA42_GATE0,CORE_ID);
           INTC.SSCIR[1].B.CLR = 1;

memcpy(&ms3_radar_data[ms3_rdata_index],
buf, read);
            ms3_rdata_index += read;
            core_z7_1_counter++;
           // we are done - lets unlock
shared memory
Unlock(SEMA42_GATE0);
```

Figure 4.6: Hardware Semaphores for Inter-core communication in core_1

segment which can be seen in the red box of the figures 4.5 and 4.6. It is essential to ensure that while one core is executing in its critical section, the other core should not be allowed to execute in its critical section. This means that the access to a particular section must be an atomic action. One core can execute its critical section by waiting until the next core has entered and left its critical section. In other words, only one core at a time is allowed to execute its critical section. Therefore, we made this process more robust by using interrupt signals in the code segment of semaphores for each core. From figure 4.5 and 4.6, it can be seen that the interrupt signal set by core_0 is cleared by the core_1 to write the data in the shared memory and leaves

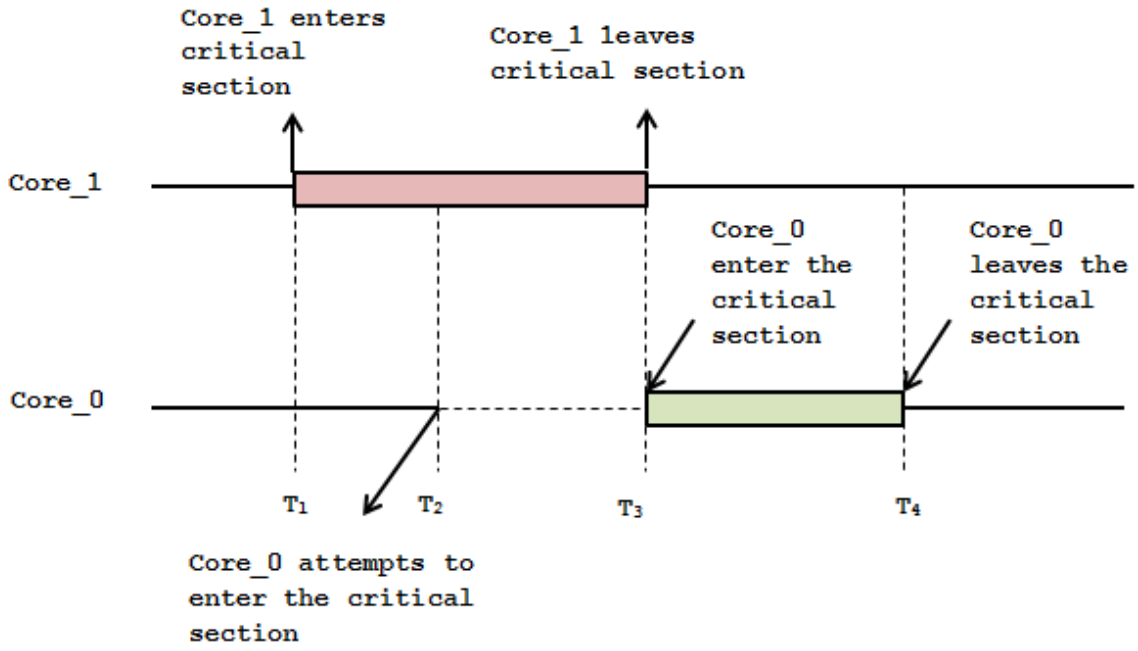


Figure 4.7: Mutual exclusion principle

the critical section. Following this, core_1 executes its critical section and sets a signal to the core_0 that it can execute its critical section. This mechanism loops for as long as there is data to write and read without causing any synchronization issues as seen in 4.7.

4.2. LFM- BPSK Implementation

To achieve the goal of phase coding the linear frequency modulated chirps, the MCU is programmed in association with the timing engine that is located on the radar front-end. The timing engine and the MCU are synchronized with the same clock and connected using a serial peripheral interface (SPI) communication protocol. In order to transmit the information, the wave has to be initially phase coded (with 0 or π), which means that each chirp can be coded with one bit at a time. There are two ways to control the phase shift from MCU:

- Using timing engine module settings: timing engine consists of 4 chirp profiles. Each chirp profile is a template that defines the chirp timing. In other words, each chirp in the frame of a sequence of chirps that is transmitted is associated with one of the four profiles.

- Using the phase shifter: make the phase shifter follow the level on an I/O pin via the hardware BPS hard switches as seen in the figure 3.1. This makes the phase shifter on the radar board to toggle on and off at

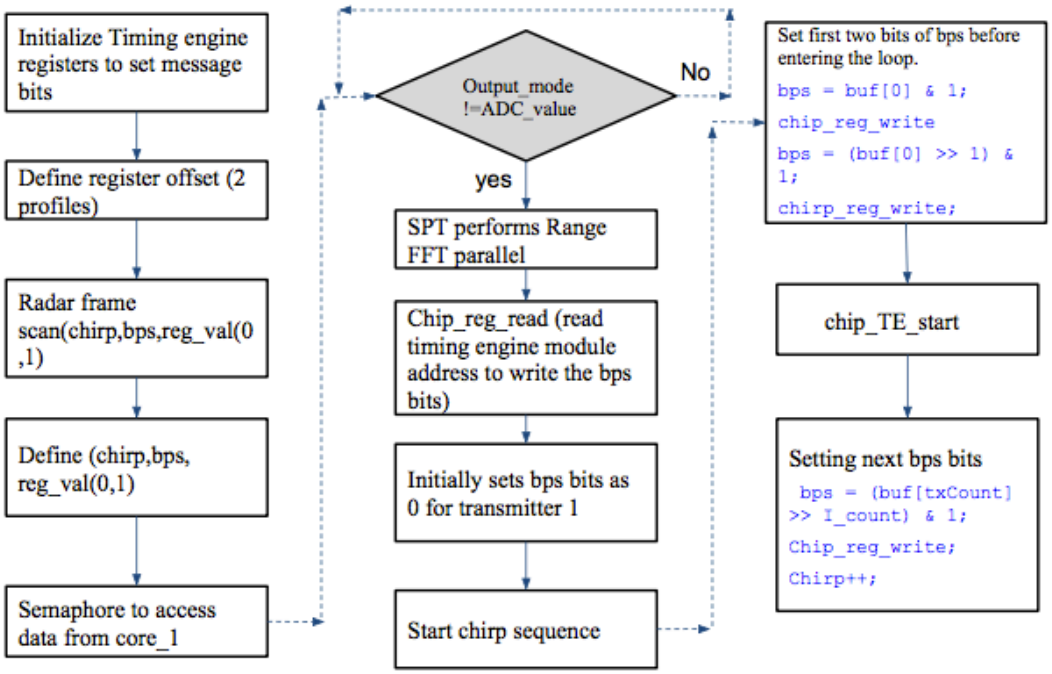


Figure 4.8: Software Implementation of BPSK-LFM modulation on MCU on core_0

quick succession between the chirps.

The second technique allows for a higher bit rate from the communication perspective but is harder to decode and also harder to retain the normal radar functionality. Therefore, we implement the phase coding using the first technique which utilizes the hardware timing engine registers. The flowchart of the implementation is shown in figure 4.8 We first initialize the timing engine module located on the radar front-end from the MCU in order to send the bits of information from MCU to the radar front-end transmitter end.

One of the ways of sending the bits to the radar front end is by accessing the API calls. However, for ease of implementation and to avoid unwanted overheads, we directly access the timing engine registers that also hold the phase shift settings. Once the timing engine registers are initialized, it is essential to simultaneously initialize the semaphore module as it is required for the core_0 to get the information bits from core_1 as all of the radar processing and sending of bits to the timing engine module is taking place in the core_0. Once the core_0 gets message bits, it proceeds with signal processing. In this implementation, the ADC values and the processing of range takes place simultaneously in the SPT. The timing engine register where the bits will be sent is first read by the MCU via the API call ”Chip_reg_read”. The register contains three bits for the phase shift settings, one for each transmitter. We make use of only one transmitter to toggle the phase between the chirps. These bits are collected in the variable binary phase shift (BPS) correspondingly in each profile that is responsible for enabling a 180-degree phase

shift for each transmitter.

Essentially, a profile is a collection of radar front-end settings corresponding to the chirps, like frequency and timing settings, and - more importantly, the BPS settings. The timing engine can be configured to use a different profile for each chirp. So, by creating profiles with different BPS settings, we can control the BPS setting in the transmitters. Since only four profiles can be defined, which is far less than the number of bits in the communication message, we cannot create a profile for each message bit due to hardware and memory limitation. Therefore, we apply a strategy here: we configure the timing engine to alternate between profile 0 and 1 for the consecutive chirps. Moreover, while the timing engine is using profile 0, we update profile 1 with a new BPS setting, and vice versa as illustrated in the figure 4.9. Here, we considered one ASCII character associating with the communication message of our interest, "*H*". The binary bits corresponding to the letter "*H*" are 01101000. From the above strategy, we can observe that while sending the first bit "0" in the profile 0, we are already configuring the second bit in the profile 1. Then, while sending the second bit the profile 0, we are again configuring the profile 0 with the third bit. This continues for all the bits in the message.

In the sequence of the chirps in a frame which is encoded with one bit each, we keep the values of the first two chirps constant. These associates with the message passing frame where the payload data can be easily identified in the stream of different radar frames.

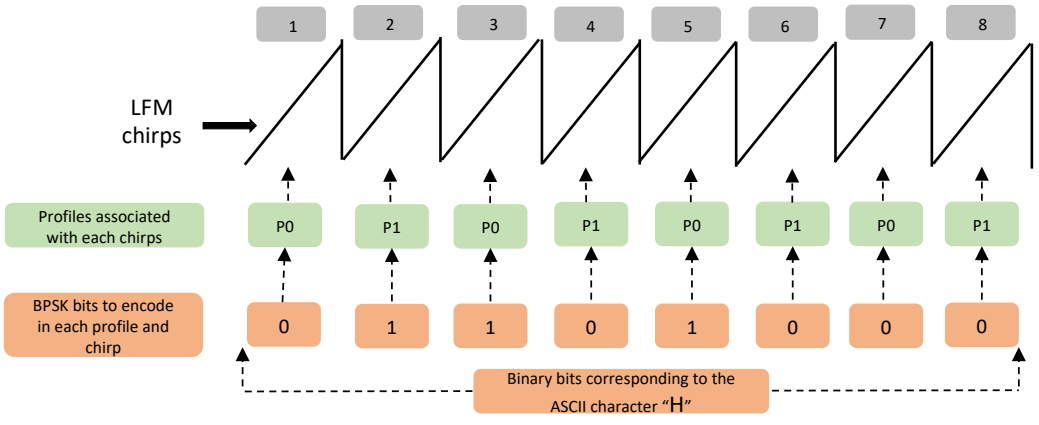


Figure 4.9: Software Implementation of BPSK bits in each chirp

The information bits received from the core_1 are stored in a buffer of core_0. To send these bits to the register, we consider each byte of the message and implement it in an 8-bit counter. The 8-bit counter triggers the bit at the least significant position of the byte at every iteration and keeps

shifting for eight times and masks it with and operation. By doing this, at every instance, the BPS variable gets one bit from the byte of information and writes this to the timing engine register using the chirp_reg_0 API call. These registers are associated with each profile.

It is only possible to send and reconstruct the data bits if the number of bits is equal to the number of chirps. Therefore, for the configuration involving 256 chirps, it is possible to send 256 bits. For example, consider a message with message size as 10 bytes. From the implementation perspective, it is only possible to embed one bit per chirp. This means that in order to send $10 * 8bits = 80bits$ of data, we still require $256 - 80bits = 176bits$ of dump message bits. In order to overcome the dump message bits, we optimized our client (host PC) from where the message is initially sent, by making the client append 0's after the message ends if the message size is less than 256 bits. In this way, we achieve 256 chirps with 256 bits of information.

4.3. MATLAB Post-processing

The center goal of this thesis is to extract phase information from the beat signals in order to read each bit of information encoded in the chirp as it is known that the beat signals contain the information of range, doppler, and phase. To realize this, it is important to understand the digital spectral analysis of FMCW radar. This analysis is broken down into range and doppler processing.

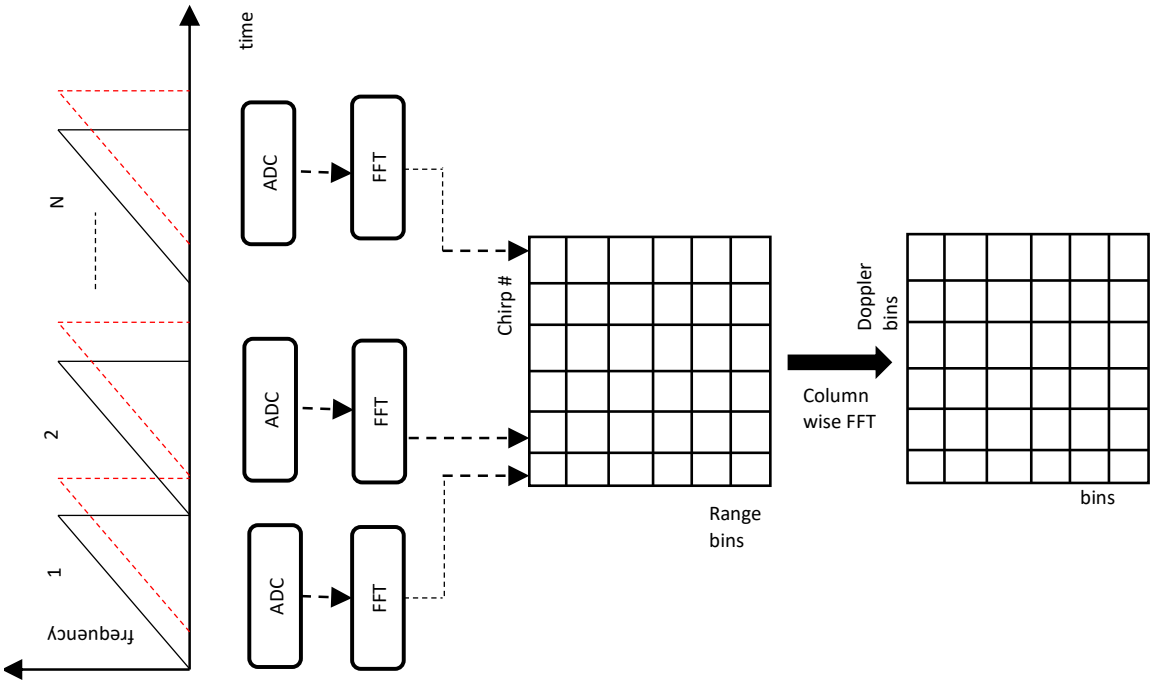


Figure 4.10: Range/doppler processing

The relation between the target's range and beat signal is shown in the equation 2.18. The target's range can be computed from the beat frequencies[22]. The beat frequencies can be obtained by sampling the beat signal and applying FFT along the sampled sequence. This results in obtaining the range dependent amplitude and phase values.

This means that each chirp generates M samples. For N chirps, a block of N*M samples is formed. These sampled values are stored in a data matrix comprising of rows and columns, and a specific memory location on MCU is dedicated to it, after which, the data matrix is subjected to the range and doppler processing. Each row in the data matrix now consists of all the samples of a single chirp. In a sequence, these rows, one by one undergoes FFT resulting in N*M block. Each row comprises with the baseband signals of the corresponding chirps. The row data in the matrix is commonly referred to as slow-time. To obtain velocity information, FFT is applied on the columns of the data matrix yielding the range/doppler spectrum. This is referred to as fast time. Since we can extract all three output modes: ADC samples, range FFT values and doppler FFT values from the MCU after performing signal processing on the acquired samples, we choose to extract values of both raw ADC samples and doppler FFT. By doing so, we show that it is possible to reconstruct the communication message by looking at the phase information in the case of the stationary target by post processing both the ADC samples and the doppler FFT values in the results chapters.

4.4. Validation test case scenario

To validate our BPSK-LFM implementation, we considered a validation test scenario with a stationary target. The stationary target considered for this purpose is a strong corner reflector. Corner reflectors are very useful to reflect strong radar echoes from the target making it useful for radar system calibration.

In this validation test case scenario, the DCC module is kept constant at a fixed distance from the corner reflector. Once this setup is arranged, we initialize the output mode from the MCU to be either raw ADC samples, range FFT or doppler FFT. Once the initialization is defined, the UDP server on the host PC is launched to start collecting the required data from the output mode selected and store it in a binary file (.bin) for further MATLAB post-processing. Further, we considered the values shown in the table 4.1 to configure the chirp parameters. Two configurations were provided in the demo kit initially to configure the radar front-end. However, to accomplish our implementation, we made use of the configuration_1.

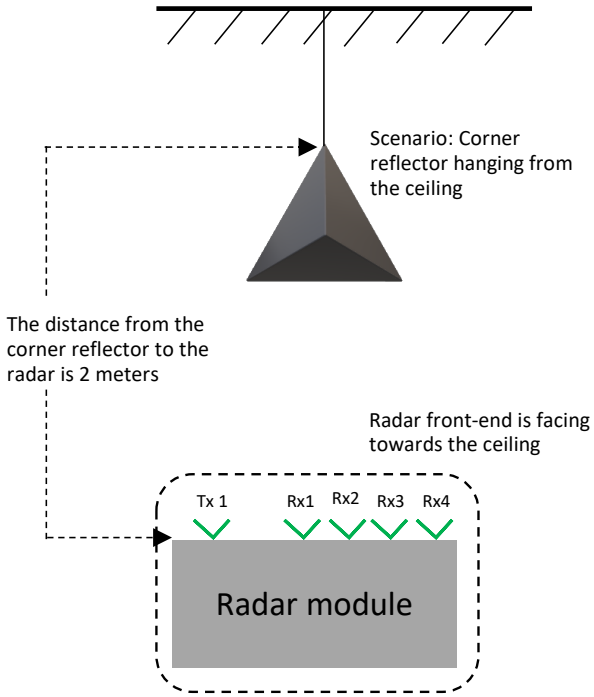


Figure 4.11: Validation test case scenario setup

Chirp parameters configuration			
Parameter	Unit	Configuration_1	Configuration_2
Start frequency	GHz	78.5	78.5
Number of chirps (N)		256	128
Dwell time	us	40.05	13.8
Settle time	us	5	5
Jumpback time	us	0.25	0.25
Reset time	us	5	5
ADC Samples (M)		256	512
ADC sample rate	Msps	10	20
Minimum chirp rate	us	26.3	49.5
Chirp bandwidth	GHz	1.5	375 MHz
Data rate	MBPS	240	240
Receiver channels		4	4

Table 4.1: Chirp, profile and frame configuration parameters

4.5. Conclusions

To fulfill the aim of embedding communication message in an LFM waveform, we first built a client on the host PC as a source of sending the message bits to the MCU which acts as s server to receive the message bits. The client-server model is built on the TCP/IP protocol concerning the ethernet connection between the host PC and the MCU. We chose to implement the TCP/IP protocol due to its data integrity and message delivery guarantee. The incoming message bits are stored in the local memory of the core_1 of the MCU as the ethernet stack is located there. However, the message bits can be further sent to the radar front end only from the core_0 as it is responsible for configuring the radar and for radar processing. To solve this issue, we used hardware semaphores to create an inter-core communication between the core_0 and core_1. Following this, the software implementation of the BPSK-LFM waveform was achieved.

To validate the communication message we sent across the radar LFM waveform, we consider extracting two output mode values on the host PC offered by the MCU where the SPT performs algorithms to compute the range and velocity of the target. The two output modes are raw ADC sample values and doppler FFT values. We require these two values to further post process in MATLAB to obtain the phase information. Therefore, we built a client-server model using a UDP protocol where MCU is the client that outputs the two measurements to the server (host PC).

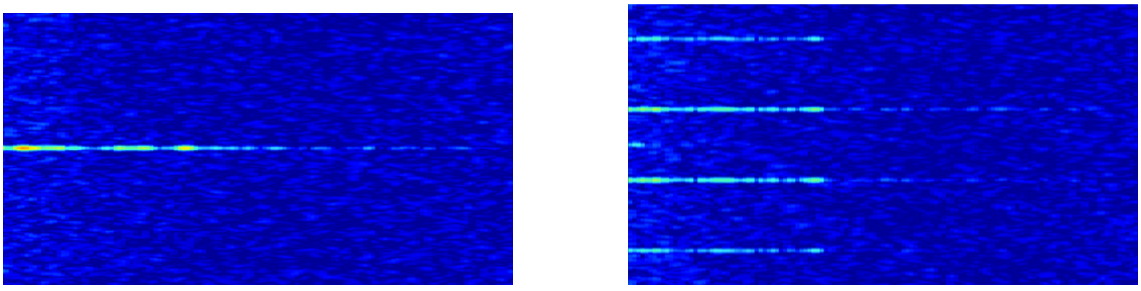
5

Results and Validation

This chapter of the thesis is dedicated to present the most relevant results of our implementation. We follow the flow diagram shown in the figure 3.6 to obtain the end goal which is the phase information corresponding to the communication message. After showing the results, we validate the implementation. In the first two sections, we show the validity of the real-time 2D FFT spectrum with the MATLAB simulated spectrum in a stationary target scenario. The next part focuses on the real-time 2D FFT plots obtained from the application provided by NXP. The following sections are dedicated to showing different plots achieved from collecting different values related to range, raw ADC samples, and doppler FFT with and without phase coding. The final section presents the reconstruction of the message encoded in the LFM chirp and reflects on the validation of reconstructing the message bits from the chirp using MATLAB.

5.1. Real-time 2D FFT spectrum with and without phase coding

In this section, we start by presenting the real-time 2D FFT spectrum that plots range on the X-axis and velocity on the Y-axis as seen in the figure 5.1. The plot was achieved by a 2D FFT visualizer provided by the NXP.



(a) (b)

Figure 5.1: Real-time 2D FFT spectrum: (a)without phase coding, and (b)with phase coding

We do not have more information regarding the application and therefore can only attempt to explain what has been perceived from the plot as much

as we could. This plot is a result of acquiring the UDP packets from the MCU after the SPT performs radar signal processing on the raw ADC samples to obtain range/doppler measurements.

5.2. Validation of 2D FFT spectrum with and without phase shift

In this section, we present the 2D FFT output plot of a stationary target simulated by MATLAB as shown in the figure 5.2 for 256 chirps. This simulation reflects the spectrum of 2D FFT when there is no phase coding in the LFM waveform.

It can be seen here that for doppler frequency of 0KHz, the target is located at 100m corresponding to -60 dB magnitude.

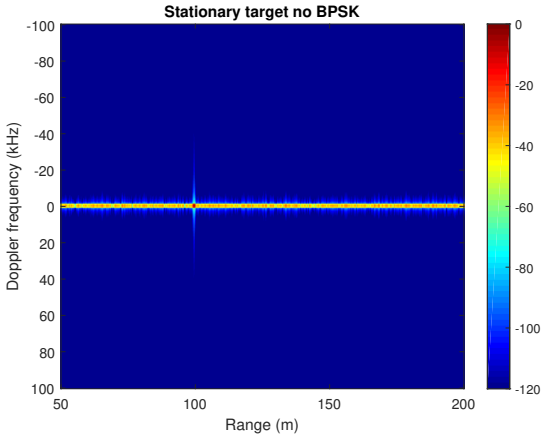


Figure 5.2: MATLAB simulated 2D FFT spectrum without phase coding

This section presents the output plot simulated in MATLAB for stationary targets. In this simulation, phase coding was applied, and the result can be seen in the figure 5.3. We applied a sequence of ”11110000” corresponding to the BPSK signal. The signal representation of the BPSK-LFM waveform can be seen in the figure 2.29.

If we observe the output of both real-time and MATLAB simulated 2D FFT spectrums, it can be seen that the spectrum is a representation of the frequencies corresponding within a chirp (representing the distance along the X-axis) and the frequencies found between the consecutive chirps (representing the velocity along the Y-axis). With the phenomenon of phase modulating the LFM waveform at the transmitter end, we are changing the phase between the chirps. By doing this, we are causing an additional phase shift in addition to the phase shift obtained due to doppler shift, which means that we are adding extra frequencies to the received signal and interfering with what is interpreted as the velocity at the receiver end. To obtain velocity, we look at the frequencies between the consecutive chirps. These additional frequencies are interpreted as added velocities in the 2D spectrum as seen in the figure 5.1b for real-time and figure 5.3 for MATLAB simulated output.

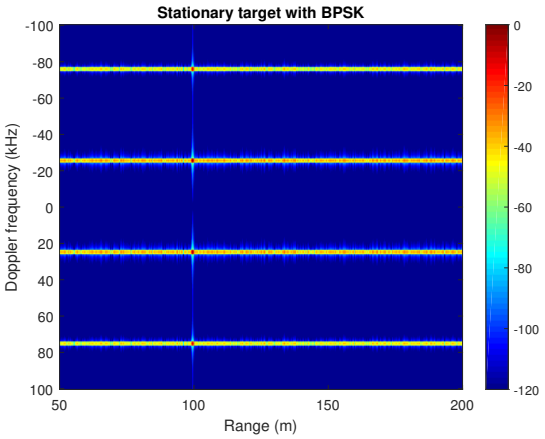


Figure 5.3: MATLAB simulated 2D FFT spectrum with phase coding

5.3. MCU output mode validation

The ADC module on the radar front-end converts all the received analog signals into digitized samples and sends them to the MCU and is then stored in the SRAM on the MCU. To enable target detection and to achieve information on the range and the velocity of the target, the SPT module on the MCU performs 1D and 2D FFT respectively. Therefore, the MCU is able to deliver the computed values of raw ADC samples, range FFT and doppler FFT. The objective of this thesis work which is sending the bits in the LFM waveform and then reconstructing the waveform to achieve the message bits is fulfilled by extracting raw ADC samples and doppler FFT values from the MCU and post process the values on the host PC to observe and validate the phase change corresponding to the communication message bits.

In this section, we present the most relevant plots achieved by enabling the MCU to output raw ADC sample values and doppler FFT values. The initial MCU output mode was configured to doppler FFT and therefore we first implemented the doppler FFT output mode. After we collected the doppler FFT values, we configured the MCU to output the raw ADC samples.

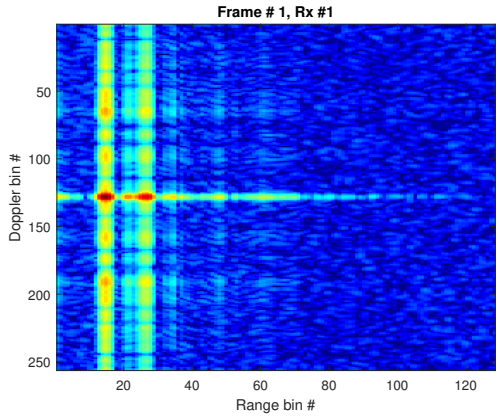
5.3.1. FFT output mode validation

After presenting the 2D FFT spectrum plots in both cases of real-time and simulated for stationary with and without the phase coding, we now illustrate the results and validation of range, doppler, beat signals and phase variations obtained by configuring the MCU to operate in the first output mode namely doppler FFT. We considered four different strings of information bits sent across the radar via MCU from the host PC.

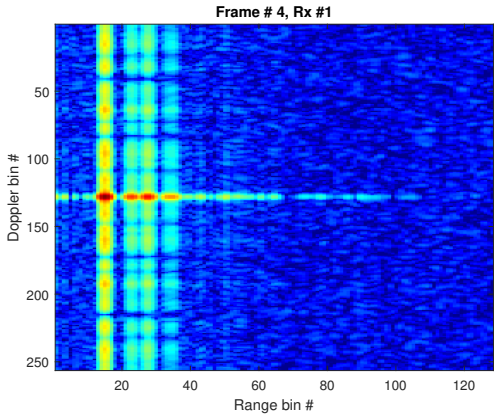
2D FFT(range-doppler) Spectrum

Figure 5.4 shows the range/doppler spectrum with different test strings for the corner-reflector target placed at a 10 m distance from the radar module. It is important to note that the 2D FFT response was plotted only for one frame (Frame1) obtained from one receiver (Rx1).

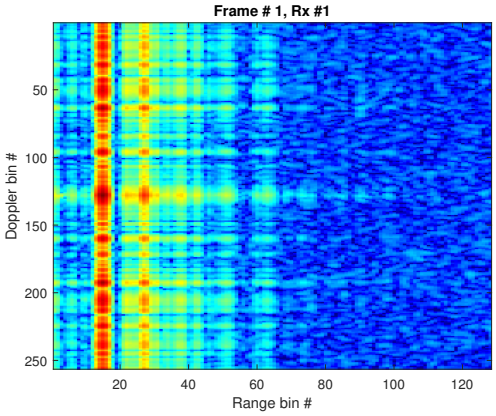
To obtain the range/doppler plot, the SPT in the MCU performs two FFT operations on the acquired raw ADC samples in the SRAM. The first FFT operates on the samples within a chirp and separates the data over distance (into range bins). This was done for each chirp. The second FFT operates across consecutive chirps, it takes the sample from a particular range bin from each chirp. This separates the data over velocity (into doppler bins). This was done for each range bin. The spectrum is a result of processing the doppler FFT values accumulated in a .bin file in MATLAB. The explanation for such an unstable spectrum that was shown in the previous section is also valid for this case.



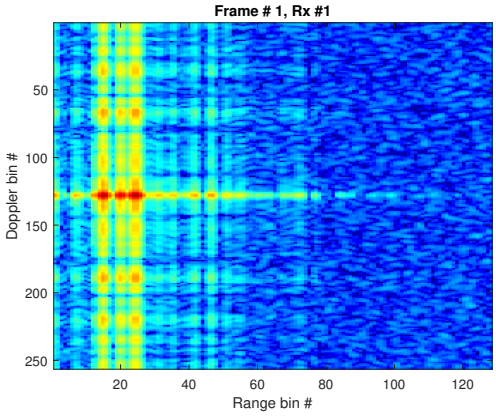
(a) String 1 (helloworld)



(b) String 2 (greenlight)



(c) String 3 (iamrobotand icandrive)



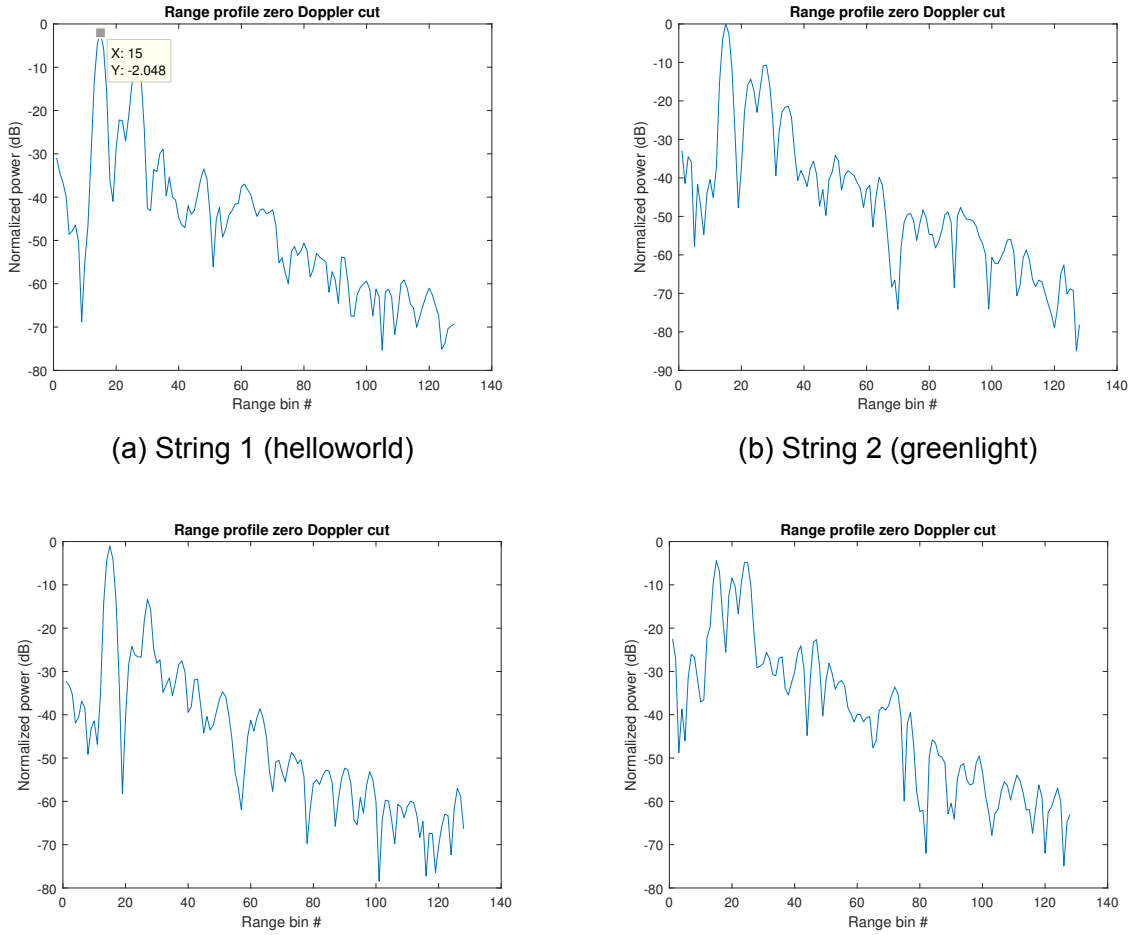
(d) string 4 (STOP!@123GO)

Figure 5.4: 2D FFT spectrum for four different test strings

1D FFT (range) spectrum at zero doppler cut

Once the doppler spectrum has been processed from the doppler FFT values received from the MCU on the host PC, we proceed with processing the range spectrum between the radar and the stationary target. When the target is in the moving state, it is difficult to identify any phase shift between the chirps caused by encoding the bits as there is always a relative velocity between

the radar and the moving target and this introduces doppler effect. Due to this reason, having the velocity as zero will make it possible for observing the phase shifts between the chirps caused by information bits. Therefore, in the



(a) String 1 (helloworld)

(b) String 2 (greenlight)

(c) String 3 (iamrobotandidrive)

(d) string 4 (STOP!@123GO)

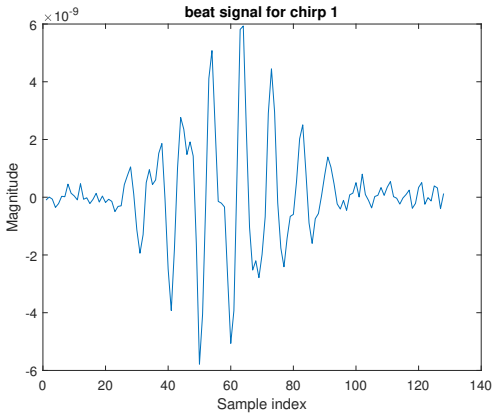
Figure 5.5: Range Plot at zero doppler cut

figure 5.5, we show the range spectrum of the radar and the stationary target at zero doppler cut when the target was placed at a fixed distance of 10 m from the radar. From the (1D) FFT analysis at zero doppler cut, the peak signal values differs for every string. For string 1, the peak signal was located at −4.127 dB. This peak corresponds to the target in the range bin 15 which is equal target distance of 10. From the analytical point of view, it was also observed that the range-bin corresponding to target was found to be equal in all the chirps for every frame and for all the receivers. This is because both target and radar were stationary throughout the entire measurement time.

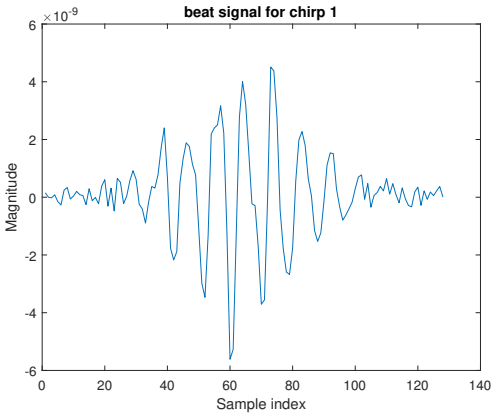
Beat signals

The beat signals are obtained either from acquired raw ADC samples or from the doppler FFT values. To achieve beat signals, we made use of the doppler FFT values. In MATLAB, after processing the range and doppler plots, we

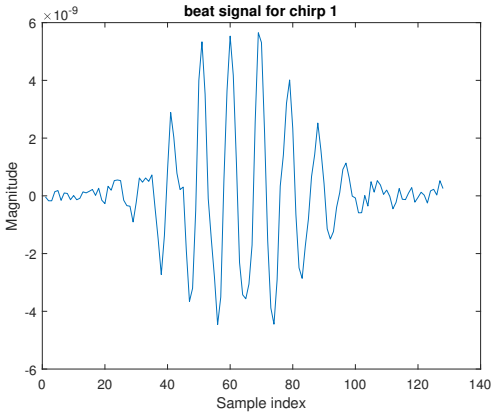
applied inverse FFT over the dimension of the range. We performed the IFFT for all the four different test cases. The resulting beat signal plots are shown in the figure 5.6 corresponding to one chirp. Further, we also show beat signals for all the 256 chirps in the figure B.1 from the appendix B section.



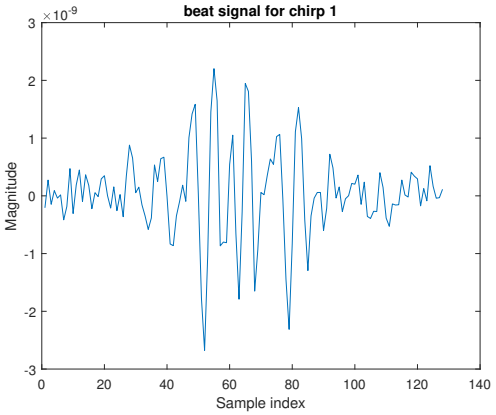
(a) String 1 (helloworld)



(b) String 2 (greenlight)



(c) String 3 (iamrobotandidrivegreenlight)



(d) string 4 (STOP!@123GO)

Figure 5.6: Beat signals for 256 chirps acquired from doppler FFT output values for four different test strings

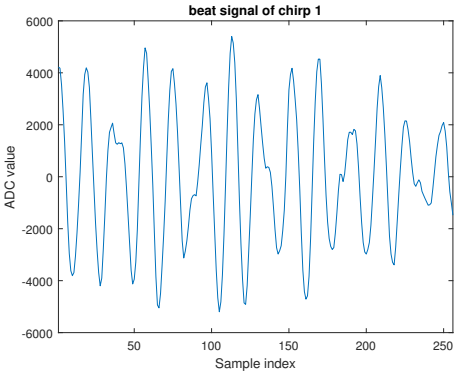
It is interesting to note that the beat signal achieved from the doppler FFT values is preceded by a windowing operation. By default, this is a Dolph-Chebyshev -110dB window with 16-bit real coefficients. Further, due to windowing, it becomes difficult to observe the phase toggles between the chirp as will be discussed in the upcoming sections.

5.3.2. ADC samples output mode validation

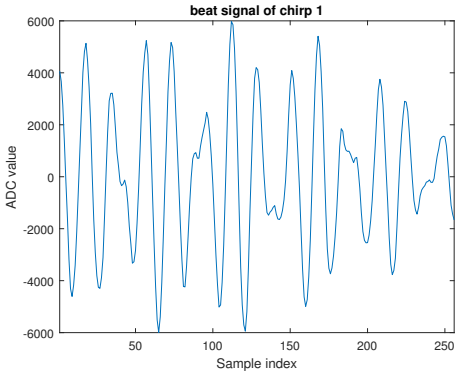
In the previous section, we have illustrated various plots concerning range, doppler, beat signals and phase variations obtained by configuring the MCU to operate in the first output mode namely doppler FFT. In this section, we present the plots by configuring the MCU to output raw ADC samples values.

Beat signals

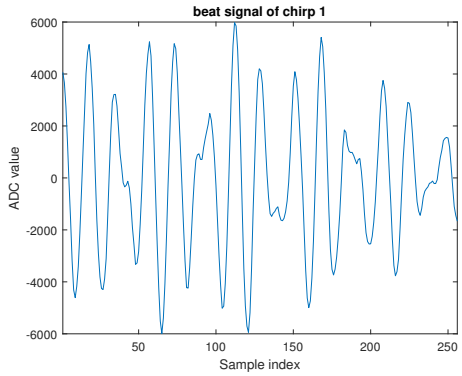
For the output mode transferring raw ADC data, we can get clear beat signals without any windowing, thereby, achieving a clear plot of phase variations between chirp to chirp. The output plot of beat signals is shown in the figure 5.7 that corresponds to one chirp. Further, we also show beat signals for all the 256 chirps in the figure B.3 from the appendix B section.



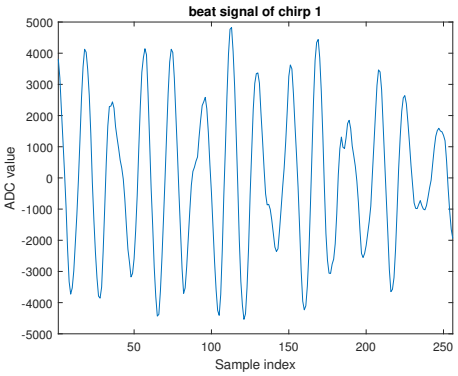
(a) String 1 (helloworld)



(b) String 2 (greenlight)



(c) String 3 (iamrobotandidrive)



(d) string 4 (STOP!@123GO)

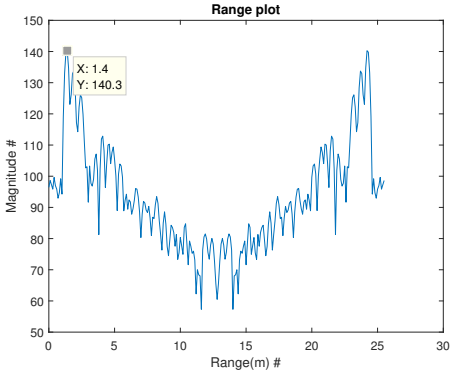
Figure 5.7: Beat signal corresponding to one chirp for four different strings

1D FFT spectrum

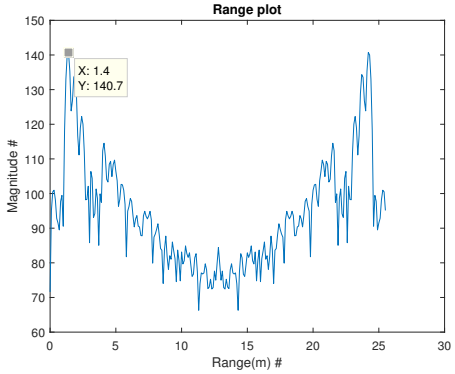
After obtaining the beat signals from the raw ADC samples, we now validate the range of the stationary target by performing first FFT operation over the time samples. In contrast to the FFT output mode, where the SPT in the MCU is performing the FFT operation on the time samples, here, we obtain the range information of the target by post processing the raw ADC samples

in MATLAB. We applied a window function preceded by the range FFT, and, in our case, the windowing is a dolph chebyshev function with -110 DB level. As NXP's default window function is the dolph chebyshev, we considered the same for accuracy.

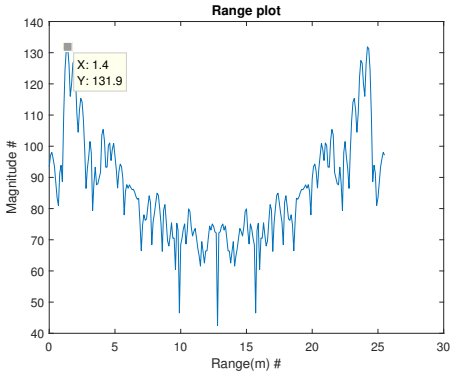
From the table 4.1, we use configuration_1 to program the chirp parameters in this thesis work. Considering the values of chirp duration T and bandwidth of the chirp B, we compute the round trip time delay of the transmitted and received chirps and find out the frequency shift which is also called as beat frequency and is denoted as f_b. To obtain the range R of the target, we use the equation 2.18, where the relation between the beat frequencies f_b and the range R is shown. The range plots for four different strings is shown in the figure 5.8. For all the scenarios, the range of the target from the radar was found to be approximately 1.4m. This is because we considered the target to be stationary for all the test string cases. The validation of the range from beat signals indeed shows us that the range computed from the beat frequencies is same as that of the ground truth range.



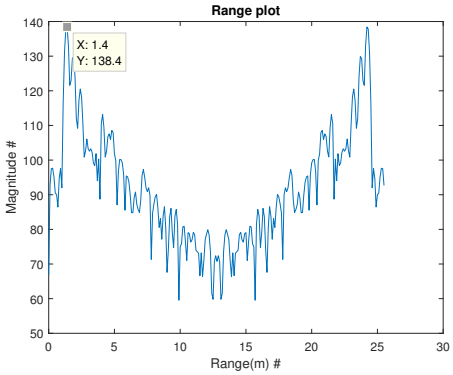
(a) String 1 (helloworld)



(b) String 2 (greenlight)



(c) String 3 (iamrobotandidrive)



(d) string 4 (STOP!@123GO)

Figure 5.8: Range plots from the beat frequencies for different strings

5.4. Message reconstruction

As detailed in the implementation chapter, where we have encoded the message bits in the LFM waveform, we show the reconstruction of the message.

The message on the client was entered in American Standard Code for Information Interchange (ASCII) characters. Each character is represented as 1 byte which extends into 8 bits. We reconstruct the message in MATLAB from the doppler FFT (.bin) file. The data present in the .bin file is in terms of unsigned 8-bit values.

String	h	e	l	l	o	w	o	r	l	d
Reconst. String	h	e	l	l	o	w	o	r	l	d

(a)

String	g	r	e	e	n	l	i	g	h	t
Reconst. String	g	r	e	e	n	l	i	g	h	t

(b)

String	i	am	robot	and	i	can	drive
Reconst. String	i	am	robot	and	i	can	drive

(c)

String	S	T	O	P	!	@	1	2	3	G	O
Reconst. String	S	T	O	P	!	@	1	2	3	G	O

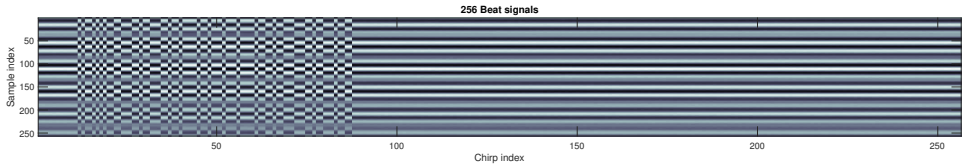
(d)

Table 5.1: Message reconstruction

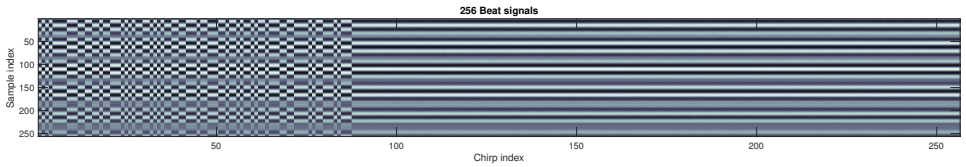
After receiving the uint_8 values, we convert it to double values in MATLAB and reconstruct the message by looking at the phase variations and obtain the messages as shown in the table 5.1. In the process of reconstructing the message, we have to note that the message received from the client is stored in the big-endian format in the buffer of MCU. After the binary phase shift setting is applied on the message bit by bit, the message is stored in the UDP packets in big-endian. To be able to recover the message bit by bit in the correct order, in MATLAB, the entire string is flipped from right to left for every 8 bits in an array of 32 bytes. In addition to reconstructing the message, it is also possible to validate the phase variations of the signal with respect to the message encoded by two ways: the first way is by processing the raw ADC samples, and other is to use the doppler FFT values.

Phase variation plot from raw ADC samples output mode

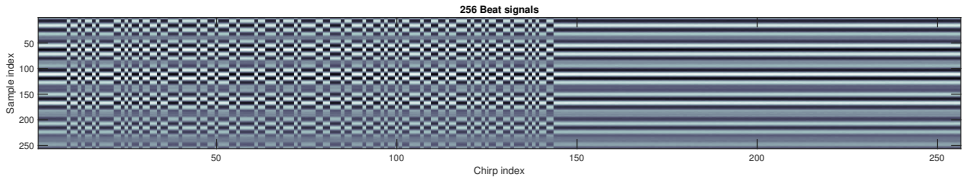
In the figure 5.9 which is the result of phase variations obtained from the beat signals, the phase toggling can clearly observed for 256 chirps and 256 ADC samples. The change in two colors successively corresponds to the bit toggle of 0 to 1 and 1 to 0. For example, let us consider the phase toggling along the 256 chirps at the sample index 113. We obtain the figure 5.10. In principle, we can look at any sample index, but we chose the sample index at which the beat signal has the highest peak from figure 5.7 for all the strings.



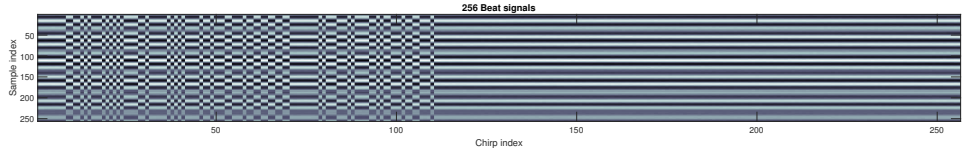
(a) String 1 (helloworld)



(b) String 2 (greenlight)



(c) String 3 (iamrobotandidrive)

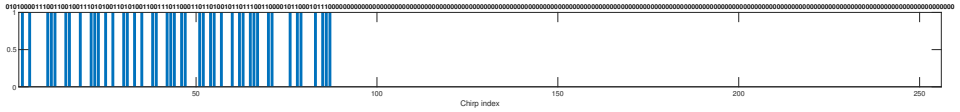


(d) String 4 (STOP!123@GO)

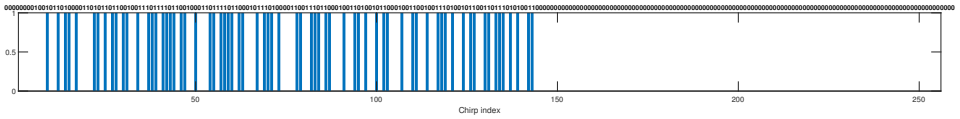
Figure 5.9: Phase variations from beat signals acquired from raw ADC samples

For string 1 which specifies *""helloworld"*, the binary value is extracted as *"10110101001100011011000110111101101110111011110110010011100011000100011"* preceded by 9 bits of header and appended by 0's until the 32bytes is reached as shown in the figure 5.10. Similarly, all the other strings also can be extracted.

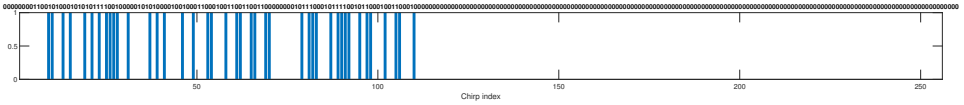
(a) String 1 (helloworld)



(b) String 2 (greenlight)



(c) String 3 (iamrobotandidrive)

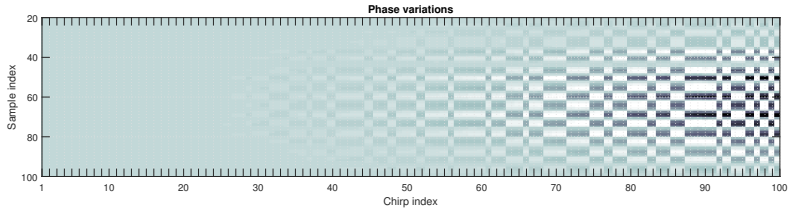


(d) String 4 (STOP!123@GO)

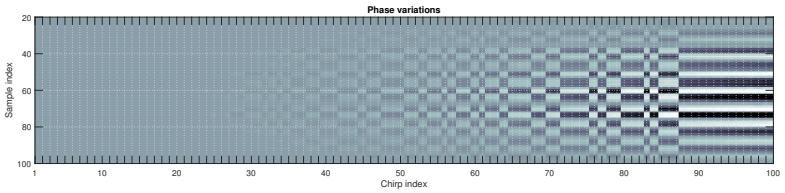
Figure 5.10: Phase variations beat signals and corresponding binary values for four different test strings acquired from raw ADC samples

Phase variation plot from FFT output mode

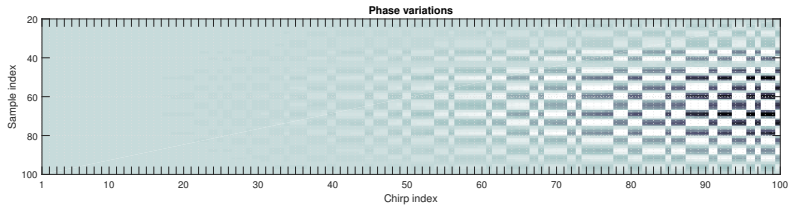
In the figure 5.11 which is the result of phase variations obtained from the FFT output mode, the phase toggling can be observed for 256 chirps and 256 ADC samples. For example, let us consider the phase is toggling along the 256 chirps at the sample index 60. In principle, we can look at any sample index, but we chose the sample index at which the beat signal has the highest peak from figure 5.6 for all the strings. The change in two colors successively corresponds to the bit toggle of 0 to 1 and 1 to 0. It is hard to observe the phase toggling as the doppler FFT values extracted are a result of applying windowing, as the side lobes are suppressed in contrast with the phase variation plot obtained from raw ADC samples. For string 1 which specifies



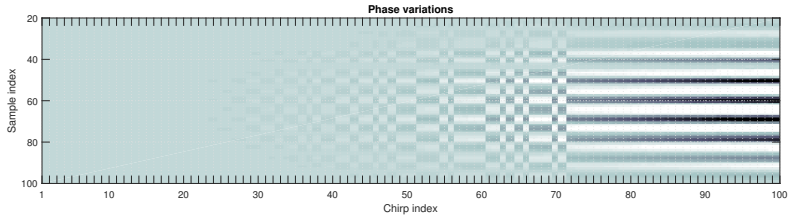
(a) String 1 (helloworld)



(b) String 2 (greenlight)



(c) String 3 (iamrobotandidrive)

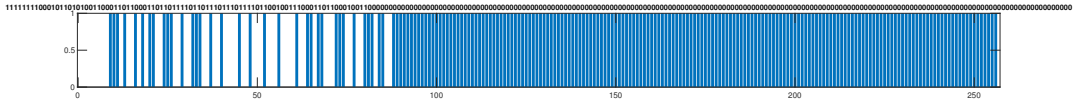


(d) String 4 (STOP!@123GO)

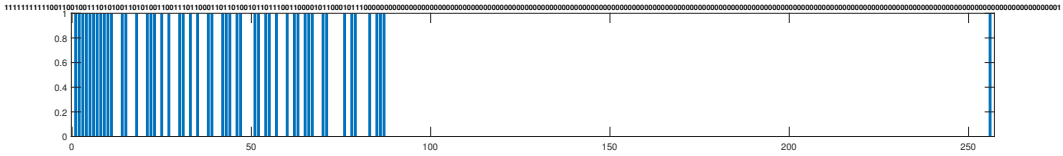
Figure 5.11: Phase variations from beat signals acquired from FFT values

”*”helloworld”*, the binary value is extracted as
”1011010100110001101100011011110110111011101111011001001

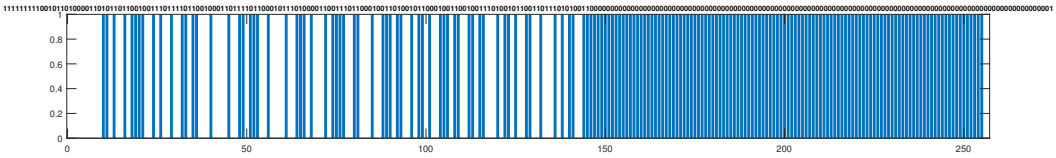
110001100010011" preceded by 9 bits of header and appended by 0's until the 32bytes is reached as shown in the figure 5.12. Similarly, all the other strings also can be extracted. It is interesting to note that the phase variation plot from the beat signal is quite clear than the phase variations from doppler FFT values. The reason for such behavior is windowing. A windowing operation precedes both the range and doppler FFT. By default, this is a Dolph-Chebyshev -110dB window with 16-bit real coefficients. This is not the case for the beat signals obtained from the raw ADC samples where the beat signals are obtained before applying any windowing. Therefore, it is ideal to look at the beat signals from raw ADC samples to correctly verify the phase coding.



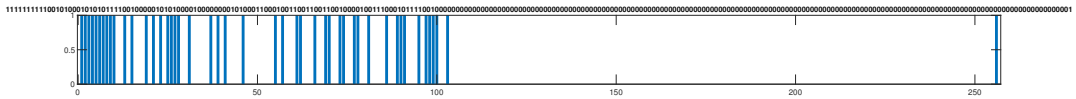
(a) String 1 (helloworld)



(b) String 2 (greenlight)



(c) String 3 (iamrobotandidrive)



(d) String 4 (STOP!@123GO)

Figure 5.12: Phase variations beat signals Phase variations beat signals and corresponding binary values for four different test strings acquired from FFT values

5.5. Conclusion

One of the first results corresponds to the validation of the real time 2D FFT spectrum by implementing LFM-BPSK in MATLAB. We considered four different strings to acknowledge the implementation of LFM-BPSK waveform and present different plots related to its behavior such as its 2D FFT spectrum, the number of receiver channels per frame, range profile sequence with zero doppler cut, range plot and phase variations between chirps at a selected single range bin selected. In addition to this, we show beat frequency plots and also prove the phase variations from the plot.

6

Conclusion and Future work

6.1. Conclusion

The essence of exploiting both sensing and communication in the same waveform transmitted from the radar is the true motive behind conducting this thesis work. We aim to implement one of the widely acknowledged form of modulation techniques based on phase to embed information bits in to the radar waveform using a MCU as the autonomous cars today are increasingly becoming software driven requiring high performance processing speeds. The thesis was carried out on hardware platform provided by NXP, namely the DCC, which is an integration of high performance 32bit S32R274 MCU that provides strong signal processing acceleration and a radar front-end TEF810X which is based on fast-chirp modulation.

In order to achieve a real-time information exchange from radar to enable dual functionality of both sensing and communication in a vehicle, we have performed the following stages of implementation. In the end of the implementation, we successfully were able to send the message in the LFM chirps from the radar in real time. Following this, we also reconstructed the message correctly by post processing both the raw ADC samples and range/doppler FFT values which are a result of the SPT toolbox performing computations on the received signals in the MCU.

In the first stage of the implementation, we built a Graphical User Interface (GUI) interface between the host PC and MCU based on TCP/IP communication protocol to send message bits to the MCU by setting up a listening echo server on the DCC board. Moreover we implemented a client connection on the PC which makes communication possible between MCU and any host making it bidirectional in nature. Following this, to receive the UDP packets that contain range, doppler and raw ADC samples in order to further conduct post-processing in MATLAB, a UDP listening server was built on the host PC and client on MCU. The MCU which saves the processed range, doppler and ADC samples, outputs them into the UDP packets and stores them into the

SRAM of the MCU.

From the system implemention, it can see that there was a challenge with respect to sending the message bits from MCU to the radar front end as the data is stored in the local memory of core_1 and all the radar processing happens in core_0. Therefore, we implemented a technique to establish inter core communication between core_0 and core_1 using hardware semaphores. Once the communication is established, we send few test strings of message from the host PC to the MCU and begin the process of sending them to the radar via timing engine registers bit by bit. It was important to consider one test case scenario to conduct the experiments, being, the stationary target. We did not perform the BPSK-LFM implementation on the moving target due to the fact that in order to the doppler information is obtained by looking at the phase shift which is a result of doppler phenomenon between consecutive chirps. When we are encoding the chirp with message bits, we are causing addition phase shifts. In the current implementation, where we are observing the phase shifts on the echo received signals on the same radar board as that of the transmitter, the receiver interprets the phase shift as the doppler information. This makes it impossible to reconstruct phase shifts corresponding to message bits. Therefore, we keep the radar at a fixed distance from the target to obtain the phase information. When using this implementation for radar-to-radar communication, the transmitting radar would compensate for the phase shifts in its receiver before processing the data, thus obtaining a clean spectrum usable for range/velocity measurements. The receiving radar will not compute the full 2D spectrum from the signal, but instead analyze the phase jumps between consecutive chirps to reconstruct the bit stream.

While it is possible to observe the phase toggling from both the raw ADC sample values and the doppler FFT values outputted by the MCU, it was deduced that, indeed, the raw ADC samples provide clear beat signals without any windowing unlike the beat signals obtained from post processing the doppler FFT values. However, we also included and analyzed the beat signals and phase information obtained by the doppler FFT values as the end application offered by the DCC board is a doppler spectrum that provides the range information of the target contributing to target detection. In other words, the SPT in the MCU is able to perform the acquisition of the ADC samples and compute the range and velocity of the target in parallel. We take this advantage and prove that indeed it is possible to reconstruct the message with the doppler FFT values as well.

Finally, coming back to our research question of achieving joint sensing and communication functionality in a radar via a MCU, to facilitate real time processing, we successfully accomplished the following goals. Firstly, the software architecture for the interface between the NXP's S32R274 MCU, TEF810X transceiver and the host PC was proposed. Secondly, we implemented the binary phase coded linear frequency modulated waveform (BPSK-

LFM) on the MCU to facilitate real time information exchange and lastly, the correctness of phase coding was validated by establishing a test scenario with a stationary target.

6.2. Future scope

Lastly, we conclude our work with some final future work suggestions that involves message reconstruction and the number of bits encoded per chirp. The immediate followup field of investigation to this thesis would be to work on building a robust decoder at the receiver end that will reconstruct the message without any inconsistency in the message bits. It is also important to note that the message sent from the client to the MCU is a continuous stream of ASCII characters without a space. Conventionally, it is required to send messages with spaces between the words. In the limited span of this thesis work, we could not identify where the issue lies. However, this could be valuable to start working with instantly. Furthermore, it is interesting to notice from research conducted that it is possible to encode one chirp with more than one bit of information by switching the chirp not only at 0 or 180 degrees but also at other angles. This technique is referred to as reduced binary phase shift keying.

A

Hardware setup

The Dual Credit Card (DCC) radar sensor board is an integration of TEF810X1 77GHz RADAR transceiver chipset and the S32R274 automotive MCU that can be used for radar software development, performance/system evaluation and application demonstration.

A.1. TEF810X Transceiver

The figure A.1 depicts the front and rear side of the TEF810x transceiver. The transceiver board is connected to the S32R274 MCU via the stacking connectors that is also located on the rear side of the MCU.

A.2. S32R274 MCU with debugger

The S32R274 Microcontroller (MCU) board component of the DCC includes a high-speed connector for Nexus debug hardware which is Nexus Aurora debug compatible debug hardware and can be connected directly to this connector. For cases where the available debug hardware supports only JTAG connection to the microcontroller, a debug adapter board (X-JTAG-CON) is needed to expand this connector to provide the standard 14-pin JTAG connector, shown in figure A.2. The JTAG connector is used by the P&E Micro Multilink debug hardware Interface hardware to provide the debug interface to S32R274 and support reprogramming of the embedded flash memory with a new firmware binary.

A.3. Communication GUI using the Transmission Control Protocol/Internet Protocol (TCP/IP) ethernet protocol

The snippet of the communication application on host PC (client) from where the message is sent to the MCU (server) is shown in the figure A.3

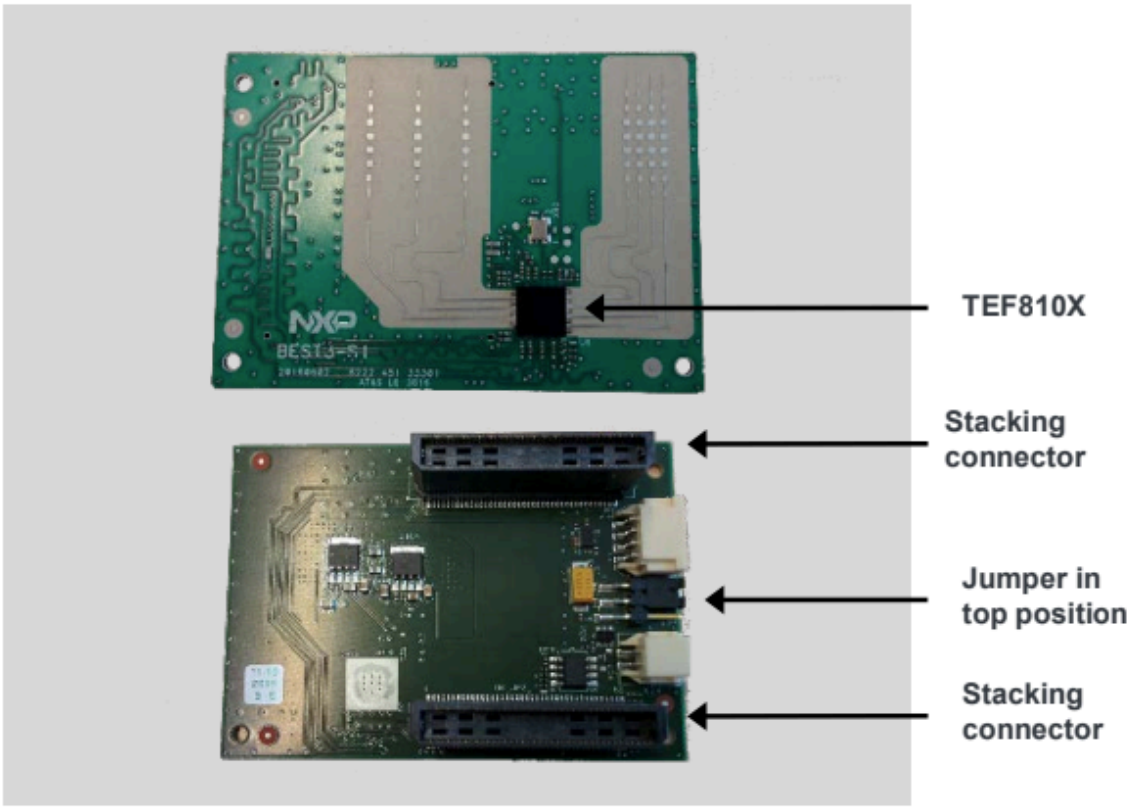
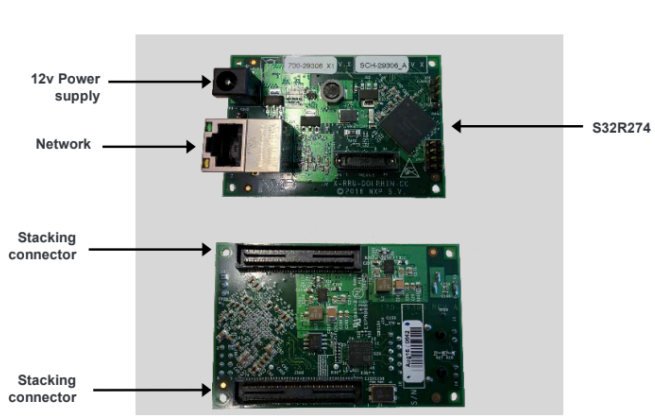


Figure A.1: NXP's TEF810X transceiver front and rear side



(a) NXP's S32R274 MCU front and rear side[**?**]

(b) NXP's S32R274 with JTAG connector[**?**]

Figure A.2: NXP's automotive S32R274 MCU with Debug Adapter card Attached

A.4. MCU output mode packet receiver GUI using UDP protocol

The snippet of the receiver application on host PC (server) on which the processed raw ADC samples, range/doppler FFT values are sent from the MCU

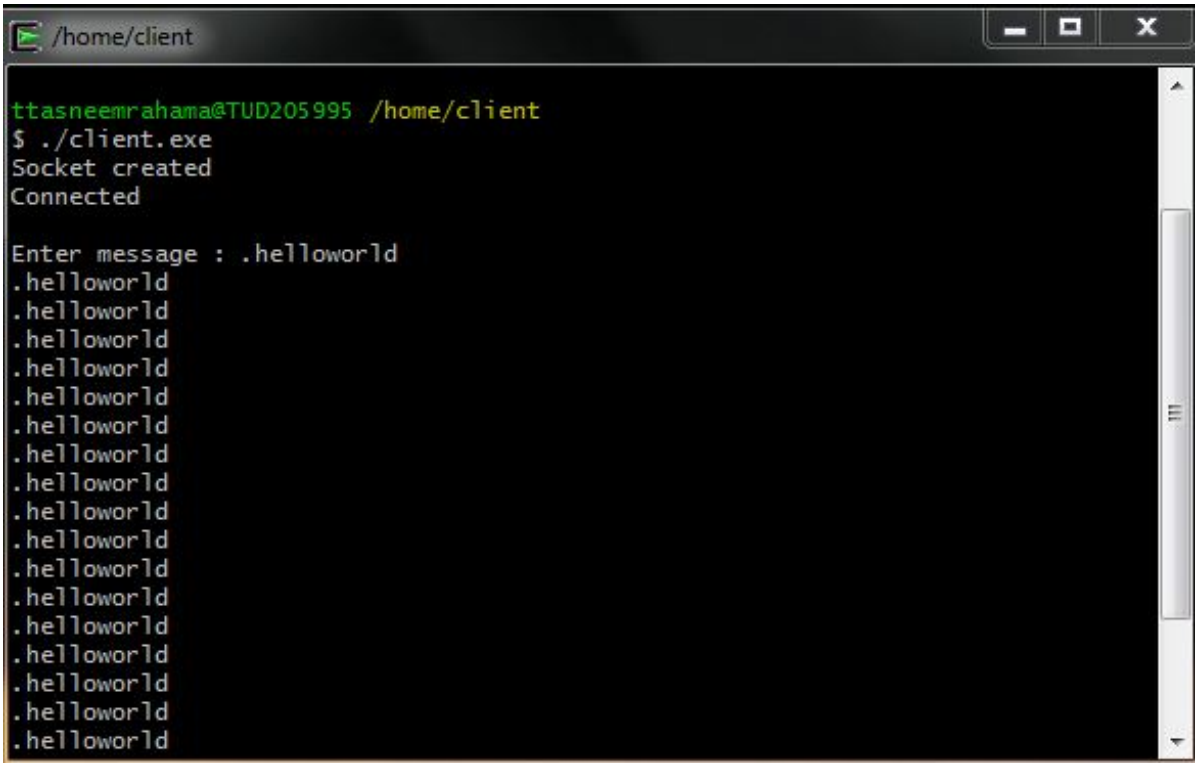


Figure A.3: TCP/IP client-server GUI on host PC to send messages to the MCU

(server) is shown in the figure A.4

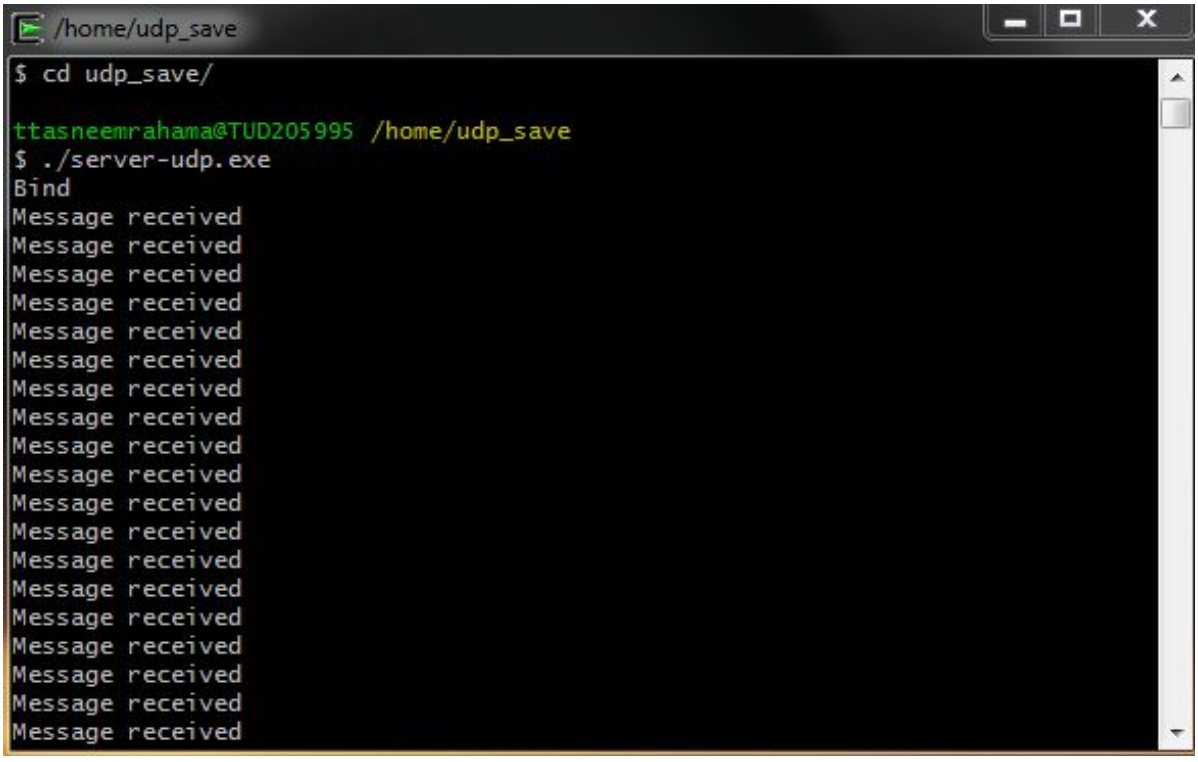


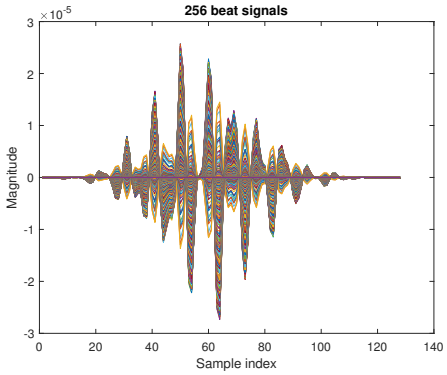
Figure A.4: UDP client-server GUI to receive packets from MCU on the host PC

B

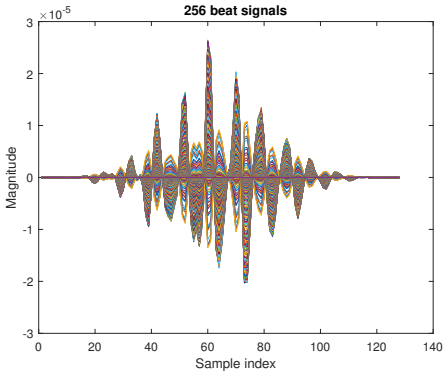
Simulations

B.1. 256 beat signals from FFT output mode

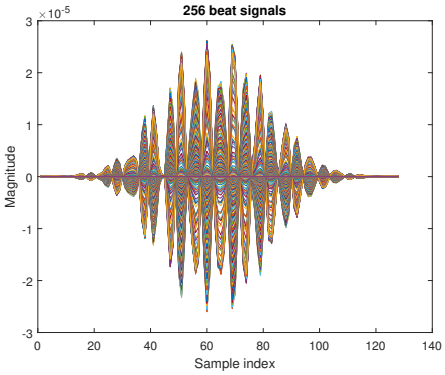
The figure B.1 plots all the 256 beat signals corresponding to 256 chirps for four test strings. These beat signals are obtained from processing the FFT values in MATLAB that is sent from the MCU to the host PC where the post processing is occurring.



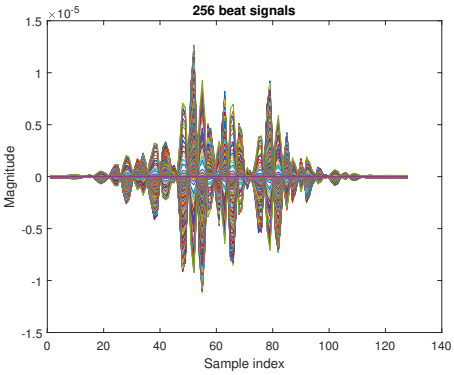
(a) String 1 (helloworld)



(b) String 2 (greenlight)



(c) String 3 (iamrobotandidrive)



(d) string 4 (STOP!@123GO)

Figure B.1: Beat signals for 256 chirps obtained from FFT values for four different strings

B.2. Beat signals in a 3D view from FFT Output mode

This section illustrates the beat signals in 3-dimensional axis obtained from the doppler Fast Fourier Transform (FFT) values. In MATLAB, after processing the range and doppler plots, we applied inverse FFT over the dimension of the range. We performed the IFFT for all the four different test cases. The resulting beat signal plots are shown in the figure B.2. The plots shows number of samples and corresponding number of chirps. This plot gives a clear representation of beat signals with both samples and chirps index.

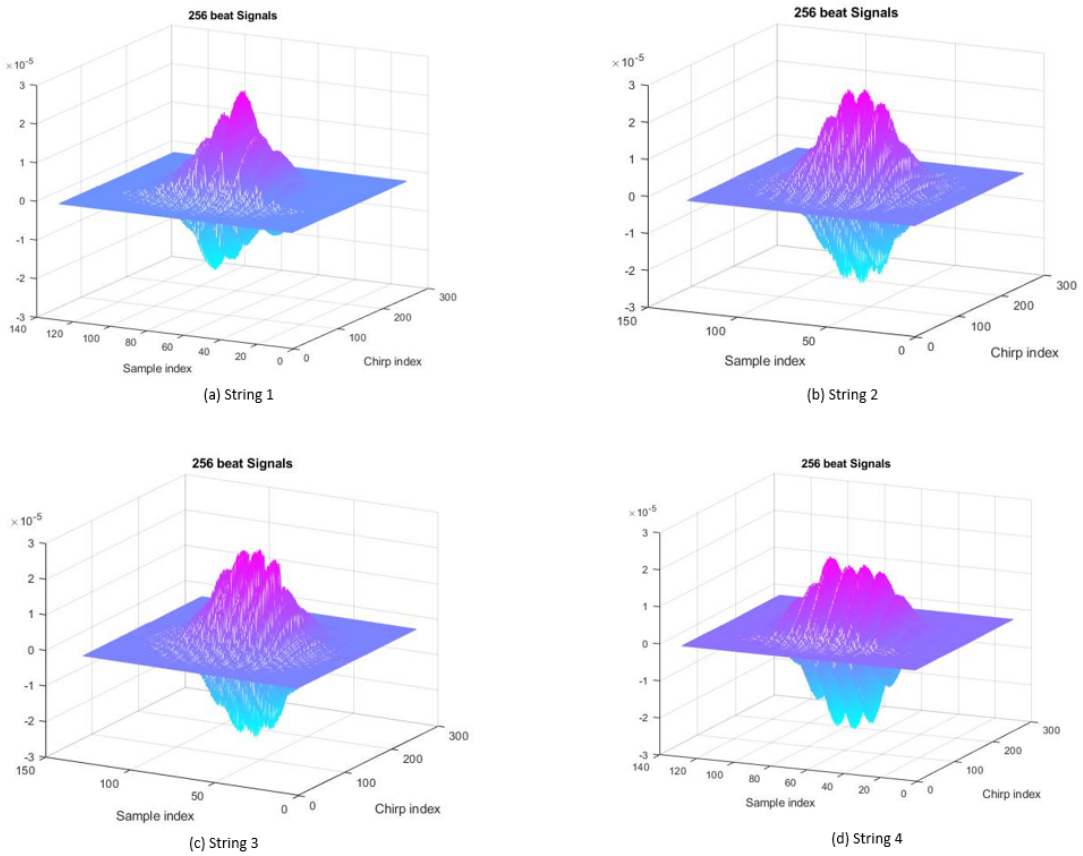
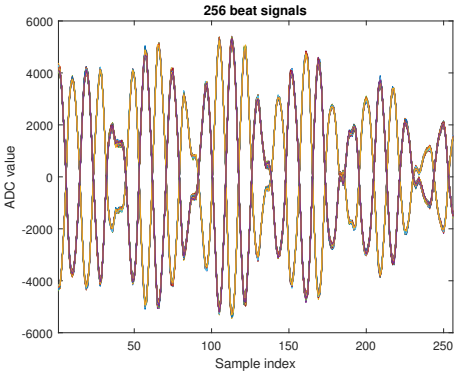


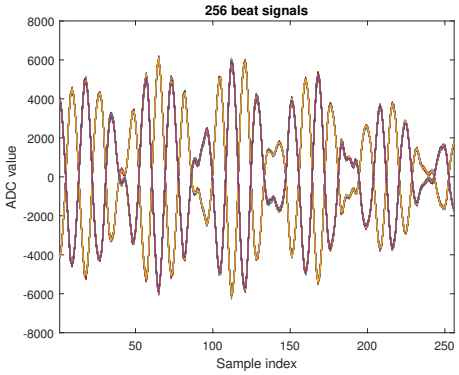
Figure B.2: Beat Signals in a 3D view

B.3. 256 beat signals from ADC output mode

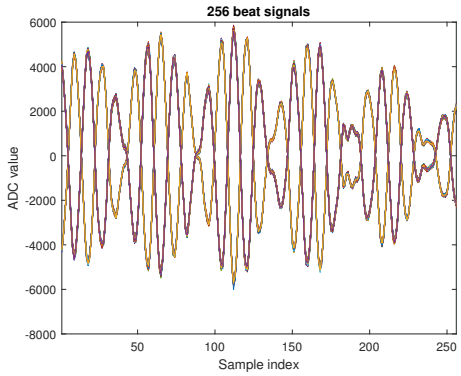
The figure B.3 plots all the 256 beat signals corresponding to 256 chirps for four test strings. These beat signals are obtained from processing the raw ADC sample values in MATLAB that is sent from the MCU to the host PC where the post processing is occurring.



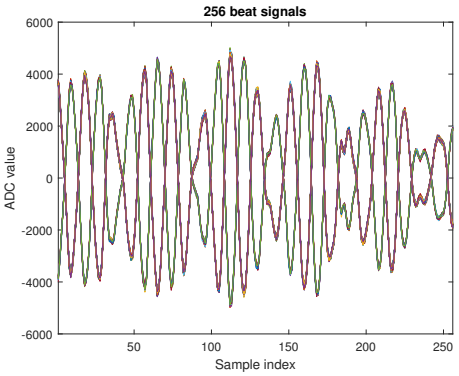
(a) String 1 (helloworld)



(b) String 2 (iamrobotandidrive)



(c) String 3 (greenlight)



(d) string 4 (STOP!@123GO)

Figure B.3: Beat signals for 256 chirps obtained from raw ADC sample values for four different strings

C

Matlab code

These MATLAB codes correspond to the post processing of the UDP packet data received from the MCU that contain the ADC samples and FFT(range/doppler) values to achieve the results as shown in the results chapter. The code was written in collaboration with ir.Pascal Aubry in the MS3 Dept. of TU Delft.

C.1. BPSK-LFM signal representation

```
t = linspace(0,.5,1001);

Y = chirp(t);
figure;plot(Y)

Z = repmat(Y,1,6);

code = [1 1 -1 1 -1 -1];

ZZ = reshape(Z,1001,6);
ccode = repmat(code,1001,1);


ZZZ = ZZ.*ccode;

ZZZZ = reshape(ZZZ,1,6006);
cccode = reshape(ccode,1,6006);

tt = linspace(0,1,6006);
figure;
subplot(3,1,1);plot(tt,cccode);ylim([-1.2 1.2]);title('Binary Data')
subplot(3,1,2);plot(tt,Z);title('Unmodulated LFM Chirps Signals')
subplot(3,1,3);plot(tt,ZZZZ);title('BPSK Modulated LFM')
```

C.2. 2D FFT spectrum with and without phase coding

```
clear; clc; %close all
```

```
2
3  c = 3e8;
4  fc = 100e6;
5  B = 50e6;
6  T = 5e-6;
7  K = B/T;
8
9  fs = .5e9;
10 dt = 1/fs;
11 t = 0:dt:T;
12
13 Nsamp = length(t);
14 Nsweeps = 256;
15
16 SNR = 10; % 10dB
17
18 ThetaLFM = ((K*t-B)/2+fc).*t;
19 Txchirp = (exp(1i*2*pi*ThetaLFM));
20
21 f = linspace(0,fs,length(Txchirp));
22 figure;plot(f/1e6,20*log10(abs(fft(Txchirp))))
23
24 tgtR = 100;
25 tgtDlay = 2*tgtR/c;
26
27 Rxchirp = ifft(fft(Txchirp.*exp(-2*pi*1i*(f+rand
28 (1,Nsamp)*1e5).*(tgtDlay+rand(1,Nsamp)*1e-11)))).*rand(1,Nsamp)/SNR;
29
30 BS = Txchirp.*conj(Rxchirp);
31
32 maxRange = c*T/2;
33 Range = linspace(0,maxRange,Nsamp);
34
35 PRF = 1/T;
36 fd = linspace(-PRF/2,PRF/2,Nsweeps);
37
38 figure;plot(Range,20*log10(abs(fft(BS.*hamming(Nsamp)'))))
39
40
41 %% BPSK
42 Pattern = [ones(1,4) -1*ones(1,4)]; % equivalent to 11111000
43 Code = repmat(Pattern,1,Nsweeps/length(Pattern)); % repeat Pattern ...
       over number of sweeps
44
45
46 BURSTcoded = Code'*BS;
47 BURST = ones(256,1)*BS;
48
49 window = hamming(Nsweeps)*hamming(Nsamp)';
50
51 RngDopp = fft2(BURST.*window);
52 RngDoppcoded = fft2(BURSTcoded.*window);
53
54
55
56
```

```
figure;
imagesc(Range,fd/1e3,fftshift(20*log10(abs(RngDopp)/max(abs(RngDopp(:))))
,1));
caxis([-120 0]);xlim([50 200]);colormap jet;title('Stationary ...
    target no BPSK')
xlabel('Range (m)');ylabel('Doppler frequency (kHz)')
figure;
imagesc(Range,fd/1e3,fftshift(20*log10(abs(RngDoppcoded)/
max(abs(RngDoppcoded(:)))),1));
caxis([-120 0]);xlim([50 200]);colormap jet;title('Stationary ...
    target with BPSK')
xlabel('Range (m)');ylabel('Doppler frequency (kHz)')


% [a, b]=max((abs(fft(BS))));
% figure;plot(rad2deg(angle(RngProf(:,b))));
```

C.3. Extracting raw ADC samples from UDP packets and reconstructing the message

```
fid = fopen('O:\thesis_code_final\matlab_final\adc_measurments\
adc_phase_stationary_diamrobot.bin');

NSamp = 256; % Samples per chirp
NChirps = 256; % Chirps per frame

k=1;    % k = UDP packet number
while ¬feof(fid)
    DataTypeID(k,:) = (fread(fid, 2, 'uint8'));
    FrameNumber(k) = fread(fid, 1, 'uint16','b');
    ChirpSeqNum(k) = fread(fid, 1, 'uint16','b');
    PacketNum(k) = fread(fid, 1, 'uint8','b');
    TotPacketNum(k) = fread(fid, 1, 'uint8','b');
    buffer(:,k) = fread(fid,4*183,'int16','b');
    k = k+1;
%     if k==512
%         break
%     end
end
fclose(fid);

% find start of first and last full frames
idx1stFrame = find(ChirpSeqNum==0,1,'first');
idxlastFrame = find(ChirpSeqNum==0,1,'last')-2;
Data = buffer(:,idx1stFrame:idxlastFrame);
[M,N] = size(Data);

% re-arange data per Rx Channel
% stitch 2 packages together
Data = reshape(Data,M*2,N/2);

[M,N]=size(Data);
n = 1;
for m = 1:32:M
    A(n:n+7,:) = Data(m:m+7,:);
    B(n:n+7,:) = Data(m+8:m+15,:);
    C(n:n+7,:) = Data(m+16:m+23,:);
    D(n:n+7,:) = Data(m+24:m+31,:);
    n = n+8;
end

A = A(1:NSamp,:);
B = B(1:NSamp,:);
C = C(1:NSamp,:);
D = D(1:NSamp,:);

A = reshape(A,NSamp,NChirps,N/NChirps);
B = reshape(B,NSamp,NChirps,N/NChirps);
C = reshape(C,NSamp,NChirps,N/NChirps);
D = reshape(D,NSamp,NChirps,N/NChirps);

```

```
52 figure;imagesc(A(:,:,1));
53 colormap bone;xlabel('Chirp index');ylabel('Sample index')
54 title('256 Beat signals')
55
56 figure;plot(A(:,:,1))
57 xlim([1 256]);title('256 beat signals');
58 xlabel('Sample index');ylabel('ADC value')
59
60 AA = A(113,:,1);
61 AAA = AA;
62 AAA(AA>0) = 0;
63 AAA(AA<0) = 1;
64
65 RxBinStr =  num2str(AAA') ';
66
67 figure;plot(AA)
68 xlabel('Chirp index');ylabel('ADC value');
69 title('Beat signal ADC value for sample #113')
70
71
72 figure;bar(AAA)
73 xlim([1 256]);title(RxBinStr,'fontsize',7);
74 xlabel('Chirp index')
75
76 RxSTR_char = [];
77 jj = 1;
78 startbit = 9;
79 for ii = startbit:8:256-startbit
80     tmp = RxBinStr(ii:ii+7);
81     tmp2 = fliplr(tmp);
82     RxSTR_char(jj) = bin2dec(tmp2);
83     jj=jj+1;
84 end
85 char(RxSTR_char)
```

C.4. Extracting FFT values from UDP packets and reconstructing the message

```
load('O:\thesis_code_final\matlab_final
\FIGURES_helloworld\Data4_helloworld_v1.mat')

[NDop,NRange,NRx,NFrame]=size(Data4);
figure;
for n = 1:NFrame
    imagesc(fftshift(20*log10(abs(squeeze(Data4
    (:,:,1,n))/max(abs(Data4(:))))),1));
    caxis([-80 0]);colormap jet;
    title(['Frame # ' num2str(n) ', Rx #1'])
    xlabel('Range bin ')
    ylabel('Doppler bin ')
%     M = getframe;
%      writeVideo(v,M);
    pause(.1);
end
colormap cool
% close(v);

 figure;plot(fftshift(20*log10(abs(squeeze(Data4
 (1,:,1,1))/max(abs(Data4(:))))),1));
xlabel('Range bin')
ylabel('Normalized power (dB)')
title('Range profile zero Doppler cut')

%% PHASE CODING CHECK
RP = ifft(Data4); % perform IFFT to get range profiles sequence
figure;imagesc((20*log10(abs(squeeze(RP(:,:,1))/max(abs(RP(:)))))));
xlabel('Range bin ')
ylabel('Doppler bin ')
title('256 range profiles')
colormap jet

% select strong target range bin
tgtBin = 15;    % manually

[maxv, maxi] = max(max(abs(squeeze(RP(:,:,1))))); % programmatically


Rx_phseq = angle(squeeze(RP(:,maxi,1,1)));
figure; stem(rad2deg(Rx_phseq),'o');
xlabel('Chirp index')
ylabel('Phase (deg)')
title('Phase of range bin 15')

Rx_binseq = uint8((cos(Rx_phseq-min(Rx_phseq))+1)/2);
figure;bar(abs(double(Rx_binseq)-1));
RxBinStr =  num2str(Rx_binseq)';                % V1
%RxBinStr =  num2str(abs(double(Rx_binseq)-1))'; % V2

title(RxBinStr,'fontsize',7)
```

```
52
53
54 RxSTR_char = [];
55 jj = 1;
56 startbit = 9;
57 for ii = startbit:8:256-startbit
58     tmp = RxBinStr(ii:ii+7);
59     tmp2 = fliplr(tmp);
60     RxSTR_char(jj) = bin2dec(tmp2);
61     jj=jj+1;
62 end
63 char(RxSTR_char)
64
65 %% Beat Signals
66
67 BS = ifft(RP,[],2);
68
69 figure;plot(squeeze(real(BS(:,:,1)))')
70 title('256 beat signals');
71 xlabel('Sample index')
72 ylabel('Magnitude')
73 % figure;imagesc(squeeze(real(BS(:,:,1)))')
74 %
75 % figure;plot(20*log10(abs(RP(128,:,1)))')
76
77 figure;mesh(squeeze(real(BS(:,:,1)))')
78 shading interp
79 colormap cool
80 xlabel('Chirp index')
81 ylabel('Sample index')
82 % zlabel('Magnitude')
83 title('256 beat Signals')
84
85 figure; ...
      imagesc(log10(squeeze(real(BS(:,:,1))-min(min((real(BS(:,:,1))))))')')
86 colormap bone
87 caxis([-5 -4.5])
88 xlabel('Chirp index')
89 ylabel('Sample index')
90 % zlabel('Magnitude')
91 title('Phase variations')
```

Bibliography

[1] Peter Alley. Introductory microcontroller programming. Master's thesis, WORCESTER POLYTECHNIC INSTITUTE, 2011.

[2] G. Brooker. Understanding millimetre wave FMCW radars. *1st International Conference on Sensing Technology*, page pp. 152–157, Nov 2005.

[3] Prof. R.V. Rajakumar, Prof. Saswat Chakrabarti. *Lectures on Module:1 Introduction to Digital Communications and Information Theory*. Nptel, IIT Kharagpur.

[4] J.Choi, N.Gonzalez-Prelcic, R.Daniels, C.R.Bhat, and R.W.HeathJr. Millimeter-wave vehicular communication to support massive automotive sensing. *IEEE Communications Magazine*, vol. 54(no. 12):pp. 160–167, December 2016.

[5] S. Quan, W. Qian, J. Guq, and V. Zhang. Radar-communication integration: An overview. *In The 7th IEEE/International Conference on Advanced Infocomm Technology*, pages 93–103, 2014.

[6] Texas Instruments. URL `https://e2e.ti.com/blogs_/b/behind_the_wheel/archive/2017/10/25`.

[7] N. Levanon. Multifrequency complementary phase-coded radar signal. *IEEE Proceedings-Radar, Sonar and Navigation*, pages 147(6):276–2843, 2000.

[8] J. Li and P. Stoica. MIMO radar with collocated antennas. *IEEE Signal Processing Magazine*, 24:106 – 114, September 2007.

[9] Mast, 2018. URL `http://www.masttechnologies.com/automotive/`.

[10] *TEF810X Fully-Integrated 77 GHz Radar Transceiver*. NXP Semiconductors.

[11] *S32R274 DCC Software Guide: Technical Guide to S32R274 and TEF810X Dual Credit Card Sensor Software v1.3*. NXP Semiconductors Semiconductors, 2017.

[12] S. Rao. *Introduction to mmWave sensing: FMCW radars*. Texas Instruments (TI) mmWave Training Series, 2017.

[13] T. S. Rappaport. *Wireless communications: principles and practice*, volume 2. Prentice Hall PTR, New Jersey, 1996.

[14] P.Papadimitratos, A.LaFortelle, K.Evenssen, R.Brignolo and S.Cosenza. Vehicular communication systems:enabling technologies, applications, and future outlook on intelligent transportation. *IEEE Communications Magazine*, vol. 47(no. 11):pp. 84–95, 2009.

[15] Ph.D. Rodrigo Romero. *EE 3376 Microprocessor Systems I*. University of Texas at El Paso.

[16] NXP Semiconductors. *Guide to Dolphin API*, 2017 .

[17] Mong Sim. *A Practical Approach to Hardware Semaphores For MCP56xx and MPC57xx Multi-core Qorivva Devices*. NXP Semiconductors Semiconductors, 2014.

[18] Christian Sturm and Werner Wiesbeck. Waveform design and signal processing aspects for fusion of wireless communications and radar sensing. *Proceedings of the IEEE*, 99(7), July July 2011.

[19] Y. L. Sit, C. Sturm, and T. Zwick. Doppler estimation in an OFDM joint radar and communication system. *German Microwave Conference*, pages 1–4, 2011.

[20] WHO. Global status report on road safety. Technical report, 2015.

[21] Z. Zhang, M. Nowak, M. Wicks and Z. Wu. Bio-inspired RF steganography via linear chirp radar signals. *IEEE Communications Magzine*, pages 2–6, June 2016.

[22] V. Winkler. Range doppler detection for automotive FMCW radars. *European Microwave Conference*, page pp. 1445–1448, Oct 2007.