FMCW radar based communication for automotive applications

Master Thesis Report

Simone Orru’

Student number: 4748212
For the achievement of the Master of Science
in Embedded Systems

FMCW radar based communication for automotive applications

Master Thesis Report

by

Simone Orru'

To obtain the degree of Master of Science in Embedded Systems from the faculty of EEMCS at the Delft University of Technology.

To be defended publicly on Thursday July 16, 2020 at 10:00 AM.

Student number:	4748212	
Thesis number:	Q&CE-CE-MS-2020-01	
Project duration:	March 1, 2018 – July 16, 2020	
Thesis committee:	Dr. S.D. Cotofana,	TU Delft, Associate Professor
	Dr. ir. A. J. van Genderen,	TU Delft
	Dr. F. Uysal,	TU Delft

TU Delft

Abstract

In the field of automotive technology, the last decade has seen a rise in projects focusing on self driving vehicles. With this renewed interest from the industry and academic world, new solutions to improve the reliability and safety of these vehicles are required.
In this thesis, a new method for information sharing between different vehicles is discussed. This method makes use of radar devices, already in use in the automotive sector for obstacles detection. More specifically, a communication channel will be created between two frequency modulated continuous wave (FMCW) radar devices, using the same waveform utilized for radar sensing.
This work will focus on the obstacles to overcome in order to achieve correct data transmission between two radar devices. Different solutions will be evaluated, and a working system, achieving data transmission, will be implemented using radar evaluation boards provided by NXP semiconductors.

Acknowledgements

Developing a new topic in uncharted territory is always challenging, luckily I could count on the help provided by my thesis committee members. In particular I would like to thank Faruk Uysal that was always looking after me in a propositive and engaged way. I would also like to give a special thanks to Arjan van Genderen whose feedback helped me immensely during the development of this thesis.
However my largest thanks goes to my family. Thanks mom and dad for giving me the chance of following my dreams, thanks for all the sacrifices you did over the years to offer me all the possibilities you did not have.

Simone Orru'
Delft, July 2020

Contents

1 Introduction

In this chapter an introduction to the context in which this thesis is developed as well as the main issue this work tackles are presented. In addition a short outline of this document is proposed, to help the reader orient himself in the text.

1.1. Context

The history of self driving cars dates much further back than the last decade. Already in the nineteen-eighties autonomous vehicles were object of scientific research and some proof of concepts such as the VaMoRs self driving van were developed [7]. Thanks to the last two decades of technological development, the objective of commercially available self driving cars seems now closer than ever. Multiple car manufacturers such Tesla and Chrysler are already proposing on the market models with partial self driving capabilities.
Most of the examples of partially self driving vehicles present on the market, such as Tesla cars, rely heavily on cameras and computer vision, but also integrate radars in order to identify obstacles [21]. Radar technology has multiple advantages over cameras in this application, specifically radars can sense obstacles even through fog, snow, or light vegetation. Radars are furthermore not subject to ambient light variations and can detect accurately radial velocities.
The combination of multiple technologies allows for a more robust sensing, making radar technology a fundamental part of autonomous vehicles.

1.2. Problem definition

For completely self driving vehicles to enter the market and being declared road legal, safety is of key essence. The correct detection of obstacles on the road depends on the amount of data collected by the sensors as well as on the processing method applied to this data.
Due to the changing road environment, data collection can become a challenge and the solution until now has been the integration of multiple types of sensors with different characteristics [21].
With this thesis we aim to give our contribution in the development of a new data sharing system based on radar waves. With the aim of allowing multiple vehicles to share sensors readings, increasing the data sets available to each car, all without the need for additional sensors or communication devices.

1.3. Previous research

This thesis will be the natural continuation of the work developed by the master student Tasneem Rahaman Khan in the course of her master thesis [12]. In her work, Tasneem reprogrammed a 77GHz FMCW radar to transmit a waveform with encoded information in the form of phase shifts. Her implementation works as a proof of concept for the possibility of embedding information in a radar waveform. However no transfer of information was achieved, and her experimental results were limited to transmitting a radar wave and receiving the scattered result on the same radar.
Actual transmission of information involves multiple other challenges, as will be explained in the next chapters.

1.4. Research objective and questions

The final objective of this thesis is to achieve transmission of information between two completely independent radars, embedding information in their respective waveforms.
In order to render more feasible the utilization of our system in real world applications, this thesis includes two more objectives: a substantial throughput increase and a more manageable radar programming.
The three following research requirements have been formulated in order to express clearly the objectives of this thesis.

1. Find an optimal method to synchronize multiple radar units enabling transmission of information between them.

2. Find a method to allow multiple bits to be sent during a single chirp

transmission. Increasing the system throughput.

3. Build a user interface, command line based, allowing a user to modify the functioning of the radar module, without a firmware flash.

Satisfying these three research requirements leads to the demonstration of a new and innovative communication system. A system that exploits an existing technology and that with minimal overhead enables autonomous or semi-autonomous vehicles to share sensor readings or additional information. Increasing the information available for navigation will make self driving cars more reliable and safer, contributing to their spreading.

1.5. Thesis outline

This thesis will be structured as follows:

- Chapter 2 will be dedicated to provide the reader an introduction on the topic of frequency modulated continuous wave radars.
  It will also include a short introduction on the platform used.
- Chapter 3 will introduce the reader to the previous work and the various improvements and modifications done before the start of this work.
- Chapter 4 will take care of the second research requirement.
- Chapter 5 will host the discussion for the first research requirement.
- Chapter 6 will include the work developed on the third and last research requirement.
- Chapter 7 will summarise the result of this thesis and propose future improvements.

The order in which the research requirements are laid out in this document respects the chronological order in which these problems have been tackled during this research. The need to follow the chronological order is due to the evolution of the different test setups through time, with the addition of new hardware and software components for each new research question faced.

2
Theory and backgound

In this chapter a short introduction will be given on the physical principles and equipment utilized during the course of the thesis. First, the functioning principle of a FMCW radar will be discussed, then a short description of the BPSK modulation technique will be provided. Following GPS and Ethernet basics concepts will be covered. Finally a short explanation on the development platform used will be given.

2.1. Radar

Radar technology is divided in two main branches, pulsed radars and continuous wave radars.
Pulsed radars send a time finite signal, at regular intervals. In the time interval between transmission they remain listening in order to capture any refracted signal. They then compute distance estimation from a target using the elapsed time between transmission and reception.
Continuous wave (CW) Radars send instead a continuous waveform. They continuously listen through the medium, receiving the scattered waveform from eventual targets. CW radars can estimate the velocity of the target comparing the received waveform to the transmitted one and exploiting the Doppler effect that induces an apparent frequency shift in the refracted signal. In this design however range estimation is impossible without adding any type of modulation to the transmitted waveform. To overcome this limitation the transmitted signal can be modulated in amplitude or frequency, making possible to estimate the distance from the target.

2.1.1. FMCW radar

FMCW radars are a particular case of CW radars. They involve the use of frequency modulation to make possible target distance estimation[3].
FMCW radars transmit a FM sinusoidal wave. The modulation is divided into periodic segments, named *chirps*. Each chirp has a similar form, starting with a defined base frequency, then increasing it linearly to a defined maximum frequency and finally sharply decreasing the frequency back to the starting point. At the receiver end the refracted signal will be mixed with the original signal, in order to obtain a frequency constant waveform to facilitate signal post processing.
A collection of chirps is named *radar frame*, typically comprised of 256, 512 or 1024 chirps in a single frame. In this work every mention of frames will refer to radar frames unless specifically mentioned.

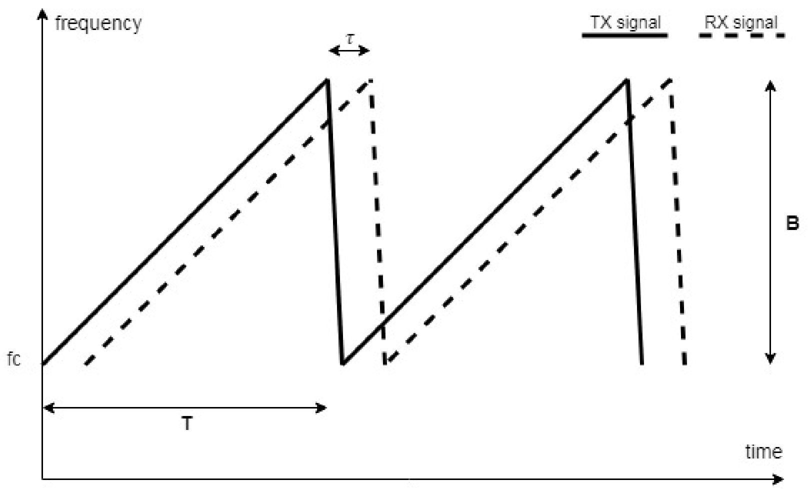


Figure 2.1: Chirp frequency representation

The linear increase of the frequency is a frequency sweep over a bandwidth B in a time T. As so it can be written as:

$$f(t) = f_c + St \tag{2.1}$$

where f_c is the carrier frequency and S is the steepness of the linear increase and corresponds to:

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

Based on the sign of S the chirp is defined as up-chirp ($S > 0$) or down-chirp ($S < 0$). As stated previously, the transmitted waveform of a FMCW is a cosinusoidal wave, in the form.

$$X_{tx}(t) = A_{tx} \cos(\psi(t)) \tag{2.3}$$

To find the correct angle $\psi(t)$ to obtain the required frequency sweep, equation 2.1 has to be integrated over time [8].

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

Leading to:

$$\psi(t) = 2\pi \left(f_c t + \frac{S}{2} t^2 \right) + \psi_0 \tag{2.5}$$

which can be substituted in equation 2.3.

$$X_{tx}(t) = A_{tx} \cos \left(2\pi \left(f_c t + \frac{S}{2} t^2 \right) + \psi_0 \right) \tag{2.6}$$

In this equation A_{tx} is the signal amplitude, f_c is the carrier frequency, S is the steepness of the curve, ψ_0 is the phase at time 0.
In an FMCW radar, the waveform is scattered from the target to the receiver. The finite speed of the waveform propagation allows for estimation of the distance R between target and receiver. First supposing the target is stationary, we can deduce how the time delay between transmission and reception is:

$$\tau_{stationary} = \frac{2R}{c} \tag{2.7}$$

were R is the distance between target and antenna, c is the speed of light and the factor 2 is needed because of the round trip delay. Now, supposing the target is moving at constant radial velocity v (the target is moving away from the receiver), the additional τ is:

$$\tau_{moving} = \frac{2vt}{c} \tag{2.8}$$

where vt is the additional distance from the receiver that the target has travelled after time t, considering $t = 0$ as the instant where the target was at distance R. While c is the speed of light.
The total delay is then:

$$\tau_{total} = \frac{2\left(R + vt\right)}{c} \tag{2.9}$$

The received signal will be:

$$X_{rx}(t - \tau) = A_{rx} \cos\left(\psi(t - \tau)\right) \tag{2.10}$$

The received signal will then be mixed with the sent signal. The equation of a mixer with two cosinusoidal waves as input is the following:

$$X_{IF} = \frac{A_1 A_2}{2} \left[\cos\left(\psi(t)_1 + \psi(t)_2\right) + \cos\left(\psi(t)_1 - \psi(t)_2\right)\right] \tag{2.11}$$

In the case of FMCW a low-pass filter is placed on the receiver side, eliminating the $(\psi(t)_1 + \psi(t)_2)$ component. After filtering, mixing the RX and TX signals leads to:

$$X_{IF} = \frac{A_{rx} A_{tx}}{2} \left[\cos\left(\psi(t) - \psi(t-\tau)\right)\right] \tag{2.12}$$

By substituting 2.5 in the above equation:

$$X_{IF} = \frac{A_{rx} A_{tx}}{2} \left[\cos\left(2\pi\left(f_c t + \frac{S}{2}t^2\right) + \psi_0 - 2\pi\left(f_c(t-\tau) + \frac{S}{2}(t-\tau)^2\right) - \psi_0\right)\right] \tag{2.13}$$

$$X_{IF} = \frac{A_{rx} A_{tx}}{2} \left[\cos\left(2\pi\left(f_c t + \frac{S}{2}t^2 - f_c(t-\tau) - \frac{S}{2}(t-\tau)^2\right)\right)\right] \tag{2.14}$$

$$X_{IF} = \frac{A_{rx} A_{tx}}{2} \left[\cos\left(2\pi\left(f_c \tau - \frac{S}{2}\tau^2 + St\tau\right)\right)\right] \tag{2.15}$$

In this equation, the τ can be substituted with equation 2.9.

$$X_{IF} = \frac{A_{rx} A_{tx}}{2} \left[\cos\left(2\pi\left(f_c \frac{2\left(R + vt\right)}{c} - \frac{S}{2}\left(\frac{2\left(R + vt\right)}{c}\right)^2 + St\frac{2\left(R + vt\right)}{c}\right)\right)\right] \tag{2.16}$$

$$X_{IF} = \frac{A_{rx} A_{tx}}{2} \left[\cos\left(4\pi\left(\left(\frac{f_c v + Svt + SR}{c} + \frac{Sv^2 t - 2SRv}{c^2}\right)t + \frac{f_c R}{c} - \frac{SR^2}{c^2}\right)\right)\right] \tag{2.17}$$

A series of simplifications can furthermore be applied. This is because $c^2 >> c$, as well as $c >> R$, $c >> v$, $c >> S$.

$$X_{IF} = \frac{A_{rx} A_{tx}}{2} \left[\cos\left(4\pi\left(\frac{f_c v + Svt + SR}{c}t + \frac{f_c R}{c}\right)\right)\right] \tag{2.18}$$

One additional consideration is that $vt << R$, because of the short duration of the chirp.

$$X_{IF} = \frac{A_{rx} A_{tx}}{2} \left[\cos\left(4\pi\left(\frac{f_c v + SR}{c}t + \frac{f_c R}{c}\right)\right)\right] \tag{2.19}$$

In the above equation two components are visible in the frequency term. The first, $\frac{f_c v}{c}$ is caused by the Doppler effect and subsequently is named Doppler frequency. While the second, $\frac{SR}{c}$ is called the beat frequency. Doppler frequency and beat frequency can be used to estimate the radial velocity as well as the distance of the target.
For stationary targets and stationary radars the Doppler frequency will become zero, leading to the simplified equation:

$$X_{IF} = \frac{A_{rx}A_{tx}}{2}\left[\cos\left(4\pi\left(\frac{SR}{c}t + \frac{f_c R}{c}\right)\right)\right] \tag{2.20}$$

Subsequently, the frequency of signal X_{IF} will be:

$$f_{Xif} = \frac{2SR}{c} = S\tau_{stationary} \tag{2.21}$$

2.2. BPSK modulation

In order to transmit any type of data using electromagnetic waves a specific modulation scheme needs to be used. Different options have been developed through the years, one of the most notorious is Phase Shifting Keying (PSK).
In phase shifting keying the symbols to be sent are coded in the phase of a sinusoidal signal. Supposing to have M symbols, each one gets coded in a different phase:

$$\varphi_n = \frac{\pi}{M}(2n-1) + \varphi_0 \tag{2.22}$$

With $n = 1, ..., M$ symbol number and φ_0 constant phase. The transmitted signal is then:

$$X_{tx}(t) = A_{tx}\cos(2\pi f_c t + \varphi_n) \tag{2.23}$$

The simplest version of PSK is with $M = 2$ called Binary PSK (BPSK). In this case, the two symbols are coded with two waveforms of opposing phase.

$$X_{tx1}(t) = A_{tx}\cos(2\pi f_c t + \varphi_0) \tag{2.24}$$

$$X_{tx2}(t) = A_{tx}\cos(2\pi f_c t + \pi + \varphi_0) \tag{2.25}$$
$$= -A_{tx}\cos(2\pi f_c t + \varphi_0) \tag{2.26}$$

Thanks to this large phase difference between the only two symbols BPSK modulation is more resistant to noise compared to PSK modulations with more symbols. For example, the symbol error probability of BPSK can be estimated as [1]:

$$P_e = \sqrt{2}Q\left(\sqrt{\frac{E_s}{N_0}}\right) \tag{2.27}$$

Where $Q(x)$ is defined as the tail distribution function of the standard normal distribution, defining the probability of a standard normal random variable to exceed the threshold x. With x taking the value $\sqrt{E_s/N_0}$, in which E_s is the symbol energy and N_0 is the noise power spectral density. While the symbol error probability of the a QPSK modulation ($M = 4$) is:

$$P_e \simeq 2Q\left(\sqrt{\frac{E_s}{N_0}}\right) \tag{2.28}$$

It's visible how the BPSK has a lower error per symbol compared to QPSK. Due to the time limitation our implementation does not include error correction over the radar channel, for this reason we consider the error per symbol more important than the error per bit.
The low error per symbol as well as the ease of implementation are some of the reasons we utilize BPSK.

2.3. GPS

The Global Positioning System, in short GPS, is a global wide service provided and owned by the USA military. It consists of space-based positioning, navigation and timing signals provided for free to users.
Such service is provided through a network of satellites flying in medium earth orbit. In the beginning of 2019 the US had 31 operational satellites and ensures to users at least 24 satellites in orbit 95% of the time. A 24 satellites mesh ensures that users have at least 4 satellites visible in any part of the planet [10].

2.3.1. Trilateration

GPS receivers use trilateration to determine their position, speed as well as synchronize the internal clock to the atomic clock contained in the satellites [4].
Knowing the position of each satellite in orbit, the receiver can estimate its distance from each one of them using its internal inaccurate clock. We can define t_r the time offset of the receiver compared to the satellite clock, and δ_n the real travel time of a signal from each satellite to the receiver (with

$n = 1, .., 4$). The distance is at first estimated as follows.

$$d_n = (t_r + \delta_n)c \tag{2.29}$$

This brings the receiver to have 4 known points in a tridimensional space at a known distance. A sphere with as radious the distance can be drawn around each satellite. Figure 2.2 shows an example restricted to two dimensions (where only three satellites are necessary) for ease of representation. Considering how the distance between satellites is known, the

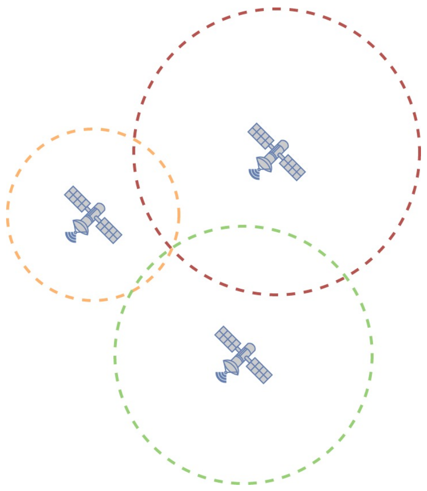


Figure 2.2: 2D representation of a trilateration scenario

problem becomes estimating the intersection between four spheres. And can be expressed by the following system of equations (restricted for the 2D scenario) [22].

$$(x - x_1)^2 + (y - y_1)^2 + (z - z_1)^2 = d_1^2 \tag{2.30}$$
$$(x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = d_2^2 \tag{2.31}$$
$$(x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 = d_3^2 \tag{2.32}$$

Where x_n, y_n, z_n are to coordinates of each satellite.
The above system has a single solution for the vector: $d = (d_1, d_2, d_3)$.
Subsequently, the time offsets δ_n can be calculated and the local clock can be aligned withe the atomic clock on the satellites.
Following, knowing the distance from each satellite, the position of the receiver is easily estimated. Furthermore the velocity of the receiver can be estimated using the position just calculated or exploiting the Doppler effect on the received signals.

For the purpose of this thesis the GPS technology is evaluated only as an effective method to obtain an accurate clock with precision inferior to the microseconds. Different low cost modules are available for this purpose and they claim precision within to 60ns for a pulse per second signal (99% of the times)[23].

2.4. Ethernet and UDP/TCP

2.4.1. Ethernet

The Ethernet is composed of hardware and software elements and has the function to connect two different computing units together, creating a channel to transmit information [19].
Three components can be distinguished in the Ethernet structure:

- The Ethernet frame: a sequence of bits with a well defined structure used to transmit an information.
- The Media Access Control protocol: consists on a series of rules that arbitrate how different machines can transmit over the channel.
- The physical medium: the cables and other hardware used to physically send the frames over the network.

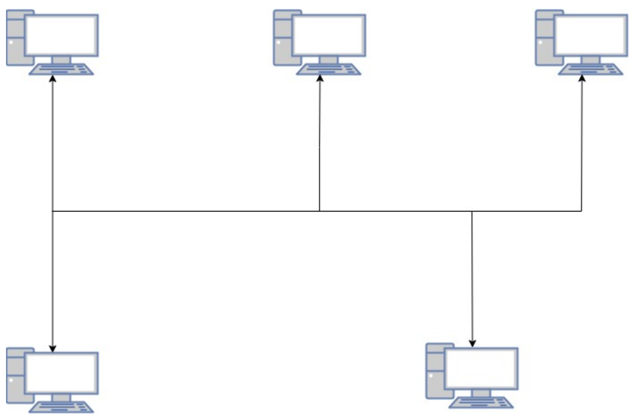


Figure 2.3: Typical Ethernet network configuration

The Ethernet frame

An Ethernet frame is the basic information unit sent over Ethernet, it is a sequence of bits with a well defined structure. The standard Ethernet frame can be divided in three parts:

- The header: a sequence of bits with a standard structure carrying information about the frame itself, the main fields are: preamble, destination address, source address, length.
- The data payload: contains the bits of information to carry from source to destination. The length of this section is defined in the header.
- Frame check sequence: contains a CRC (cyclic redundancy check) of the data payload, is used to check if the frame received contains errors.

The Media Access Control

The Ethernet can be used to connect multiple devices to the same physical medium. For its intrinsic structure no master device is defined and a Media Access Control protocol has to be implemented to avoid collisions.
The original protocol used is the CSMA/CD protocol [20]. In more modern networks however faster speeds are reached when connecting each device directly to a network switch eliminating the need for MAC protocol. MAC protocol can still be employed when using Ethernet with speeds of 10 or 100Mb/s but any faster connection does not support it.

The physical medium

The physical medium is composed by different elements:

- Cables, used to transmit the electrical signals.
- Transmit and receive devices on the computational units that exploit the Ethernet for communication.
- Switches, active elements connecting different parts of the network together. They connect to multiple links and forward packets from one link to another (or others).

2.4.2. IP TCP/UDP

The Internet Protocol (IP) is a layer above Ethernet and while the Ethernet protocol takes care of transmitting frames over a single Ethernet network, the IP protocol can send packets (composed by multiple frames) over multiple Ethernet networks (or other types of networks).
The IP protocol is still a connectionless protocol, meaning that no connection is established between sender and receiver [15].
The IP protocol supports the two most used transport layer protocols UDP and TCP.

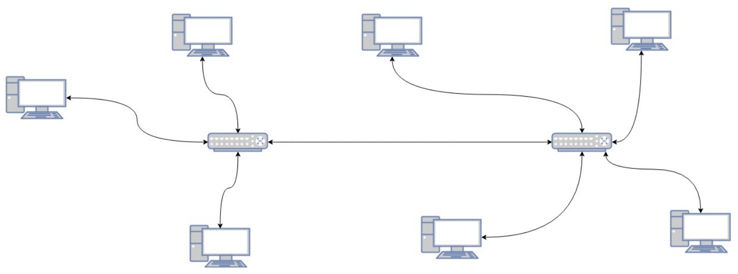


Figure 2.4: Typical small network configuration

TCP

TCP is a transport layer protocol that is used for reliable transmission of information between to devices. It's a connection based protocol, meaning that before transmitting information a connection is established between the two devices and during transmission it's ensured that all the sent information will be received at the other end of the connection.
TCP ensures correct transmission of data, however this is at the expenses of transmission speed when compared to connectionless protocols like UDP [6].

UDP

UDP is a transport layer protocol that is used for transmission of data when no guarantee of correct reception is needed.
In the UDP protocol the sender (named server) broadcast the information over the medium, without giving any consideration if any receiver (named client) is receiving the information. At any moment a client could start listening over the medium and receive any of the packets sent by the server. It's important to notice how, especially in large networks, no guarantee of correct delivery is given, the packets could be lost during transmission or be received in wrong order.
The main advantage of UDP over TCP is the faster transmission speed given by eliminating the overhead of the connection.

2.5. Uncertainty propagation rules

In chapter 6 it will come useful to know some concepts about uncertainty propagation theory. For this reason we refresh a simple formula for weighted sums and differences between measurements [13].

If we observe the measurement z, defined as:

$$z = ax \pm by \pm cq ... \tag{2.33}$$

With a, b, c... constants without uncertainty.
And x, y, q... are uncorrelated measurements with uncertainties $\delta x, \delta y, \delta q$...
Then, the following relation on the composed measurement z uncertainty δz is true:

$$\delta z = \sqrt{(a\delta x)^2 + (b\delta y)^2 + (c\delta q)^2 + ...} \tag{2.34}$$

A special case of this equation is if a single operation with null uncertainty is performed on a measurement. In such case, the equation simplifies to:

$$\delta z = a\delta x \tag{2.35}$$

This simplified equation assumes the scaling constant a contains no uncertainty, and under this assumption will be used in this thesis.

2.6. Radar development platform

The development platform used is a dual PCB small factor system provided by NXP. It consists of two PCBs in a credit card format and the product name is simply: ”TEF810X / S32R274 DCC development kit”.
The first PCB contains a TEF810X chipset as well as three transmission antennas and four receiving antennas. The second one contains a power supply circuit, a S32R274 chipset, an Ethernet module, a CAN module and a small selection of general purpose input/outputs.

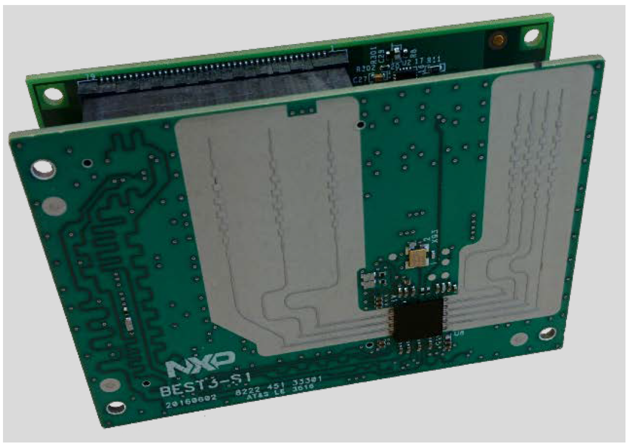


Figure 2.5: DCC Development Kit

2.6.1. TEF810X

The TEF810X block diagram can be seen in figure 2.6. At the transmission side, using an extenal 40MHz oscillator, the Chirp Generator module

generates the waveform to be transmitted (chirp). The Timing Engine (TE) included in the Chirp Generator module takes care of enabling the different transmission and receive lines, as well as determining any phase shift to be applied to each individual line.
For each transmission line, the TEF810X has available 4 different transmission profiles, each profile contains chirp specific settings. The profiles can be switched dynamically between chirps.
At the receiver side, the incoming signal is mixed with the internally generated chirp, this in order to obtain the X_{IF} signal. Following this a high pass filter is applied to remove low frequency noise, then the signal is amplified. Next a low pass filter is applied in order to remove one component of the received waveform, as described in equation 2.11. Finally the signal is sampled by an ADC module.
Before transmission the sampled values can be decimated based on the chip settings. After this, the ADC samples from the different receive antennas are serialized and transmitted.
In our configuration the chip is controlled through SPI and as such only write and read from/to registers commands are available. The transceiver outputs the sampled values through MIPI CSI-2, a serial interface commonly used for cameras [11].

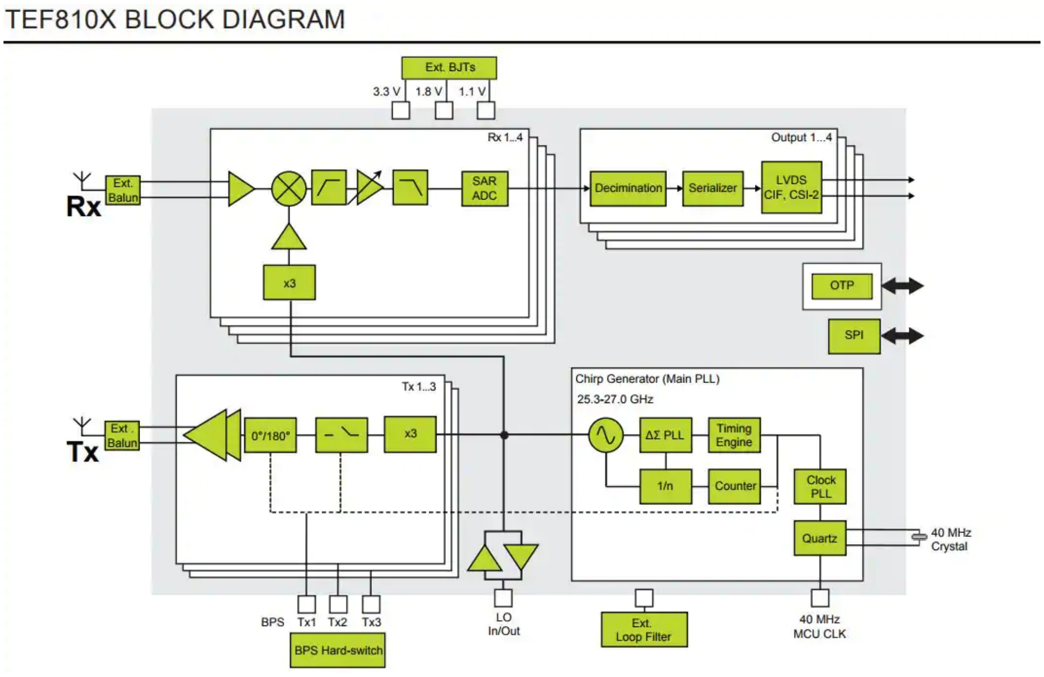


Figure 2.6: TEF810X Block Diagram [18]

2.6.2. S32R274

The s32r274 chip is a multi-core power architecture based MCU [17]. Appositely designed to control radar modules. It contains an optimized radar signal processing accelerator module. With a signal toolbox capable of performing FFT and with direct memory access.
This multicore architecture is composed by four cores, two independent Power Architecture e200z7 32-bit CPUs and two Power Architecture e200z4 32-bit CPUs one of which is used as checker core. This means that the two e200z4 cores perform the same operations, for redundancy reasons, and can be treated as a single core by the user.
In terms of Input Output it contains dedicated modules for Ethernet and CAN.
A testing firmware version for this device has been provided by NXP. Such

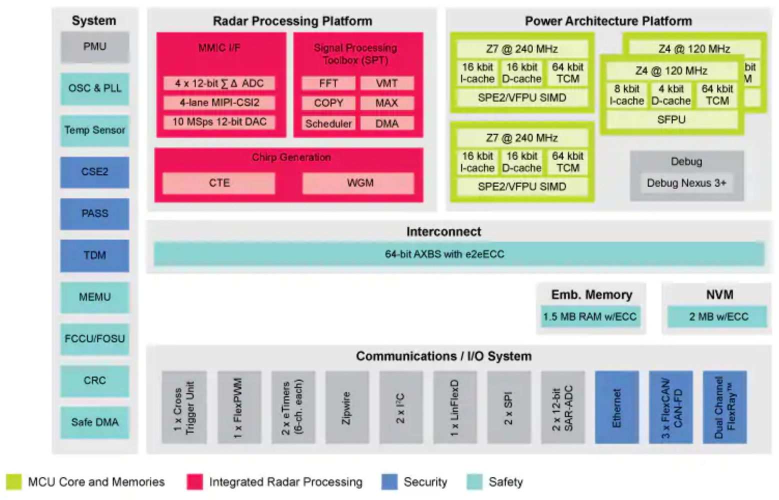


Figure 2.7: S32R274 Block Diagram [17]

software makes use of two of the three available cores.
Core 0 is used to control the TEF810X chipsed through SPI and a dedicated API, called Dolphin API. Core 0 is also responsible to control the Radar Processing Platform deciding the type of computation to be performed on the raw ADC samples received from TEF810X.
Core 1 takes care of transfering the processed data through Ethernet. It uses UDP to broadcast the values on the channel. Core 1 makes use of a modified version of fnet embedded TCP/IP stack.

S32R274 firmware

The Dolphin API allows to control the TEF810X chipset through a series of functions. The more important functions are:

- *chip_TE_ChirpTrigMode()* configure the chirp (or the chirp sequence) transmission, to be triggered by SPI command or by a pin of TEF810X.
- *chip_TE_StaticConfig()* configure profile independent settings of the timing engine, such as the number of samples per chirp and the profile (or sequence of profiles) selected.
- *chip_Chirp_Program()* profile dependent system-level API which configures a chirp profile in one call. Most of the chirp parameters can be configured through this function.
- *chip_TX_ProfileConfig()* configure the transmission channels profiles. Configures for each profile transmission channel parameters such as transmission gain and LO trippler gain.
- *chip_RX_ProfileConfig()* configure the receive channels profiles. Configures for each profile receive channel parameters such as receive gain, low pass filter and high pass filter.
- *chip_CC_SerializerInterfaceSet()* and *chip_CSI2_Config()* configure the CSI2 interface to transmit back the ADC values.
- *chip_reg_Read()* performs a single read through SPI from a register block of the TEF810X chip. Each register block is 32 bits wide.
- *chip_reg_Write()* performs a single write through SPI to a register block of the TEF810X chip. Each register block is 32 bits wide.
- *chip_TE_ChirpStart()* this API triggers one chirp sequence (frame).

Through these functions, different aspects of the transmitted waveform can be modified. The parameters that can be modified include:

- Chirp period $T_{chirp\_interval}$: total time of one chirp is modified through the change of the sub-times composing the chirp;
- Dwell time T_{dwell}: time before starting the chirp;
- Settle Time T_{settle}: time to wait after starting the chirp before ADC acquisition starts;
- Sample Time T_{active}: time interval in which the ADC acquisition is performed. This time is indirectly selected by setting sampling rate as well as number of samples each chirp;

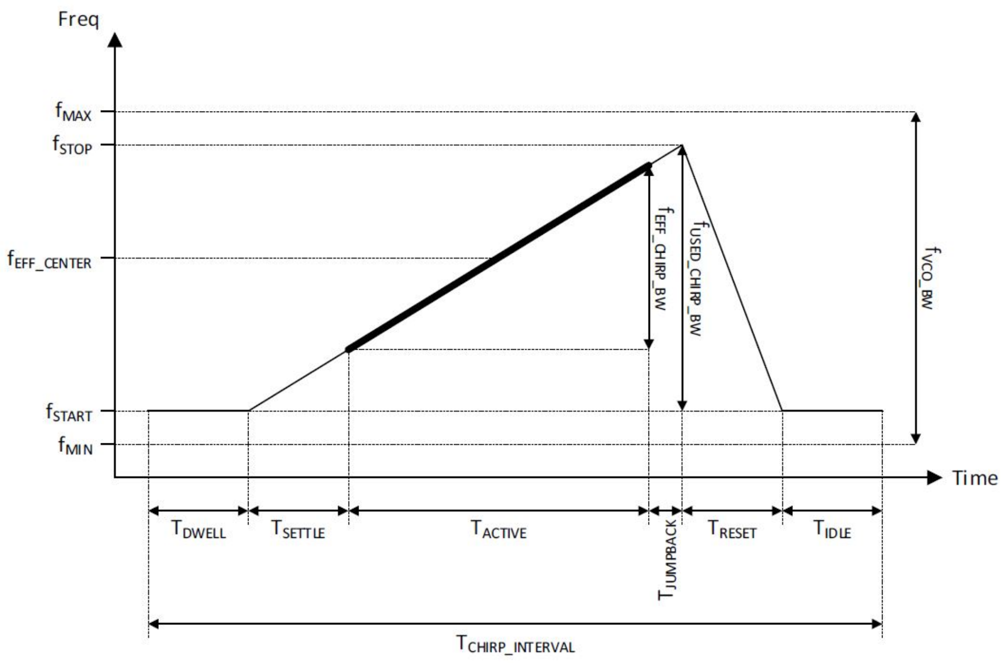


Figure 2.8: Chirp with different parameters [18]

- Jumpback Time $T_{Jumpback}$: time after ADC capture ended and before end of linear frequency increase;
- Reset Time T_{reset}: Time for the frequency to drop to the initial value;
- Center frequency (effective) f_{eff_center};
- Chirp bandwidth (effective) $f_{eff\_chirp\_BW}$;
- Number of ADC samples to be collected each chirp;
- Activation and deactivation of each TX and receiver line;
- Phase state of each TX channel;
- Sampling frequency of the ADC module. One of three values can be selected: $40Msps$, $20Msps$ and $10Msps$.

Additional parameters that can be controlled through the API are: decimation value, transmission and receive gain.

STM hardware timing modules

The S32R274 micro-controller includes three System Timer Module (STM), 32-bit timer modules designed to support commonly required system and application software timing functions.
Each STM includes a 32-bit up counter and four 32-bit compare channels with a separate interrupt source for each channel. The counter is driven by the appropriate system clock divided by an 8-bit prescale value (1 to 256). In our implementation the clock selected to drive the timing module is the $PBRIDGE$, set at $60MHz$. This clock is obtained from the external crystal oscillator at $40MHz$ by scaling it up by a factor 6 and then dividing it of a factor 4 outside of the STMs [14](page 151).

3 Previous work

In this chapter, an in depth explanation of the previous work developed by Tasneem R Khan during the course of her thesis will be presented. Understanding such system is necessary to the reader since the implementation presented in the following chapters is based on it. Furthermore, because of how unstable and covered by bugs the previous work was, a new stable version has been developed. Some of the changes are also included in this chapter.

The system developed by T.R. Khan in [12] can be described as a distribuited system. It runs on two different computational units, a personal computer and the development board described in chapter 2.
It can be divided conceptually in 4 components (as visible in figure 4.1 present at the beginning of chapter 4.):

- Client application to run on PC.
- Firmware running on the S32R274 chipset. This software can be furthermore be divided in two blocks:
  - Software to run on Core 0.
  - Software to run on Core 1.
- Server application to run on PC.
- Matlab script for post processing.

Following the path of a message, the implementation can be divided in the following steps.

- The user enters a message on the PC client application. The client forwards the message through Ethernet to the Radar Board.
- The Radar Board encodes the message in the sensing waveform and sends it.
- The Radar Board receives the scattered sensing waveform and forwards the ADC raw samples to the PC server through Ethernet.
- The PC server application receives the ADC samples and stores them in a binary file.
- The binary file is opened and the ADC samples are processed through matlab. The message is extracted from the ADC samples and returned to the user.

To discuss this implementation, this chapter will follow the message path and as so will analize each step explained above.

3.1. PC client application

This application is written in C, and is developed to be used on Linux machines. On Windows counterparts Cygwin can be installed.
The application utlizes the library *sys/socket.h* to create a TCP socket. It then tries to establish a connection to the server application on the Radar Board.
If successful, the application enters a infinite while loop. In each iteration of the loop it performs a serie of operations.

- Prompts the user to insert a string.
- Reads from the standard input a string of maximum length 32. If the read string is empty then the subsequent steps are not performed and the loop starts from the beginning.
- Sends the string through TCP to the server.
- Waits until the server has replied and prints the reply to the standard output.

The loop is repeated indefinitely. The program can be stopped by pressing ESC and then ENTER when promped for a new string.

3.2. Radar Board reception of message

On the Radar Board, Ethernet communication is accomplished by using a version of fnet [5] modified by NXP to transmit back radar values.
To receive the string from the PC Client, a socket is created, configured and binded on Core 1.
A infinite while loop is then started on Core 1, inside this loop a series of operations is performed.

- A connection from a client is accepted. If no client is requiring a connection the while loop starts from the beginning.
- A nested infinite while loop is started. It performs the following sequence of actions.
  - Receives the message sent from the PC client. And stores it in a private location of memory for Core 1. If the connection is closed by the client, then exits this loop.
  - Sends back the message it just received without performing any computation.
  - Locks (reserves for current core) an inter-core shared memory location.
  - Writes the message in such location only if the received message has length larger than 0.
  - Unlocks the shared memory location.

This loop repeats indefinitely and if no message is received from the client no operation is performed.

3.3. Shared memory and semaphore implementation

In order to regulate the access to shared memory locations between cores, the s32R274 chipset makes available 32 shared registers between cores.
The system implemented by T. R. Khan consists in creating a new shared memory location of $32K$ through the modification of the dedicated linker files. The location of this memory section has been modified from the one chosen by T.R. Khan since it was conflicting with other memory sections and placed in an unused location.
The essential parts of the linker files can be seen in the below code extract.

```
MEMORY {
    [...]
    /* Shared memory location */
    /* Located at the end of the shared memory between */
    /* cores. */
    shmem :     org = 0x40000000+ LENGTH(c0_sram)+
                    LENGTH(c1_sram)+ LENGTH(c2_sram)+
                    1K+LENGTH(all_sram)-32K,    len = 32K
}
```

```
SECTIONS{
    [...]
    /* SHARED MEMORY SECTION */
    .shared_memory (NOLOAD):
    {
        *(.shared_mem.o)
    } > shmem
    [...]
    /*shared memory*/
    __shmem_struct = ADDR(.shared_memory);
}
```

The register *SEMA42.[SEMA42_GATE0].R* regulates access to this shared memory location. If a core wants to access the shared location it first checks the status of the *SEMA42.[SEMA42_GATE0].R* register. If the register is clear then it changes its value to it's core number plus 1 and access the memory, otherwise it waits until the register is clear. After the access is complete, the core clears again the register.
This method eliminates the risk of a conflict during memory access.

3.4. Radar Board transmission of waveform

At startup Core 0 initializes the TEF810X through the Dolphin API and immediately starts sending chirps following the settings hard coded in the firmware.
In T.R. Khan's code these settings have been slightly modified. The radar sends always using a single antenna and receives using a single antenna.

The main modification from the original settings is that the radar alternates each chirp between transmission profile 0 and transmission profile 1. At startup both profiles are set as identical. Furthermore, the Radar Board is configured to output always ADC signals.
The original main function executed by Core 0 takes care of initializing the radar and periodically triggering a sequence of chirps to be sent (defined as frame).
The function that triggers the transmission of a frame has been modified to allow for transmission of the message. Before triggering the transmission, the following additional operations are performed.

- Reads the shared memory location in the same way as Core 1 does and stores the content in a local buffer.
- Reads, using *chip_reg_Read()* the value of the register block containing the profile 0 phase inversion value. Stores it in a local variable.
- Reads, using *chip_reg_Read()* the value of the register block containing the profile 1 phase inversion value. Stores it in a local variable.
- The values that have just been read are 32bit wide, however the phase inversion is controlled by a single bit for each TX antenna. Preforming a bitwise AND operation with a mask, only the phase shift bits are set to 0.
- Using *chip_reg_Write()* profile 0 settings are saved with phase inversion 0. This in order to send a single bit of header before the actual message (this passage eases the post processing and has been added after [12]).
- Using a bitwise OR with the message saved in the local buffer, the first bit of information to be sent is read.
- The bit containing the phase inversion in profile 1 of TX antenna 1 is changed in order to reflect the first bit of the message.
- Using *chip_reg_Write()* the modified register block containing the phase inversion value of profile 1 is saved on the TEF810x chip.

After this passage, the TEF810x is configured to send the first chirp with no phase inversion and the second bit with phase inversion only if the first bit of the message is 1.
The frame transmission is then started using the function *chip_TE_ChirpStart()*.

For the next steps is assumed that the function *chip_TE_ChirpStart()* returns after a constant time smaller than the first chirp transmission. If this was not the case, coding the second chirp would be impossible.
After starting the frame transmission a first while loop is entered. As guard of such loop is the condition *SPT.GBL_STATUS.B.FRM_ACQ_DONE == 0*, this condition will keep the loop running until TEF810X signals the end of acquisition of the current frame.
Inside this loop a series of operations is performed.

- A check is performed to ensure if there are still bits to send. If not, the loop is broken.
- Wait until the the current chirp acquisition is finished.
- Extract the next bit to send from message.
- Use the position of the current bit to decide in which profile to encode the bit (using odd or even).
- Using *chip_reg_Write()* save the phase inversion setting to the TEF810X chipset.
- Increment the number of sent bits.

This function will be repeated indefinitely as it is. As a consequence, until a new message is sent through TCP from the PC Client, the radar will keep transmitting the same message.

3.5. Radar Board reception of the waveform

The receive part of the waveform has not been developed by T.R. Khan, but uses a demo code provided by NXP. The selected output data are ADC samples.
A short explanation on the functioning of the receive path of the radar will be given here.
Being the selected output data ADC samples, the Signal Processing Platform is bypassed in our case. No operation is performed other than saving the data in a shared memory location. The reception of the scattered signal is performed in parallel with the transmission. For this reason the function *chip_TE_ChirpStart()* called by Core 0 also triggers the acquisition of the received data.
On Core 0 after all the chirps in one frame have been transmitted an interrupt is raised to Core 1. Core 0 waits until the interrupt is cleared by Core

1.
The function *c1_fnet_RADAR_output_isr* is triggered on Core 1 by the interrupt. This function transmits all the ADC values through the Ethernet port in the form of UDP packets.
The UDP packets are sent to the server application on the PC side. For the connectionless nature of UDP, some packets could be lost in this process.

3.6. PC server reception

The PC server application is written in C and meant to operate on Linux systems. It uses the same library as the PC client application (*sys/socket.h*) to create a socket and a server application.
It then requests the user for a time duration of the acquisition of data from the radar. This time determines only the time the PC application will register data sent from the radar, but will have no influence on the radar functioning.
The application saves the received data, without any computation in a binary file and once the user set time for acquisition is over, it automatically terminate execution.

3.7. Matlab post processing

The raw ADC values are now saved in a binary file and can be accessed with a Matlab script in order to extract the sent message as well as compute range and Doppler FFT when required.
The script has been modified in almost every part from T.R. Khan's work. This was necessary in order to provide a stable platform to visualize the effects of the software developed in this thesis.
For simplicity, the Matlab script has been divided in steps, and will be discussed here.

Step 0: opening binary file

A few variables have been placed at the beginning of the script to ease global changes. First, since the ADC samples received are from multiple frames and a single one is needed to receive the message, the frame number to extract the message can be selected. Second, the number of bits that compose the header can be selected.
The file is then opened and each packet content is divided into the different parts composing it. Each packet contains:

1. *DataTypeID*: Contains the type of data saved in the packet (ADC in this case)

2. *FrameNumber*: Contains the frame number of the data contained in the packet.

3. *ChirpSeqNum*: Contains the number of the chirp (relative to the frame) of the data contained in the packet.

4. *PacketNum*: Contains the packet number relative to the chirp. In this case each chirp is divided over 2 packets, this field will be 0 or 1.

5. *TotPacketNum*: Contains the number of packets for each chirp, always 2 in our case.

6. *Buffer*: Contains the data payload, each packet carries 732 values.

Only the buffer and FrameNumber fields will be needed and are stored in matrix and array form. The buffer matrix contains one packet for each column.

Step 1: cutting half frames

In this step, the first and last frames are deleted. This because the acquisition starts and finishes at a random place and the first and last frame recorded are often incomplete.
The buffer matrix and the FrameNumber array are shaped accordingly. The buffer matrix is renamed Data.

Step 2: dividing into frames

This step reshapes the Data matrix in order to obtain a three dimensional matrix where: each column is one packet and each third dimension contains only packets from one frame.
Due to the UDP lossy protocol used to send the data packets, this passage cannot be performed using the reshape function included in matlab. It will require some additional computation in order to exclude any frame containing any lost packet.

Step 3: composing chirps

Each chirp is divided in two packets, now a reshape function can be used to stitch together couple of packets obtaining a Data matrix with for each column one Chirp.

Step 4: extracting data from antenna 1

In the Data matrix, each chirp contains the received data from all the antennas. In our case only one RX antenna is used and for this reason only the samples from this antenna will be needed.
This step extracts the data from antenna 1 and stores it back in Data, resizing it accordingly.
Furthermore, only 256 samples are recorded by the receiver antenna, Data will be resized to contain only the first 256 values of each column.

Step 5: plot

A plot of the selected frame for data extraction is displayed. The two dimensional matrix is displayed as an grayscale image.
This allows for graphical inspection of the phase inverted chirps.

Step 6: selecting a good ADC sample to extract the message

In order to extract the message, the phase inversion has to be estimated for each chirp. This can be done by taking a strong ADC sample and checking the sign for each chirp.
This step takes care of selecting the highest ADC sample through all the chirps and evaluate the sign of all the chirps for the same ADC sample.
After this step, the first bit sent as header, the value of which is known, is used to distinguish between sent bit 1 and sent bit 0.
An array of 1s and 0s is constructed at the end of this step.

Step 7: fixing the phase shifted chirps

This step flips the chirps with phase inversion, returning a matrix containing the ADC samples as if no transmission where ever made.
This matrix can be used to extract Doppler FFT and range FFT.

Step 8: conversion of binary sequence into string

This step converts the binary array returned after step 6 in a readable character based string. It outputs this string on the command line to be read.

3.8. Summary

By executing all the programs described above, it's possible to demonstrate how information can be encoded and decoded correctly from the waveform used for sensing.
The above has been tested with a stationary radar in an room with stationary targets and performs reliably.

4 Test setup evolution

In this chapter, the different test setups developed will be explained.

During the course of this work, the test setup used has changed considerably. This short chapter has been created to facilitate the understanding of the different test setups used in the following sections.

4.1. Initial test setup

The initial test setup, that will be used in chapter 5 is the same as the one used by T.R. Khan for her work.
A single radar will be pointed straight to a stationary target at a distance of approximately two meters. The stationary target will be a strong corner reflector, chosen in order to maximize the reflected signal for post processing. This setup is visible in figure 4.1.

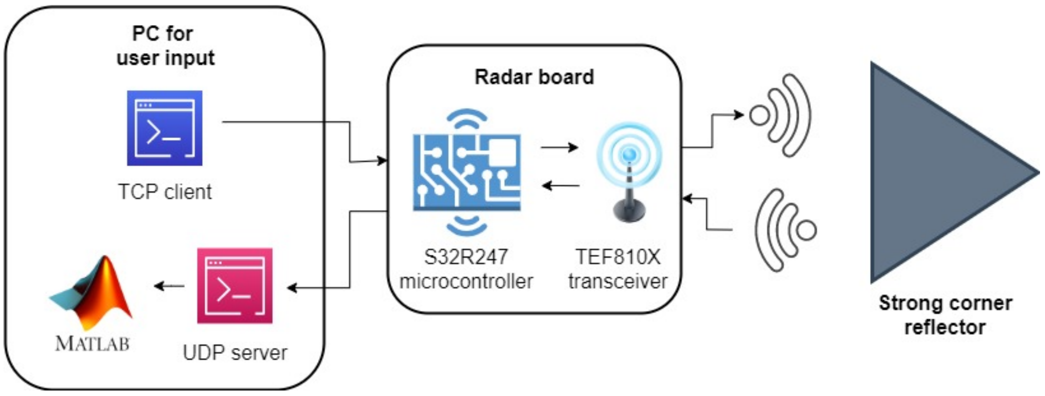


Figure 4.1: Diagram of T.R. Khan's setup

4.2. Test setup with GPS

From chapter 6 the setup used will change. Instead of a single radar, now two radars will be used. The two radars will face each other at a distance of one and a half meters. Each radar will be connected to a dedicated PC as well as a GPS module.
This new setup configuration can be seen in figure 4.2. Due to limitations in the mounting system of the hardware devices both radars will be placed on the same table, as so we can expect some interference caused by the surface of the table.

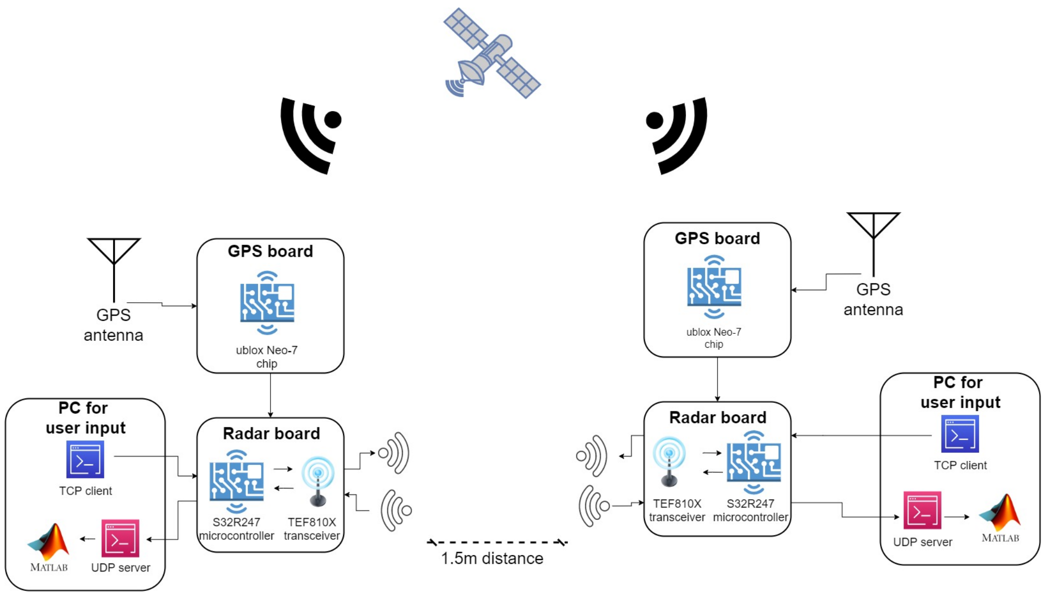


Figure 4.2: Diagram of the setup used from chapter 6

4.3. Additional test setups

The system implemented in this thesis has been used with similar configurations during multiple experiments conducted with doctor F. Uysal for the Microwave Sensing, Signals and Systems department. These configurations include multiple targets, moving and stationary, they are however out of the scope of this thesis and will not be described [16] [9].

5
Multiple symbols encoding in single chirp

In this chapter the second research requirement will be discussed. Namely how to encode multiple bits, equivalent to multiple phase shifts, into a single chirp.

In order to improve the throughput of the system developed by T.R. Khan in [12] multiple bits of information can be sent during a single chirp. This can be accomplished by inverting multiple times in a single chirp the phase of the transmitted signal.
This method could be used to increase the throughput of a transmission system based on this technology. As we will see in this chapter the main limitation in the scaling factor will come from the switching behaviour of the radar hardware architecture.
To accomplish phase inversion two approaches have been evaluated:

- Inversion through the dedicated pin on the TEF810X chip.
- Inversion through SPI commands from the S32R274 to the TEF810X.

Both methods have been evaluated and tested independently. While the using SPI commands brought no results, it was possible to successfully switch phase using the dedicated pin.
The process as well as the results and limitations will be discussed in this chapter.

5.1. Radar settings and setup

In all scenarios the radar has been configured with the following parameters:

- Effective bandwidth $1.5GHz$.
- Samples per chirp 256.
- Sample rate of ADC module $10Msamples/s$.
- Chirps per frame 512.
- $T_{dwell} = 40.05us$
- $T_{settle} = 5us$
- $T_{jumpback} = 0.25us$
- $T_{reset} = 5us$

In order to facilitate post processing, only one TX antenna has been used for transmission and only one RX antenna will be used for receiving the scattered signal.

For simplifying the extraction of encoded bits by providing a clear scattered signal, a stationary strong corner reflector has been placed at a distance of $R = 2m$ from the radar.
The data received on the PC side are raw ADC samples with no post processing performed on the radar board.
Figure 5.1 shows the ADC samples of the received mixed signal after being filtered and mixed in the TEF810X chip, this signal has been defined as X_{IF} in chapter 2.
The frequency of such signal is proportional to both distance and velocity of the targets, being the target stationary the velocity component of the frequency can be ignored. The frequency of the signal can be estimated using equation 2.21 to:

$$f_{XIF} = (2SR)/c = 781KHz \tag{5.1}$$

Where: c is the speed of light, R is the distance from the target, and S is the steepness of the linear increase $S = B/T$.

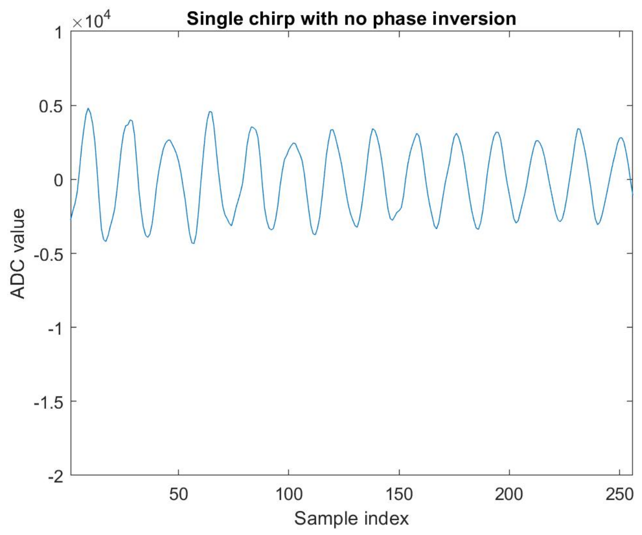


Figure 5.1: ADC values of a single chirp.

5.1.1. Development platform upgrades

Some modifications to the firmware of the radar have been performed. This in order to facilitate post processing and analysis of the raw data with phase shifts.
Three modes of operation are selectable in the firmware of the radar:

- No phase shift: no phase shift is applied on the signal. Can be used for comparison.
- Constant phase shift: each chirp has the same sequence of bits encoded in the three phase shifts. Can be used to compare the chirps in a frame with each others.
- Random number phase shift: a random number is created on computer side (and saved locally) the radar then sends it over a frame. Can be used to verify the effective transmission of information.

5.2. Timing functions

In order to automate the post processing of the data, independently from the method used, is of critical importance to shift frequencies in the same point for each chirp.
In the code running on core 0 of the S32R274 chip, a function is present that triggers each time a chirp transmission starts. What is missing in such code is a delay function that can set the processor idle, until a predefined

amount of time has passed.
This function has been developed, allowing us to insert a known delay in the S32R274 code, after each chirp has started. The hardware system timer module (STM) included in the microcontroller has been used for this purpose, this module includes multiple hardware counters and comparators.
The counters increase at the PBRIDGE clock frequency, set at 60MHz. Furthermore, a divider can be set to make the counters increase at a fraction of the clock frequency.
Three functions have been created:

- Start function: starts the timer
- Stop function: stops the timer
- Wait function: a blocking function that returns after the set amount of clock cycles has elapsed.

The code can be seen below:

```
void STM_2_start(void){
    // Channel enable CH0
    STM_2.CHANNEL[0].CCR.R = 0x1;
    // Divide system clock by 1 & enable timer
    STM_2.CR.R = 0x0001;
    //(Counter increments each 1/60MHz = 0.01667us)
}
void STM_2_stop(void){
    // Disable timer
    STM_2.CR.R = 0x0;
}
void STM_2_wait(uint32_t time){
    // Set compare value CH0
    STM_2.CHANNEL[0].CMP.R = time;
    // Clear counter
    STM_2.CNT.R = 0;
    // Clear flag (w1r)
    STM_2.CHANNEL[0].CIR.R = 0x1;

    // Wait until compare value matches counter
    while (STM_2.CHANNEL[0].CIR.R == 0x0);
    // Clear flag (w1r)
    STM_2.CHANNEL[0].CIR.R = 0x1;
}
```

Thanks to these timing functions a known delay can be placed anywhere in the code.

5.3. Estimating the correct delay

In order to test if it is possible to change phase during a chirp with an SPI command, we need to estimate after how much time from a chirp start we will be in the middle of the chirp.
The parameters of the sent chirp are hard coded in the S32R274 software and can be seen below.

- Samples per chirp 256.
- Sampling rate of ADC module $10Msamples/s$.
- $T_{dwell} = 40.05us$
- $T_{settle} = 5us$

From the *Samples per chirp* and the *Sampling rate* parameters, the time duration of the chirp slope can be estimated and subsequently a γ time constant, representing the time from the beginning of the chirp to the middle of the slope can be estimated.

$$\gamma = T_{dwell} + T_{settle} + \frac{Samples\ per\ chirp}{2 * Sampling\ rate} = 58.3us \tag{5.2}$$

A delay corresponding to this value can be then inserted after the start of the chirp using the STM modules.

5.4. Using SPI commands

The connection between the S32R274 and the TEF810X chips is established through SPI. As explained in chapter 2 a series of functions are available to change the TEF810X settings. However all of them use SPI to write and/or read register blocks on the TEF810X chip.
In practice *chip_reg_Read()* and *chip_reg_Write()* are the only relevant

functions. Furthermore, the only way to modify the phase inversion bit in the register is to write the entire block of registers containing it as done in [12].
It was unsure if changing through SPI the value of such registers in the middle of chirp transmission would cause a phase shift.
Using the delay functions presented in the previous sections we made a simple addition in the code running on core 0 of the 32r274 chip, as visible below.

```
//T is set approx to 70us of delay (taking a margin
//for the execution of the SPI command)
STM_2_wait(t);

//Changing phase modulation
//Imposed always at 0 in middle of chirp
chip_reg_Write(TENG_MODULE_ADDRESS,
    reg_offset[(chirp-1) & 1],
    reg_val[(chirp-1) & 1] |
            (0 << bps_position[(chirp-1) & 1]),
    NULL);
```

The register offset as well as the module address have been chosen based on the registers list provided by NXP.

5.4.1. SPI results

This approach yield no result, the transmission is performed without the occurrence of a phase shift in the middle of the chirp.
In order to exclude a wrong setting in the delay functions we gradually decreased the value of the delay, until the point to have close to zero delay between the start of one chirp and the register change, we also tried to increase the value of the delay bringing it higher of 200us. None of these methods brought any change.
One possible explanation for this failure is related to the internal functioning of the TEF810X chipset.
Once one chirp acquisition is started, the phase settings are read from the registers and saved in a different location, and for the entire duration of the chirp, the used settings are kept constant. For this reason, changing the registers values in the middle of a chirp transmission brings no change to

the current transmission.
This unfortunately means that no phase change in the middle of a chirp can be accomplished through SPI communication.

5.5. Using the dedicated pin

The second option in order to achieve multiple phase shifts in a single chirp is to use a dedicated pin present on the TEF810X transceiver.
Due to a text/figure misalignment on the data-sheet of the development board utilized in this work, this pin was assumed for a long time to be left disconnected.
After reviewing multiple times the data-sheets we identified how the pin is instead connected directly to one of the GPIO pins of the S32r274 micro-controller.
To accomplish this phase inversion, the pin on the S32r274 chip is configured and controlled as output and will be triggered using the delay functions developed previously.

5.5.1. Code for pin switch

The code needed to perform the pin switch can be divided in three parts:

- The delay function already visible in the timing functions dedicated paragraph.
- A setup function for the GPIO pin to be correctly configured.
- A small addition in the frame sending function present in code 0 of the S32r274 chip.

The small addition in the code of core 0 consists of the following:

```
//Wait for new chirp
while (SPT.GBL_STATUS.B.CHRP_ACQ_DONE == 0) {   }

//t contains the required delay time
STM_2_wait(t);
//bit contains a binary value
//with 1 phase shift and 0 no phase shift
SIUL2.GPDO[NUM_PS].R = bit;
```

5.5.2. Configuration of output pin

In order to utilize a pin on the S32R274 as output we need to configure some registers in the SIUL2 module, this module is the controlling module for all general purpose IO.

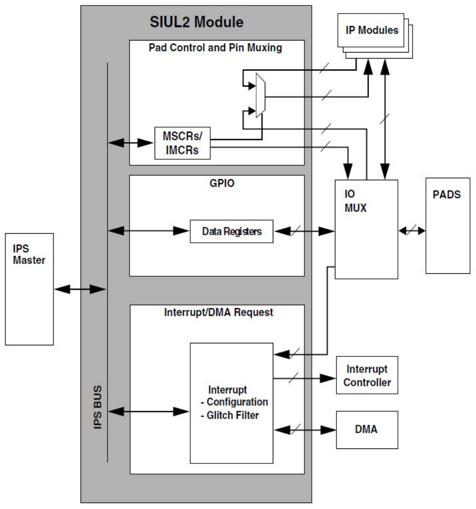


Figure 5.2: The SIUL2 module [17]

Excluding the configuration registers needed for interrupt triggering, the used configuration registers are:

- MSCRn: used to select the signal to connect to a pin as well as to enable functionality of the pads.
- IMCRn: used to select the input pad for each module. In case of a pin with only one input function the pin is directly directed to the module.

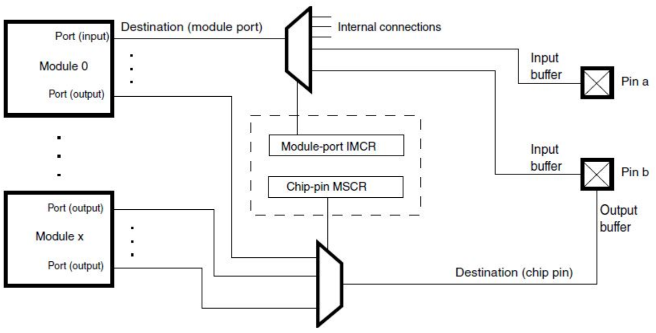


Figure 5.3: The Input Output muxing.

The used fields in the MSCRn register, with their assigned values, can be seen in table 5.1. For a more detailed description of the functionality of each register the s32R274 reference manual can be examined [14].

Field	Value	Description
SRC	0b11	Controls slew rate of output pin.
OBE	0b1	Enables the output buffer.
ODE	0b0	Enables the open drain at the pin.
SMC	0b1	Safe mode control.
APC	0b0	Analog pad control.
IBE	0b0	Enables the input buffer.
HYS	0b0	Enables the hysteresis reading the pin in digital mode.
PUS	0b0	Selects pull up or down.
PUE	0b0	Enables selected pull.
INV	0b0	Inverts pin value.
SSS	0b0000	Source signal, can be set using the IO multiplexing tables.

Table 5.1: Description of the fields in the MSCR register.

In order to change the state of the pin, the PDO field of the GPDOn register can be changed writing a single byte on it.

5.5.3. Pin switch results

With this approach we were able to switch phase in the middle of the chirp. An example with two phase inversions in each chirp can be seen in figure 5.4 on the right.

Switching overshoot

However an undesirable effect has emerged, due to hardware characteristics a large overshoot is generated when the phase is shifted.
Such overshoot can be recognized in figure 5.4 by comparing the signal without phase shift to the one with phase shift. As can be seen this overshoot has a large influence on the overall chirp duration.
The large overshoot can be seen even better in figure 5.5, where multiple chirps, some without phase coding, and others with identical phase coding are superimposed on a single graph.
This effect leads to a loss in precision in the radar functionality of the device when a large number of phase shifts are applied.

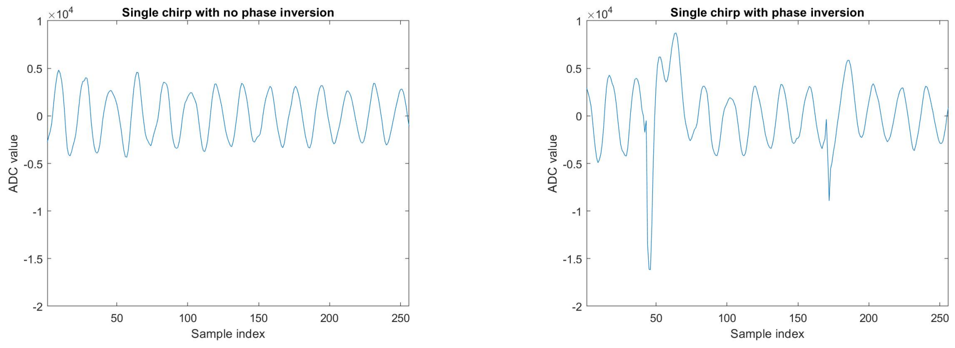


Figure 5.4: No phase shift vs phase shift with only three bits coded.

We decided to send four bytes of information in each chirp, this in order to find a compromise between information throughput and the accuracy of the radar readings.

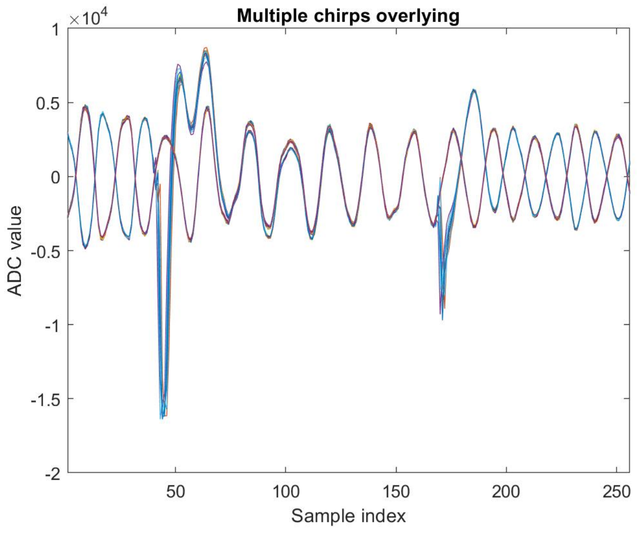


Figure 5.5: Superimposition of phase shifting and ordinary transmission.

5.5.4. Data post processing

In our experiment raw data is processed and sent to the PC through Ethernet.
All the post processing is then performed on the PC using dedicated matab scripts. This allows users unfamiliar with the firmware of the device to implement different post processing techniques.
Two aspects of the data post processing are important:

- Extraction of radar information
- Extraction of phase coded message

The extraction of the coded message from the received signal can be performed independently from the radar information processing, the two functions can be parallelized by having them performed by two different microcontrollers.

Radar information processing

To perform radar processing and extracting information such as distance and velocity of targets the receiver signal has to be reconstructed, eliminating the phase shifts that would interfere with classical radar processing algorithms.
After reconstruction common techniques can be used to obtain Doppler FFT and range FFT.
Reconstructing the signal means inverting back the phase shifted areas as well as eliminating the distorted areas caused by the overshoot in the signal.
This type of processing goes beyond the scope of this thesis, however the interested reader can find an example of such technique in [16].
This paper utilizes the radar system developed in this thesis to propose a smart filtering method to post process the radar data.

Phase coded message processing

In order to decode a chirp, the phase shifts need to be clearly identifiable, to do so a simple Matlab script has been developed.
How to extract the phase shifts for a single chirp will be explained below.
The method to extract the phase shifts can be divided in 4 steps, also identifiable in figure 5.7, such figure refers to the chirp in figure 5.6.

a) The phase angle of the received signal can be extracted in matlab by applying the *angle()* function (returning the phase angle) over the *hilbert()* function (returning the Hilbert transform).

b) Using the *unwrap()* function is then possible to correct the phase angles by keeping a maximum distance of π radians between consecutive values, obtaining a continuous function.

c) The second derivative of such signal is used to clearly identify the phase changes, exploiting the disturbance created by the phase switching.

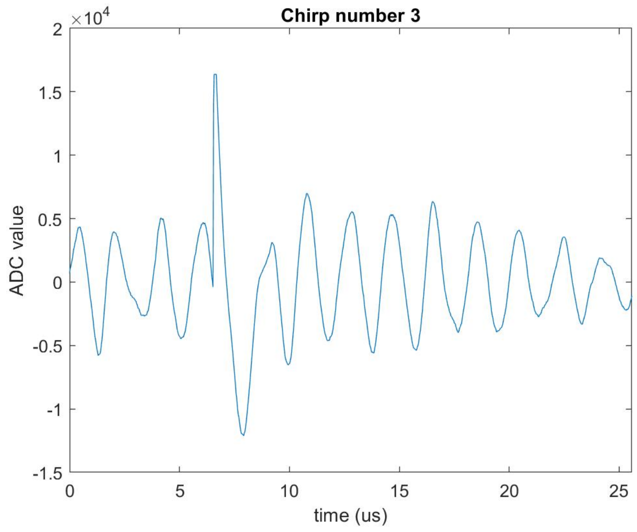


Figure 5.6: The chirp used for step by step phase inversion identification.

d) By taking the absolute value of such derivative it's possible to set a threshold for identification of the phase change.

This method allows us to extract the valid bits of information from the signals using a threshold based method.
Due to the switching overshoot we described previously, the threshold method might have to be slightly adapted to avoid detecting two phase changes in a long overshoot. However the phase changes happen always in a predefined point, and the overshoot behaviour is very regular and similar between chirps. Simplifying the process of identifying correctly a phase change.
The same result could be obtained performing a similar computation on the S32R274. Two problems arises with this approach:

- The SPT (signal processing toolbox) included in the micro-controller does not incorporate dedicated hardware to perform the Hilbert function or to extract phase angles. This would lead to high computational load.

- The limited memory on the S32R274 is now fully utilized to store ADC samples. Performing additional computation on the board would require space to store the results, limiting the number of ADC samples that can be collected.

Due to these issues we think the implementation of on board post processing should be delayed to a later stage of development, with more special-

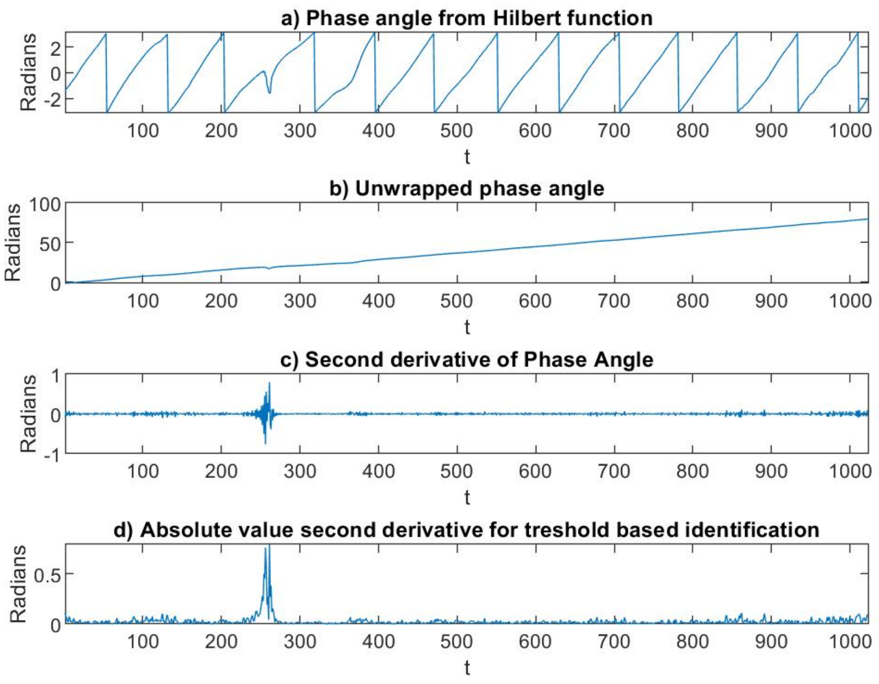


Figure 5.7: Step by step process for phase inversion identification of the single chirp in figure 5.6

ized hardware and a more defined communication protocol between the radars.

6
Multiple units synchronization

The second research requirement we dealt with focuses on the synchronization of two radar modules, in order to enable transmission of information between them. In this chapter, firstly the need for synchronization will be discussed, secondly a comparison between different synchronization solutions will be made, and finally the implementation derived from the chosen synchronization method and its results will be discussed.

6.1. The need for synchronization

The main objective in this chapter is to obtain transmission of information between two radar devices. To obtain such a result, the main obstacle is the synchronization between the two radar units.
In fact, using two synchronized radars, the information signal transmitted from one of them will be seen by the receiving radar in the same way as a scattered signal from a target. This will make possible to use the same method developer in chapter 5 to extract the transmitted information.
In the functional block diagram of the radar transceiver utilized, present in figure 2.6, it is visible how the received signal is subject to two filters right after mixing.
The filtering applied is first from a high-pass filter and then from a low-pass filter. As discussed in chapter 2.1.1 a low pass filter is required to suppress one of the components of the X_{IF} signal, while a high-pass filter is used to eliminate unwanted clutter.
Due to this high and low pass filtering, an asynchronicity in the two radar modules could lead to a filtering of the signal carrying information. In other words, if the received signal and the generated signal of the receiver board are not aligned, the resulting X_{IF} could be out of the filters boundaries.

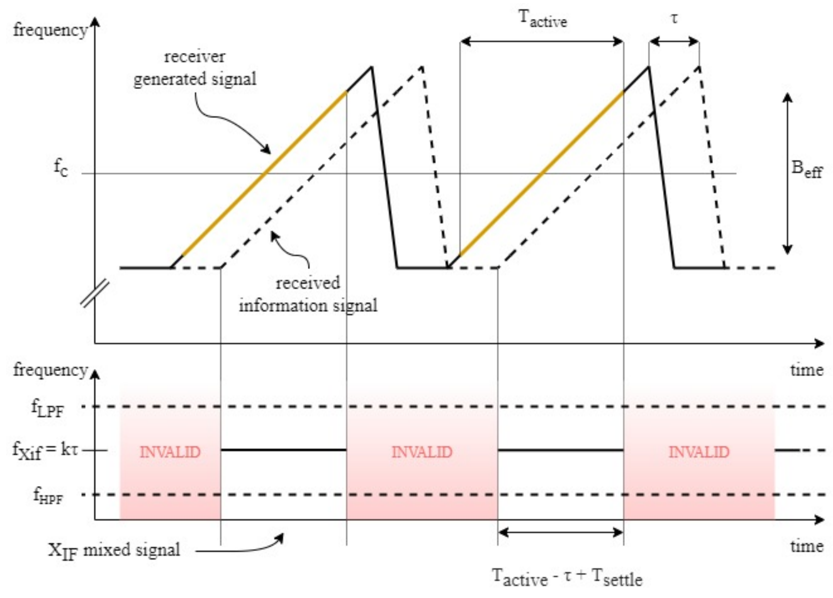


Figure 6.1: Receiver device signals for two stationary radars (f_{Xif} neglects Doppler effect).

This problem can be visualized with the aid of figure 6.1, where the signals present in the receiving device are shown.
Two graphs are present in such figure, the top one representing the two signals present on the receiving board before mixing, while the bottom one represents the mixed signal that is then used for the extraction of information.
The *receiver generated signal* is the signal generated on the receiver board and is then mixed directly with the *received information signal* that as the name suggests carries the information to be extracted. The *received information signal* can be treated similarly to a scattered signal from a target, allowing us to call the resulted mixed signal X_{IF} and to use the mathematical formulations derived in chapter 2.1.1.
The main differences with a normal scattered radar signal are two, first the distance travelled by the radar wave will be half the one in a normal scattering scenario, and second τ will now incorporate a delay caused by the synchronization error between the radars.

$$\tau = \tau_{distance} + \tau_{sync}$$

A good synchronization, is equivalent with assuming a small τ. A small τ means:

- The resulting shift in frequency of the useful part of the mixed signal will be relatively small. Meaning the value of $f_{Xif} = S\tau$ will remain in the margins of the low pass filter.
- The useful part of the mixed signal will cover the majority of the chirp. In other words, the smaller τ is, the smaller the invalid region of X_{IF} will be.

In the case of two radar devices with bad synchronization, τ will grow and the two conditions discussed above won't be valid anymore.

- The resulting shift in frequency of the useful part of the mixed signal will become large. In other words, the value of $f_{Xif} = S\tau$ will be large and the signal could be filtered out by the low-pass filter of the radar transceiver.
- The invalid region of the signal X_{IF} could become much larger, making the useful part of X_{IF} too small to be evaluated.

For these reasons, synchronization between the two radar devices is a prerequisite to obtain transmission of information.

Our radar module in the current configuration sends a chirp with a T_{active} of $25.6us$ and a T_{settle} of $5us$. The cut off frequency of the low-pass filter is set to $30MHz$.
This values can be used as reference to estimate the maximum allowed synchronization error.
Two limits can be considered:

- The value of τ should not be larger than $T_{settle} = 5us$ in order to preserve the integrity of the signal during the whole duration of T_{active}. This translates to $\tau \leq 5us$.
  If $\tau > T_{settle}$ then the sampling in the receiver will start before the starting of the slope in the received chirp, meaning that the first part of X_{IF} will be corrupted.
- The value of the frequency f_{Xif} should not exceed the low-pass filter frequency. Meaning that $f_{Xif} \leq 30MHz$

Both these conditions must be true in order to extract such signal correctly, and both need to be translated to be expressed with relation to τ_{sync}.
For the first condition, we know the relation $\tau = \tau_{distance} + \tau_{sync} \leq T_{settle}$, from which follows:

$$\tau_{sync} \leq T_{settle} - R/c \leq 4.93us \tag{6.1}$$

Where $R = 20m$ is the distance between the two radars, and c expressed in $[m/s]$ is the speed of light.
For the second condition, using the formulation present in chapter 2.1.1, and for two radars at a distance of $R = 20m$ follows that:

$$f_{Xif} = S\tau = S(R/c + \tau_{sync}) \leq 30MHz \tag{6.2}$$

Where $S = B_{eff}/T_{active}$, $B_{eff} = 1500MHz$ is the effective bandwidth of the radar, and $c = 299792458m/s$ is the speed of light.
Inverting this inequality we find that $\tau_{sync} \leq 445ns$. This value was extrapolated using a $R = 20m$, a realistic value for real world applications in the automotive industry as well as for laboratory conditions.
This will be the lower admitted synchronization limit in order to allow communication in our system.

6.2. Different ways to synchronize

To achieve synchronization we evaluated two different approaches:

- Synchronizing using an Ethernet wired connection.
- Synchronizing utilizing a couple of GPS receivers.

The pros and cons of both options will be discussed below.

6.2.1. Ethernet synchronization

The simplest option for synchronization is to use the Ethernet port present on the Radar Board to connect the two radars together.
By creating a server/client application on the two boards this would allow to send a synchronization UDP or TCP message.
This method however presents two main problems:

- The Ethernet ports are already used to send back raw ADC samples from the Radar Board to the PC, this creates traffic on the Ethernet channel, causing additional delay. Furthermore, at least one switch would be needed in this scenario, adding latency to the connection.
- Due to the wired nature of the Ethernet network, synchronization could only be used as a proof of concept. To make the technology proposed in this thesis useful in real world applications a wireless synchronization method will have to be developed.

6.2.2. GPS synchronization

GPS synchronization is the second evaluated solution for synchronization. As discussed in chapter 2.3, a cheap GPS module can provide a synchronized PPS (pulse per second) signal with a precision of $60ns$. This value is much lower than the required limits calculated chapter 6.1, leading to the possibility of providing synchronization through GPS.
Furthermore, GPS technology is of wireless nature, making it possible to utilize our system in real world applications such as in the automotive sector.

PPS precision verification

In order to verify the claims of high performances provided by cheap GPS devices present on the market, we bought two devices. Two small PCBs with mounted a NEO-7M GNSS module from U-Blox. The NEO-7M chip claims in 99% of time an accuracy on the pulse per second (referred in the datasheet as time pulse signal) of 60ns.[23]
They include a small PCB SMD antenna that was found inadequate due to the indoor environment of our laboratories. For this reason two external antennas have been used, the two GPS specific antennas utilized are however not from the same manufacturer.
The PPS signal is directly available on one of the pins of the GPS boards. In order to confirm the provided specifications we used a digital oscilloscope, powering up both radars at the same time and checking for the differences in the PPS signal. One reading from the oscilloscope can be observed in figure 6.2. From this tests we deducted how the two signals have the same

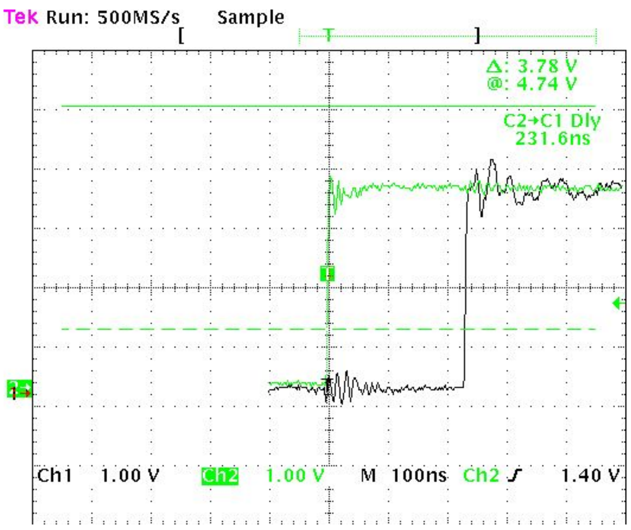


Figure 6.2: Oscilloscope reading of the two PPS signals (time difference $231.6ns$).

frequency but their phase difference is higher than the one specified in the datasheet.
This phase difference oscillates between 200ns and 500ns, values that are still in the boundaries set in chapter 6.1. As a consequence, the system can be used to obtain synchronization.

6.2.3. Synchronization method chosen

After the evaluation of both synchronization methods our choice fell for utilizing GPS technology. This mainly due to the wireless nature of GPS technology. Having a wired connection between the two radars would be unrealistic in a real world application of our system.

6.3. Synchronization implementation

After evaluating the two options for GPS synchronization we focused on building a test setup and a software implementation capable to achieve synchronization.
In this section we will describe the experimental setup, firstly in the hardware part, secondly in the software.

6.3.1. Hardware

The experimental setup utilized from this chapter onward has been explained in chapter 4. Here we will go slightly more in depth in the hardware description of this setup.
We can identify four main types of devices used:

- The two radar devices, referred from NXP as *DCC development kit*.
- The two GPS modules, with each a dedicated antenna.
- Two 3.3V power supply units, one for each GPS module
- Two PCs with the duty of collecting the data from the receiver as well as serving as user input for the radars.

Both the GPS modules have to be powered at 3.3V in order to be compatible as inputs to the radar boards. For this reason the two separate 3.3V power supplies are needed to power the GPS devices.
The radar modules have been positioned on a table with the antennas facing each other. Each one of them is connected to one GPS module.
The distance between the radar devices can be adjusted in order to perform measurements with different distances.

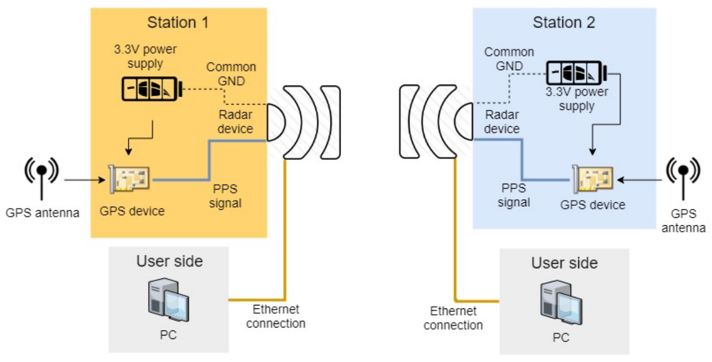


Figure 6.3: Diagram of the test setup

In figure 6.3 a diagram comprehensive of all the elements of our system is present.
As can be seen some connections are necessary between components:

- The radar device has one of his GPIO pins connected to the GPS device PPS pin.
- The 3.3V power supply provides current to the GPS module as well as is sharing the ground with the Radar Device.
  This is necessary because the radar device needs to read an input from the GPS board, without sharing a common ground the radar would have no reference to read the signal.
- The radar device has been connected to the PC through Ethernet.

In order to validate the functioning of the software, two additional GPIO pins of the radar device have been used to connect an oscilloscope during testing.

6.3.2. Software

In order to implement synchronization, the firmware of the radar device, running on the S32R274 has been modified.
In this new implementation all three cores have been utilized, compared to the two utilized in [12]. Each core covers a specific duty:

- Core 2: Takes care of detecting the input PPS signal from the GPS device and then raises one interrupt each time a chirp should be sent.
- Core 1: Takes care of the communication with the PC.
- Core 0: Takes care of triggering a frame acquisition as well as coding the information to be sent trough phase inversion.

Furthermore, the implementation developed for our purpose relies on interrupt routines in order to minimize the delay between events.
To facilitate post processing, the frequency of acquisition has been increased from $10Msps$ to $40Msps$ and the samples per chirp has been subsequently increased to 1024 compared to the setup used in chapter 5.
The functioning of each core will be described more in depth below.

Core 2 functions

This core has two main duties:

- Reading the pin connected with the PPS port of the GPS module.
- After the rising edge of each PPS signal, rise periodically an interrupt to core 0 to trigger the transmission of a frame. The period of this interrupt can be defined in the firmware, with a maximum value of one second.

Core 2 input pin configuration

The first duty will be satisfied by setting the input pin to trigger an interrupt function. This can be accomplished by setting correctly the registers of the SIUL2 module.
The MSCRn register will have to be configured in a slightly different way compared to how described in chapter 5 due to the pin being used as input and not as output. The different registers are:

Field	Value	Description
OBE	0b0	Enables the output buffer.
IBE	0b1	Enables the input buffer.
PUS	0b0	Selects pull up or down.
PUE	0b0	Enables selected pull.
SSS	0b0000	Source signal, can be set using the IO multiplexing tables.

Table 6.1: Configuration of MSCR register for input.

Different registers in the SIUL2 module are dedicated to configure the interrupt behaviour.

- DISR0: contains the interrupts flags.
- DIRER0: interrupt enable register.

- DIRSR0: selects between interrupt and DMA access.
- IREER0: rising edge event enable.
- IFEER0: falling edge event enable.
- IFER0: enables the pad glitch filter for the interrupt.
- IFMCRn: set the glitch filter period.
- IFCPR: prescale the clock for the glitch filter.

These registers have all been configured in order to trigger the interrupt.
The microcontroller is set to handle interrupts in Software vector mode. In this mode, an area of memory is allocated to contain the pointers to the triggered functions by each interrupt source.
One additional register has to be configured in order to trigger the interrupt function. It is part of the Interrupt Controller module.

- PSR[x] register field PRC_SELNn: setting this field will forward the interrupt to core n.
- PSR[x] register field PRIN: this field controls the priority of this interrupt. The interrupt is masked if the priority is 0.

The x value refers to the vector number that is connected to the interrupt source, in our case the interrupt source is the GPIO pin.
The registers to trigger an interrupt from the input pin require a configuration in a certain order. The order they are listed here is the order in which they are configured.
It is important to notice how the PSR register is set by default to enable core 0. If core 0 does not have to be enabled, the PRC_SELN0 field has to be set to 0 manually.
After this configuration, the pin triggers an interrupt in core 2. To assign a function to this interrupt, the interrupt vector table of the core has to be modified, with the addition of the wanted function in the correct vector slot. The function triggered in our case has been named *input_detected()*.

Register	Field	Value	Description
DIRER0	-	0x00	Masks interrupts for all GPIO pins, disabling them.
IREER0	IREE3	0b1	Enables rising edge triggered events for a single pad.
IFEER0	IFEE3	0b0	Disables falling edge triggered events for a single pad.
IFER0	IFE3	0b1	Enables filtering on a single pad.
IFMCR[n]	MAXCNT	0b1111	Set pad filtering period as maximum for pad n, this in order to have the best precision during the detection of an edge.
IFCPR	IFCP	0b1111	Pre-scale the filter clock as high as possible in order to obtain better filtering.
MSCR	-	-	The configuration of this register can be found in table above.
DIRSR0	DIRSR3	0b0	Selects interrupt functionality for a specific pad.
DISR0	-	0xFFFFFFFF	Resets interrupt flags.
DIRER0	EIRE3	0b1	Enable interrupt for the required pad.
PSRx	PRC_SELNn	0b1	Enable interrupt source x for core n.
PSRx	PRIN	0 to 31	Sets the priority of the x source of interrupts.

Table 6.2: Required settings to trigger an interrupt on a GPIO pad.

Core 2 triggering of interrupt for core 0

The approach taken to trigger the frame acquisition in core 0 will be based on the knowledge that more time between two frames is required than the frame transmission time. This due to the overhead from the UDP transmission of raw adc samples. The UDP transmission is in the order of hundreds of ms, a larger value compared to the time required for a single frame to be sent.

For simplicity, we started sending only two frames in a single second. One at rising edge of the PPS signal and one approximately 500ms after.

The function *input_detected()*, triggered each time a rising edge is de-

tected from the PPS signal, first raises an interrupt for core 0, second takes care of initializing one of the timer modules present in the S32R274 microcontroller.
This timer module, once the compare value set for 500ms matches, will subsequently raise a second interrupt for core 0.

Core 1 functions

The function of core 1 remains the same as in the corrected code from [12]. It takes care of sending the radar raw adc values through UDP packages once a radar frame acquisition is complete.
It also takes care of receiving configuration messages as well as the information that the user wants embedded in the transmission.

Core 0 functions

Core 0 functions remain similar to what presented in chapter 5 on the multiple phase shifts in a single chirp.
The main difference is that in this implementation each frame transmission is triggered through interrupts.
This has been achieved with a similar configuration as the one used in core 2. The function *radar_frame_scan()* encapsulates a complete frame scan and is now executed only if triggered by vector #1 in the Software mode interrupt signal vector table.
The additions to the isr vector table can be seen in the code below.

```
...
extern void radar_frame_scan(void);
...

const uint32_t __attribute__ ((aligned(4096)))
    __attribute__ ((section(".isr_vector_table")))
    IntcIsrVectorTable[] = {

/* Vector # 0 Software settable flag 0*/
(uint32_t) &dummy,
/* Vector # 1 Software settable flag 1*/
(uint32_t) &radar_frame_scan,
/* Vector # 2 Software settable flag 2*/
(uint32_t) &dummy,
...
...
}
```

With this addition, core 0 will send a frame only and only if core 2 will trigger it.

Differences between transmitter and receiver

To simplify the post processing in this first phase two different configurations have been uploaded on transmitter and receiving radar.
Two main differences are visible between the two devices:

- While the transmitting device is set to code a signal in the transmitting waveform, the receiving device will not code any information and his transmitted waveform will be the same as a normal FMCW radar.
- The transmitting antenna gain of the receiving device will be set to a low value 1 over a 256 interval. This to avoid the difficulty of distinguishing the origin of the scattered waves.

6.4. Error propagation in the synchronization process

Different sources of error that may lead to a loss of synchronization are present in our system.
The sources of error in a single radar setup can be visualized in figure 6.4.
In such figure the red blocks represent the larger sources of error, while the yellow blocks introduce less error.
The possible sources of error are:

- The GPS receiver, in our case the UBLOX NEO-7N. This block incorporates the error introduced by the different propagation paths of the GPS signal, as well as the uncertainty introduced by the chip itself.
  The maximum difference between two modules should be of $60ns$, however as discussed in chapter 6.2.2 this difference rises much higher than $200ns$.
- The crystal oscillator, in our case the device used in the radar boards is a single NX3225SA from *NIHON DEMPA KOGYO CO., LTD.*.
  This crystal with a nominal frequency of $40MHz$ provides a frequency tollerance of $\pm 15ppm$.

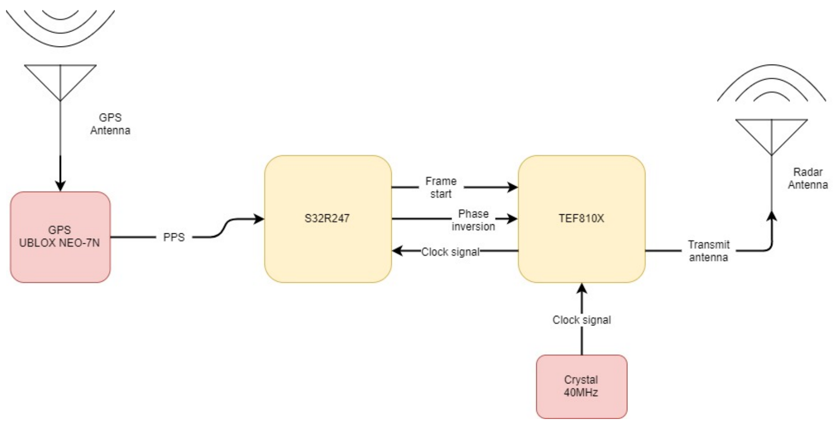


Figure 6.4: Block diagram of one radar setup.

- The S32R247 micro-controller introduces two types of uncertainty. A hardware related uncertainty, which we will consider as neglectable, due to the complexity of evaluating the differences in the silicon manufacturing process. A software related uncertainty, which we minimized by making the developed code as linear as possible, avoiding loops and branching unless necessary.
- The TEF810X radar transceiver acts on two different sources of error.
  - It buffers the clock signal from the crystal oscillator to the S32R247 micro-controller.
  - It receives the frame start signal and start the transmission of the frame. It furthermore triggers the phase inversion during each chirp.

  Both these errors are related to the silicon manufacturing processes of the transceiver, and while is important to know of their existence their study goes beyond the purpose of this work.

6.5. Synchronization results

6.5.1. The radars clock slack problem

From the first runs with the synchronization setup described above we obtained inconsistent results. Of the two transmitted frames in a single second, only the first one was received correctly.
We confirmed this by disabling the second frame in a single second and the results showed consistency.
We investigated the problem and observed how the two radar modules

loose the synchronization precision achieved at the beginning of the second quickly.
In other words, while the time difference between the two PPS signals appears to be sufficient to synchronize the two radars, there appears to be a difference in the internal timing of the two devices. One of the two radars slacks behind the other, leading to a excessive desynchronization.

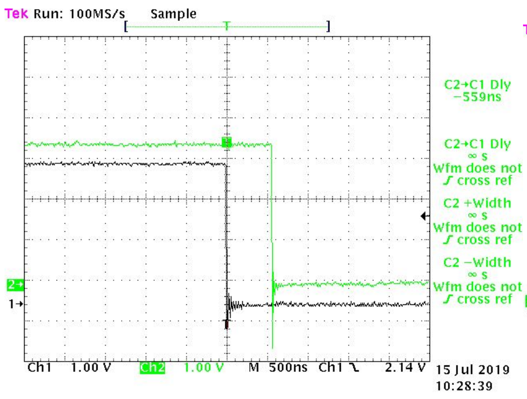


Figure 6.5: Desync at beginning of PPS: $550ns$.

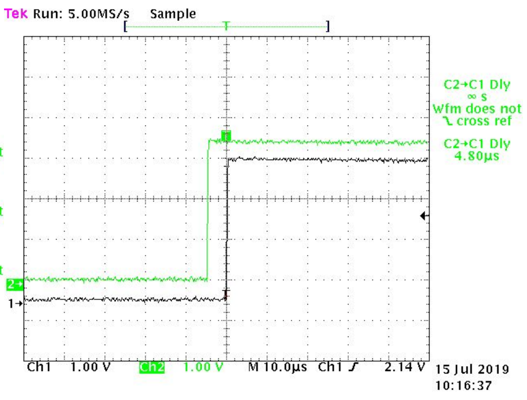


Figure 6.6: Desync at end of PPS: $5us$.

To better study this behaviour, we created a simple code that would run on an isolated core of the radar devices and change the status of the two pins right after a certain delay from the rising edge of the PPS.
We then used an oscilloscope to read such outputs, the results can be seen in figures 6.5 and 6.6. Figure 6.5 shows the synchronization difference shortly after the PPS edge, while 6.6 shows the same after more than $0.8s$ passed from the PPS edge.
There is a significant difference between the two systems synchronicity. At the beginning of the PPS a barely acceptable delay of $550ns$ is present, while at the end of the PPS we found a delay of $5us$. Meaning that the useful signal will be filtered out and consequently the reception of any data will be impossible.
The increase of 10x in the misalignment of the two signals is independent from the GPS system and indicates a problem residing in the functioning of the radar device.
We supposed two reasons for this behaviour can be identified, namely:

- A difference based on the software running on the two S32R274 devices.

- A difference in the delay introduced by the hardware timing module in the two devices.

The software on the radar device has been designed in order to minimize the time uncertainties. This has been achieved by utilizing hardware interrupts on the input pin as well as the hardware timing module of the S32R247 micro-controller to obtain a constant delay.
We verified how the root of our issue does not resides in the software by observing how the slack between the two radars is proportional to the time elapsed since the PPS edge. Since the delay is generated by the hardware timing module, this excludes a software issue.
We were then able to identify the source of the problem in the delay generated by the hardware timing module. This delay is generated using a counter on the clock signal, and for this reason the main suspect for this mismatch is a clock signal difference in the two devices. The three hardware components involved in the clock path to the timing module are the TEF810X transceiver, the S32R247 micro-controller, and the external crystal oscillator used to power the board.

Error introduced by the TEF810X transceiver

The The TEF810X transceiver simply buffers the clock signal 1 to 1 from the crystal oscillator to the S32R247. We will consider as neglectable the uncertainty introduced by this component.

Error introduced by the S32R247

The clock utilized by the S32R247 micro-controller timing modules is a $60MHz$ clock signal. This signal has been generated by first scaling up of a factor 6 the external clock from $40MHz$ to $240MHz$. Secondly the $240MHz$ clock has been divided by a factor 4 to obtain the wanted $60MHz$ signal.
If the external clock contains one uncertainty of δX, the clock powering the timing modules will contain the uncertainty:

$$\delta Y = K_{S32R247}\delta X = \frac{6}{4}\delta X \tag{6.3}$$

Our analysis does not take into account the possible differences in the multipliers and dividers of the S32R247 micro-controller, these modules could bring additional uncertainties.

Error introduced by the Crystal

Manufacturers list their crystal oscillators with maximum values for a series of parameters. In this specific case, the frequency tolerance is the one we

should pay close attention to.
The frequency tolerance is defined as: *The allowable frequency deviation plus and minus the specified crystal frequency. It is specified in parts per million (PPM) at a specific temperature, usually +25 degrees C.*[2]
In other words, the frequency tolerance indicates the maximum allowed difference in frequency of a crystal from the nominal value. The stability over time and temperature of the effective frequency of the crystal is represented by other values, and is often much better than the frequency tolerance.
As stated previously, the crystal used in the radar boards is the NX3225SA from *NIHON DEMPA KOGYO CO., LTD.*, this devices is rated for a frequency tolerance of $\pm 15ppm$.

Error propagation analysis

In order to understand if the crystal oscillator frequency tolerance and the uncertainty added in the S32R247 chip are the cause of the different behaviour of the timing modules, an analysis on the error propagation should be done.
The frequency tolerance of the crystal used in the radar boards is $\delta X = \pm 15ppm$ meaning that in a crystal of frequency $1MHz$ the possible deviation would be of $15Hz$. In the case of a $40MHz$ clock signal, the possible deviation would be of $\pm 15 * 40 = \pm 600Hz$. By taking the worst possible difference between two crystals, we obtain a possible mismatch of $1200Hz$.
The maximum frequency variation can be obtained with the generic equation:

$$\Delta f_{crystal} = 2 f_{crystal} \delta X = 1200Hz \tag{6.4}$$

Where $f_{crystal}$ is the frequency of the clock signal generated by the crystal, δX is the frequency tolerance of the signal and the factor 2 takes into account how we want to find the maximum distance between two crystals.

The clock signal will then travel through the TEF810X chip without any alteration of Δf. After being buffered by the TEF810X chip, the clock will be scaled and divided in the S32R247 chip, leading to and uncertainty of:

$$\Delta f_{S32R247} = K_{S32R247} \Delta f_{crystal} = 1800Hz \tag{6.5}$$

The timing module will then use a counter on the clock to produce the required delay. We can estimate how much the maximum difference between two timing modules powered by two different crystals will be using the frequency tolerance.

To translate a frequency mismatch to a time mismatch over a defined time interval, we can define the two radars as one with frequency $f_1 = f_{S32Clock} - \Delta f_{S32R247}$ and the other with frequency $f_2 = f_{S32Clock}$.
In an arbitrary interval of $t_1 = 0.7s$, the first radar would run $n = t_1 * f_1$ clock cycles. The same number of clock cycles would be obtained by the second radar in a time $t_2 = n/f_2 = t_1 * f_1/f_2$.
The resulting skew will be:

$$\Delta t = t_1 - t_2 = t_1 - t_1\frac{f_1}{f_2} = t_1 - t_1\frac{f_{S32Clock} - \Delta f_{S32R247}}{f_{S32Clock}} \tag{6.6}$$

$$\Delta t = t_1 - t_1\frac{f_{S32Clock} - 2k_{S32R274}f_{crystal}\delta X}{f_{S32Clock}} \tag{6.7}$$

Since $f_{S32Clock} = k_{S32R274}f_{crystal}$ we can simplify equation 6.7 to:

$$\Delta t = t_1 - t_1\frac{1 - 2\delta X}{1} \tag{6.8}$$

$$\Delta t = 2t_1\delta X \tag{6.9}$$

And finally substituting the values obtained in our case:

$$\Delta t = 2 * 0.7 * 15 * 10^{-6} \approx 21us \tag{6.10}$$

This value represents the worst case mismatch between the two devices after $t_1 = 0.7s$. In our case, the mismatch found experimentally and visible in figure 6.6 is well between the bounds we just computed.

Possible solutions

The cause of the synchronization problem has been identified above in a possible mismatch between the crystals that leads the timing modules of the S32R247 chip to generate different delays.
We also obtained a general equation for our system:

$$\Delta t = 2t_1\delta X \tag{6.11}$$

Where δX is the frequency tolerance of the crystal, and t_1 is the overall time we evaluated the delay on.
From this general equation, we can understand how the Δt can be reduced by following two approaches:

- Reducing the time interval evaluated. This would require a more frequent synchronization signal and would lead to a lower Δt.

- Installing a more precise oscillator. The lower value for the frequency tolerance for commercially available devices is around $\pm 0.5PPS$. One example is the *XTCLH40M000CHJA0P0*, 40MHz oscillator.

Improving these two points could lead to a much more reliable synchronization and to a solution of the radar clock slack problem. Using a crystal with frequency tolerance of $0.25pps$ in our system would lead to a maximum slack of $500ns$, a close value to the boundary condition for synchronization calculated in chapter 6.1.

6.5.2. Collected data analysis

Unfortunately in our case we could not modify the radar hardware so drastically to circumvent the radar clock slack problem. We had to limit the number of frames transmitted in a single second to one. This lead to a large decrease in throughput and a loss of precision in radar sensing.
Despite the issues with the on board radar clock, lowering the frames frequency resulted in reliable detection of each frame.
The resulting superimposition of all the received chirps in a single frame that was received correctly can be observed in figure 6.7. The three peaks that can be seen in such figure correspond to the overshoot of the phase shifts in the transmitted signal.

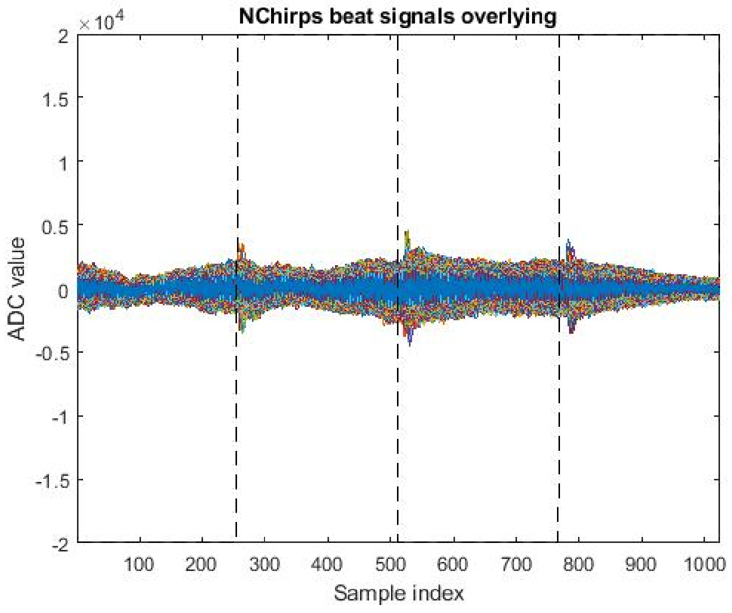


Figure 6.7: Received signal from receiving radar.

Decoding of transmitted message

In order to extract the information present in the received chirps, the same post processing described in chapter 5.5.4 can be performed.
A single chirp containing 4 bits coded can be observed in figure 6.8, the coded information bits are 0101 as can be observed by the phase changes in between the different bites.
Figure 6.9 shows the result of the decoding process, a peak is created for each phase change which can be extracted by a simple threshold. This under the assumption that the signal-to-noise ratio (SNR) available for communication is high.

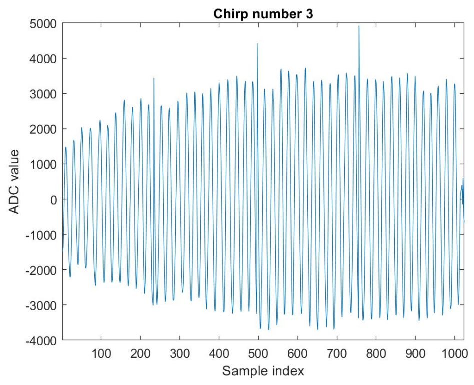


Figure 6.8: Single chirp before derivative

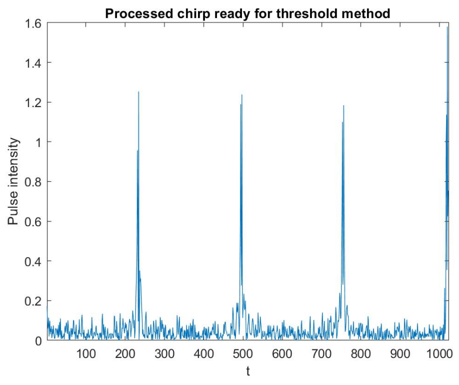


Figure 6.9: Coded values for a single chirp, containing the sequence 0101

One important drawback of such decoding system is that while detection of phase shifts results is quite simple, the detection of the initial phase compared to the previous chirp phase is difficult.
This means that in each chirp, the first bit of information has to be always set to a known value, limiting the useful transmitted bits of information to 3 each chirp.
We leave to future work the development of a detection system for the initial phase of consecutive chirps. This would bring back the usefulness of the first bit of information, increasing the overall throughput.

Processing of radar signal

To process the radar signal and obtain the distance and radial velocity of targets, first the phase shifts have to be eliminated from the X_{if} signal.
As stated in chapter 5.5.4, this process goes beyond the scope of this thesis and the interested reader can find an example in [16].
Once the data phase shifts have been corrected, common algorithms can

be applied to obtain the targets information.

7
User interface

We aim to make the work developed in this thesis useful for researchers with minimum overhead from their side. The initial implementation discussed in chapter 3 required changes in the firmware of the radar devices for selecting parameters as duration of chirps, number of chirps and many others. In order to overcome this problem, a small user interface has been developed, providing users with ease of use in setting up the radars.

7.1. Functionalities required

In order to provide control over the radar functionality, the user must be able to modify a series of parameters.
We will divide these parameters in three classes:

- Parameters of the radar requiring no reconfiguration of any hardware module.
- Parameters requiring a reconfiguration of the TEF810X transceiver.
- Parameters requiring a reconfiguration of the TEF810X transceiver plus a reconfiguration of multiple hardware modules in the S32R274.

As will be explained further in this chapter, of these three classes the first is the simpler one to implement, while the last the harder.
Furthermore, during the development of this thesis, multiple changes have been performed on the radar device, this user interface will focus on the last iteration of our firmware.

7.1.1. Parameters not requiring a TEF810X reset

These parameters can theoretically be changed in between chirps without any message to be sent to the $TEF810X$ transceiver.
Only the following five parameters can be changed in this way. They have all been created by us during the development of this thesis.

- Continuous or GPS frame transmission. This parameter allows to select how the frame transmission should be triggered. Two modes are available:
  - GPS triggered transmission. The transmission of a frame starts each time a PPS rising edge is detected.
  - Continuous frame transmission. Frames are sent one after each other, independently from the GPS.
- Coding mode: This parameter selects the coding mode, three options are available.
  - No phase coding. The radar will not code any data in the chirps, it will then send the same chirps of a normal FMCW radar.
  - Alternating phase coding. The radar will alternate between phases, changing phase three times each chirp.
  - Coded string or random number. The radar will code a string or random number in each frame.
    After this setting has been selected, the radar will wait for a random number or string through TCP and only after it has been received it will start coding.
- First phase change delay: This parameter will select the delay from the beginning of a chirp to the first phase shift. The resolution of this value is $1/60us$.
- Second phase change delay: This parameter will select the delay from the first phase shift to the second phase shift in each chirp. The resolution of this value is $1/60us$.
- Third phase change delay: This parameter will select the delay from the second phase shift to the third phase shift in each chirp. The resolution of this value is $1/60us$.

7.1.2. Parameters requiring a reconfiguration of the TEF810X transceiver

This class of parameters requires a reconfiguration of the $TEF810X$ transceiver. However, it does not require any hardware module of the $S32R274$ micro-controller to be reconfigured.
In other words, the firmware changes required for these parameters do not require us to modify the client or server programs running on the user side, and do not require complex reconfiguration procedures to be performed on the micro-controller.
These parameters are:

- ADC module sampling rate.
- Chirp centre frequency.
- Effective Bandwidth of chirp.
- T_{dwell}, T_{settle}, $T_{jumpback}$, T_{reset} as defined in chapter 2.
- Low pass filter cut-off frequency.
- High pass filter cut-off frequency.
- Transmission antenna gain.
- Receiver antenna gain.

For information on the supported values of these parameters we refer the reader to the reference manual of the $TEF810X$ transceiver.

7.1.3. Parameters requiring reconfiguration of hardware module in S32R274

The third and last class of parameters requires a much more complex procedure to provide run time changes.
Apart from the $TEF810X$ transceiver, three additional modules have to be reconfigured, the MIPI module, the SPT module and the FNET stack.
The MIPI module provides a high speed communication channel between the $S32R274$ micro-controller and the $TEF810X$ transceiver. This channel is used only for the transmission of the sampled ADC values.
The SPT module is the Signal Processing Toolbox included in the radar processing platform of the $S32R274$ micro-controller, it takes care of performing post-processing on the data collected. It also decides in which position and fashion to store the received samples.

The FNET Stack is a TCP/IPv4 Stack, it manages the communication between radar device and PC, for both TCP and UDP packets. This Stack has been modified by NXP to provide initial functionality.
The parameters which will require these modules to be altered are two:

- Number of samples per chirp
- Number of chirps each frame

The reason these two parameters require additional changes are related to two aspects: memory occupation and UDP packet structure.
Two independent modules of the $S32R274$ need to access the same data:

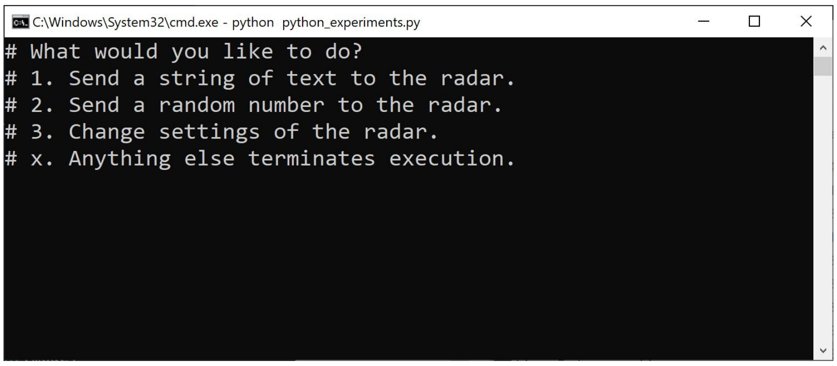
- The Radar Processing Platform, stores the received ADC samples according to dimension of frame and number of samples.
- Core 1 extracts the samples from memory, estimating the location from the number of samples and frame size. It then divides them in appropriate sized packets to be sent.

Communication between these two modules is essential and was not supported in the firmware developed in the previous chapters.

7.2. User interface design

The user interface design started from the easier parameters, scaling up of difficulty with the progression of the work.
We started the development by choosing a programming language. Our choice fell on python for the ease of implementation as well as for the portability of such language.
We then implemented a simple client program to substitute the one described in chapter 3. This client is a command line based user interface and allows us to modify the setting of the radar with an intuitive series of multiple choice menus.
The starting screen can be seen in figure 7.1. From the beginning three options are available:

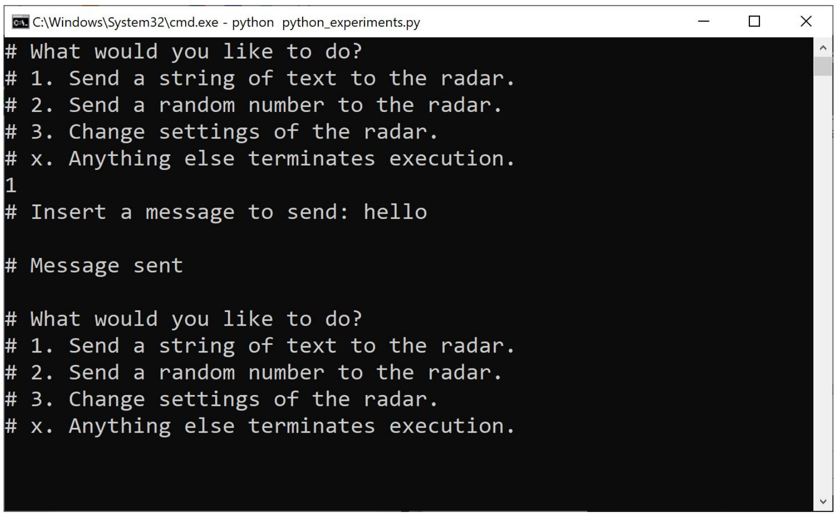
- Sending a string to the radar to be coded and then transmitted.
- Sending a random number to the radar to be coded and transmitted. This random number is also saved locally in a binary format.
- Changing the settings of the radar device.



```
C:\Windows\System32\cmd.exe - python  python_experiments.py
# What would you like to do?
# 1. Send a string of text to the radar.
# 2. Send a random number to the radar.
# 3. Change settings of the radar.
# x. Anything else terminates execution.
```

Figure 7.1: Starting screen for User Interface

From this menu, the first two options allow the user to send information to the radar to be encoded. Such functionality can be seen in figures 7.3 and 7.2.
Once the user inserts a string or selects the random number, the program will restart, going back to the initial screen.



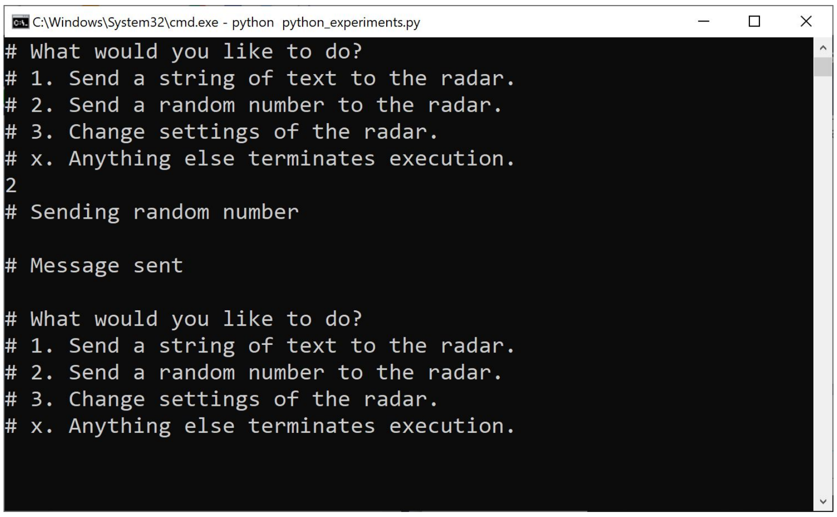
```
C:\Windows\System32\cmd.exe - python  python_experiments.py
# What would you like to do?
# 1. Send a string of text to the radar.
# 2. Send a random number to the radar.
# 3. Change settings of the radar.
# x. Anything else terminates execution.
1
# Insert a message to send: hello

# Message sent

# What would you like to do?
# 1. Send a string of text to the radar.
# 2. Send a random number to the radar.
# 3. Change settings of the radar.
# x. Anything else terminates execution.
```

Figure 7.2: User procedure to send a string.

The third option allows the user to change the settings of the radar. Selecting this option will bring to a separate menu, where the user will be able to insert the desired parameters.
The current list of modifiable parameters can be seen in figure 7.4.

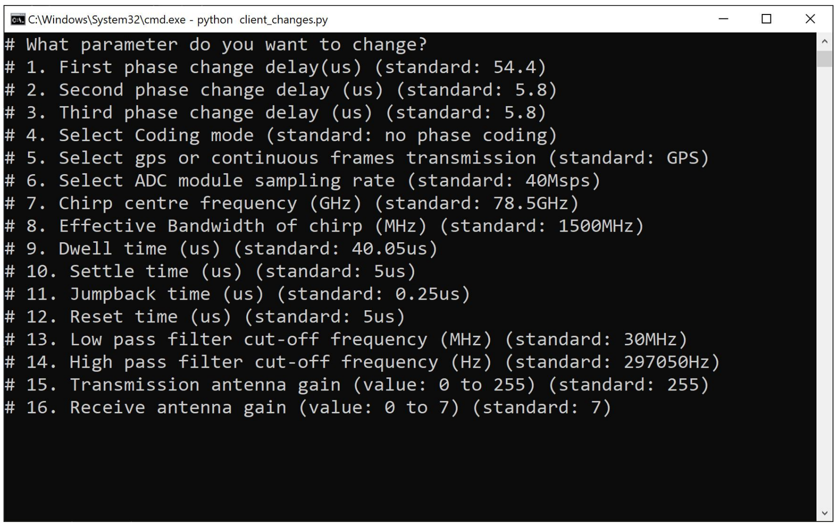


```
C:\Windows\System32\cmd.exe - python  python_experiments.py
# What would you like to do?
# 1. Send a string of text to the radar.
# 2. Send a random number to the radar.
# 3. Change settings of the radar.
# x. Anything else terminates execution.
2
# Sending random number

# Message sent

# What would you like to do?
# 1. Send a string of text to the radar.
# 2. Send a random number to the radar.
# 3. Change settings of the radar.
# x. Anything else terminates execution.
```

Figure 7.3: User procedure to send a random number.



```
C:\Windows\System32\cmd.exe - python  client_changes.py
# What parameter do you want to change?
# 1. First phase change delay(us) (standard: 54.4)
# 2. Second phase change delay (us) (standard: 5.8)
# 3. Third phase change delay (us) (standard: 5.8)
# 4. Select Coding mode (standard: no phase coding)
# 5. Select gps or continuous frames transmission (standard: GPS)
# 6. Select ADC module sampling rate (standard: 40Msps)
# 7. Chirp centre frequency (GHz) (standard: 78.5GHz)
# 8. Effective Bandwidth of chirp (MHz) (standard: 1500MHz)
# 9. Dwell time (us) (standard: 40.05us)
# 10. Settle time (us) (standard: 5us)
# 11. Jumpback time (us) (standard: 0.25us)
# 12. Reset time (us) (standard: 5us)
# 13. Low pass filter cut-off frequency (MHz) (standard: 30MHz)
# 14. High pass filter cut-off frequency (Hz) (standard: 297050Hz)
# 15. Transmission antenna gain (value: 0 to 255) (standard: 255)
# 16. Receive antenna gain (value: 0 to 7) (standard: 7)
```

Figure 7.4: Menu for selecting the parameter to be changed.

As can be seen two parameters are missing. More specifically, the number of samples per chirp and the number of chirps per frame.
This is caused by the difficulties encountered while resetting the FNET stack. This stack takes care of the transmission through UDP of the sampled data and has been modified from NXP in order to transmit packets with a custom format.
The complexity of FNET makes it very difficult to create a single function to reset its status while simultaneously changing the required parameters.

8 Conclusion

Obtaining communication between two radar devices by modulating the sensing waveform remains a complex issue, requiring multiple stages of development. In this work, the first steps to create a physical layer for such communication technology have been laid down.
Two contributions to FMCW radar communication have been given:

- Real communication has been achieved between two separate radar devices. This result was obtained using two physically separated units composed by a FMCW radar and a GPS receiver each.

- Phase inversion during chirp transmission was achieved, increasing the overall throughput of this communication system by a factor three.

Furthermore, a software interface has been developed allowing inexperienced users to configure the developed firmware. This with the objective to ease the accessibility to our system and future developments in this technology.

While the three main research objectives have been accomplished, large obstacles have also been encountered.
The main limitation we faced bounds the maximum speed of communication between two stations. At its source are two hardware characteristics of our devices:

- The difficulty to match nominal frequency between crystals installed in different radar devices leads to a loss of synchronization between the two devices. Limiting the data transmission to a single radar frame each second.

- The large overshoot after a phase inversion on the transmitted waveform limits the amount of data that can be transmitted within a radar frame without compromising the sensing functionality of the radar devices.

Another limitation we encountered rises from the complexity in changing certain parameters during run time in the radar system. For this reason, two parameters could not be included in the user interface, namely the number of samples per chirp and the number of chirps per frame.

In conclusion, the implementation proposed in this work achieves the objectives we set off to achieve in chapter 1.4, these positive results are however limited from different factors.

8.1. Future work

Solutions to the limitations described above span out of the scope of this thesis. However, these obstacles could be overcame in future research.
Both the frequency discrepancy between radar devices and the large overshoot during phase inversion could be solved with a follow up master thesis or phd research. The large frequency difference would require a more accurate source of clock, something that can be implemented as an addition to the transceiver and micro-controller used during this thesis, requiring only a new PCB design. Correction of the overshoot can only be accomplished with a redesign of the transceiver module, requiring expertise in the field of integrated circuits design.
Due to the hardware nature of these problems, a collaboration with a semiconductor company, such as NXP, would be greatly beneficial and probably essential. Such a partnership would allow the development of dedicated hardware. Additionally, dedicated firmware could be developed ground up to meet the requirements of the researchers, without the limitations imposed by the current transceiver.

Bibliography

[1] Nevio Benvenuto. *Principles of Communications Networks and Systems*. John Wiley & Sons, Incorporated, 2011.

[2] Steven Bible. *Crystal Oscillator Basics and Crystal Selection for rfPIC and PICmicro Devices*. Microchip Technology Inc., 2002.

[3] Graham M. Brooker. Understanding millimiter wave fmcw radars. *1st International Conference on Sensing Technology*, pages 152–157, 2015.

[4] Kar-Ming Cheung. Accuracy/computation performance of a new trilateration scheme for gps-style localization. *2018 IEEE Aerospace Conference*, 2018.

[5] FNET Community. fnet, embedded open source tcp/ip stack, 2011.

[6] Irfaan Coonjah. Experimental performance comparison between tcp vs udp tunnel using openvpn. *International Conference on Computing, Communication and Security (ICCCS)*, 2015.

[7] A. Zapp E. D. Dickmanns. Autonomous high speed road vehicle guidance by computer vision. *IFAC Proceedings Volumes*, 1987.

[8] Ahmed M. Eid. System simulation of rf front-end transceiver for frequency modulated continuous wave radar. *International Journal of Computer Applications*, 2013.

[9] F. Tigrek S. Orru A. Alvarado F. Willems A. Yarovoy F. Lampel, F. Uysal. System level synchronization of phase-coded fmcw automotive radars for radcom. *in Proceedings 14th European Conference on Antennas and Propagation, EuCAP 2020*, March 15-20, 2020.

[10] B. Hofmann-Wellenhof. *GPS Theory and Practice*. Springer Wien New York, 1992.

[11] MIPI Alliance Inc. *MIPI Camera Serial Interface 2*, 2017.

[12] Tasneem R. Khan. Automotive radar: real-time implementation of joint sensing and communication waveform on a microcontroller. *Master Thesis TU Delft*, 2018.

[13] James Kirchner. *Data Analysis Toolkit n5: Uncertainty Analysis and Error Propagation*. Berkeley Seismology Laboratory. University of California., 1995.

[14] NXP. *S32R274 reference manual*, 2019. URL https://tinyurl.com/ybs5r9c4.

[15] Defense Advanced Research Projects Agency Information Processing Techniques Office. *RFC 791 - INTERNET PROTOCOL*, 1981.

[16] F. Uysal S. Orru. Phase coded fmcw automotive radar: Application and challenges. *in Proceedings IEEE International Radar Conf.*, May 2020.

[17] NXP Semiconductors. *S32R274 datasheet*, 2019. URL https://www.nxp.com/docs/en/data-sheet/S32R274DS.pdf.

[18] NXP Semiconductors. *TEF810X datasheet*, 2019. URL https://www.nxp.com/products/rf/radar-transceivers/tef810x-fully-integrated-77-ghz-radar-transceiver:TEF810X.

[19] Charles E. Spurgeon. *Ethernet: The Definitive Guide*. O'Reilly Media, 2014.

[20] IEEE standards association. *802.3 - IEEE Standard for Ethernet*, 2018.

[21] Inc. Tesla. Tesla vehicle autopilot functionality, 2020. URL https://www.tesla.com/autopilot.

[22] Federico Thomas. Revisiting trilateration for robot localization. *IEEE transactions on robotics*, 2005.

[23] Ublox. Neo-7 datasheet, 2013. URL https://www.u-blox.com/sites/default/files/products/documents/NEO-7_DataSheet_%28UBX-13003830%29.pdf.