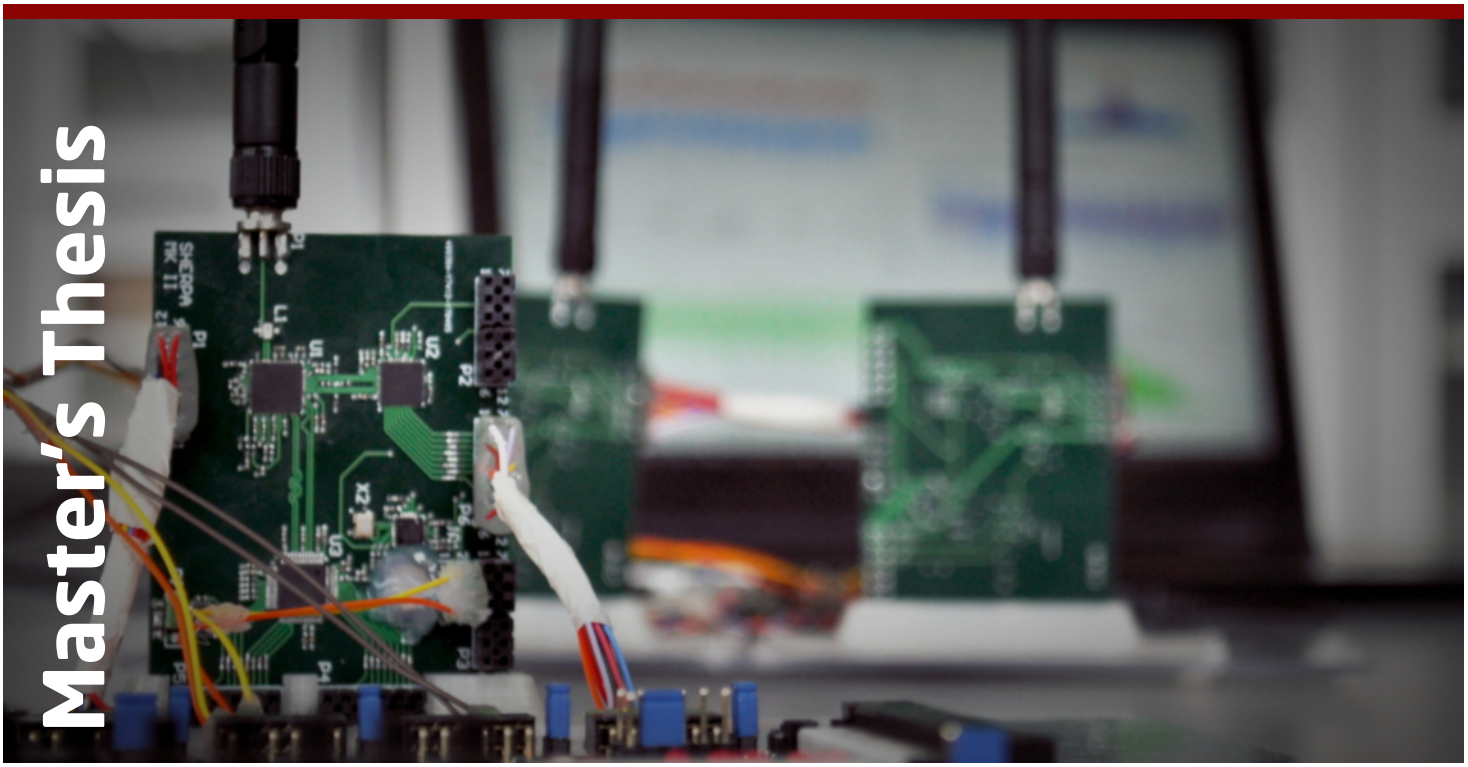
System for Hardware-based Estimation of Relative Position and Attitute

National Space Institute

S.H.E.R.P.A



Lukas Christensen
June 2017

Supervisor:
José M. G. Merayo

DTU Space
National Space Institute

M.Sc. Thesis
Master of Science in Engineering

DTU Space
National Space Institute

System for Hardware-based Estimation of Relative Position and Attitude

S.H.E.R.P.A.

Lukas Christensen (s123745)

Kongens Lyngby 2017

DTU Space
National Space Institute
Technical University of Denmark

Elektrovej, building 327
DK-2800 Kgs. Lyngby
Phone (+45) 4525 9500
Fax (+45) 4525 9575

www.space.dtu.dk

Abstract

This master's thesis deals with the development of a low-complexity relative navigation system based on time of flight measurements of radio transmissions. The choice of technology is based upon previous work by the author which is briefly outlined to provide background information. The theory necessary to determine the relative position and attitude of multiple vehicles is described and used as the basis for developing a proof of concept system.

The system architecture, including both hardware configuration and software structure, is developed with the time of flight measurements being performed by means of a novel approach to data sampling.
As no suitable radio transceivers were found to satisfy the needs of such a system, a custom one is developed, tested, and used to prove the viability of the measurement method. From this, the performance of a complete relative navigation system based on these transceivers is analyzed through simulations.

It is found that oscillator drifts make it difficult to determine distance better than within one meter, however, if the drift can be compensated for, distances can be determined with an accuracy of a few centimeters for ranges up to at least 45 m. Based on this, simulations show that a time of flight navigation system with 1 m baselines between individual sensors can achieve positional and rotational accuracy better than 5 cm and 5°, respectively, for distances up to 100 m using hardware costing less than €1000 in total.

Preface

This master's thesis was prepared at the Division of Measurement and Instrumentation, National Space Institute at the Technical University of Denmark in fulfillment of the requirements for acquiring a master's degree in Earth and Space Physics and Engineering. The study was supervised by Prof. José M. G. Merayo, National Space Institute.

Kongens Lyngby, June 23, 2017

Lukas Christensen (s123745)

Acknowledgements

I would like to thank my thesis adviser Professor José M. G. Merayo for excellent supervision and guidance throughout, and for allowing me to undertake this frankly quite challenging, but very rewarding project.

I would also like to extend my gratitude to the entire Measurement and Instrumentation Division of DTU Space for their wonderful assistance and hospitality, especially Assistant Engineer Kim Andersson and Scientist Thorbjørn Helvig Christensen.

Finally, my thanks go out to my fellow Earth and Space Physics and Engineering thesis candidates with whom I've shared many moments of both joy and despair while working on our respective projects. You have been my brothers and sisters in arms.

Contents

CHAPTER 1

Introduction

In these years there seems to be a new tendency building in the world of space exploration and research. On account of the ubiquitousness of smart phones, computer and sensor technology is becoming smaller, cheaper, and more powerful by the day. This means that constructing smaller spacecraft without compromising on functionality and reliability is becoming ever more viable. This can be seen in the increasing popularity of, for example, cubesats which are no longer just small curiosities launched by university students, but have entire companies dedicated to making and developing them. This opens for completely new possibilities for exploring the universe by, instead of having one monolithic spacecraft, combining the efforts of multiple smaller vehicles working together in unison.

For example, consider an infrared space-borne telescope such as the soon-to-be-launched James Web space telescope that cost several billion dollars to develop. If the main mirror was replaced by several smaller sections each constituting an individual spacecraft, the cost could be reduced considerably as these small vehicles would not need the special manufacturing techniques required when grinding very large parabolic mirrors, but could even be mass produced. Other mission critical elements could also be replaced by multiple small spacecraft that could offer redundancy should any one of them fail. Another case is the measurement of large scale potentials such as the Earth's magnetic or gravitational fields. Having a single satellite in a polar orbit enables to complete mapping of the planet, however, it takes a long time from a given point is mapped until the satellite passes over the same spot once again. As such, only limited information can be obtained about the temporal variation of the observables. Having many smaller spacecraft flying in formation would enable measurements of the time derivative while simultaneously also providing data about the spatial gradient, which in itself is very valuable.

Many other uses within a diverse set of scientific and technological disciplines can be imagined, but common for all of them is that the spacecraft need some way of determining their relative position and attitude. Translation can potentially be determined using *global navigation satellite systems* (GNSS), while various attitude sensors exist that can provide orientation information. However, GNSS currently only works in close vicinity to the Earth and require fairly complex receiving hardware to achieve high levels of accuracy. Moreover, attitude sensors are often complex, expensive and separated from the positioning system. If navigation could be determined accurately using a self-contained system of low complexity, it could potentially enable endless possibilities.

CHAPTER 2

Problem Statement

This project strives to perform the initial steps in the development of a relative navigation system based on time-of-flight measurements. These steps include the following:

1. Showcasing that the measurement principle is valid when performed with low complexity, preferably *comercial-off-the-shelf* (COTS) parts.
2. If the method proves unfeasible, another solution must be identified and investigated.
3. A high-level design of a complete system must be created.
4. The prospective performance of the design must be evaluated using simulations to ensure that it lives up to the requirements specified below.
5. A prototype must be developed and characterized to investigate whether or not the simulated performance is achievable in actual hardware.

The purpose of the system is to provide accurate estimates of relative position and attitude between two or more space vehicles for the purposes of formation flight, rendezvous and docking, and gradiometric measurements. As such, the system should be able to perform adequately in ranges from sub-meter to multiple kilometer scales and preferably work equally well both in deep space and for terrestrial applications.

A satisfactory system will live up to the following specifications:

1. A rotational accuracy better than $\pm 5^\circ$.
2. A positional accuracy better than $\pm 10\,\mathrm{cm}$.
3. These accuracies should be attainable for ranges up to at least $100\,\mathrm{m}$.
4. The system shall be of sufficiently small size that it may be of use for spacecraft with volumes less than $1\,\mathrm{m}^3$.

Beside these specifications the system shall ideally also be of low-complexity and cost such that it may be utilized for many-spacecraft missions without being a limiting factor in the design process.

CHAPTER 3

Project Plan

This section holds the project plan and self-evaluation as required by the DTU master's thesis hand-in guidelines.

3.1 Original Plan

This project deals with the development of a relative navigation system for use within various fields of spaceflight. The goal is to design a system that satisfies the requirements of the problem statement (cf. chapter 2) and investigate the viability of the design using a combination of custom measurement hardware and simulations.

The plan for this project is to split up the development of the relative navigation system into five discrete parts with an approximately equal amount of time dedicated to each. These are: method validation, system design, hardware development, software development, and testing.

At the onset of the project, time will be spend on validating the viability of the fundamental measurement method in the context of *radio frequency* (RF) transmissions. This includes finding a ready-made hardware solution for radio communication and developing any necessary control software. Should this early analysis prove that the measurement method is not usable; another one must be quickly identified and verified before the project can continue. Naturally, this will involve major restructuring of the project plan. Should the method instead prove to be viable, the project will progress to the next phase.

Before hardware and software development can be initialized, some degree of system design is needed. Ideally, an overall architecture can quickly be developed paving the way for the next project phases to be initiated. The plan is to continue the system design throughout the project, but at a lower priority while the software and hardware is being developed.

The software development will run concurrent with system design, but at a high priority meaning that, in general, more time will be dedicated to the software. However, it is expected that the software will be completed in steps according to what is needed in the current stage of the project as a whole. There will therefore be extended periods of time without the need for programming, why the time spent on this part of the project should end up being equivalent to that of the overall design.

The hardware development phase will almost certainly include some time waiting for shipping and manufacturing, as such this shall be started early and be of top priority. Once it is done, the testing phase will take over the top priority spot and continue for the remainder of the project.

Once everything else is completed, the final few weeks of the project will be dedicated to report writing. Of course, documentation will also happen during the development processes, but at a lower priority. A prospective timeline of all of this can be seen in figure 3.1.

3.2 Evaluation

The initially developed project plan turned out to be fairly realistic. Some parts of the project actually ended up requiring less time than expected, while others were extended due to unexpected delays. The actual time line can be seen in figure 3.1.

The initial validation process was longer than expected due to the fact that no ideal RF solution could be identified. Therefore, a suboptimal compromise had to be made which resulted in additional time being spent on configuration.

Both the software development and the overall project design progressed more or less as expected, with the main difference being that the software actually took a shorter time to develop than expected. In addition, the increased focus on the initial validation resulted in the coding process being postponed significantly.

The hardware development took up more time than was alloted to it in the original project plan, however, this was not completely unexpected as that is the nature of the beast. Various delays and extended manufacturing times were the main cause of this.

Testing the developed system components also required more than the allotted time. This was due to both the delayed hardware construction as well as various measurement issues that required modifications to software and the planned testing procedures. The result of this ultimately led to the development phase being extended into what was originally dedicated to documentation.

Regardless of the delays, it is the opinion of the author that the initial project plan was of sound design based on the knowledge available at that point. Furthermore, the project was finished on time with no significant task being delayed by more than a few weeks. Therefore the project as a whole must be considered successful and the work load adequate, as the original goals were accomplished.

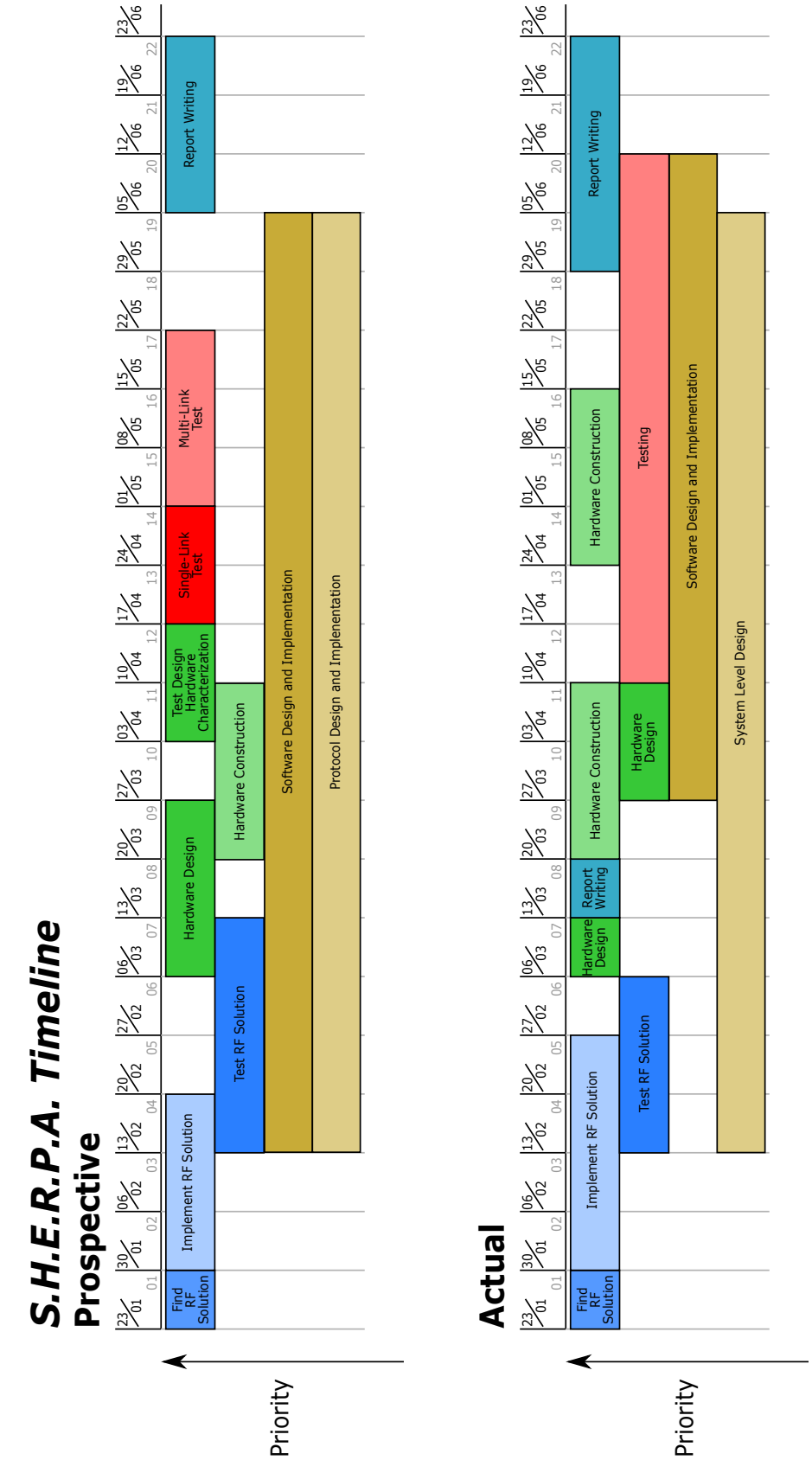


Figure 3.1: Timeline of the project, both prospective and actual.

CHAPTER 4

Previous Work

Before the start of this project, a smaller feasibility study was performed to determine whether or not it would be possible to develop a relative navigation system meeting the prescribed requirements. This study consisted of an examination of relevant literature with the goal of finding a suitable technology, followed by the development of simple test hardware. The hardware performance was compared to theory and simulations were performed to analyze the potential performance of the technology[9].

4.1 Relative Navigation Methods

In total, six different technologies were considered based on the literature study, namely:

1. *Time of flight* (ToF)
2. *Line of sight* (LoS)
3. Photogrammetry
4. Inertial sensing
5. Differential GNSS
6. Structured light

Time of flight measurements work on the basis of measuring the time it takes a signal to propagate from one place to another, which is equivalent to the distance if the travel speed is known. This is the basis of all current GNSS solutions and is therefore very well proven in its potential for accurate navigation. Various methods exist for measuring the time delay, but they all share a common issue: Very stable clocks are needed on both the transmitter and receiver of the signal to provide accurate results, especially when using electromagnetic waves as the communications medium. This requirement is somewhat diminished if two way communication is employed as possible clock offsets can be estimated this way. If the distances between a target and a series of points are known, the target position can be estimated using trilateration. If the positions of multiple points on the surface of a given target are thus determined, the orientation and position of the target can be estimated. Using GNSS accuracies on the scale of millimeters are possible[7].

Line of sight systems work by measuring the direction to a target by means of various sensors. Measuring the direction to the target from at least three different points enables the determination of its position through triangulation, which, similarly to time of flight measurements, can be used to estimate the navigational parameters of an arbitrary target. The direction can be obtained using, e.g., cameras[26] or custom designed sensors. Simulations show that accuracies on the scale 1 cm positional, 1° rotational are achievable at distances around 100 m[11].

Using images of a target it is possible to extract information about position and orientation relative to the imaging system. This process is referred to as photogrammetry. Photogrammetry has the potential of being very accurate (position error < 1 mm for short distances[6]), however, it is quite sensitive to illumination conditions and is computationally expensive. Furthermore, the target needs to extend over a significant number of pixels in the gathered image if any information is to be extracted which means that the range is generally limited to less than 100 m[6]. The distance over which it works can be extended by using long focal length lenses at the cost of limiting the field of view. If the target is so far away that it constitutes only a few pixels, this method essentially converts to line of sight sensing[26].

Inertial sensing refers to measuring acceleration caused by the sensor not constituting an inertial reference frame. By integration of the acceleration it is possible to estimate position and attitude through dead reckoning with potentially high accuracy and frequency. One large disadvantage of this type of system is that errors will accumulate over time resulting in gradually decreasing accuracy. This leads to the requirement of external information from other types of sensors to provide stable measurements over extended periods of time. In the context of a relative navigation system, inertial measurements from two different vehicles could be subtracted from each other and then integrated to provide relative navigation parameters, however, this does require communication accurately synchronized in time. Furthermore, some extra method is needed to ensure that the inertial measurements do not drift away from the true values[15].

Differential GNSS works by comparing the GNSS signal from two or more receivers and looking at the differences. This way many distorting effects can be eliminated and an estimate of the relative positions of the receivers can be made with a much higher accuracy than is possible using just a single receiver in relation to the Earth. This method works very well on the surface of, or in low orbit around, the Earth, but is useless in deep space or around other planets as no GNSS satellites are available. In addition, to achieve high accuracies (potentially millimeter precision) complex and expensive receiver hardware is needed[7].

Using emission of structured light to determine orientation and translation can potentially lead to very high accuracies, but a limited ranges. The basic principle consists of emitting a specific light pattern and measuring where and/or when a target object is illuminated. Based on this information the morphology of the target can be determined, or, if this is already known, the position and orientation can be estimated with accuracies and at ranges similar to those of photogrammetry[24]. The main drawback of

this approach is that the expanding nature of the illumination pattern means that the accuracy quickly diminishes with increasing range. In addition, the emitter hardware is somewhat complex and the method is generally quite computationally intensive[24].

4.2 Project Contents

Through weighing of the pros and cons of the different measurement technologies it was decided that time of flight would be most appropriate for the project. Actually, a combination of ToF and LoS would be ideal as one provides very good distance estimates while the other results in accurate directions, however, due to the limited time available it was decided to focus on a single technology.

A simple reference design was developed consisting of two vehicles each with four transceivers distributed over their surfaces constantly measuring the inter-transceiver distances. It was decided that the transceivers would communicate using visible light as this would be easier to implement compared to radio transmissions. Thus, a simple transceiver was designed and constructed consisting of a high power *light emitting diode* (LED) and a reverse bias photodiode connected to an *operational amplifier* (OpAmp) based amplification circuit.

The actual distance measurements were performed by emitting amplitude modulated red light and then correlating the received signal with a copy of the modulation code (see section 5.2.1). A novel approach to sampling the signal was devised and implemented resulting in high potential accuracy at low sampling rates. This method has been much expanded upon during the course of this current project and will be detailed in section 5.2.2.

To determine how well the designed system could perform theoretically, regardless of how the test hardware might turn out, simulation software was also developed.

4.3 Results

The developed hardware turned out to be only somewhat functional. It was able to transmit and receive the coded signal, however, issues with the amplifier necessitated an introduction of low pass filters which had the unwanted effect of severely distorting the signal. Figure 4.1 shows plots of *cumulative dot product* (CDP) (see section 5.2.2) functions that were obtained using the transceivers at a distance of 50 mm from each other.

The CDP function should ideally have a pyramid shape, with the location of the peak indicating the time delay between transmitter and receiver. As is clearly evident, it is difficult to determine anything from the produced graphs. The ”flattening” of the peak appears to be caused by the aforementioned low pass filters distorting the signal. Even though no exact delay can be determined, this result still indicates that the measurement

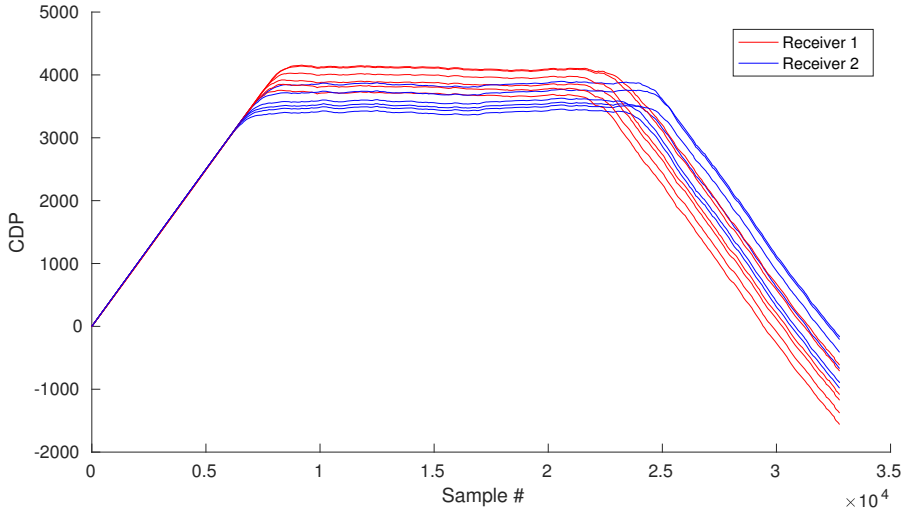


Figure 4.1: The cumulative dot product calculated over five cycles using the hardware developed in the pre-project. The distance between the two transceivers was 5 cm[9].

method does indeed work since some semblance of a pyramid is produced. Most likely, this method could be made to work properly with visible light if more time was spent on amplifier design. However, with the photo diodes used in the pre-project even a better amplifier would not result in a usable system as the magnitude of the signal would reach the dark current levels at a distance of just 5 m. Potentially other optical sensors could be employed extending the range, but it was ultimately decided that radio transmissions would be a better option for any future project based on this measurement principle.

The simulations showed that for a data rate of 100 MHz, distances could be determined to within one tenth of a millimeter resulting in uncertainties of around 5 cm positional and 5° rotational at a distance of 100 m and baselines of 1 m. These results were obtained using a modulation code length of 32767. It was also shown that the potential accuracy varies linearly with both code length as well as data rate. Furthermore, distance errors are constant regardless of baseline while both transverse positional and angular errors are inversely proportional to baseline length. These results matched well with theory why they were considered to be realistic. Selected graphs showing these results are available in appendix A.

Based on these results it was decided that the measurement method showed enough promise to continue with this project albeit using radio instead of visible light for transmissions.

CHAPTER 5

Theory

5.1 Relative Navigation

In the context of this project, relative navigation refers to the process of finding the position and attitude of a target vehicle within the coordinate frame of a base vehicle. As such, the navigation parameters are only known relative to the base, hence the term. This is distinct from other forms of navigation where orientation and translation are determined with respect to a (semi)-inertial reference frame such as the Earth or the stars, as is the case with, e.g., GNSS.

The relative navigation parameters of a target vehicle can be calculated if the positions of a series of points on the target vehicle are known in the frame of the base vehicle. One way to determine these positions is by means of distance measurements.

5.1.1 Distance Measurements

Consider two points, p_1 and p_2, with known coordinates located in a 2D-plane and separated by a distance b, as well as a target point p_t located at an arbitrary point in the plane. This situation is illustrated in figure 5.1.

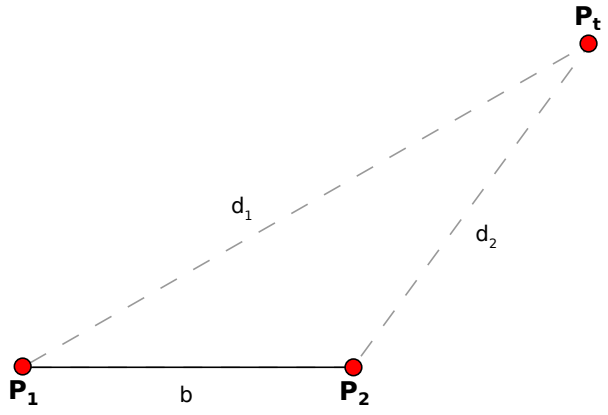


Figure 5.1: Geometry for determining 2D position from distances.

Defining the distances from p_t to p_1 and p_2 as d_1 and d_2, respectively, it follows that

$$d_i = \sqrt{(p_{i,x} - p_{t,x})^2 + (p_{i,y} - p_{t,y})^2} \tag{5.1}$$

where $p_{i,j}$ is the j-coordinate of point i. If d_1 and d_2 are known, the coordinates of p_t can easily be found by solving the above set of equations numerically. This can be extended to three dimensions with the addition of a z-coordinate, however, now three points are needed to determine the position of p_t unambiguously.

5.1.2 Differential Distance Measurements

If the target is far away from p_1 and p_2 it will hold that the angle θ between the vectors $\mathbf{p_1p_t}$, $\mathbf{p_2p_t}$ and the baseline vector $\mathbf{p_1p_2} = \mathbf{b}$ will be approximately equal. This is illustrated in figure 5.2.

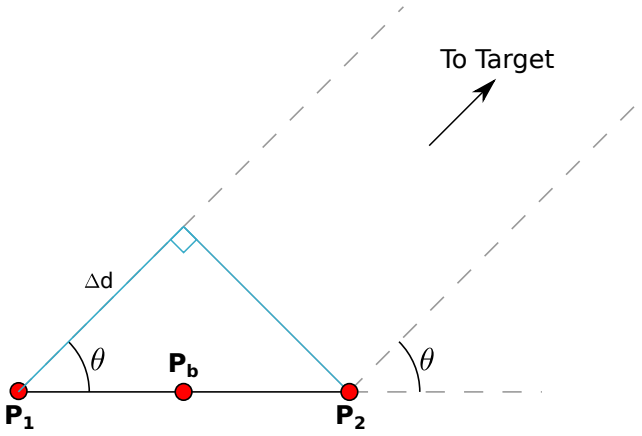


Figure 5.2: Geometry for determining direction from angle and/or distance difference.

It follows that

$$cos\theta = \frac{\Delta d}{b} \tag{5.2}$$

with Δd being the difference in distance to the target as measured at p_1 and p_2. Defining a vector going from the center of the baseline, p_b to the target

$$\mathbf{d} = [p_{t,x} - p_{b,x}, p_{t,y} - p_{b,y}]^T \tag{5.3}$$

equation 5.2 can be rewritten as

$$\frac{\Delta d}{b} = \frac{\mathbf{b} \cdot \mathbf{d}}{b|\mathbf{d}|} \Leftrightarrow \Delta d = \frac{\mathbf{b} \cdot \mathbf{d}}{|\mathbf{d}|}. \tag{5.4}$$

It follows that the angle to the target can be determined using the differential distance of two known points, however, if another baseline is added it is possible to find a vector pointing in the direction of the target. Therefore three points are needed in the 2D case. Interestingly enough, three points are also sufficient in 3D space as three separate baselines can be formed connecting three points as long as they are not all located on the same line.

If the target is not far away, the position can still be found, however, the governing system of equations now becomes

$$\Delta d_{ik} = \sqrt{(p_{i,x} - p_{t,x})^2 + (p_{i,y} - p_{t,y})^2} - \sqrt{(p_{k,x} - p_{t,x})^2 + (p_{k,y} - p_{t,y})^2}. \tag{5.5}$$

5.1.3 Navigational Parameters

If the target point is located on a vehicle that has some translation and rotation associated with it, the target position can be written as

$$\mathbf{p_t} = \mathbf{t} + \mathbf{R}\mathbf{p'_t} \tag{5.6}$$

where $\mathbf{t}$ and $\mathbf{R}$ are the translation vector and rotation matrix of the vehicle, respectively, while $\mathbf{p'_t}$ is the position of the target in the coordinate frame of the vehicle.

Now assuming 3D space, it is often advantageous to represent the rotation using quaternions instead of, say, Euler angles as this avoids the issues of gimbal lock and also reduces the need for trigonometric functions which are computationally expensive[15]. Defining a quaternion describing the vehicle rotation as $\mathbf{e} = [e_0, e_1, e_2, e_3]^T$ results in the following form of the rotation matrix[31]:

$$\mathbf{R} = \begin{bmatrix} e_0^2 + e_1^2 - e_2^2 - e_3^2 & 2(e_1e_2 + e_0e_3) & 2(e_1e_3 - e_0e_2) \\ 2(e_1e_2 - e_0e_3) & e_0^2 - e_1^2 + e_2^2 - e_3^2 & 2(e_2e_3 - e_0e_1) \\ 2(e_1e_3 - e_0e_2) & 2(e_2e_3 - e_0e_1) & e_0^2 - e_1^2 - e_2^2 + e_3^2 \end{bmatrix}. \tag{5.7}$$

Since a quaternion used for rotation must satisfy $|\mathbf{e}| = 1$ a total of six navigational parameters are needed to describe the transformation of the vehicle: three for position and three for rotation.

Inserting equation 5.6 into the equations 5.1 to 5.5, these can be solved for the vehicle transformation parameters directly instead of finding the individual target locations first. However, for this to work a minimum of three target points and three reference points are needed to provide a unique solution. One way to solve the equations is least squares estimation.

5.1.4 Least Squares Estimation

An arbitrary set of data $\mathbf{d}$ that can be described by a linear model can be expressed in the following way

$$\mathbf{d} = \mathbf{G}\mathbf{m} + \varepsilon \tag{5.8}$$

where ε is a data misfit error term and $\mathbf{G}$ is a design matrix that relates a set of model parameters $\mathbf{m}$ to the observed data. This formulation, i.e. calculating the data from the model, is referred to as the forward problem while determining the model parameters from the data is referred to as the inverse problem. One method for solving the inverse problem is ordinary least squares which determines the set of model parameter that minimizes the L_2-norm of the error term[4]. Using this method $\mathbf{m}$ is calculated according to

$$\mathbf{m} = (\mathbf{G}^T {\mathbf{C_d}}^{-1}\mathbf{G})^{-1}\mathbf{G}^T {\mathbf{C_d}}^{-1}\mathbf{d} \tag{5.9}$$

where $\mathbf{C_d}$ is the covariance matrix describing the uncertainty associated with the data. This works well in many cases, however, if dealing with an ill-posed problem it will not always provide a usable solution[4]. Under such circumstances, more advanced methods are needed that can provide some boundaries on the solution. One such method is damped least squares (also known as Tikhonov regularization) which bounds the problem by trying to minimize the norm of the model vector simultaneously with error term. Assuming equal uncertainties on all observations, this method is formulated as

$$\mathbf{m} = (\mathbf{G}^T\mathbf{G} + \alpha^2\mathbf{I})^{-1}\mathbf{G}^T\mathbf{d} \tag{5.10}$$

with α being a small number that determines how much weight is put on minimizing model norm compared to data misfit, and $\mathbf{I}$ being the identity matrix with the same dimensions as $\mathbf{G}^T\mathbf{G}$[4].

For weakly non-linear problems described by a model function g, it is still possible to use the least squares method, however, it has to be modified. The modification consists of linearizing the problem by first choosing an initial model estimate $\mathbf{m_0}$ and then using the following equation

$$\delta\mathbf{m} = (\mathbf{J}^T {\mathbf{C_d}}^{-1}\mathbf{J})^{-1}\mathbf{J}^T {\mathbf{C_d}}^{-1}\delta\mathbf{d} \tag{5.11}$$

where $\delta\mathbf{m}$ is a change in the model parameter estimate and $\delta\mathbf{d}$ is the difference between the actual data and what can be estimated using the current model parameters. $\mathbf{J}$ is the Jacobian matrix defined as

$$J_{i,j} = \frac{\partial g_i}{\partial m_j}. \tag{5.12}$$

By adding the change to the current estimate and then executing the equation again and again in an iterative fashion the model parameter estimate will eventually converge towards some constant value, provided that the problem is not too non-linear and the starting values are chosen intelligently[4]. Damped least squared can be linearized in a similar way.

5.2 Time of Flight Measurements

The distance between two points can be determined by emitting a signal from one of the points and then measuring the time it takes for the signal to propagate to the second point. The distance is then equal to

$$d = \Delta t \cdot c \tag{5.13}$$

with Δt being the propagation time and c being speed of propagation, which, for radio signals in a vacuum, is equal to the speed of light ($299\,792\,458\,\mathrm{m/s}$).

5.2.1 Delay Estimation

Consider two radio *transceivers* (XCVR), arbitrarily positioned in space, sharing a common clock source. If XCVR 1 transmits a radio signal of finite duration at time $t = 0$ and XCVR 2 starts recording at the same time it is possible determine the propagation time of the signal by noting when it arrives at XCVR 2. However, if a third transceiver starts transmitting simultaneously with XCVR 1, some method is needed to separate the two signals.

5.2.1.1 CDMA

Code-division multiple access (CDMA) is a method that allows for multiple different signals being emitted on the same frequency that can all be separated from each other upon reception. The separation is performed by having each transceiver modulate the transmitted signal using a binary *pseudo-random number* (PRN) code that is unique to it. The PRN code is generated at a chirp-rate much higher than the data rate such that an integral number of complete PRN codes are transmitted for each actual data bit. Whether the data bit is a 0 or 1 is determined from the polarity of the PRN code[17].

If, upon reception, the combined signal from multiple transceivers is cross-correlated with a local copy of the PRN associated with a specific transceiver a peak will be present at the point associated with the transmission delay. However, if the sought for signal is not present little to no correlation will be observed.

This principle is illustrated figure 5.3 where the left plot shows the auto-correlation function of the PRN code associated with a certain transceiver, while the right plot

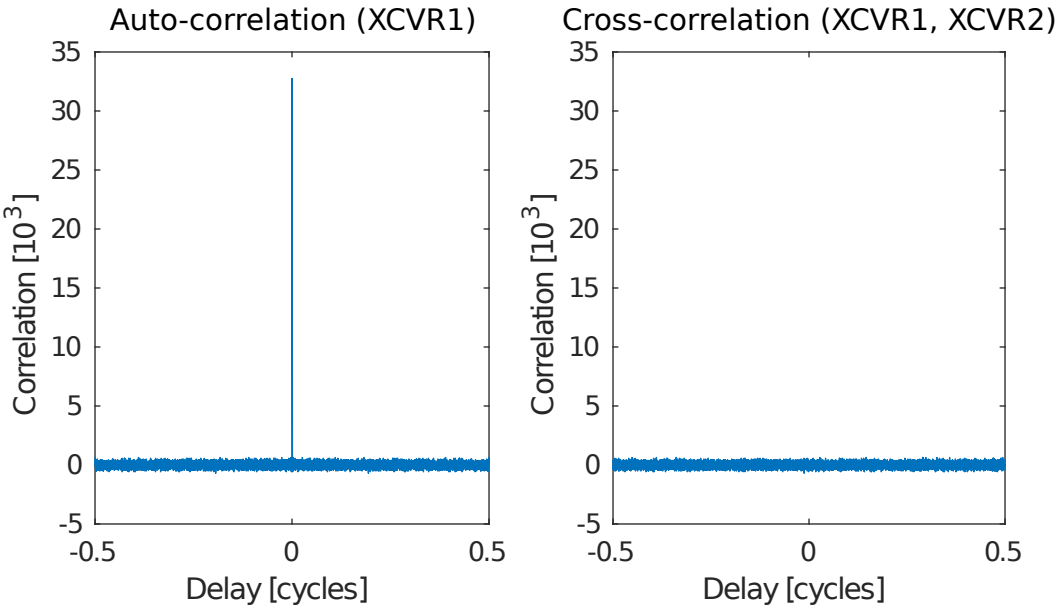


Figure 5.3: Comparison between the auto-correlation of the 32767 bit Gold code used for XCVR 1 and the cross-correlation between the codes used for XCVRs 1 and 2.

shows the cross-correlation between the same PRN code and that of another transceiver. As can be seen, there is no noticeable correlation between the codes and hence these signals can be separated from each other.

In addition to allowing the separation of different signals, CDMA also has the effect of spreading signal information over a wide bandwidth which improves robustness to noise. In fact it is possible to receive data from signals with *signal-to-noise ratios* (SNR) $\ll 1$[7].

5.2.1.2 Gold Code Generation

For CDMA to be the most effective the PRN codes must be selected such that they have as small cross-correlations profiles as possible and are ideally easy to generate. A class of pseudo-random number that satisfies this is the Gold Code.

Linear feedback shift registers (LFSR) can be used to generate pseudo-random bit sequences. They consist of standard shift registers of n-bit length where some of the intermediate outputs are `xor`'ed together and fed back to the input, which results in a pseudo-random bit sequence at the output[19]. This is illustrated in figure 5.4.

The exact outputs that are tapped to generate the feedback are commonly described with the characteristic polynomial, which for the shift register of figure 5.4 is equal to

$$P_1(x) = 1 + x^8 + x^{15}. \tag{5.14}$$

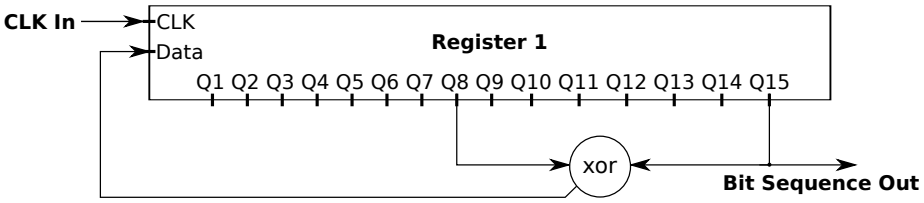


Figure 5.4: A 15 bit linear feedback shift register.

Certain feedback configurations, referred to as *maximum length sequences* (M-sequences) generators, have the special property that the output sequence repeats itself after $2^n - 1$ bits. $P_1(x)$ represents once such configuration. This type of sequence has a number of interesting properties including a very well defined single peak auto-correlation function. However, the cross-correlation between different M-sequences often have multiple significant peaks which makes them unsuitable for CDMA[19].

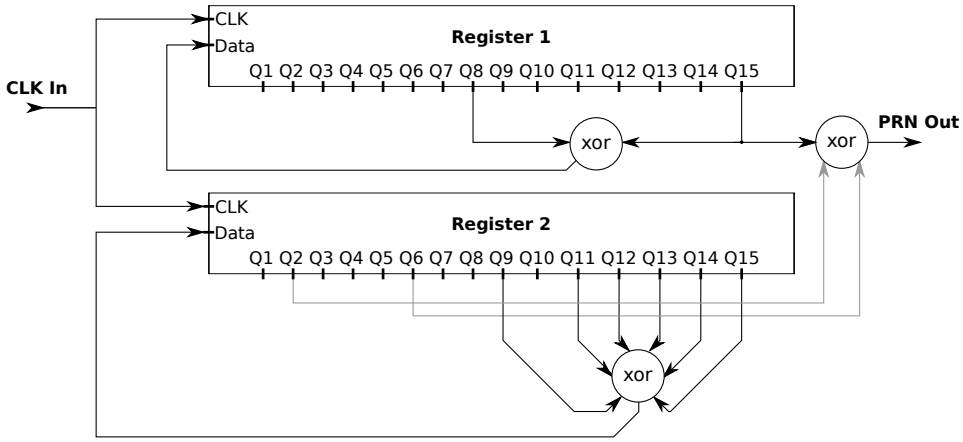


Figure 5.5: The 15 bit shift register configuration which generates one of the 32767 bit PRN codes that are used in this project. The lines that connect the second shift register to the output are colored gray to indicate that these can be moved around to generate other PRN sequences.

If, however, the outputs of two different M-sequence generators are `xor`'ed together, as is shown in figure 5.5, another type of pseudo-random bit sequence is generated - the Gold code. These codes have auto-correlation functions with slightly less well defined peaks, but cross-correlations that appear as low magnitude random noise (see figure 5.3), which makes them ideal for CDMA[17].

Different Gold codes can be generated using the same LFSR configuration by shifting the phase of one of the shift registers. This can be achieved by either changing the initial state of the register, or, alternatively, using an `xor` combination of several tabs as the output instead of simply tabbing the last bit. The second approach is what will be used

in this project together with the LFSR configuration of figure 5.5, i.e., a combination of $P_1(x)$ and

$$P_2(x) = 1 + x^2 + x^6 + x^9 + x^{11} + x^{12} + x^{13} + x^{14} + x^{15}. \tag{5.15}$$

5.2.2 Fractional Sampling

When a CDMA signal is sampled at a frequency of f_s, the location of the auto-correlation peak, and hence the time delay, can be resolved with an accuracy of $1/f_s$[7]. This means that to achieve high accuracy, high sample rates are generally needed. In many advanced *global positioning system* (GPS) receivers this issue is overcome by measuring the phase of the carrier wave in addition to the code delay which enables higher accuracy, but leads to ambiguities that need to be resolved before the position can be determined[7]. However, by sampling the signal at a specific frequency that is slightly lower than the chirp rate, it is actually possible to achieve an accuracy that would otherwise require unfeasible sampling rates.

Consider an analog non-return-to-zero (NRZ) binary signal as shown in figure 5.6(c). This can be considered as being the convolution of a single square pulse (5.6(a)) and a series of delta functions representing the corresponding digital signal (5.6(b)). Any delay of the analog signal can be described by a delay of the square pulse, which means that by deconvolving the analog signal with the series of delta functions, the square pulse can be reconstructed and the delay can be determined. A delay of 0.5 bit has been used for the purposes of this illustration.

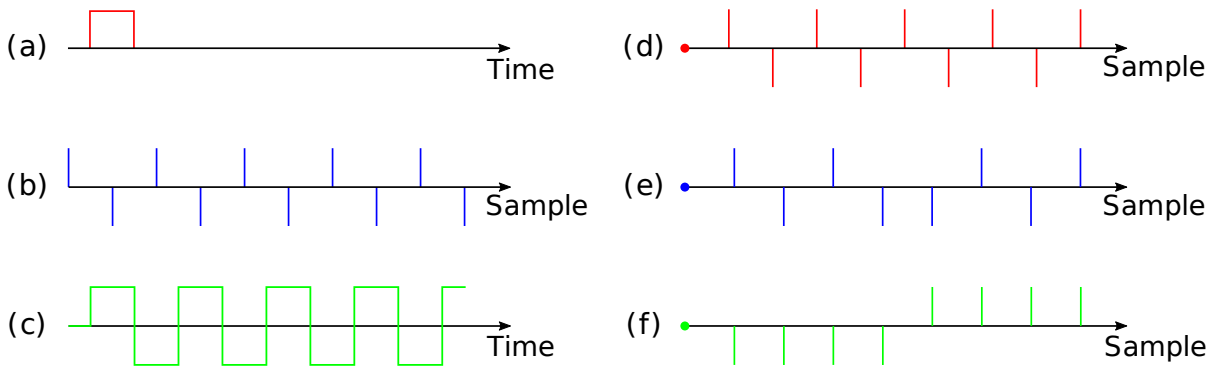


Figure 5.6: This figure illustrates the concept of fractional sampling: (a) is a single square pulse with a width of τ_b; (b) is a series of delta functions describing a digital signal; (c) is the convolution of (a) and (b) corresponding to an analog signal; (d) is the digital signal that comes from sampling (c) at $1/\tau_b = f_b$; (e) is the digital signal produces by sampling (c) with $0.9f_{frac}$; and (f) is the result of performing element-wise multiplication of (b) and (e).

Figure 5.6(d) shows the digital signal that is produced when the analog signal is sampled at the chirp frequency f_b and is just a copy of the transmitted signal with opposite polarity because of the delay. This means that the delay can only be determined to within one bit. Meanwhile, 5.6(e) shows the signal when sampled at a frequency of $0.9 \cdot f_b$. This offset in frequency means that there will be 9 samples of (e) in the time that 10 samples of (d) will be produced. Therefore, each bit of (c) will be sampled at different positions resulting in the production of (e).

Assuming amplitudes of 1, element-wise multiplication of (b) and (e) will produce figure 5.6(f). This essentially corresponds to the first $1/f_b$ seconds of (a) sampled at a rate of $10f_b$, while mixing (e) with a version of (b) that is delayed by 1 sample will represent the next $1/f_b$ seconds of (a) and so on. This means that the original pulse can be reconstructed and the delay can be determined with the same accuracy as if the signal had been sampled with a rate of $9f_b$.

Of course, the shown example uses a simple square wave signal where the polarity shifts for each bit, which shows an ideal situation for this method. If a different signal is transmitted instead, say a PRN code, the resulting pulse reconstruction will be less clean as the pseudo-random nature of the signal will introduce some correlation where there should be none. This can be seen in figure 5.7.

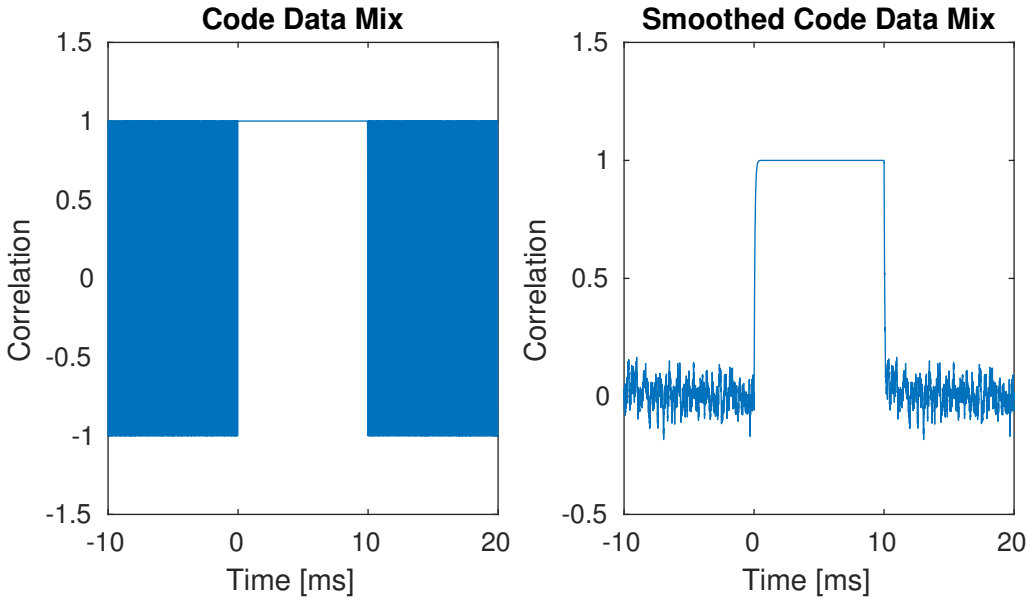


Figure 5.7: The left graph shows pulse recovery by mixing the received data with an early, an on-time and a late copy of the modulation code. The right graph is the same as the left one after a low pass filter has been applied.

The left plot of this figure shows the pulse recovery of 32767 bit NRZ Gold code, while the right plot shows a smoothed version of the same data. As can be seen, the mixing produces all ones during the actual pulse and random data outside of it. The signal has

been sampled with a rate of $f_b \cdot 32766/32767$ resulting in 32766 samples per cycle. In this case f_b has been selected to be 3276700 Hz. This sampling rate is ideal as it ensures that the sampled signal is realigned with the emitted signal after exactly one code cycle. This can be generalized to a sample rate of

$$f_s = f_b \frac{N-1}{N} \tag{5.16}$$

where N is the number of bits in the code. Following this method means that the delay of a signal modulated by a given PRN code can theoretically be determined with an accuracy on the scale of $1/f_b/(N-1)$. In practice, the accuracy will not be quite as high because of the noise introduced by the nature of the PRN code. Regardless, if a 32767 bit code is transmitted at a rate of 1 MHz, a sampling rate of 3.2766 GHz (!) would be needed to achieve to same level of accuracy as can be achieved with this method.

To capitalize on the benefits of this fractional sampling scheme a method is needed to determine the exact position of the recovered pulse shape. One method is to mix the received signal with three copies of the PRN code - one delayed by 1 sample, on one time, and one that is 1 sample early - and then low pass filtering the result to perform a pulse reconstruction. This is how figure 5.7 was created. The time delay can then be found by simply noting when the resulting signal crosses a certain threshold. This, however, is fairly computationally intensive, and might be problematic for real time applications.

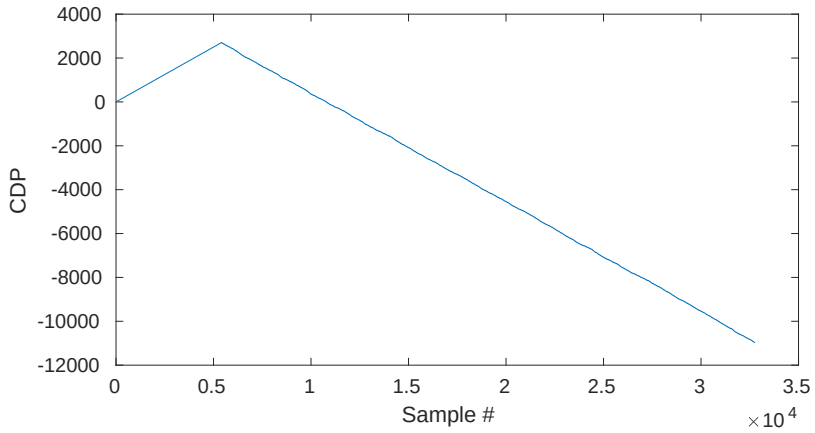


Figure 5.8: Plot of a noise free CDP for a delay of $0.165/f_b$.

Another method is to introduce the concept of a *cumulative dot product* (CDP) defined as

$$CDP(k) = \sum_{i=1}^{k} (S(i) \cdot C(i-1) - 0.5) \tag{5.17}$$

where $S(i)$ and $C(i)$ are the recorded signal and the PRN code at index i, respectively. That is, the cumulative sum of the element-wise product of the signal and a copy of the PRN code delayed by 1 sample. The -0.5 has the effect of "rotating" the resulting function such that it produces a distinct peak instead of an increasing slope followed by a flat plateau. An example of the CDP is plotted in figure 5.8, once again, using a 32767 bit Gold code for the signal. As can be seen, the CDP produces a peak at the sample corresponding to the time delay, why it constitutes a less computationally heavy alternative to the direct recovery of the pulseform. However, the CDP is somewhat more sensitive to noise and can result in ambiguities when the delay is close to integer multiples of the chirp length.

Regardless of how the fractional offset is determined it can be converted to distance using

$$d = \left(\omega_i + \frac{\omega_f}{N-1}\right)\frac{c}{f_b} \tag{5.18}$$

where ω_i and ω_f are the integer and fractional delay, respectively.

5.2.3 Error Sources

When using the propagation time of a signal from one transceiver to another as a way of measuring distance, there are a number of error sources that can reduce the effective performance and should therefore be taken into account.

5.2.3.1 Timing Issues

No matter how well-calibrated a system is, it is impossible to have two independent clocks be completely aligned in time. This means that if a transceiver, B, starts to transmit at time $t = 0$ and another transceiver, A, starts to receive at time $t = 0$, the time at which the signal arrives at A will, according to A, be equal to

$$t_A = \frac{d}{c} + \delta t \tag{5.19}$$

where d is the distance between A and B and δt is a time offset caused by the fact that A and B do not completely agree about when $t = 0$ occurs. This situation is illustrated in figure 5.9.

Similarly, if the signal is instead emitted from A to B the measured time of arrival will be

$$t_B = \frac{d}{c} - \delta t. \tag{5.20}$$

Because the time difference is added for one way and subtracted from the other, it is possible to estimate both the distance and the offset by combining the two measurements:

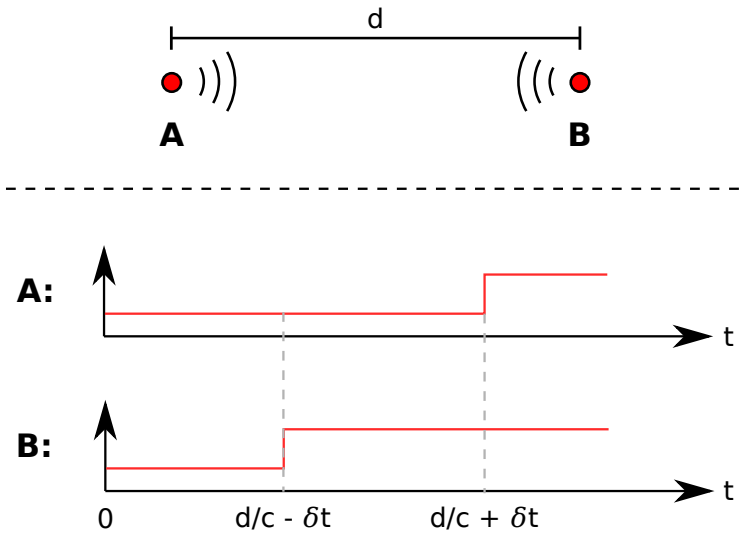


Figure 5.9: If a signal is emitted from two transceivers, A and B, at slightly different times because of clock differences, the time offset can be determined by combining the measurements of both transceivers.

$$t_a + t_b = 2\frac{d}{c} \Leftrightarrow d = \frac{c}{2}(t_a + t_b) \tag{5.21}$$

$$t_a - t_b = 2\delta t \Leftrightarrow \delta t = \frac{1}{2}(t_a - t_b). \tag{5.22}$$

Naturally, this implies that two way communication is required for distance determination, however, if multiple transceivers are connected to each clock it is in some cases possible to estimate the time offset using unidirectional transmissions. This is, for example, the case for GPS where it is assumed that the individual GPS satellites are synchronized in time and only the receiver has an offset. By combining the signals from four different satellites, both receiver position and time offset can be estimated simultaneously, however, this only works well if the satellites are distributed over a large area of the sky. If this is not the case, and the satellites are gathered close together, the uncertainty on the time estimate will be very large[7].

Besides the constant time offset that is present between different clocks, there are additional issues regarding time. Differences in the oscillators controlling the clocks manifest themselves in two main ways: long term frequency offsets and short term frequency instabilities.

Figure 5.10(a) shows a reference clock signal, while 5.10(b) shows a signal with a small offset in frequency. As can be seen, this has the effect of constantly changing the phase between the two signals which is indistinguishable from changes in distance, thus resulting in a varying propagation delay. If a frequency offset is present and constant

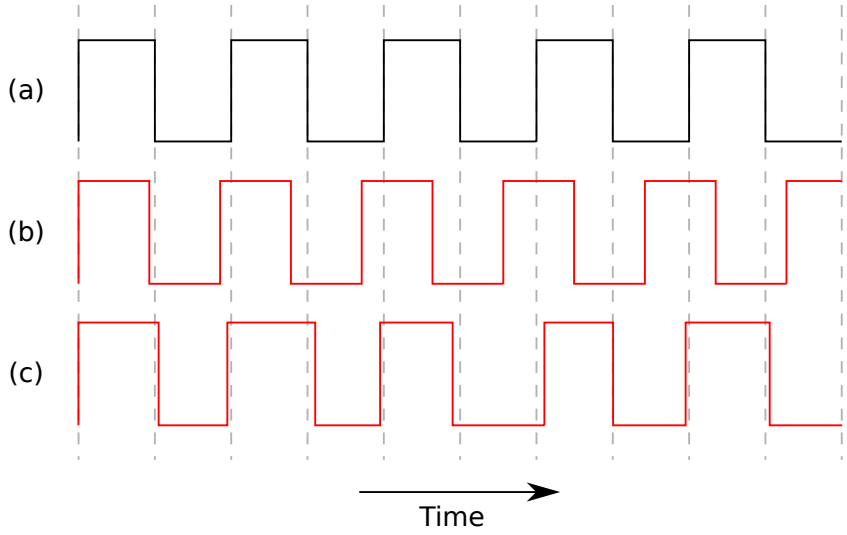


Figure 5.10: A comparison of different timing errors. (a) represents the true undistorted clock signal; (b) corresponds to a clock signal with a constant frequency offset; and (c) is a jittery clock signal with the same period as (a).

it can be accounted for and removed, however, if the frequency offset is changing in a non-deterministic way it represents a significant issue.

Frequency instabilities result in the period of the signal changing by small amounts from cycle to cycle but with some well defined mean frequency value. This is referred to as jitter and an example is shown in figure 5.10(c). Jitter can be characterized in different ways such as Period Jitter which is a measure of the deviation from the mean signal period, and Cycle-to-Cycle Jitter which describes the maximum change in period from one cycle to the next[3].

In the context of range measurements, jitter will be manifested as random noise in the estimated distance, however, since it distributed around a mean frequency value it can be mitigated by simple averaging. For a well designed crystal oscillator, the cycle-to-cycle jitter can be as low as a few picoseconds, however, for many programmable oscillators it may exceed 100 ps[3].

5.2.3.2 Multipath Distortion

Multipath refers to the fact that a given signal may reflect off different surfaces before reaching its destination. This means that the signal moves through different paths with different lengths in addition to the direct one as is illustrated in figure 5.11.

This is a big issue when using the propagation time to determine the direct line-of-sight

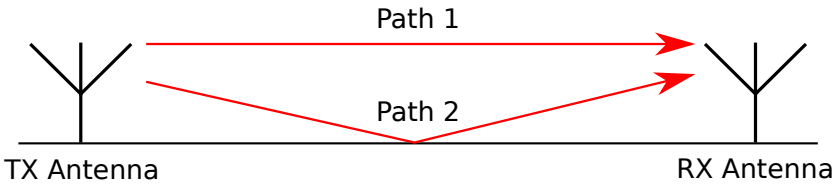


Figure 5.11: This figure illustrates the issue of multipath. The signal emitted from the transmitting antenna takes two distinct paths to the receiving antenna: path 1 goes directly between the two antennas, while path 2 reflects of the ground before reaching the target.

distance as the multiple paths will superimpose signals with different travel times on top of each other resulting is a wrong distance estimate. The exact amount of distortion that results from multipath depends on a lot of factors such as signal wavelength and the geometry and material of any objects in the vicinity of the transmitter-receiver pair. As such, it is difficult to give an estimate on the magnitude of the impact this may have on measurements.

One way to mitigate multipath is to have a circularly polarized signal. When such a signal is reflected of a surface the polarization switches direction which will drastically reduce the amount of noise caused by signals reflected an odd number of times. Of course, at the second, fourth and so on reflections the issue reappears, but since the first-reflection signal has much more energy that the subsequent reflections, this method is quite beneficial[7]. Other methods include advanced statistical analysis of the received signal, but in general, it is best to simply avoid antenna configurations that are prone to multipath in so far as is possible.

5.2.3.3 CDMA Noise

Because Gold codes are close to orthogonal to each other, and hence have low cross-correlations, any CDMA signals that are present in addition to whichever one is to be detected will appear as (pseudo-)random noise. As such, having multiple CDMA transmissions operating simultaneously will drastically reduce the SNR of the wanted signal therefore making detection less reliable.

Increasing the length of the PRN codes increases the hight of the auto-correlation peak for any given signal which means that this issue of added noise can be reduced by using longer codes. In addition, CDMA is by its nature very robust to noise why this issue is not a big hindrance. In fact, the individual signal levels of the GPS system are significantly lower than the thermal noise of most receivers, but can still be reliably extracted[7].

5.3 RF Design

5.3.1 Radio Basics

An arbitrary radio signal can be described by the following equation

$$s(t) = A(t)\cos(2\pi f_c t + \phi(t)) \tag{5.23}$$

where $A(t)$ is the amplitude of the signal, f_c is the carrier frequency, t is time, and $\phi(t)$ is a phase offset. Information can be encoded onto the signal by time varying either the amplitude, the phase or a combination thereof[15]. Alternatively, the signal can be expressed by the equivalent equation

$$s(t) = I(t)\cos(2\pi f_c t) - Q(t)\sin(2\pi f_c t) \tag{5.24}$$

with $I(t) = A(t)\cos(\phi(t))$ and $Q(t) = A(t)\sin(\phi(t))$. $I(t)$ and $Q(t)$ correspond to the real, or in fase, and imaginary, or quadrature, part of the complex baseband modulation function $b(t) = A(t)e^{\phi(t)}$. If $s(t)$ is mixed with a signal $\sin(2\pi f_c t)$ and low pass filtered, the result will be the real part of the baseband signal, while mixing with $\cos(2\pi f_c t)$ will result in the imaginary part. Conversely, if $I(t)$ is mixed with the sine function and $Q(t)$ with the cosine function, the sum of the two results will be $s(t)$. This means that any type of modulated radio signal can be produced by varying the amplitudes of the I- and Q-channel of the baseband signal without having to deal with generating high frequencies with varying phases. Furthermore, a received signal can be sampled and processed at baseband resulting in much simpler hardware than if everything had to be done at the carrier frequency[15].

A basic single antenna radio transceiver has five distinct components, the antenna, the duplexer (or switch), the *low noise amplifier* (LNA), the *power amplifier* (PA), and the mixer[15]. If any signal generation and processing is to be performed digitally, *analog-to-digital converters* (ADC) and *digital-to-analog converters* (DAC) are also needed. Figure 5.12 shows how these elements are combined to form a simple quadrature transceiver with digital in- and outputs.

When receiving, the antenna switch connects the antenna to a bandpass filter followed by the LNA. The signal is thus amplified before it is mixed with sine and cosine waves generated by a local oscillator. The resulting two signals are then low pass filtered removing the carrier frequency resulting in the real and imaginary parts of the baseband signal. Two *variable gain amplifiers* (VGA) are then used to adjust the signal level before it is converted to the digital domain through ADCs.

When transmitting the opposite process occurs. The digital baseband data is converted to the analog domain by means of DACs before being amplified and filtered to condition the signal. It is then mixed with sine and cosine waves to up-convert the I and Q data

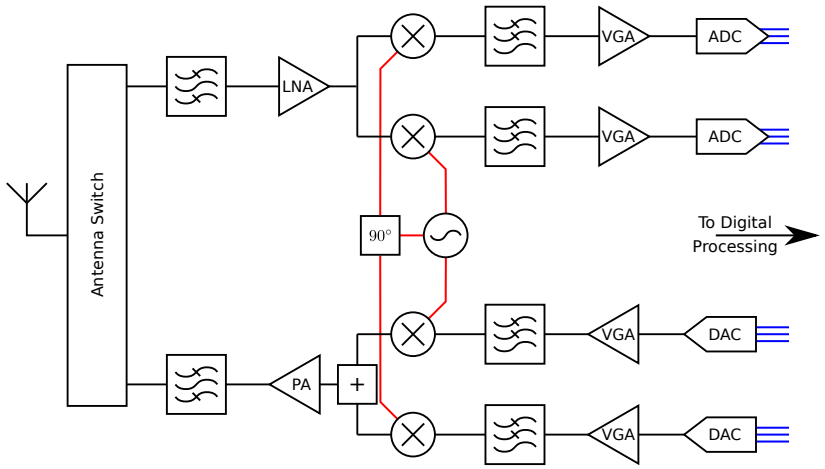


Figure 5.12: The main components of a quadrature radio transceiver.

to the carrier frequency before they are added together to form the radio signal. This is then amplified in a power amplifier, filtered and sent though the duplexer to the antenna where it is transmitted[15].

If a radio signal is sent from one transceiver to another, the power of the received signal will be equal to

$$P_R = \frac{P_T G_T G_R}{L_P L_A} \tag{5.25}$$

where P_T is the transmit power, G_T is the transmitter antenna gain, G_R is the receiver antenna gain, L_P is the path loss, and L_A are additional losses caused by signal attenuation, resistive dissipation and similar. The path loss can be calculated as

$$L_P = \left(\frac{4\pi df}{c}\right)^2 \tag{5.26}$$

with d being the distance between the transceivers, f the frequency of transmission and c the propagation speed[15]. When the signal arrives at the reception antenna a certain SNR will be associated with it. Every time the signal travels through a component (such as an amplifier or mixer) the SNR will be degraded to some extent. The amount of change is characterized by the *noise figure* (NF) which is defined as

$$F = \frac{SNR_{in}}{SNR_{out}} \tag{5.27}$$

and will always be smaller or equal to 1[15]. This degradation of signal integrity can also be classified by the *noise temperature* which is the temperature of a theoretical resistor

placed at the input resulting in the same amount of noise power. The noise figure and temperature are related by

$$T = T_0(F - 1) \Leftrightarrow F = 1 + \frac{T}{T_0} \tag{5.28}$$

where T is the noise temperature and T_0 is defined as 290 K. The power associated with a given noise temperature is given by

$$P_N = kBT \tag{5.29}$$

with $k = 1.38 \times 10^{-23}$ J/K being Boltzmann's constant and B the noise bandwidth[15]. The noise temperature can also be used to model the amount of noise that is received from the antenna due to environmental factors which provides a simple method to estimate the total noise power after a signal has propagated though receiving hardware. Defining T_A and T_R as the noise temperature of the antenna and the receiver, respectively, the total noise temperature of the system is simply

$$T_{sys} = T_A + T_R. \tag{5.30}$$

Combining the above equations, the SNR for a given radio link can thus be estimated as [15]

$$SNR = \frac{P_T G_T G_R}{L_P L_A k B T_{sys}}. \tag{5.31}$$

5.3.2 Carrier Recovery

An often used type of modulation is *binary phase-shift keying* (BPSK) which enables the transmission of binary data by having 1 bits correspond to a certain phase (often 0°) and 0 bits offset by 180°. This is especially simple to implement as in the baseband this corresponds to shifting the I-channel between 1 (1 bit) and -1 (0 bit), while the Q-channel is kept at 0.

Even in the most well designed radio systems, there will almost always be some small difference between the oscillator of the transmitter and the receiver. This gives rise to small offsets in frequency which means that after the receiver has demodulated the signal it will still have some modulation left albeit at a significantly lower frequency than the carrier. In order to extract data from the signal, this issue must be overcome. A commonly used method to achieve this is the Costas loop[7].

A diagram showing the components of a digital Costas loop is displayed in figure 5.13. The raw *intermediate frequency* (IF) signal is mixed with a local sine and cosine wave generated by a *numerically controlled oscillator* (NCO) (in an analog system this would

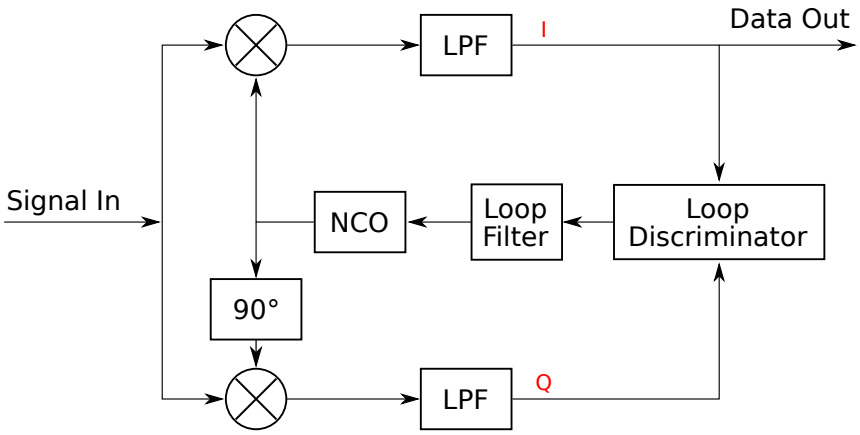


Figure 5.13: Diagram showing the components of a Costas Loop.

be a voltage controlled oscillator instead) and low pass filtered resulting in separate baseband I and Q channels. These are then sent through a *loop discriminator* which feeds into the *loop filter* that in turn controls the frequency of the NCO.

The purpose of the loop discriminator is to transform the raw IQ signals into a measure of the phase offset ϕ between the NCO and the modulation. The most accurate discriminator is $\tan^{-1}\left(\frac{Q}{I}\right)$ which is equal to the instantaneous phase difference, however, it is a fairly processing intensive function. Because of this, $I \cdot Q$ and $\text{sign}(I) \cdot Q$ are sometimes used which are proportional to $\sin(2\phi)$ and $\sin(\phi)$, respectively. It should be noted that these discriminators are designed to work with BPSK data where all the information is in the I channel, for other modulation types other discriminators are available[7].

The loop filter is commonly implemented as a *proportional integral* (PI) filter that converts the phase difference into NCO frequency. Such a filter needs tuning to work properly which can be a complicated process which depends on the exact nature of the oscillators[7].

If the signal has already been down-converted to baseband IQ data, the Costas loop has to be modified slightly. Figure 5.14 shows such a modified loop where the two mixers have been replaced by a complex multiplier that ”rotates” the input signal according to the phase difference. Because of this, the low pass filters are not necessary as there is no sum frequency to be filtered out as is the case when using mixers, but they can still be used for noise reduction purposes.

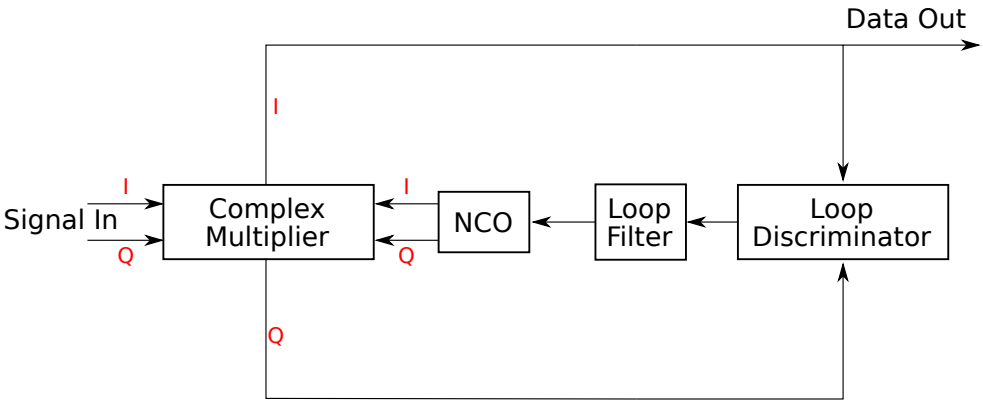


Figure 5.14: Diagram showing the component of a Costas Loop modified for use with complex baseband data.

5.3.3 Hardware Design

When working with electronics that operate in frequencies significantly above *direct current* (DC), such as is the case for RF components, care must be taken when designing circuits as the characteristic impedance of the current carrying elements, i.e. *transmission lines*, becomes an important factor.

If a non-DC signal is sent through a transmission line with a characteristic impedance of Z_0 which is terminated in a component with an input impedance of Z_L, a certain amount of energy will be reflected back into the transmission line. This has the effect of distorting the input signal and reducing the power throughput, which should be avoided whenever possible. The ratio of the voltage of the reflected signal to that of the input signal is given by

$$\Gamma = \frac{Z_L - Z_0}{Z_L + Z_0} \tag{5.32}$$

and is, in general, a complex value with magnitude ≤ 1[8]. As can be seen, this reflection coefficient goes to zero when the impedance of the transmission line approaches that of the load. Because of this, RF circuits should be designed with impedance matching in mind.

A common conduit for high frequency signals is the strip line which can be easily realized on *printed circuit boards* (PCB). It consists of a narrow conducting signal plane that is separated from one or two conducting reference planes by a dielectric. Various different strip line configurations are used for circuit design[10] with the simplest being the surface microstrip line which is shown in figure 5.15(a).

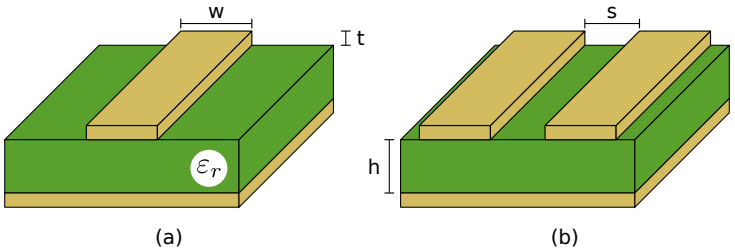


Figure 5.15: Overview of selected strip lines. (a) is a simple surface microstrip line while (b) is an edge-coupled surface microstrip line.

As can be seen, this is effectively the same as an ordinary PCB signal trace above a ground plane. The characteristic impedance of such a transmission line is given by

$$Z_0 = \frac{\eta_0}{2\pi\sqrt{2}\sqrt{\varepsilon_{r,eff}+1}} \ln\left(1 + 4\frac{h}{w'}\left[4\frac{h}{w'}\left(\frac{14\varepsilon_{r,eff}+8}{11\varepsilon_{r,eff}}\right) + \sqrt{16\left(\frac{h}{w'}\right)^2\left(\frac{14\varepsilon_{r,eff}+8}{11\varepsilon_{r,eff}}\right)^2 + \frac{\varepsilon_{r,eff}+1}{2\varepsilon_{r,eff}}\pi^2}\right]\right) \tag{5.33}$$

where h is the height of the dielectric, $\eta_0 = 119.9169832\pi\ \Omega$ is the free space impedance, $\varepsilon_{r,eff}$ is the effective relative permittivity and w' is the effective signal line width which is given by

$$w' = w + \frac{t}{\pi}\ln\left(\frac{4e}{\sqrt{\left(\frac{t}{h}\right)^2 + \left(\frac{t}{w\pi+1.1t\pi}\right)^2}}\right)\left(\frac{\varepsilon_{r,eff}+1}{2\varepsilon_{r,eff}}\right) \tag{5.34}$$

with w and t being the physical width and thickness of the trace, respectively. The effective relative permittivity can be calculated as

$$\varepsilon_{r,eff} = \begin{cases} \frac{\varepsilon_r+1}{2} + \frac{\varepsilon_r-1}{2}\left[\sqrt{\frac{w}{w+12h}} + 0.04\left(1-\frac{w}{h}\right)^2\right], & \frac{w}{h} < 1 \\ \frac{\varepsilon_r+1}{2} + \frac{\varepsilon_r-1}{2}\sqrt{\frac{w}{w+12h}}, & \frac{w}{h} \geq 1 \end{cases} \tag{5.35}$$

where ε_r is the relative permittivity of the dielectric[10]. This type of strip line works well for single ended signals, however, for differential signals it must be extended to the edge-coupled surface microstrip as seen in figure 5.15(b). In this case the differential impedance becomes the important factor, which is given by

$$Z_{diff} = 2Z_{0,o} \tag{5.36}$$

with $Z_{0,o}$ being the odd-mode impedance. This parameter can be estimated as

$$Z_{0,o} = \frac{Z_0\sqrt{\frac{\varepsilon_{r,eff}}{\varepsilon_{r,eff,o}}}}{1 - \frac{Z_0}{\eta_0}Q_{10}\sqrt{\varepsilon_{r,eff}}}. \tag{5.37}$$

The parameters $\varepsilon_{r,eff,o}$ and Q_{10} are calculated through a total of 17 different equations, which are included in appendix B. In addition to the impedance considerations, care should also be taken when designing RF circuitry making sure that the design adheres common highspeed electronics guidelines. This includes, but is not limited to, proper decoupling of power traces, using large radius turns of the transmission lines when bends cannot be avoided, having separate ground planes, and making sure that different *alternating current* (AC) signals are isolated from each other so as to avoid crosstalk[5].

CHAPTER 6

Initial Validation

Based on the results of the pre-project it would seem that visible light is not ideal as a transmission medium. It was therefore decided that another solution should be identified and proven to be suitable before the development of the actual navigation system could begin.

6.1 Hardware Selection

The other obvious mean of transmission that was initially avoided due to the relative complexity in designing the hardware is, of course, electromagnetic waves in the radio spectrum. As such, an investigation into what, if any, ready-made hardware was available that could be used in the context of this project was initiated. After a thorough scanning of the market it was decided that the 2.4 GHz to 2.5 GHz *industrial, scientific, and medical* (ISM) band is the best option regarding the frequency range. This is first of all because this is an open band requiring no license to operate in. Secondly, since it is the frequency range that is utilized by both Wifi and Bluetooth a multitude of COTS hardware is available. Of course, this also means that the band is heavily polluted in urban environments, however, since this project is designed to be operated in space, it should not prove an issue.

The search for ready-made RF transceivers found that the available hardware mostly fall into three categories (see figure 6.1):

1. Complete RF modems
2. Transceiver Front-ends
3. Wideband Programmable Transceivers

The complete RF modems include everything that is needed to establish a functional RF connection. Many of these even include packet handling modems and are basically operated as simple serial connections. Most operate according to either Bluetooth or Wifi specifications and are as such not suitable for this project. However, a subset allow for what different manufacturers call ”Smart RF”, ”Proprietary RF” or ”Direct RF” which enables the use of custom communications protocols. These include Analog Devices ADF7242[20], Nordic nRF2401[30] and several others that might seem to work in

the context of this project. Unfortunately, none of them allow for exact control over the clock signals that govern transmission and reception individually. As such, it would not be possible to implement fractional sampling severely limiting the accuracy achievable with such devices forming the base of the project.

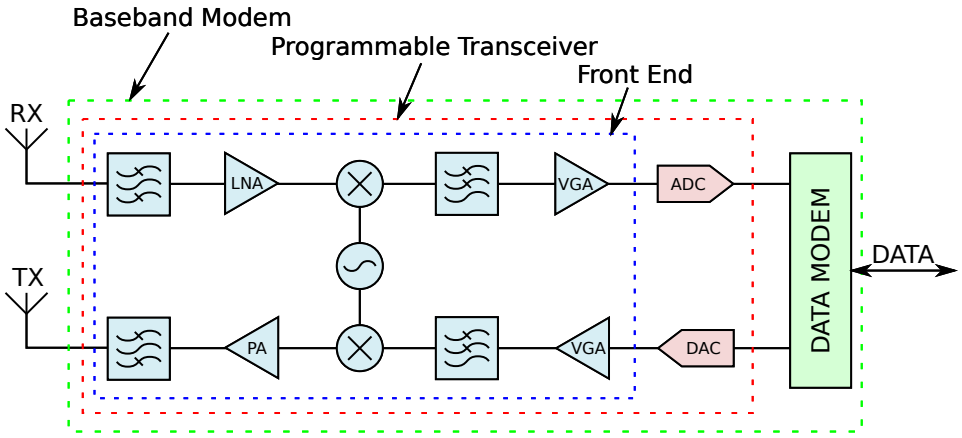


Figure 6.1: A block diagram showing the basics of a baseband transceiver. The complex signal paths have been omitted for simplicity. The colored dotted lines indicate what is included in various types of ready made components.

The transceiver front-ends combine amplifiers, signal conditioning and mixers in a single package which enables complete control over the signal path, an example being the Maxim Integrated MAX2830[1] chip. However, in order to have a complete transceiver system these chips need to be paired with ADCs, DACs and matching circuits which would require considerable amounts of time to implement. So, while these kinds of chips could potentially be used for the final implementation, they were not suitable for this initial feasibility test.

Finally, the wideband programmable transceivers, referred to as *field programmable radio frequency* (FPRF) or agile transceivers by different manufacturers, are similar to the complete RF modems except they do not include the packet handler, but instead output raw IQ data for analysis on external hardware. Furthermore, as they are primarily designed to function as the hardware section of *software defined radios* (SDR), they work over very large frequency intervals often from hundreds of megahertz to several gigahertz and allow for complete control over the timing of both reception and transmission. The downside is that they are quite expensive compared to the other solutions, often costing tens to several hundreds of Euros compared to the sub € 20 price tag of the alternative solutions. In addition, the extremely wideband nature of these chips is not needed in this project, why the programmable transceivers are somewhat overqualified. Examples include the Lime Microsystems LMS6002D[23] and the Analog Devices AD9361[29].

Based on these considerations it was chosen to acquire a wideband programmable

transceiver for the initial verification of using RF signals and, if this proved to be viable, build a custom solution based on a transceiver front-end for the actual system development. The development boards available for programmable transceivers are too prohibitively expensive to be used in this project, but fortunately a series of SDR boards are available which include both the programmable transceiver as well as the hardware (often *field-programmable gate arrays* (FPGA)) needed to interface with these. Four such SDR boards were identified namely the Ettus Research USRP B200[28], the Lime Microsystems LimeSDR[22], the Nuand bladeRF[25], the Great Scott Gadgets HackRF[16] and the UmTRX UmTRX 2.2[14]. A comparison of these is shown in table 6.1.

Table 6.1: Comparison of the capabilities of the various software defined radio boards[22][14].

	USRP	**LimeSDR**	**bladeRF**	**HackRF**	**UmTRX**
Frequency Range [GHz]	0.070 - 6	.0001 - 3.8	0.3 - 3.8	0.001 - 6	0.3 - 3.8
RF Bandwidth [MHz]	61.44	61.44	40	20	1000
Sample Depth [Bits]	12	12	12	8	12
TX Channels	1	2	1	1	1
RX Channels	1	2	1	1	1
Duplex	Full	Full	Full	Half	Full
Oscillator Precision [ppm]	2	1	1	20	0.1
TX Power [dBm]	10+	10	6	15	17
Price [€]	650	285	395	285	895

As can be seen, the HackRF is only half-duplex which means that it is not possible to test transmission and reception simultaneously resulting in the need for at least two devices in order to test the measurement principle. In addition, it is inferior to the other models without being cheaper severely limiting its usefulness. The USRP B200 and the UmTRX are both significantly more expensive than the other devices and provide no significant benefits in the context of the project. The LimeSDR and the bladeRF have similar specifications with the LimeSDR having a few significant advantages such as having multiple channels and a higher bandwidth while also being cheaper.

From this information the LimeSDR is clearly best fit for this project. Unfortunately, at the time when the SDR board had to be procured the limeSDR would not be shipping for another month, it was therefore chosen to go with the bladeRF board instead. This is not very problematic as the two boards have similar feature-sets with the limeSDR ultimately winning mainly on account of its cheaper price point.

6.2 Configuration

With a suitable early validation solution identified it was quickly procured. A larger than expected amount of time was spent on configuring the bladeRF device before tests could be performed to determine whether or not the measurement method would work in practice.

The bladeRF board consists of the Lime Microsystems LMS6002D programmable transceiver chip that is connected to an Altera Cyclone IV FPGA which, in turn, is connected to a host *personal computer* (PC) through a Cypress FX3 *universal serial bus* (USB) controller. In addition, a Silicon Labs Si5338 Clock Generator, fed by a *voltage controlled temperature compensated crystal oscillator* (VCTCXO), generates the timing signals associated with data sampling as well as various auxiliary clock signals. In the standard setting, the Cyclone IV is configured with an embedded Nios II micro controller that takes care of the serial interfaces used for programming both the Si5338 and the LMS6002D. A block diagram of this system is displayed in figure 6.2.

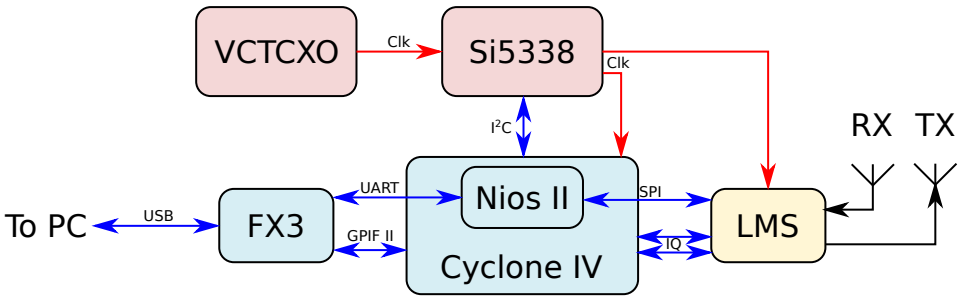


Figure 6.2: A block diagram showing the different components of the bladeRF and how they communicate with each other.

In order for the bladeRF to be utilized for fractional sampling the Nios II processor, the Si5338, the LMS6002D and the Cyclone IV all needed to be reprogrammed to various extends. The bladeRF software is all open source and is available under the MIT license, which makes the programming relatively simple as the existing code can be used as a starting point. However, due to the complexity of the system, a great deal of time still had to be dedicated to the configuration as the exact nature of the system had to be understood in detail before the reprogramming could be achieved (cf. chapter 3).

Using the full duplex capabilities of the bladeRF, the system was set to transmit a 32767 bit Gold code at a chirp rate of 3276700 Hz, equaling a transmission time of 10 ms for the full code, while simultaneously receiving data at a rate of 3276600 Hz. A frequency of 2.448 GHz was used for the carrier which was modulated using basic BPSK. The code generation was performed in hardware on the FPGA while the received data was transferred to a computer over USB for later analysis, utilizing a *first in first out* (FIFO) buffer to accommodate the different clock frequencies of the FPGA and the PC.

On the PC side, the data was recorded in 100000 sample sections, encompassing three total code transmissions, and loaded in to MATLAB® for the actual analysis. The starting bit of each code transmission was found through cross-correlation with a local copy of the emitted signal after which the cumulative dot product was calculated to find the fractional delay.

6.3 Results

The bladeRF has two *subminiature version A* (SMA) antenna plugs, one for reception and one for transmission, which are located approximately 5 cm from each other. Standard commercial Wifi antennae were connected directly to these outlets without any extension cables. The measurements made therefore only represent this relatively short distance. However, as the purpose of these early experiments was simply to validate the measurement principle, the exact distance between the antennae is not critical and the effects of varying distances will be analyzed in detail later.

Figure 6.3 shows the cumulative dot products calculated from three PRN cycles during a single recording.

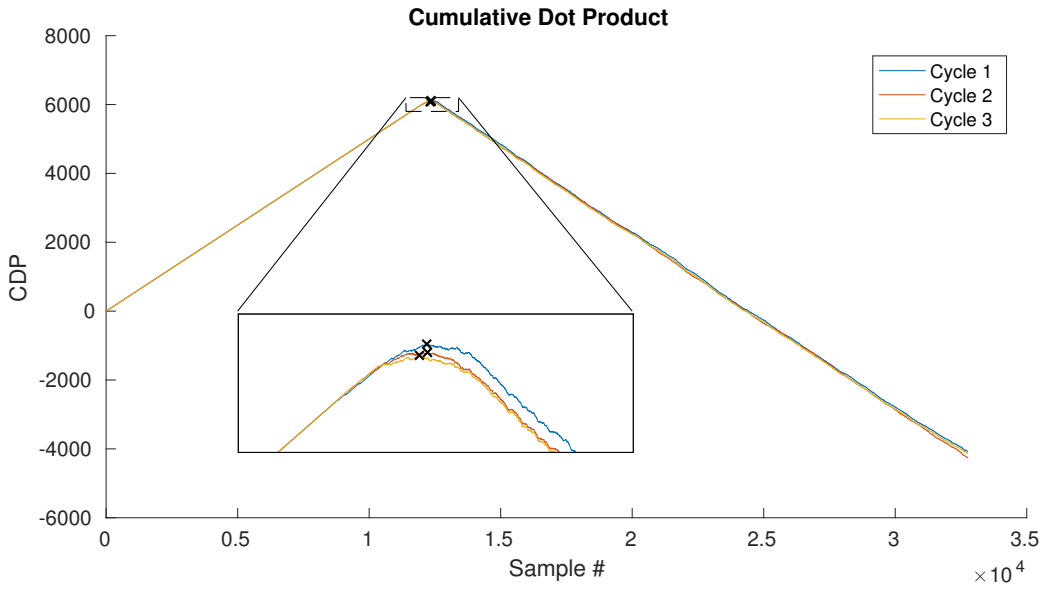


Figure 6.3: The cumulative dot product as calculated over three PRN cycles using the bladeRF. The maximum difference between any two peaks, indicated by the black crosses, is equal to 50 samples corresponding to a distance difference of 14 cm.

As can be seen, the CDP appears to forms a well-defined peak representing the fractional delay, however, when magnifying the area around the peak it is evident that the peak is

not as sharp as theory would predict. This is not unexpected as external noise, such as Wifi communication, is present in the measurements. It is worth comparing this graph with a similar plot that was obtained using the optically based transmission hardware that was developed in the pre-project as is reproduced figure 6.4 for convenience.

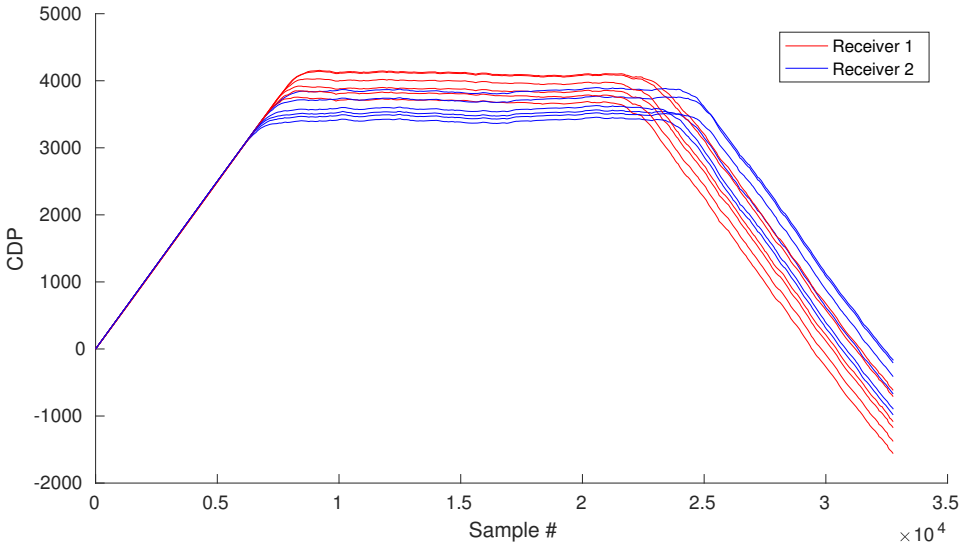


Figure 6.4: The cumulative dot product calculated over five cycles using the hardware developed in the pre-project. The inter-transceiver distance was 5 cm[9].

It is clearly evident that while the optical solution completely fails to provide usable data, the RF based solution produces a very well defined peak that indicates the fractional offset. For the three PRN cycles that are shown in figure 6.3 there is a variation between the detected peaks of up to $\approx$ 50 samples, corresponding to a distance variation of $\approx$ 14 cm. While this is clearly not ideal, it represents the very fist data before any real calibration has been performed, why it can likely be improved significantly. Furthermore, this only represents three cycles of the signals and averaging over several additional cycles will most likely result in a much more precise measurement. Increasing the bit rate or the length of the PRN code can also increase the precision to extends that are only limited by the exact configuration of the hardware.

It should be noted that results similar to the ones obtained using radio signals could probably be realized using optical signals with more careful hardware design. The real reason that RF is preferable is that optical systems have a quite limited range unless the light is focused into very narrow beams, which is undesirable in regards to this project which aims to have the transceivers be as omni-directional as possible.

Based on these results, especially the stark contrast between the measurements made with the bladeRF and the pre-project hardware, it was concluded that the fractional sampling method showed promise and that it was worth pursuing the development of a fully fletched relative navigation system based upon this technology.

CHAPTER 7

Design

This section describes a reference design of a relative navigation system based on time of flight measurements obtained using fractional sampling of CDMA signals.

7.1 Overall System Configuration

To unambiguously determine the relative navigation parameters between two vehicles using time of flight measurements, at least three transceivers must be placed on each vehicle. However, it is not possible to place three points on the surface of a 3D object and have all three points be visible regardless of orientation, as the vehicle will occlude some points for some rotations. This issue can be overcome by adding one additional transceiver and placing the transceivers at the vertices of a tetrahedron. For such a configuration, at least three transceivers will always be visible regardless of orientation. Naturally, this implies that each transceiver is outfitted with an omni-directional antenna. Based on this principle, figure 7.1 shows an overview of the major components that make up a complete navigation system.

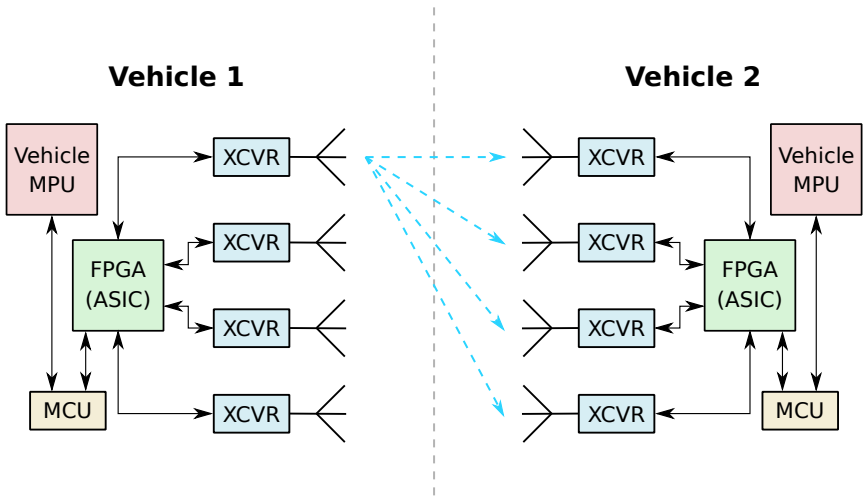


Figure 7.1: An overview of the major components of the navigation system.

Each vehicle has four transceivers which convert between radio signals and digital baseband data. The baseband data consists of CDMA modulated navigtion information that is generated and processed on an FPGA, or potentially an *application specific integrated circuit* (ASIC). The FPGA computes the signal transmission delay and feeds this information to a *micro controller unit* (MCU) that combines the delay measurements from all four transceivers and computes the relative navigation parameters. This information is then passed on to the *main processing unit* (MPU) of the vehicle where it can be used for control purposes or whatever the specific application requires.

The reason that the signal generation and processing is not performed on the MCU as well is that it requires high clock rates to perform in real time which would put stark requirements on the micro controller. Furthermore, many of the processing steps can be parallelized which makes then well suited to be implemented in hardware rather than software.

The radio signal is transmitted on the 2.4 - 2.5 GHz band using BPSK modulation. BPSK modulation is used as it is robust to noise and relatively simple to implement compared to other more advanced modulation schemes[15]. The choice of frequency comes from the fact that it is part of the ISM band which means it can be operated without a license. Furthermore, it provides a decent compromise between the long transmission range of low frequencies and small antenna size of high frequencies. At $\approx$ 2.4 GHz the wavelength is 12.5 cm meaning that even simple dipole antennas can be made fairly unobtrusive. In addition, since 2.4 GHz is widely used for terrestrial purposes (WiFi, Bluetooth, etc.) hardware is easy to obtain and relatively inexpensive.

The baseband data is transmitted at a rate of 100 Hz and modulated using Gold codes with a chirp rate of 3.2767 MHz and a length of 32767 bits. This configuration means that each data bit corresponds to one full transmission of the Gold code which makes processing simpler. The received signal is sampled at 3.2766 MHz by accord of equation 5.16.

7.2 Signal Processing

The processing of the raw baseband information can be separated into two parts: initial acquisition that is performed at first contact between the vehicles, and continuous tracking that performs the actual delay estimation. An overview of this is shown in figure 7.2.

During initial acquisition nothing is known about the target, not even what transceivers are visible. Therefore a blind search is performed by recording 32766 samples (corresponding to 10 ms of data) for each transceiver and cross-correlating this data with copies of the Gold codes used for each transceiver. To decrease the computational expense, the cross-correlation is performed by first transforming the recorded data to the spectral domain by means of a *fast Fourier transform* (FFT), multiplying the result with the complex conjugate of the Gold code spectrum, and then transforming back to the

time domain using an *inverse fast Fourier transform* (IFFT). If a signal from a given transceiver is present a sharp peak will appear in the resulting correlation function at the index corresponding to the integer time offset.

In principle, a search of signal frequency should also be performed to account for oscillator offsets as is done in most GPS receivers. However, the frequency search is performed mainly to account for Doppler-shifts caused by the high relative velocities of the satellites, but if the offset is relatively small, i.e., less than 1000 Hz this is not strictly necessary as the cross-correlation will still provide a strong peak. Therefore this step is not needed for the present navigation system.

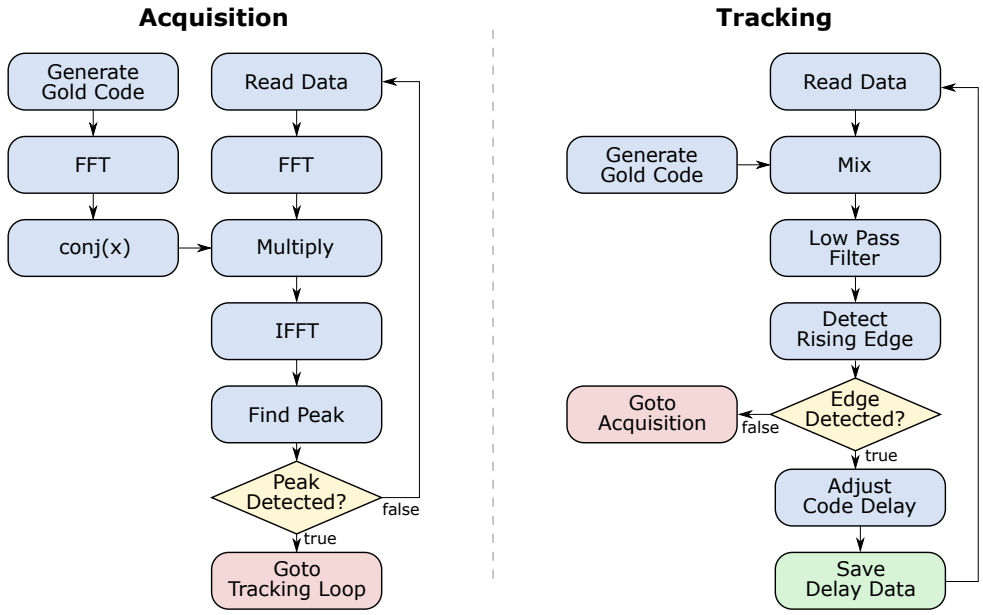


Figure 7.2: Flowchart of the basic processing steps in determining delay.

With the initial integer delay determined, the system moves on to continuous tracking which determines the fractional offset through mixing with the corresponding Gold code. Because of the fractional sampling the peak in the cross-correlation will actually be split up into to two adjacent samples with the distribution of power depending on the exact fractional delay. This can potentially lead to ambiguities when determining the fractional offset using the CDP method, however, this is not an issue when performing pulse reconstruction as this method allows for the integer delay to be off by up to ± 1 sample. If the fractional offset moves to the extremes, i.e., close to 0.0 or 1.0, the integer delay is decremented or incremented accordingly ensuring that the signal stays locked. Regardless, should the lock on the target be lost, the systems returns to the acquisition process until a new lock is achieved.

In addition to determining the signal delay, the tracking process also estimates the phase and frequency of the carrier wave through the means of a Costas loop. This information

is used to remove any potential offsets as well as extracting the navigation data that is superimposed on the Gold codes.

7.3 Navigation Loop

Once the signal delay has been determined through the signal tracking it is sent to the MCU for further processing. The steps involved in this are outlined in figure 7.3.

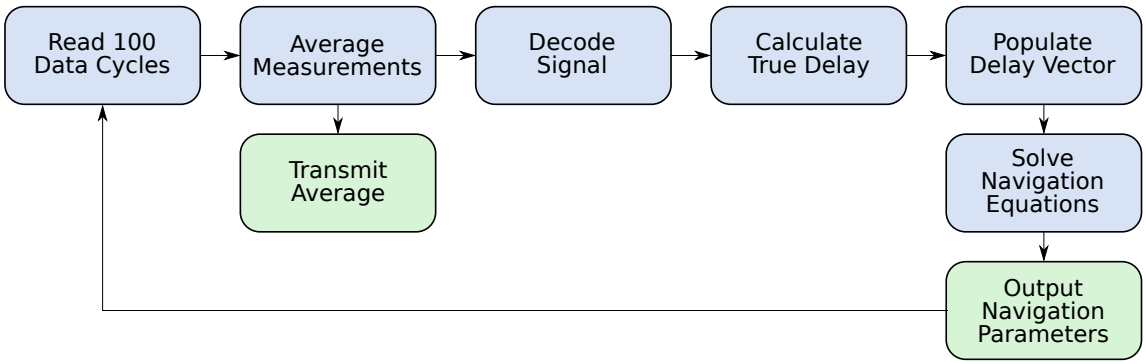


Figure 7.3: Flowchart describing the major steps of the navigation loop.

First, the determined delay is used to update a running average that is sent to the FPGA for transmission to the other vehicle. Only one delay value is transmitted per 100 Gold code cycles (see section 7.4) it therefore makes sense to simply average all the measurements until a new one is transmitted.

The incoming navigation data is translated and combined with the running average to generate the true propagation time between each transceiver pair. This is then converted to spatial distances and used to calculate the navigational parameters of the vehicles. The parameter estimation is performed by a least squares solution of the equations described in section 5.1. During this computation it is assumed that the vehicle performing the calculations is located at position $(0, 0, 0)$ with an identity rotation matrix, thus the solution provides the navigational parameters of the other vehicle within the coordinate frame of the current vehicle.

7.4 Communications Protocol

Since it cannot be guarantied that the two vehicles are time synchronized, two way communication is needed to ensure that the time offset can be estimated. Figure 7.4 shows the data protocol that is used to transmit distance information between individual transceivers.

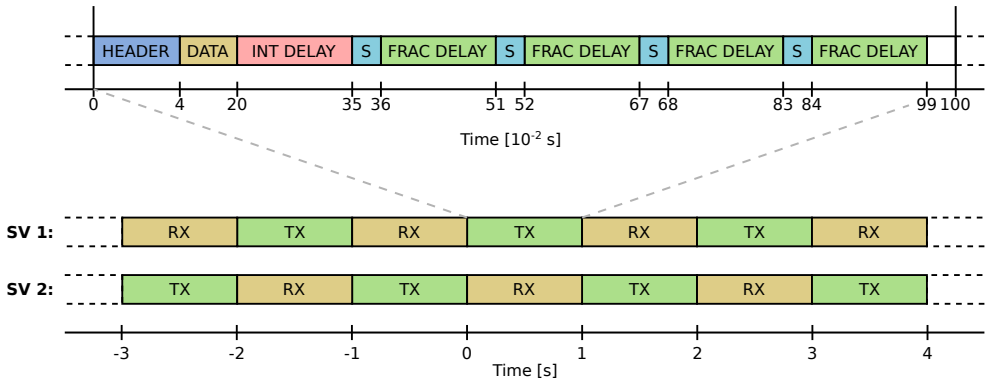


Figure 7.4: Overview of the inter-transceiver communications protocol. SV 1 and SV 2 refer to individual (space) vehicles.

At any given time all the transceivers of a single vehicle are performing the same operation whether transmitting or receiving. Each vehicle transmits for 1 second followed by 1 second of receiving with the two vehicles synchronized such that one is receiving while the other is transmitting. Of course, this means that some synchronization procedure is needed for the initial signal acquisition, however, this should be trivial to implement.

With a data rate of 100 Hz, 100 bits of information can be emitted during a single transmission. These 100 bits are different for each transceiver and are defined accordingly:

Bits 0-3 make up a 4 bit header of binary number 1100 that is used to resolve phase ambiguities at the receiving end;
Bits 4-19 are 16 bits of application specific information, which, since at least three transceivers are always in view, results in up to 6 bytes of information per transmission;
Bits 20-34 consists of the integer delay, that is, the number of whole chirps the signal is delayed. 15 bits are used for this number as this is the maximum unambiguous delay when using 32767 bit Gold codes;
Bit 35 is the sign of the fractional delay that the transmitting transceiver has calculated relatively to XCVR 1 of the currently receiving vehicle, while bits 36 to 50 are the magnitude of the fractional delay as a number of 0-32765;
Bits 51 to 66 contain the same information about XCVR 2 of the receiving vehicle;
Bits 67 to 82 that of XCVR 3; Bits 83 to 98 that of XCVR 4;
The final bit, 99, is not actually transmitted as this time period is used to switch transceiver mode from transmit to receive. This is needed as the transceivers cannot

switch instantaneously on account of using the same antenna for transmission and reception.

This transmission scheme only works for two vehicles. If the system is to be extended to work with more vehicles, the protocol also has to be extended. Bits 4 to 19 could be used to provide vehicle ID instead of arbitrary data, however, more advanced time allocation would also be needed to ensure that all vehicles receive data from all other vehicles.

CHAPTER 8

Implementation

To test the validity of the designed system, elements of different subsections have been implemented and evaluated. This includes the development of custom transceiver hardware that supports fractional sampling as well as a processing system, based on a mixture of hardware and software that is capable of transforming the raw signal measurements into delay data. Furthermore, simple simulation software has been developed that can be used to evaluate overall system performance based on the characteristics of the prototype subsystem.

The source code for both the processing system and the simulation software is available for download on a GitHub repository with more information being included in appendix D.

8.1 Transceiver Hardware

During the initial validation process of this project it was found that very few ready made transceivers on the market support fractional sampling, i.e., having different sampling rates for transmission and reception. The hardware that did support this generally came in the form of software defined radios that had many unnecessary features making them both bulky and prohibitively expensive. A custom solution therefore had to be implemented.

8.1.1 Initial Design

To keep the hardware development relatively simple it was decided to base the design of the custom solution on a ready-made radio front-end chip. This type of component includes everything needed to convert a radio signal into analog IQ baseband data. Therefore it needs to be combined with an ADC a DAC and a clock source so it can be interfaced with a digital processing system. Figure 8.1 an overview of the transceiver design.

For the front-end the choice fell on the Maxim Integrated MAX2830 RF transceiver chip which works in the 2.4 - 2.5 GHz range, has a bandwidth of up to 18 MHz and a transmission power of up to 17.1 dBm[1]. Furthermore, this chip, on the contrary to many others on the market, has a built-in antenna switch meaning that transmission

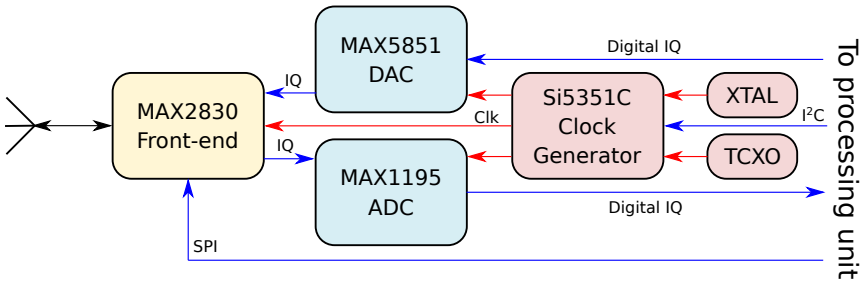


Figure 8.1: An overview of the components that make up the transceiver hardware. A complete schematic is included in appendix C.

and reception can use the same antenna. Without this ability, the navigation equations would be significantly more complicated as the signals would not travel the same length when going one way relative to the other. The MAX2830 chip can be programmed using a *serial peripheral interface* (SPI) bus meaning that it can be controlled using only a few *input-output* (IO) lines which is preferable in situations where the number of IO pins on the digital control system is limited. Both the radio signal and the baseband connections are implemented as differential pairs putting special constraints on the interfacing hardware.

The differential radio IO port of the MAX2830 must be connected to a device that can transform the signal from balanced to unbalanced so that it can be connected to an antenna. This is achieved by a simple 2.4 GHz impedance matched balun. For the antenna, a standard Wifi quarter wavelength dipole is used attached to the transceiver through a SMA connector. Such an antenna has a toroidal gain pattern, as can be seen in figure C.6 of appendix C, which is not ideal for a complete navigation system, but should be adequate for testing purposes.

For the analog to digital converter, the choice fell on the Maxim Integrated MAX1195 ADC chip which allows for data rates of up to 40 *mega samples per second* (Msps) has a differential input and an 8 bit parallel output. This device was chosen as it is designed for baseband radio sampling and is made by the same company as the front-end chip resulting in easy connection of the two. In addition, for CDMA a high resolution sampling of the signal is not necessary why 8 bits should be more than sufficient[7]. It also, again, limits the number of necessary IO lines. Many ADCs offer multiplexed outputs which brings down the number of IO lines even further, however, this often means that the I and Q channels are sampled 0.5 bits apart, which is not an issue for most applications. For fractional sampling, on the other hand, it is extremely critical that the samples are aligned in time, why it was decided to utilize parallel outputs.

The DAC is made up of the current-output 8 bit Maxim Integrated MAX5851 chip, which was chosen in large part due to the same considerations as for the ADC, and features parallel outputs at a data rate of up to 80 Msps[13]. For BPSK modulation, only the I-

channel of the baseband signal is changed meaning that 9 IO lines is sufficient to control this chip. The 9th connection is a control word signal and is used to program the DAC for different operating modes. By pulsing the control word low the chip is programmed according to the digital states of the eight Q-channel data bits. The current output is converted to voltage by means of pull-down resistors before it is sent to the front-end.

The clock is controlled by a 25 MHz *temperature compensated crystal oscillator* (TCXO) which has a reported accuracy of 0.28 *parts per million* (ppm)[27]. This oscillator is used to drive the Silicon Labs Si5351C clock generator chip which is programmed through an *inter integrated circuit* (I^2C) bus and is capable of producing 8 different phase aligned clock outputs of arbitrary frequencies between 2.5 kHz and 200 MHz[18]. The clock is also connected to a 27 MHz standard *crystal* (XTAL) which is needed to drive the internal logic of the Si5351C in addition to the TCXO.

The final major component of the transceiver design is a 3.0 V *low drop out* (LDO) voltage regulator which maintains a staple supply for the different chips. This voltage level was chosen because the ADC expects a common mode level for the differential inputs of half its supply voltage and the MAX2830 chip is capable of a maximum common mode output level of 1.5 V. The reason that a voltage regulator was utilized instead of a more energy efficient DC-to-DC converter is that noise from the constant switching of the power converter might interfere with the sensitive radio hardware.

8.1.2 Construction

To verify the transceiver design, an initial prototype PCB was created at an in-house DTU prototyping lab. This meant that only two-layer boards could be constructed with a dielectric thickness of 1.5 mm resulting in difficulties with impedance matching. For example, at these dimensions a microstrip line with a characteristic impedance of 50 Ω has a width of about 3 mm. For such large dimensions, many of the formulas for impedance calculation begin to break down. Nevertheless, a board was created and populated the result of which is shown in figure 8.2.

This prototype was largely functional with chip configuration and communication working as expected. However, no proper radio signals could be detected. Because of the impedance matching issues this was accredited to the limitations of the production system.

Following this, minor adjustments were made to the design and a new 4-layer PCB was ordered from a professional supplier. In this case `easyeda.com`. The result, after being populated with components, is shown in figure 8.3.

For this 4-layer board the top and bottom layers have been used for signal lines while the middle two layers consists of a ground and a power plane. This provides better isolation of the different signals as well as reducing the thickness of the dielectric layer between the signal and ground plane. As such, for this configuration the thickness is 0.18 mm which, with FR-4 prepreq and 1 ounce copper, results in a trace width of 0.3 mm for a

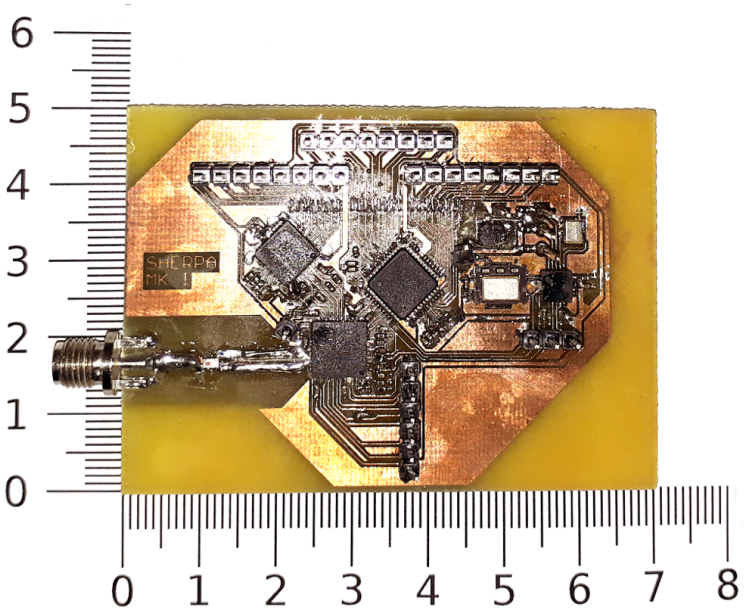


Figure 8.2: The initial prototype board. The image has been scaled so it approximately matches the real life size of the hardware.

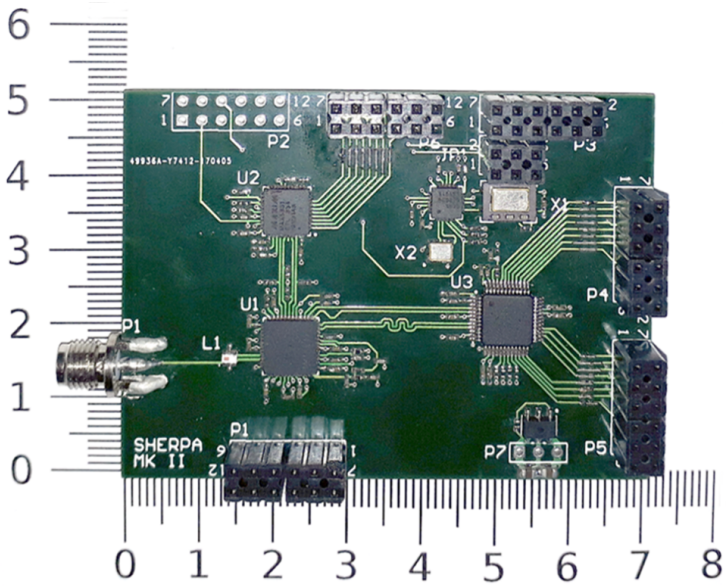


Figure 8.3: The final transceiver board. The image has been scaled so it approximately matches the real life size of the hardware.

50 Ω microstrip line. This is much more manageable and this new version turned out to work without issue.

When designing the PCB much care was put into following guidelines for high-speed

electronics. As a result of this the ground plane has been designed such that sections underlying clock signals have been separated from sections beneath digital signals which are again separated from analog parts. The different sections are only connected to each other at small junctions. This should reduce distortion of the analog signal caused by the digital signal and clock lines[12]. Furthermore, all chips have had their power pins decoupled to the ground plane using 0.1 μF capacitors placed as close to the pins as possible. Extra decoupling has been added by having the in- and output of the LDO regulator connected to ground through 10 μF and 2.2 μF capacitors, respectively, following the chip manufacturers' specifications[12][13][1].

A complete schematic and PCB layout for the designed transceiver board is included in appendix C along with a component overview.

8.1.3 Transceiver Overview & Configuration

It was decided that the interface to the transceiver should be as general purpose as possible to enable future students to use the developed hardware for whatever their purposes may be, since no proper ready made alternative can be purchased. Therefore, most interface pins are connected to external headers. However, to keep down the number of IO lines, connections enabling putting the various components into standby modes have not been exposed. This results in the transceiver using a larger amount of power than is necessary, but that should not be an issue for research purposes.

The board has six 12-pin female headers in the style of Digilent's Pmod™connectors. This makes interfacing with Digilent FPGA development boards very easy and was designed this way because several such development boards were available on hand hence removing the need to acquire expensive FPGA hardware.

In addition to the Pmod™connectors, the board also includes a 6-pin female header used for clock lines and a 3-pin male header that can be used for power connections. Alternatively, the device can be powered using the 3.3 V connections from the Pmod™connectors. A detailed overview of these connectors can be seen in figure 8.4.

As can be seen, **P1** is used for configuration of the various chips; **P2** has clock inputs to the different components as well as miscellaneous outputs that are not used in the current project. More information about these connections can be found in the data-sheets of the individual chips; **P3** is connected to the 8 clock outputs from the Si5351C chip; **P4** is the digital I-channel output; **P5** is the digital Q-channel output; **P6** is the digital Q-channel input. Whether data is transmitted on the I- or Q-channel when performing BPSK modulation is not essential as a phase offset will be introduced regardless. Furthermore, the MAX5851 chip allows for multiplexing of IQ-data using the Q-channel connections, meaning that any arbitrary modulation method can still be obtained using this configuration; **JP1** can either be used for external connections or as a jumper connecting the clock lines of the various chips directly to the clock generator outputs.

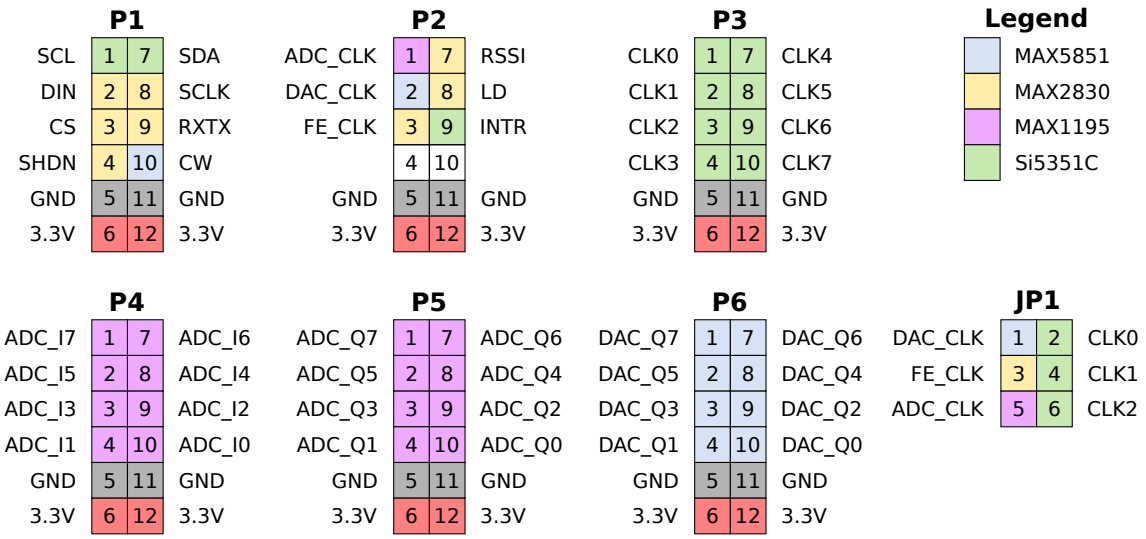


Figure 8.4: Overview of the different connectors on the transceiver board. The colors indicate which chip they are connected to.

For the purposes of this project the transceivers have been configured such that the clock generator supplies a 40 MHz clock signal which drives the MAX2830, a 3.2767 MHz clock for the DAC, and a 3.2766 MHz for the MAX1195. The front-end has a programmable bandwidth of 7.5 MHz to 18 MHz of which the lower limit was chosen. Figure 8.5 shows the power spectral density of one of the used Gold codes.

As is shown, 95.2% of the signal power is included within the first 7.5 MHz why choosing this lower limit has the result of reducing the noise level while maintaining most of the signal integrity. The carrier frequency has been set at 2.448 GHz as this is in the middle of the available frequency span and can be generated exactly by the front-end chip whereas 2.45 GHz, for example, cannot.

The configuration happens at the startup of the system and is performed by an FPGA which also serves to generate data for transmission and process it for reception. First, the clock generator chip is programmed and enabled over I^2C, after which the MAX2830 front-end is configured over a SPI bus.

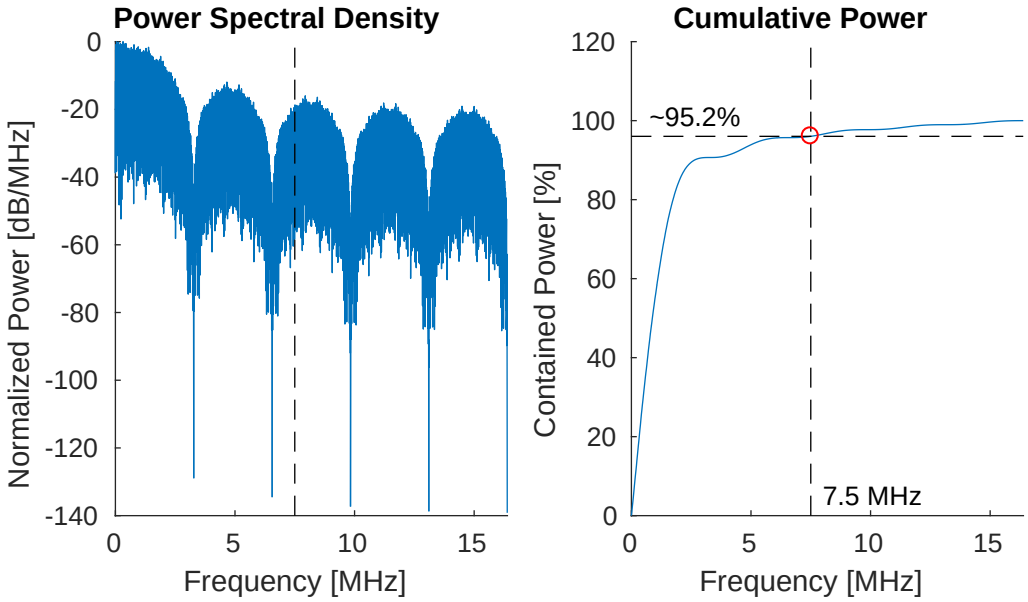


Figure 8.5: Power distribution function for the Gold code associated with vehicle 1.

8.2 Processing System

The data handling can be split up into two parts: data acquisition/generation and data processing. The acquisition and generation is performed on an FPGA as in the reference design, however, to simplify matters the raw IQ data is transferred to a PC through a USB connection where all the processing takes place.

8.2.1 Data Acquisition & Generation

A Xilinx Spartan® 6 FPGA chip on a Digilent Nexys 3™development board forms the basis of the acquisition system. This board has four 12-pin Pmod™connectors which are used to connect to the transceiver hardware as illustrated in figure 8.6.

For CDMA transmissions, using only a single bit to describe the signal results in a 2 dB reduction of the signal-to-noise ratio[7], but it is still possible to extract the data. Considering the limited number of IO lines on the development board, it was therefore chosen to only use bit 7 of the ADC data which represents the sign of the digitized signal when operating in twos-complement output mode. Sampling both I and Q data, four transceivers can thus be connected to the 8 data lines of a single Pmod™connector instead of requiring 8 separate connectors as would be the case if all bits where used. However, only a maximum of two transceivers were connected to the FPGA at any given time during testing.

To keep timing consistent across all connected transceivers, one of them is designated to be the master and feeds its ADC and DAC clock signals to the FPGA. This then

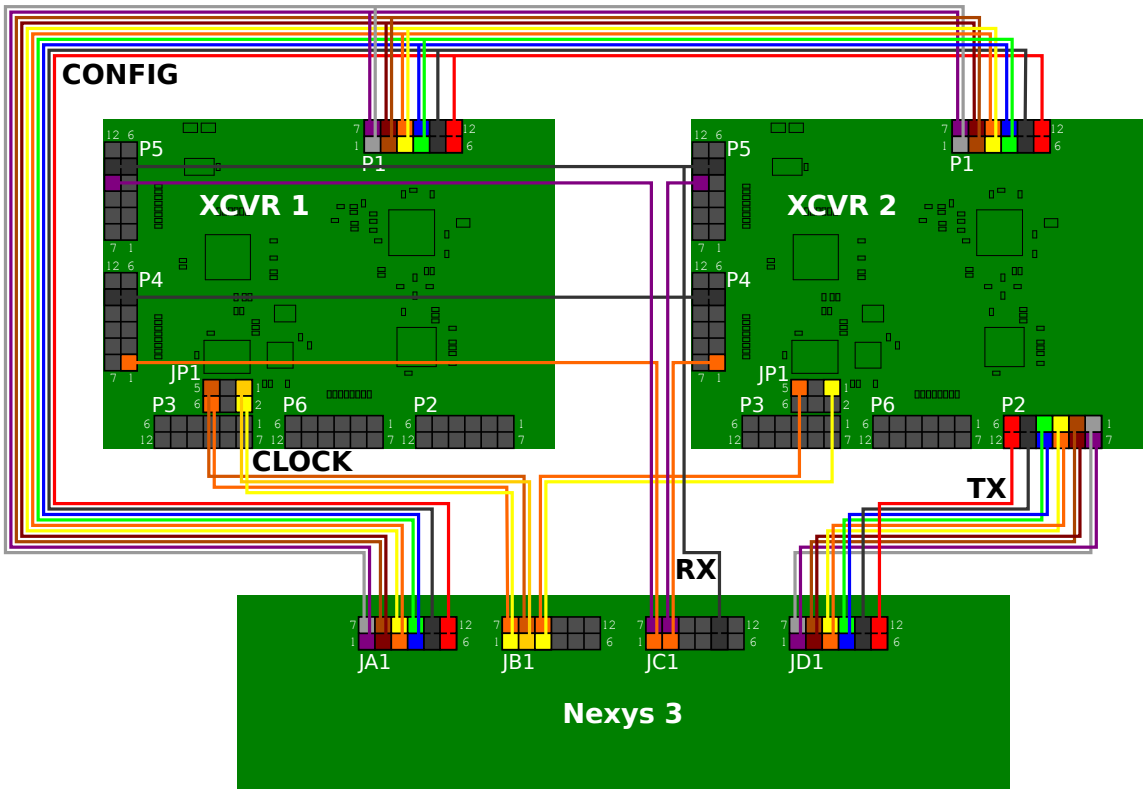


Figure 8.6: Connections between transceivers and FPGA. The setup enables reception on both transceivers, but only transmission on XCVR 2. For the actual testing the two transceivers were only connected to the receive path, while a third transceiver had its transmit path connected to a second FPGA.

distributes the signal back to the master as well as to any other connected transceivers. With such a configuration, it would be sufficient to have a single clock generating board feeding everything else, but each transceiver was outfitted with its own clock generator in order to keep them being of general purpose use. The system of feeding back the clock signal to the master has the disadvantage that only three transceivers can be connected to a single Pmod™connector, but that should be sufficient for initial testing. The 40 MHz front-end clock signal is individually generated on each transceiver as this does not need to be completely aligned with the other clocks and to avoid transferring this relatively high frequency signal over too large distances.

The final connector on the FPGA development board is connected to the transceiver DAC input. Unfortunately, the DAC chip only supports an offset binary representation of the digital data, which means that all 8 bits must be used to generate an unskewed signal. Because of this, having more than one transceiver transmitting simultaneously from a single FPGA is problematic. Potentially two can be connected at the same time,

however, this will require different configuration files for transmission and reception.

The configuration of the FPGA has been written in *VHSIC hardware description language* (VHDL) and has been constructed such that a single FPGA can serve both as signal generator when transmitting and as a link between the transceiver and a PC when receiving. And overview of the configuration is displayed in figure 8.7.

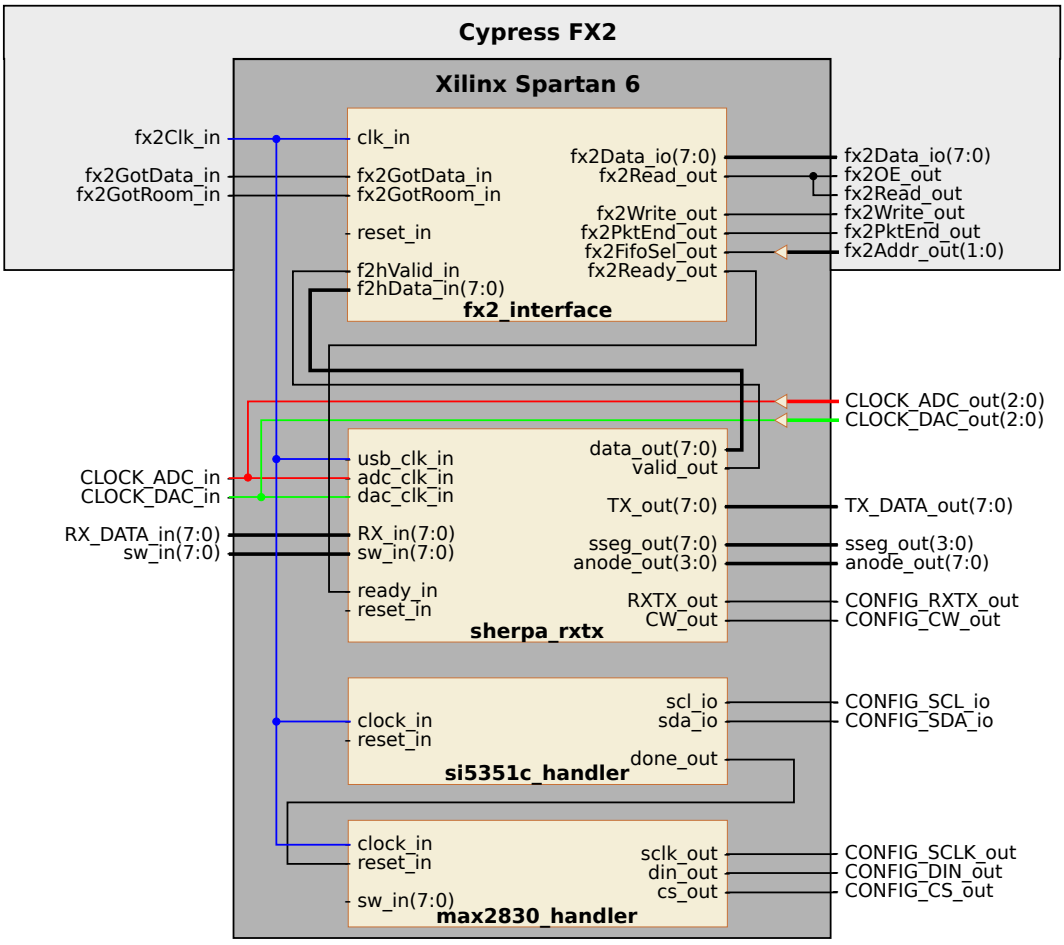


Figure 8.7: Overview of the FPGA hardware design.

Transceiver configuration is performed by the `max2830_handler` and `si5351c_handler` components which transmits configuration data stored in internal memory to the two chips using open source SPI and I^2C modules. The `fx2_interface` component is part of the open source FPGALink library, developed by Chris McClelland[21], and takes care of the USB communication between the PC and FPGA through the Cypress FX2 USB controller chip.

The `sherpa_rxtx` component handles Gold code generation and data reception. It is implemented as a simple state machine which is illustrated in figure 8.8 and includes most of the footwork needed for extending the implementation to enable transmission

of navigation data on top of the Gold codes. The `seven_seg` module simply updates a seven segment LED display for debugging purposes and is part of the FPGALink library.

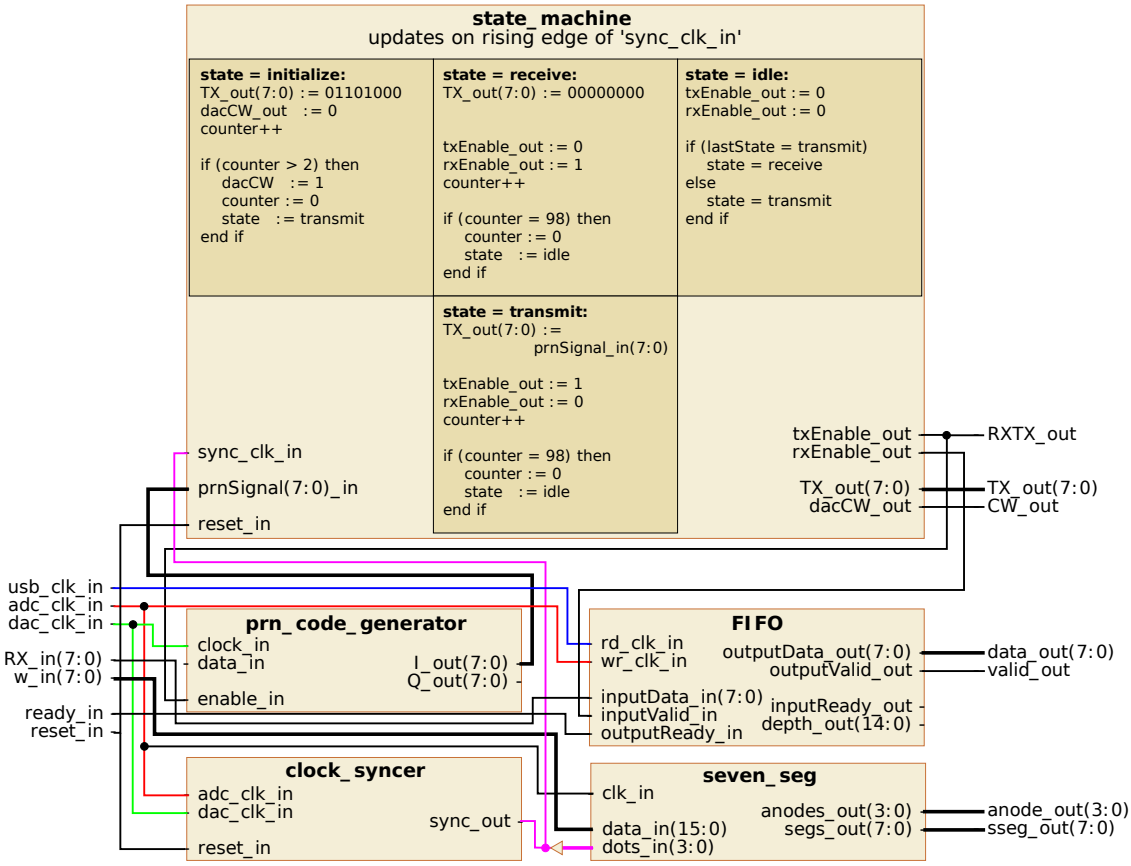


Figure 8.8: Design of the `sherpa_rxtx` component.

In the transmitting state, the transceiver DAC Q-channel input is directly connected to the output of the `prn_code_generator` module which is just an implementation of the shift register configuration shown in figure 5.5, however, it has been modified slightly such that an output of 1 corresponds to the binary number 11111111 while a zero corresponds to 00000000.

When receiving, the 8-bit transceiver ADC data is put into an 8-bit wide FIFO buffer with a depth of 32768 meaning that an entire Gold code cycle can be stored before it starts to fill up. The output of the buffer is connected to the `fx2_interface` component. This solution elegantly handles the different clock speeds between the USB interface and the ADC sampling while allowing momentary disruptions of the PC connection. It should be noted that if the FIFO buffer cannot be emptied faster or at the same rate as it is filled, data samples will be lost with the likely result of a loss of target in the subsequent navigation loop.

The state machine has been implemented such that it transmits for 99 code cyles, rests for one, receives for 99 codes samples, rests for one, and so on. For proper delay determi-

nation, it is important that the switch from transmission to reception happens exactly when the ADC and DAC clocks are phase aligned which occurs every 32767 periods of the DAC clock and 32766 periods of the ADC clock. A simple clock synchronizing module, `clock_syncer` has been implemented to take care of this. Basically, the rising edge of the ADC clock is used to sample the DAC clock, if the sampled clock signal switches from a logic low to a logic high it means that the two signals are aligned and it will continue to be high for the next 16383 ADC cycles. In practice, however, clock instabilities mean that spurious switching between high and low will occur causing the synchronization signal to not have a constant period. This issue is alleviated by extending the `clock_syncer` module to a sample and hold circuit instead of just sampling. Contrary to what might be expected, this has proven to be very reliable resulting in a synchronization signal with the expected period without any detectable jitter.

While this system of automatic switching between reception and transmission is needed in a complete navigation system, it is unnecessary and actually detrimental when testing the performance of one-way delay estimation. As such, for testing purposes, it has been disabled and transceiver switching is instead controlled by a hardware switch on the Nexys 3™. A total of 8 such toggle switches are included on the board and the remaining 7 are used for manual gain control both when receiving and transmitting.

Originally, a Costas loop was also implemented on the FPGA for carrier recovery purposes, however, this was ultimately removed as it proved to be unreliable and often degrading the recorded signal to such an extend that no information was recoverable. With more time, careful tuning could doubtlessly overcome this issue and a complete system should include such a recovery loop to achieve the best possible signal integrity.

8.2.2 Data Processing

The FPGALink project includes a C-library which enables simple USB communications to the FPGA and has been used as the basis for the processing system. This software has been written completely in C and an overview of it is included in figure 8.9.

After having initialized the FPGA connection, the main loop of the software begins, starting by reading 32766 bytes of information from the FPGA FIFO buffer. Each byte corresponds to the 1 bit I- and Q-channel data for up to four transceivers, and thus needs to be separated and transformed to NRZ data, i.e., signal values of ± 1 instead of 1 or 0. The results of this data translation is saved into individual buffers for each receiver and channel each with a size of 32766 bytes.

To find the initial integer delay, the IQ buffers of XCVR 1 are multiplied by a copy of the Gold code associated with it and the sums are computed. If the root sum square of the two sums is higher than a certain threshold, the system assumes that the correct integer delay has been determined and switches to the locked state. If this threshold is not met the Gold code is offset by one sample and the process is repeated. For each batch of data, 10 different Code offsets are tested before a new set of data is read, with experience

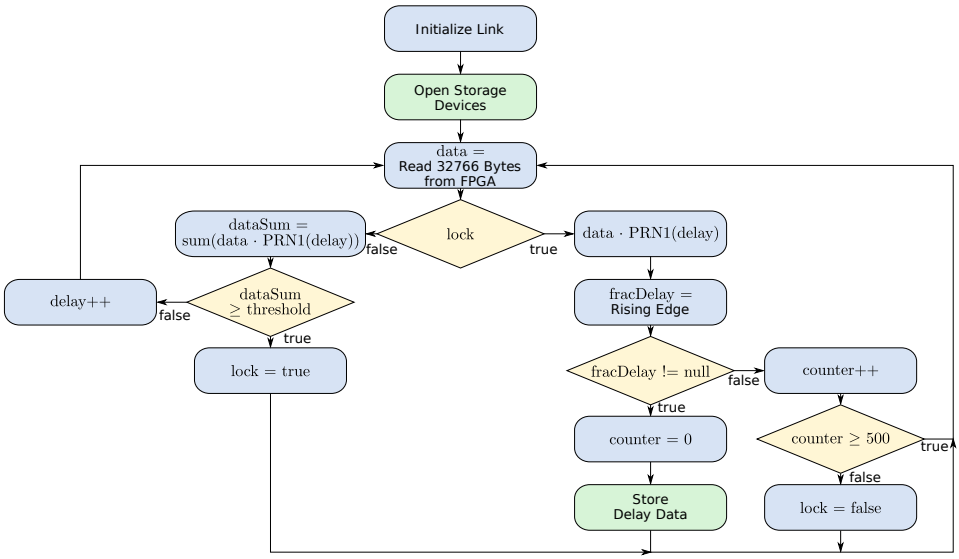


Figure 8.9: Overview of the implemented processing software. The fact that 10 different delay values are tested for each set of data has been omitted for simplicity.

showing that testing more than this number of offsets results in dropped samples. As a new set of 32766 bytes are generated from the FPGA every 10 ms, all possible code offsets are tested after about 33 seconds which thus represents the theoretical maximum time before lock is achieved.

When in the locked state, the recorded data is mixed with three copies of the Gold code: one on time (according to the determined integer delay), one 1 sample late, and one 1 sample early. This results in a new data vector of length 98298 for each channel and transceiver which is low pass filtered resulting in the reconstructed pulseform. These IQ reconstructions for a given transceiver are root sum squared together which reduces the effects of having an offset carrier. This has been implemented as an alternative to a Costas loop as the latter did not perform well in conjunction with fractional sampling. Of course, this means that the any data that is transmitted on top of the Gold code will be lost, as the squaring removes any polarity information. Because only 1 bit of data is available per channel it is necessary to perform the smoothing before the squaring as the result would otherwise always result in 1 for all bits.

The average of the newly recovered pulseform and the ones recovered from earlier sets of data is then calculated resulting in a new graph which reduces the issues of the offset carrier even further. After this, the index at which the pulseform rises above a certain threshold is determined as well as the index where it drops below it once again. The average of these indices is calculated, divided by 32766 and combined with the integer offset resulting the complete delay in units code chirps. Dividing this result by the sampling frequency and multiplying by the speed of light results in an estimate representing the distance between the transmitter and the receiver plus some offset

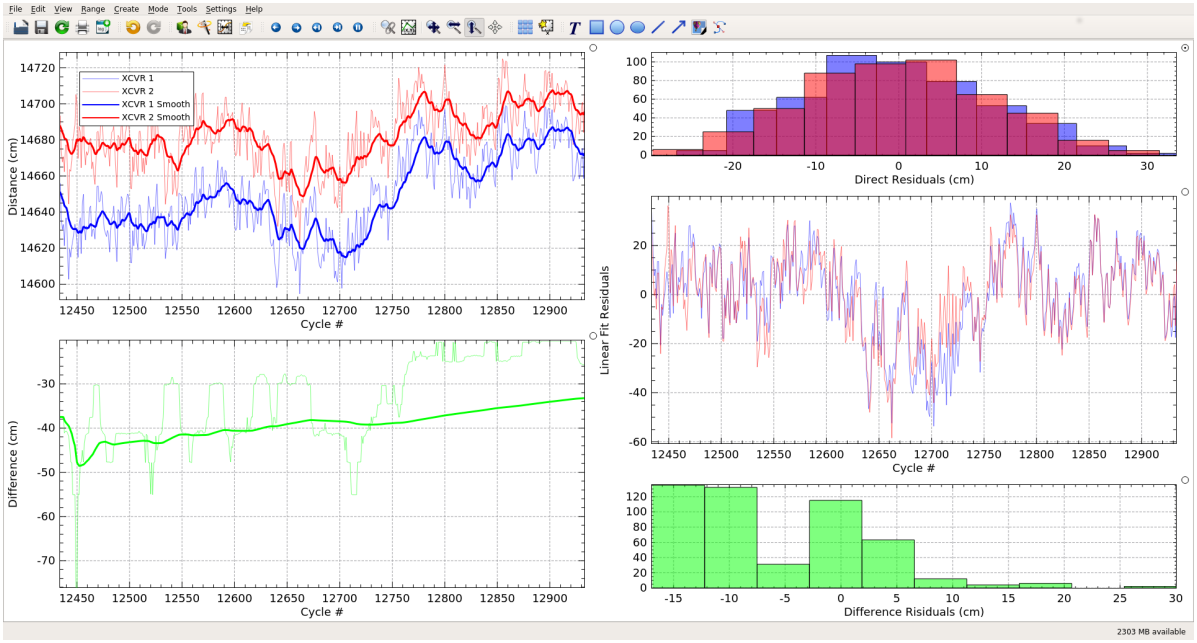


Figure 8.10: Screen shot of the Kst 2 software configured for a dual receiver setup. It should be noted that the residuals observable in the graphs do not represent the final accuracy of the system.

caused by system delays. If the recovered pulseform never crosses the threshold, a counter is incremented. Once this counter reaches a preset value, it means that the lock has definitively been lost and the system switches back to initial acquisition mode.

The recovered one-way distance estimate is saved to a simple comma-separated text file which is then read by a piece of software called Kst2[2]. This program plots the data and is able to update the plots as the file is written resulting in near real time data visualization. A screen shot of this software, as it has been configured for this project, is shown in figure 8.10.

The developed software also has the possibility of saving the data from the FPGA straight to a binary file which can then be analyzed off-line. This is beneficial when testing various tracking solutions and a copy of the C-application has also been developed in MATLAB® to allow for such experiments as well as comparisons between them.

8.3 Simulation Software

The simulation software has been implemented in MATLAB® and is quite simple in its nature. It considers two vehicles each with four transceivers placed at the vertices of a tetrahedron. One vehicle is positioned at the origin of the coordinate system with an identity rotation matrix, while the other is given a random attitude and position within a 100 m distance from the the first vehicle. The distances between all possible transceiver pairs are then calculated and Gaussian noise is added to represent actual data. These ”measurements” are then averaged a number of times to produce a more accurate estimate. The standard deviation of the errors is chosen such it matches with what is observed using the developed transceiver hardware.

With the simulated measurements completed, the data is used to calculate the position and attitude of the second vehicle by solution of the equations of section 5.1 for both direct distance measurements and differential data. For the direct distance data the equations are solved by means of simple linearized least squares which has proven to quickly converge. For the differential measurements, Tikhonov-regularization was employed as the solution would otherwise not converge. Both the direct differential method and one assuming that the target is far away has been implemented as well as a method that includes both direct and differential measurements. Common for all the different models is that the Jacobian matrix is calculated numerically as the analytic derivatives are fairly complex. With the navigational parameters computed, the results are compared to the true values and the errors are estimated.

CHAPTER 9

Results

To evaluate system performance a series of experiments were performed using the custom made transceivers. The test setup consisted of an FPGA connected to a transceiver that served as a transmitter and another FPGA connected to two transceivers serving as receivers. Measurement were made with varying distance between the two units having them completely separated from each other as well as with the same clock signal driving them both. Based on the results of these distance measurements, simulations were performed using both simulated direct and differential distance measurements with uncertainties devised from the results of the hardware evaluation.

9.1 Experiments

Figure 9.1 shows an image of the experimental setup that was used for all experiments with only minor modifications. All experiments were executed with a chirp rate of 3276700 Hz, a code length of 32767, a carrier frequency of 2.448 GHz, and a sample rate of 3276600 Hz. Higher chirp and sample rates were attempted, however, the USB2 transfer speed limited it to this number.

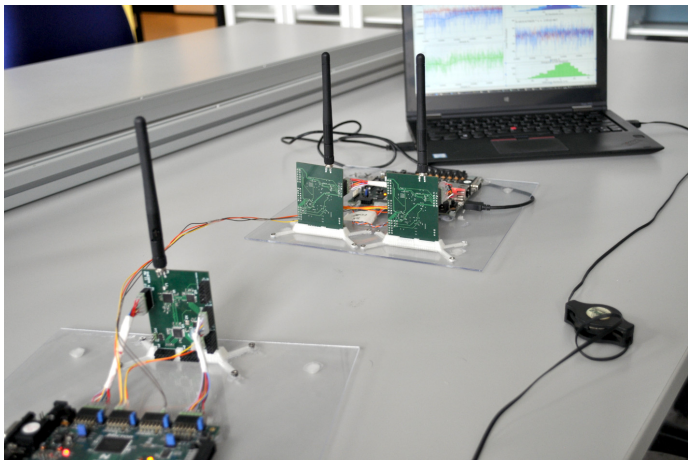


Figure 9.1: Image of the experimental setup with the transmitting section being closest to the camera. The wires connecting the two transceiver setups carry the clock signals for a single oscillator setup.

9.1.1 Dual Oscillator Measurements

For the first test the transceivers were placed such that the receiving XCVR 1 and XCVR 2 were at distances of 10 cm and 14 cm, respectively, from the transmitting XCVR 3. Cross-correlating the signal recorded by XCVR 1 with the emitted Gold code results in figure 9.2.

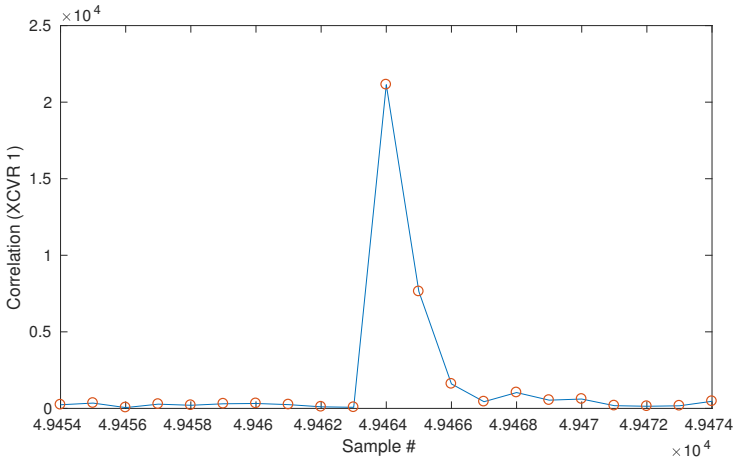


Figure 9.2: Output of the cross-correlation between the received signal at 10 cm distance and a dual oscillator setup.

As can be seen, a well defined peak is formed, however, a substantial part of the energy is in the sample immediately after the peak. This is a result of the fractional sampling and can be understood by looking at the recovered pulseform which is displayed in figure 9.3. The energy in the peak corresponds to the section of the pulseform that extends from 0 to 10 ms, while the sample after the peak corresponds to -10 to 0 ms. As the fractional delay changes so does the energy allocation in the cross-correlation function.

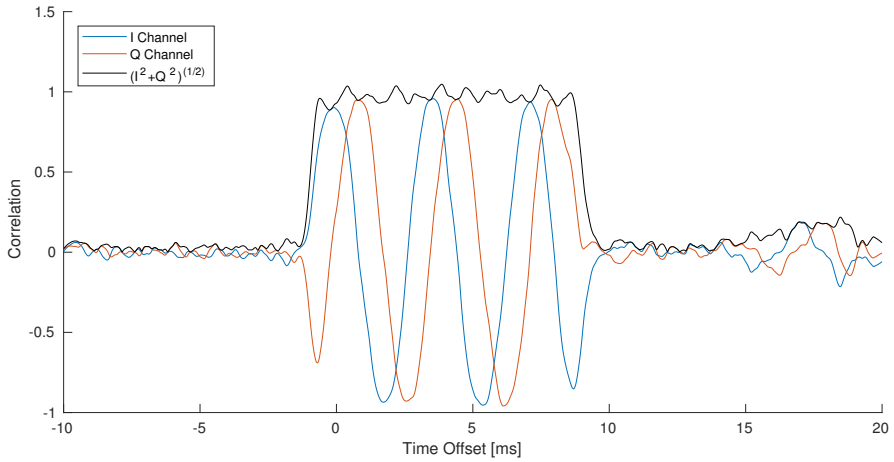


Figure 9.3: The recovered pulseform at 10 cm distance, dual oscillator.

As can be seen from figure 9.3, the method of squaring the averaged channel information works well in recovering the pulseform even though the individual channels are distorted. However, the method is not perfect and fails to retrieve the information if the carrier offset is too large. It is worth noting that the observable distortion is not caused by high frequency offsets changing the actual pulseform, but rather by the individual samples being modulated by a small offset of about 250 Hz corresponding to 100 *parts per billion* (ppb).

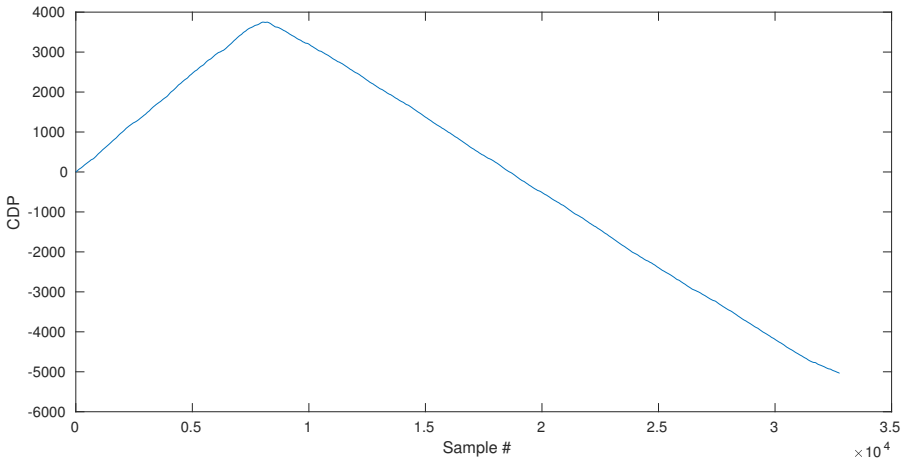


Figure 9.4: The CDP at 10 cm distance, dual oscillator.

It has been found that the pulseform recovery provides much more precise results than the CDP method, why it is used exclusively in the following. However, figure 9.4 has been included for comparison purposes. This CDP is clearly not as well defined as the one obtained using the bladeRF hardware during the initial validation. This is not unexpected as the bladeRF used the same oscillator for both transmission and reception and the transmission distance was shorter.

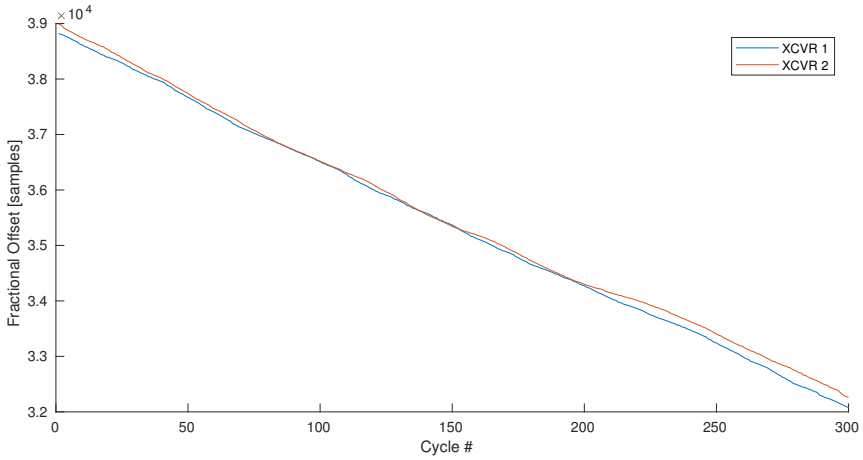


Figure 9.5: The raw estimated delay at 10 cm, dual oscillator.

Measuring the location of both the rising and the falling edge of the recovered pulseform and taking the average results in an estimate of the fractional delay which is displayed in figure 9.5. A distinct constant downward tendency is clearly visible in the data with a slope of -23 samples per cycle corresponding to an oscillator offset of 21 ppb or a velocity of 6 m/s. The reason that the observed carrier offset seemingly indicates an offset of 100 ppb is that the fractional offset is determined from the oscillator of XCVR 1 while figure 9.3 shows the pulseform for XCVR 2 and the carrier offset is determined from the individual transceivers.

Removing this constant oscillator offset by linear regression and subsequent subtraction results in the left part of figure 9.6 where the fractional delay has also been converted to distance.

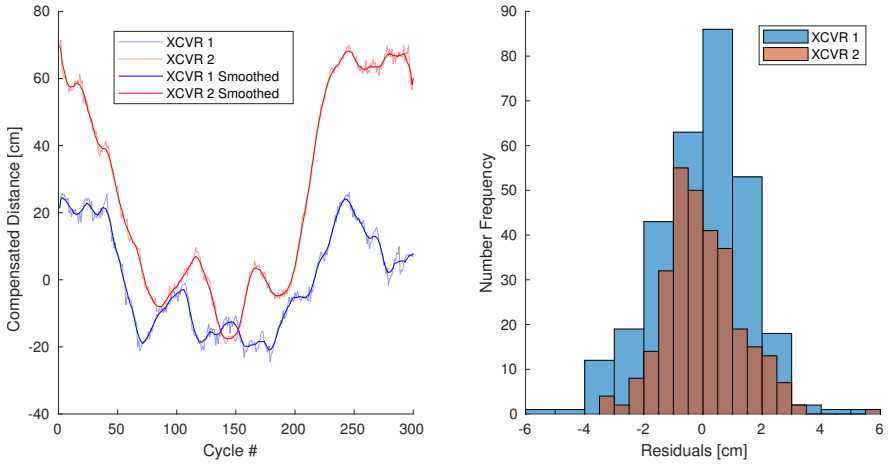


Figure 9.6: The estimated distance after correcting for the constant oscillator offset (10 cm, dual oscillator).

As can be seen, a much more stable signal emerges, however, still with a variation of about ± 50 cm. These remaining perturbations are most likely caused by short term oscillator drift and are difficult to compensate for. Subtraction of a smoothed version of the data from the data itself can be used as an estimate of the attainable accuracy should a method be found to compensate for this oscillator drift. A histogram of the result of such a procedure is shown in the right part of figure 9.6 and it appears that the resulting errors are more or less normally distributed with a standard deviation of 1.5 cm.

Changing the inter-transceiver distance to 45 m, the pulseform is somewhat degraded as can be seen in figure 9.7. The used antennas have gains of 2 dB[32] and the MAX2830 has a noise figure of 4 dB[1] which, combined with a transmit power of 17.1 dBm, results in a signal to noise ratio of 103 dB at 10 cm distance and 50 dB at 45 m. This assumes a ground temperature of 280 K and that there are no losses between the antenna and the front end system. However, the ADC limits the SNR to 50 dB[12] so the reduction in signal strength seemingly does not explain the pulseform degradation. However, the

actual transceivers do exhibit some level of loss which will make the SNR reduction have an observable effect. Still, it probably does not account for everything.

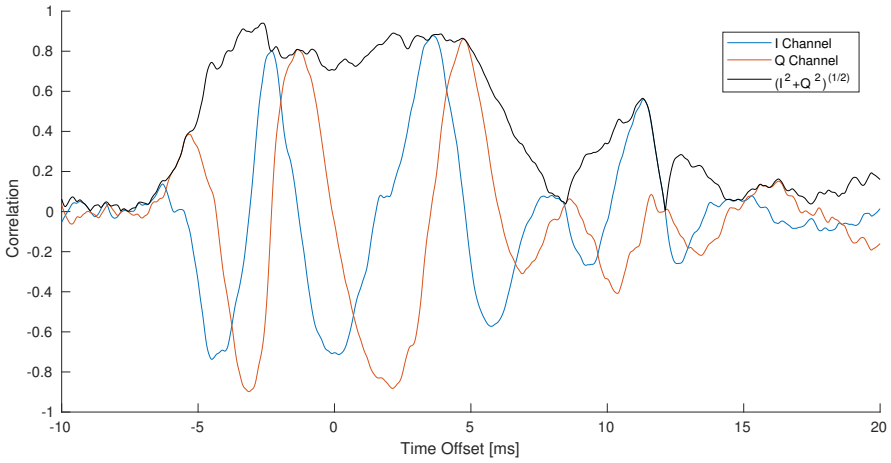


Figure 9.7: Recovered pulseform (45 m, dual oscillator).

Another factor is that the 45 m measurement was performed on the roof of DTU building 327 on a sunny day, which caused the oscillator temperatures to fluctuate independently from each other. The result of this is that the squaring method for carrier recovery does not function very well. The temperature fluctuations are also evident when plotting the calculated fractional delays as is done in figure 9.8.

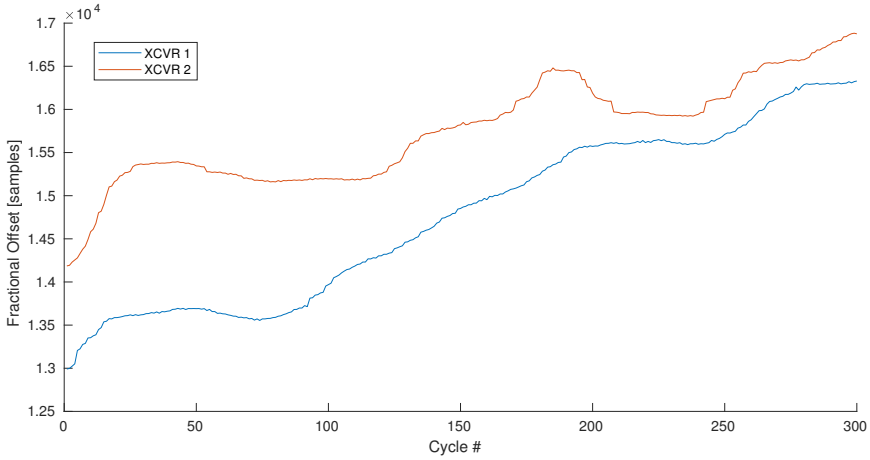


Figure 9.8: Raw delay (45 m, dual oscillator).

This graph is in stark contract to the results of the 10 cm measurement, with much larger fluctuations. Removing the mean gradient results in the left part of figure 9.9, which shows deviations on the scale of ±200 cm. Similarly to before, subtracting the running mean results in the residuals whose histogram is shown in the right part of the figure.

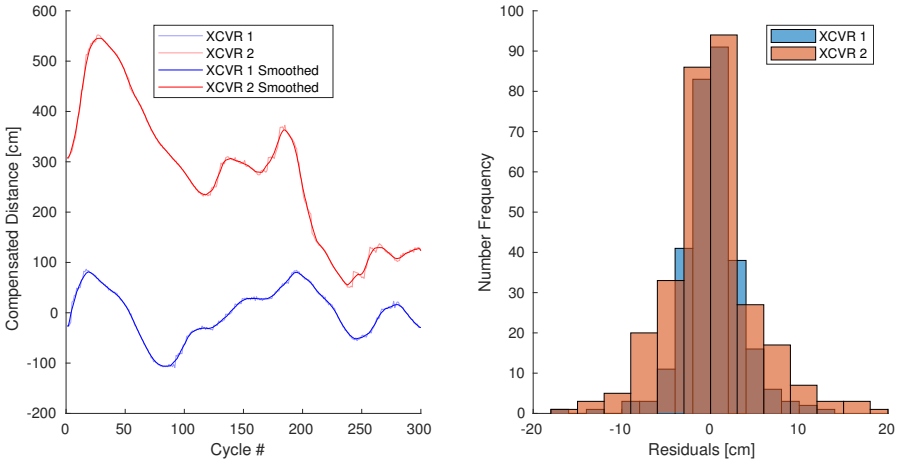


Figure 9.9: Corrected distance (45 m, dual oscillator).

The standard deviation of the residuals at this distance is around 4 cm. Whether this is caused by the change in signal strength, the temperature issues, or a combination thereof, is hard to determine.

The oscillator fluctuations mean that it is not possible to determine the direct distance using the current hardware. A possible method to overcome this issue is to use the difference in distance measurements between two transceivers sharing the same oscillator instead, as the fluctuations should be present in both signals and thus cancel out. Figure 9.10 shows the difference signal for the measurements made at 10 cm distance.

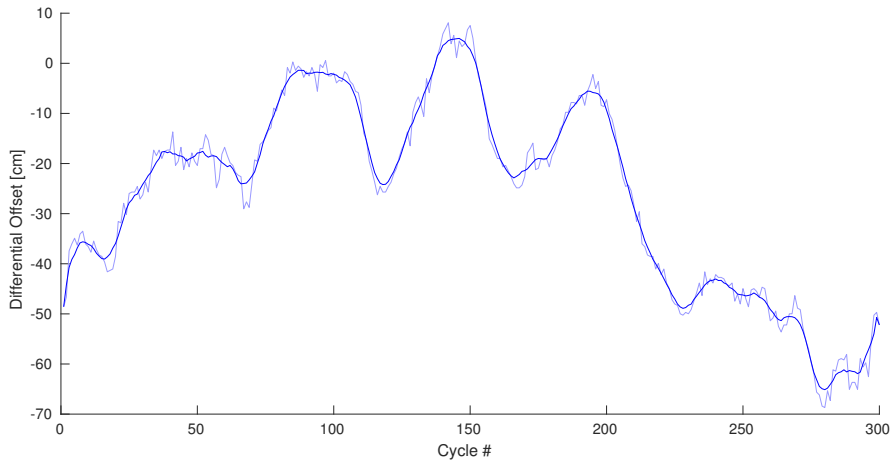


Figure 9.10: Differential distance estimate for a dual oscillator setup at 10 cm distance.

As is clearly evident, the fluctuations are still present with a similar scale to before. This could potentially be attributed to a combination of hardware differences between the two receivers, temperature fluctuations and multipath distortions. In any case, this method does not immediately resolve the issue.

9.1.2 Single Oscillator Measurements

In an effort to investigate the potential performance of the system if the oscillator fluctuations could be mitigated, the transmitter was connected to the same oscillator as the receivers. Figure 9.11 shows the calculated fractional delay for a 10 cm distance configuration.

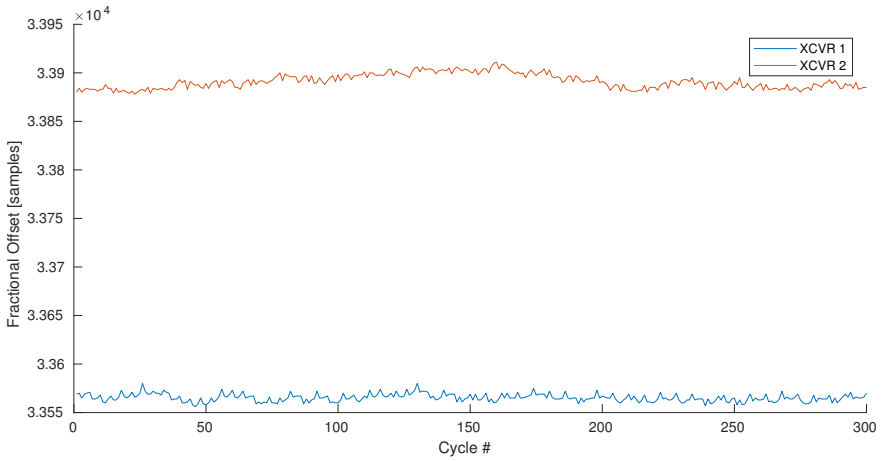


Figure 9.11: Estimated fractional offsets (10 cm, single oscillator).

The slope that was present in the earlier data is now completely gone confirming the assertion that it was caused by oscillator offset. The fact that the two curves are not located around a delay of 0 is caused by there being some inherent delay in the system which is added onto the time of flight. Converting these measurements into the corresponding distance estimates results in figure 9.12.

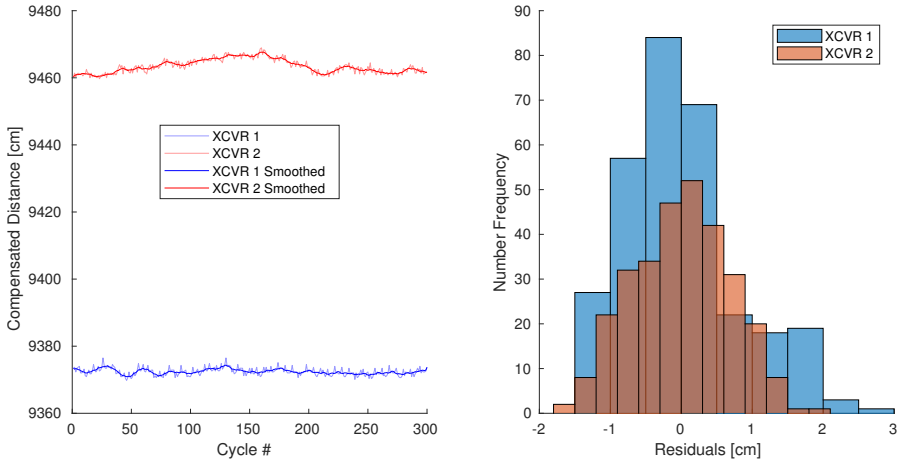


Figure 9.12: Estimated distance (10 cm, single oscillator).

In addition to removing the slope, the single oscillator configuration also results in a reduced amount of noise bringing the standard deviations of the residuals to 0.7 cm with

the total variation being on the order of ±1 and ±5 cm for XCVR 1 and 2, respectively. The difference signal between XCVRs 1 and 2 is presented in figure 9.13.

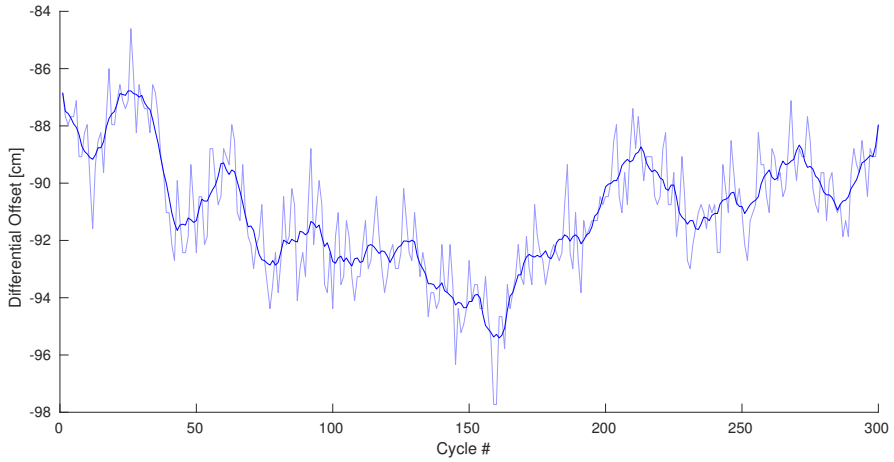


Figure 9.13: Differential distance estimate for a single oscillator setup at 10 cm distance.

It is clear that fluctuations are still present, however, the magnitude is reduced to ≈ ±5 cm. This could indicate that the measurement method itself might be the cause, however, there are a number of external noise sources which can potentially also influence the differential signal.

9.1.3 Signal Distortions

The Si5351C clock generator that is used to control timing incurs a maximum cycle-to-cycle jitter of 95 ps[18] corresponding to a distance variation of up to 2.8 cm. This places a limit on the attainable accuracy of the system, however, as the jitter is random it should even out over time.

The nature of the Gold code means that the fractional delay cannot be determined exactly even in a noise free environment on account of having bit run lengths of more than one. Figure 9.14 shows the theoretical errors that will result from this. The errors are substantial and actually larger than what is observed during actual measurements. A possible explanation for this is that by adding noise the effective run lengths are reduced which leads to the counterintuitive side effect of improved accuracy. In addition, averaging the recovered pulseforms will no doubt reduce the issue as well.

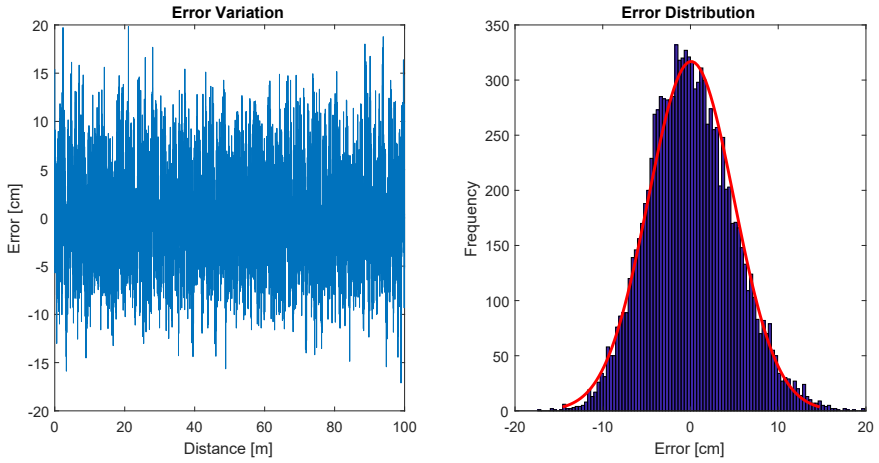


Figure 9.14: The theoretical errors for fractional delay estimation using a noise free Gold code signal.

Another significant cause of error is multipath. When performing measurements it was very clear that the distance estimates changed as a result of movement in the laboratory as is illustrated in figure 9.15.

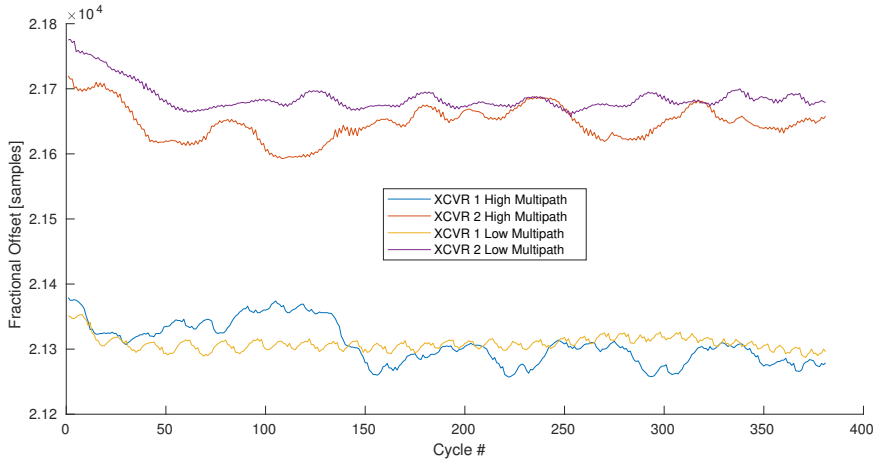


Figure 9.15: The estimated fractional offsets for a single oscillator setup with low and high levels of multipath distortion.

For the ”Low Multipath” measurements everything, including the experimenter, was kept stationary and the two graphs appear fairly constant. The ”High Multipath” plots represents a measurement where the experimenter’s hand was moved back and forth next to the communicating antennas and is the cause for the almost cyclical variation in the fractional delay which is clearly different for the two receivers. It is evident that multipath has a significant detrimental impact on the accuracy of the system.

The strong temperature dependence of the oscillators is illustrated in figure 9.16. This graph shows the fractional delays determined for a dual oscillator setup separated by a distance of 20 cm. One second in to the experiment a finger was placed on the PCB opposite to the main oscillator resulting in a slight increase in its temperature. This is clearly observable in the slopes of the curves which change drastically at the application of heat.

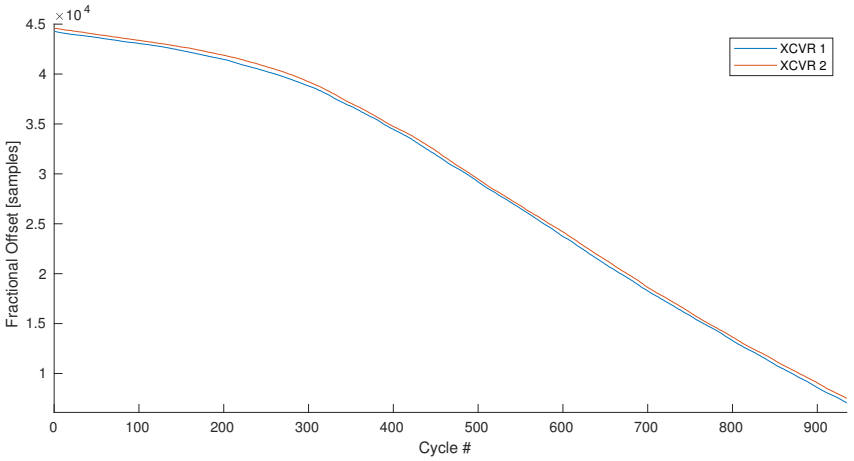


Figure 9.16: Fractional delay changes resulting from heating the oscillator of the receiver.

An additional detrimental effect is caused by the transceiver LNA. Figure 9.17 is a plot of the fractional delay in a single oscillator configuration where the gain mode of the LNA was changed from high to medium and finally to low. As can be seen, there is a significant change in delay depending on the gain mode. This appears to simply be intrinsic to the hardware and is not in itself an issue, however, if variable gain control is to implemented this could cause complications.

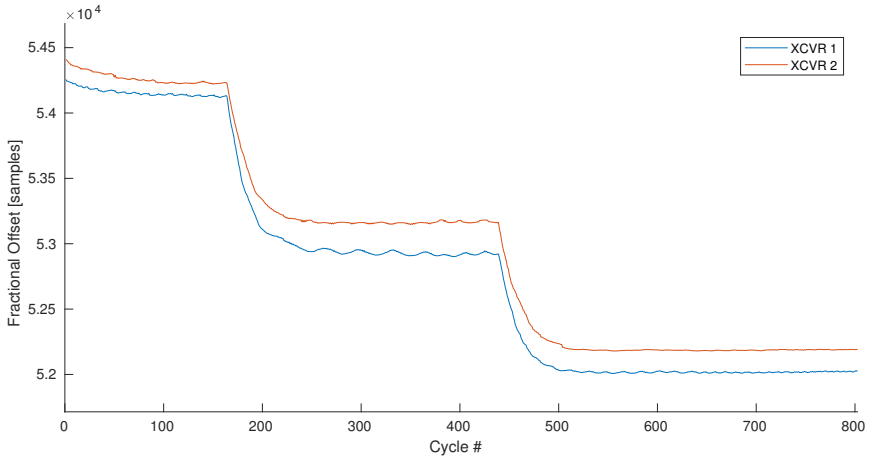


Figure 9.17: Fractional delay offsets caused by changes in gain.

Another gain related issue is illustrated in figure 9.18 which shows the pulseforms recovered from the same setup, however, instead of changing the LNA gain the attenuation of the transmitter was changed. For no attenuation the pulseform is recovered forming almost a square pulse.

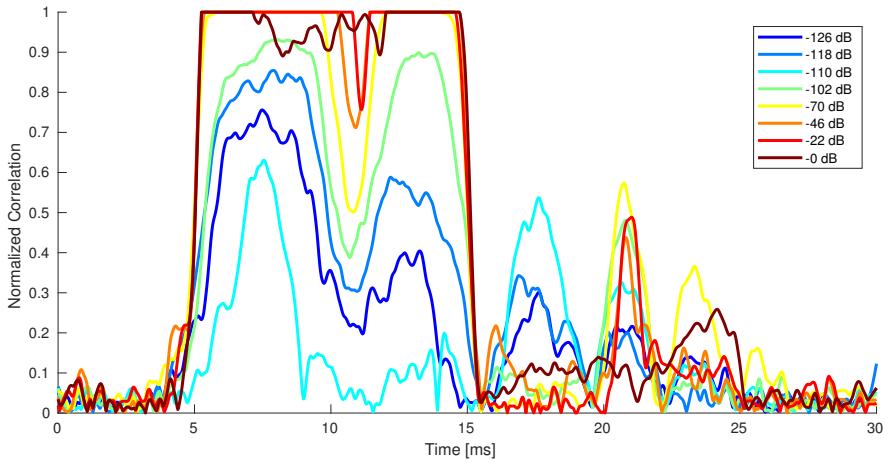


Figure 9.18: The recovered pulseforms for various levels of transmitter attenuation.

As the attenuation is increased a dip appears in the pulseform which grows larger as the signal strength diminishes and seems to be reproduced after the pulse has ended. The cause of this is not known, but one possible explanation could be that the receiver distorts the signal slightly through an apparent high-pass filtering that causes the pulse top to slope down instead of remaining constant. If this is combined with an offset zero-level in the ADC it will result in some of the samples towards the end of the pulse being recorded with the wrong polarity. This then results in them being 'reflected' immediately after the pulse on account of the processing method.

Another distortion that must also be considered is the inherent noise introduced by using CDMA. All the previous results were obtained using one transmitter and two receivers, a situation which does not accurately represent the noise environment of the complete system. Figure 9.19 shows the pulseforms that are recovered when the situation is switched around to two transmitters and one receiver. The recording was made with a single oscillator setup and an inter-transceiver distance of approximately 20 cm. The pulseforms have noticeably degraded compared to the earlier experiments, but to an extend that is different between the two transceivers. With this level of degradation the standard deviation on the distance estimates increases to $\approx \pm 4$ cm.

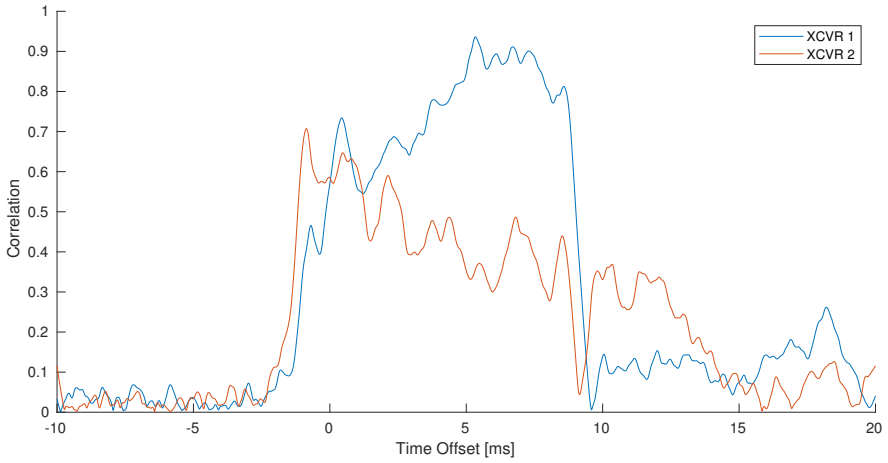


Figure 9.19: The recovered pulseforms for dual transmission at maximum gain.

9.2 Simulations

Based on the experiments, it seems that a distance error on the scale of 1 cm is attainable with various improvements to the hardware. As such, this error level has been used as a basis for a series of simulations. Common to all these simulations is the basic setup which consists of two vehicles each made up of four transceivers placed at the vertices of a tetrahedron with a baseline of 1 m. Figure 9.20 shows the error on the recovered navigation parameters when using direct distance measurements and no averaging of samples.

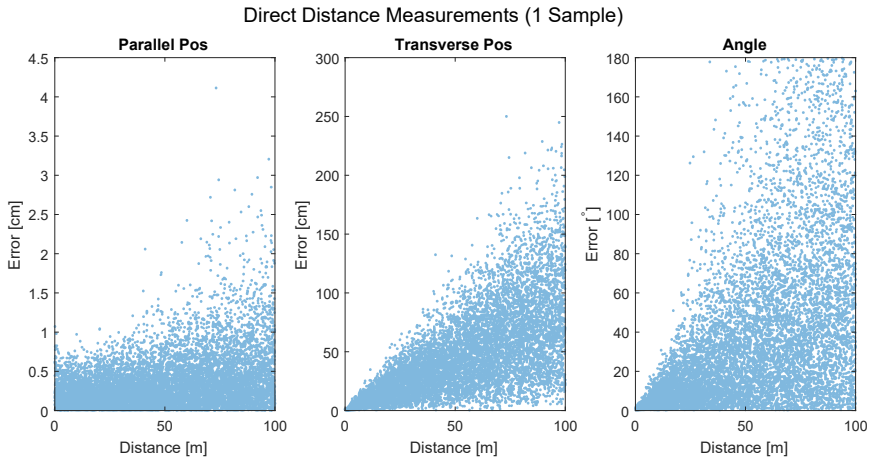


Figure 9.20: Simulated performance of the navigation system using single cycle distance measurements with a standard deviation of 1 cm.

”Parallel Pos” is defined as the magnitude of the positional error that is parallel to a vector connecting the two vehicles, basically the distance error, while ”Transverse Pos”

is the magnitude of the error which is transverse to this vector. ”Angle” refers to the magnitude of the rotation that is needed to rotate the determined orientation to the true value. As can be seen, the distance is determined to within 4 cm, the transverse position to within 3 m, and the angle is not accurately determined at all.

Increasing the number of samples used for each navigation solution to 100 and then to 1000 results in figure 9.21. For 100 samples a respectively positional and rotational

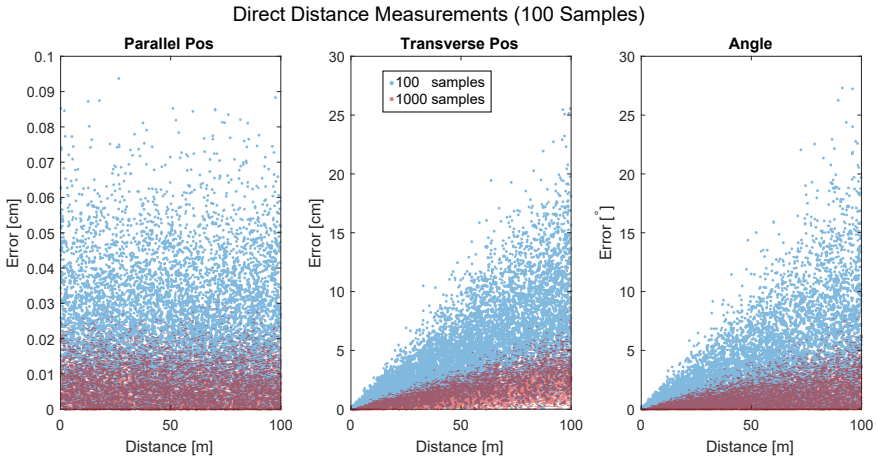


Figure 9.21: Simulation results for distance measurements averaged over 100 and 1000 cycles.

accuracy of 30 cm and 30° is observed, while 1000 samples results in 10 cm and 10°, but with most values falling within 5 cm and 5°.

Using differential measurements while still averaging over 1000 samples results in the data illustrated in figure 9.22. This shows similar, but slightly worse results for transverse

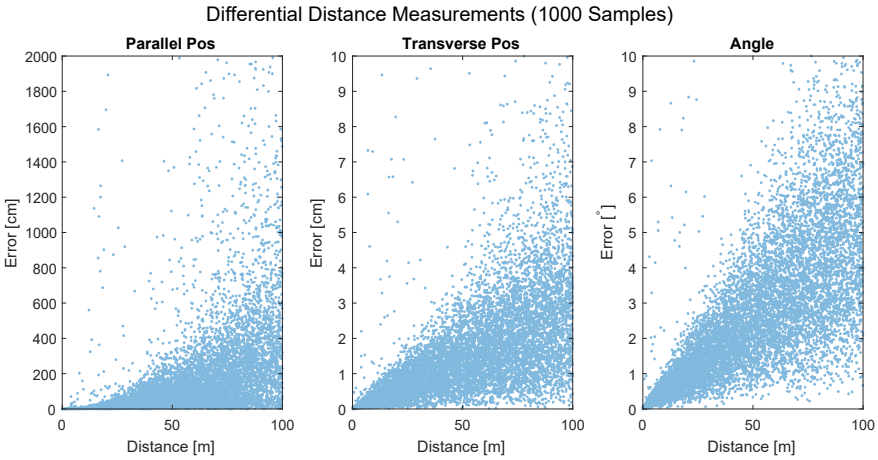


Figure 9.22: Simulation results using differential distance measurements averaged over 1000 samples.

position and angle, however, the parallel position is much worse and is only determined to within 20 m. Performing the same simulation using the assumption of large distances results in figure 9.23. This improves the accuracy in the far range slightly, but at

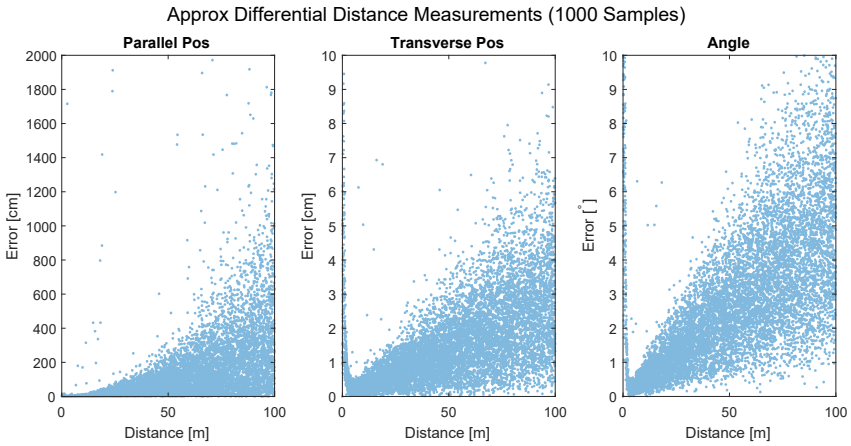


Figure 9.23: Simulation results using differential distance measurements averaged over 1000 samples using approximations for a far away vehicle.

the expense of very large errors for the extremely close range. This is not unexpected considering the approximation involved.

Combining the direct measurements with the long distance approximation gives another set of data which is displayed, compared with the direct measurements alone, in figure 9.24. The results of this are very similar to the direct distance measurements with

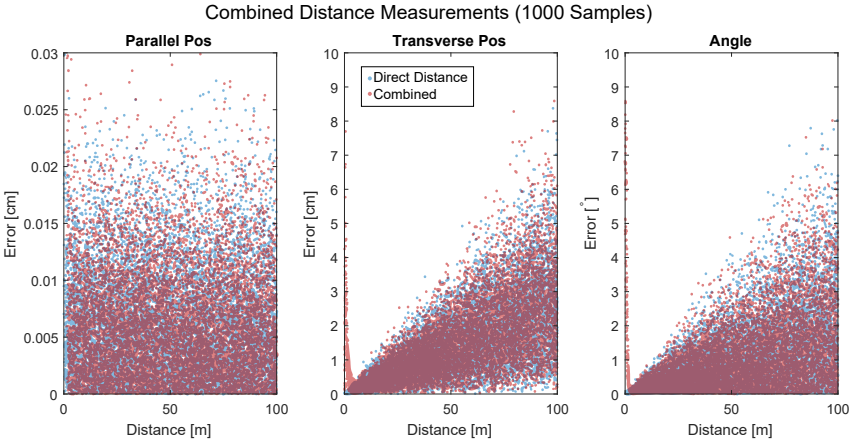


Figure 9.24: Simulation results when using both direct and differential measurements using the 'far away' assumption compared with direct alone.

the additions of large errors in the close range. There also seem to be a very slight improvement in the orientation accuracy, however, this is outweighed by the need for much more computationally expensive processing.

CHAPTER 10

Discussion

From the dual oscillator direct distance measurements it is clear that the current hardware cannot immediately be used for accurate navigation. Two-way communication could mitigate some of the effects caused by frequency offsets, however, oscillator drift happens on such short timescales that it is not possible to transmit the distance information before it is outdated. A way to combat this would be to increase the chirp rate which, in theory, would also allow for higher accuracy in general. The constant frequency offset could be resolved by exchanging the TCXO for a VCTCXO controlled by a feedback mechanism. An elegant solution would be to have the Costas loop control the oscillator directly thus removing the offset in both the carrier and the chirp rate simultaneously. Of course, differences in hardware between any two transceivers might result in the carrier and chirp frequency not being uniformly linked thus leading to the need for additional compensation.

Alternatively, differential distance measurements could be employed. These are clearly not affected by the frequency offset to as large an extend as the direct measurements and actually allow for navigation using only one-way communication. However, the simulations show that a complete system based on only differential measurements would only estimate the transverse position and orientation to an acceptable degree while the distance would be off by several meters. This might be acceptable in some circumstances, but it is clearly not ideal. In any case, there is still a significant amount of short term variation that must be dealt with.

Considering the significant change in fractional delay incurred by heating the TCXO, which is already temperature compensated, using a standard oscillator would likely have made any sort of distance measurement impossible. The temperature dependency of the oscillator is likely the cause of much of the variation exhibited in the data compensated for constant velocity. As such, thermal isolation of the oscillator combined with precise temperature control should remove a considerable amount of this effect. Efforts in this direction were made by encasing the oscillator in glue to limit its interaction with the air. This seemed to help to some extend, but the issue still remained on account of heat transfer through both the glue and the PCB. Furthermore, the other components likely also have some level of temperature dependence affecting the signal.

The transceiver hardware that has been designed costs less than €100 to make per unit, which means that a complete system based on it could most likely be realized for under €1000. In light of this the results must be considered to be quite reasonable with the system being able to track oscillator variations of less than 30 ps (corresponding to 1 cm

distance). If nothing else, the system can be used for inexpensive time synchronization purposes achieving sub nanosecond accuracy even in its current state. But with various improvements it should be possible to create a complete navigation system based on the underlying principle. This speaks to the surprisingly good performance of the fractional sampling method which works well despite its simplicity.

The level of distortion that results from having two simultaneous transmission is troublesome and could potentially result in a complete system not being feasible. However, a large part of the issue is likely caused by hardware inconsistencies as illustrated by the pulseform distortions resulting from attenuating the transmission. Even with high signal-to-noise ratios the distortions still occur - likely as the result of a combination of a DC-offset in the ADC and signal degradation in the front-end. This assertion is corroborated by the fact that the level of CDMA distortion varies by a large amount from transceiver to transceiver indicating a chip specific issue. As such, it seems likely that much better performance could be achieved with custom front-end hardware. Such measures will likely result in the system retaining close to the same level of accuracy regardless of distance since the lower signal level was partly to blame for the reduction in performance. With the remaining troubles being caused by the offset in carrier frequency.

From the simulations it is clear that direct distance measurements provides the best system performance. Furthermore, they also illustrate why it would be beneficial to combine both distance and direction sensors in a single system. The direct distance measurements clearly, and perhaps obviously, provide a very accurate estimate of the inter-vehicle distance, while orientation and transverse position are significantly worse. For the differential estimate, with represents directional measurements, the opposite is true and transverse position and angle are much more accurate than parallel position. Of course, everything is still worse than for the direct distance, however, a dedicated pointing sensor would likely achieve more accurate estimations of direction. What is of interest here is the relative accuracy between parallel and transverse position. Combining the two methods should lead to high levels of accuracy on all accounts as one performs well where the other does not and vice versa.

The geometry of the system constitutes what is more or less the least advantageous configuration for trilateration with all points being gathered close together. This explains why the simulations show much better performance at short ranges. One way to alleviate this issue is to increase the baseline length, however, this puts constraints on the size of the vehicle. Another option is to have more than two vehicles. Three or more vehicles would make the geometry much more conducive to trilateration resulting in increased accuracy for larger distances. Of course, this means that the system design must be extended to allow for the extra vehicles, but it would potentially result in higher and higher accuracy as more vehicles are added to the system. Having more than four transceivers would also improve system performance even though it does not improve the geometry because it leads to additional independent measurements.

Having additional transceivers would also enable more options regarding antenna choice.

The dipole antennas that were used for testing were not appropriate for the system design. For a tetrahedral configuration, an isotropic gain pattern would be close to ideal, but it is not realizable in practice[15]. An alternative could be a hemispherical pattern extending away from the center of the tetrahedron, which would still enable line of sight to three transceivers for most configurations. Placing more transceivers around the vehicle could lead to always having several transceivers visible even though the individual antennas might have narrow gain patterns. Another option is to use high gain antennas combined with pointing mechanisms, although this would complicate the system design considerably. It would also be beneficial to use antennas enabling circular polarization of the radio signal as this would reduce the effect of multipath which have shown to be quite significant.

Even with the potential accuracy that might be achieved if the hardware issues could be alleviated, averaging over several code cycles is still needed in order to obtain the navigational accuracy prescribed in the problem statement. Averaging over 1000 code cycles corresponds to 10 seconds for one-way communication and 20 seconds for two-way. This is clearly not ideal and severely limits the possible uses for the system. Increased chirp rates would decrease this in addition to the earlier mentioned increase in potential accuracy. Whether such measures lead to better performance in practice has unfortunately not been tested on account of the USB connection limiting the maximum sample rate. Implementing the entire signal processing algorithm in hardware would enable such tests, however, the time alloted to this project did not allow for it.

So, while the performance of the developed system only barely lives up to the requirements through excessive averaging, the final result is still quite good considering the simplicity of the hardware and processing algorithm. Furthermore, there are many ways performance can be improved upon and there is much potential in the fractional sampling method.

CHAPTER 11

Conclusion

The purpose of this project was to go through the initial steps in developing a general purpose relative navigation system for use in space and on the ground alike. This was to be achieved though measuring the time of flight of radio signal between several transceivers and converting it to distance.

An initial validation has been performed that showcased it would be possible to obtain highly accurate distance estimates by means of a novel approach to signal sampling and subsequent processing. The validation was performed using a relatively expensive software defined radio as no other suitable ready-made hardware could be identified, not even after a thorough search of available systems. Therefore, to avoid excessive development costs a custom transceiver system was designed and manufactured for use in further testing.

The custom hardware has been used to create a simple distance measurement setup and a series of tests have been performed to characterize its performance. Signal generation, acquisition and processing has been implemented as a mixture of hardware and software using a combination of an FPGA and a PC. The testing showed that it should be possible to measure distances with an accuracy on the scale of 1 cm, however, this would require modifications of the hardware. Without any changes, the hardware still enables distance estimates with accuracies better than ± 50 cm under stable temperature conditions.

The design of a complete relative navigation system using hardware similar to the developed one has been completed and includes descriptions of top level software, hardware and communication protocols. Simulations have been performed using the potential accuracy determined from the hardware experiments as a basis to evaluate the performance of the designed system. It has been found that using only the distance measurements it is possible to obtain positional and rotational accuracy better than ± 5 cm and $\pm 5^\circ$, respectively, for distances up to 100 m using baselines of 1 m. However, this requires extensive time-averaging with the result that navigation updates would only be available at a rate of 0.1 Hz.

Given the extend of the project and the relatively limited amount of time available, it is the opinion of the author that these results must be considered quite positive as the project requirements have been met and the method shown to work. With more time, and possibly the inclusion of additional sensors, it should be possible to create a complete relative navigation system that maintains the low complexity and size shown in this project, but with a significantly better accuracy.

CHAPTER 12

Future Work

Following the conclusion of this project there are a number of avenues that could potentially be interesting to explore further.

Carrier recovery through a Costas loop should lead to improved stability and accuracy, but so far implementations in both hardware and software have failed to perform satisfactory. Investigating why this has been the case and ultimately creating a reliable solution would be interesting and could potentially enable more noise-robust performance through pulse shaping.

Currently, the emitted signal consists of a series of square pulses, however, there is no reason why more advanced shapes could not be employed. Since fractional sampling enables the reconstruction of the pulseform, high levels of reliability could be achieved through cross-correlation between the recovered and transmitted pulseforms, with the exact shape determining the correlation function. Although, this might confuse the Costas loop leading to the requirement of more advanced carrier recovery methods.

Accuracy could also be improved by the addition of directional sensors working in conjunction with the distance measurements and/or having additional vehicles. Developing the extensions to the system that such measures would require should be a fairly trivial, but nevertheless time consuming process. Measurements of carrier phase might also be beneficial to include in the system design.

The whole data inversion process could also be investigated further by experimenting with different methods for solving inverse problems. The current least square solution is relatively simple and more advanced solutions might lead to better overall performance. Of course, these methods must still be fast enough to be implemented on microprocessor hardware.

Embedding the entire delay determination in hardware would enable experiments with much higher chirp rates which could potentially increase accuracy, but it is possible that the observed level is the limit of what is achievable with the current hardware. As such the development of a custom radio front-end solution should also be pursued.

Finally, an interesting subject is that of antennas. Not only their gain patterns and how conducive they are to omni-directional sensing, but also the exact location from where the signal is emitted and received. After all, when the antennas are much longer than the observed uncertainty, whether the signal is emitted from the tip, the middle, or the bottom of the antenna is quite significant and should be investigated.

APPENDIX A

Selected Pre-project Results

Here follows results from simulations performed during the pre-project.

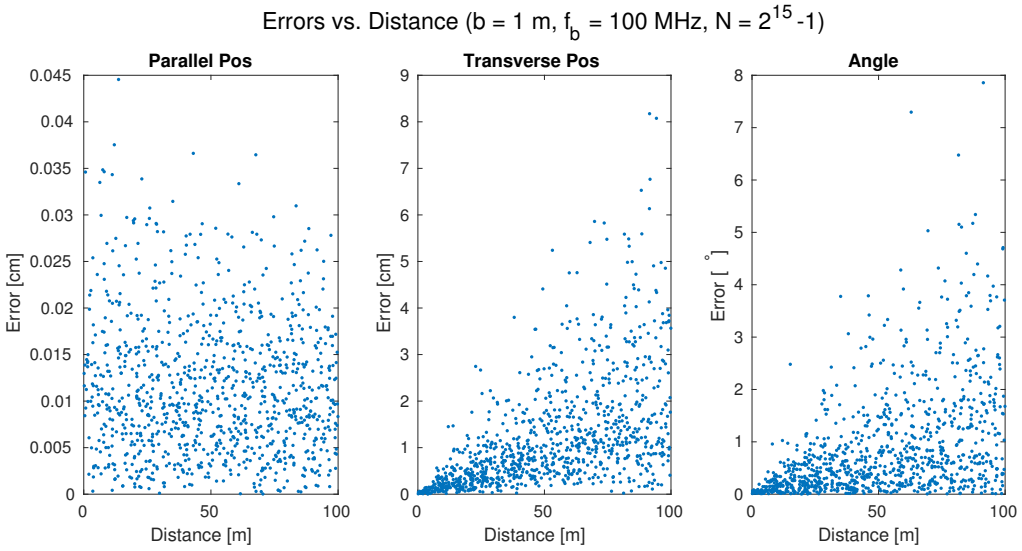


Figure A.1: Errors as a function of distance. Adapted from [9].

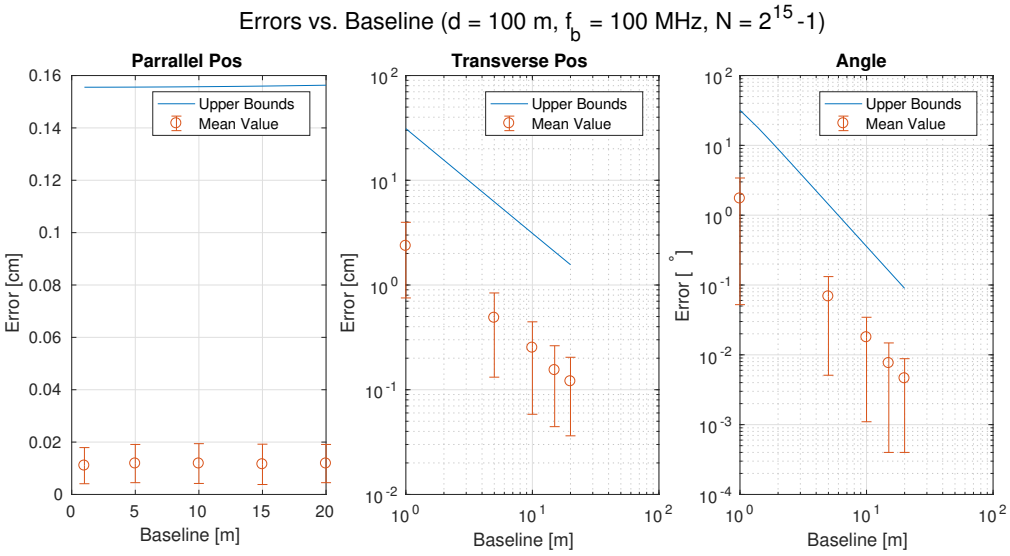


Figure A.2: Errors as a function of baseline length. The blue lines follow the upper boundaries as expected from the underlying theory. Adapted from [9].

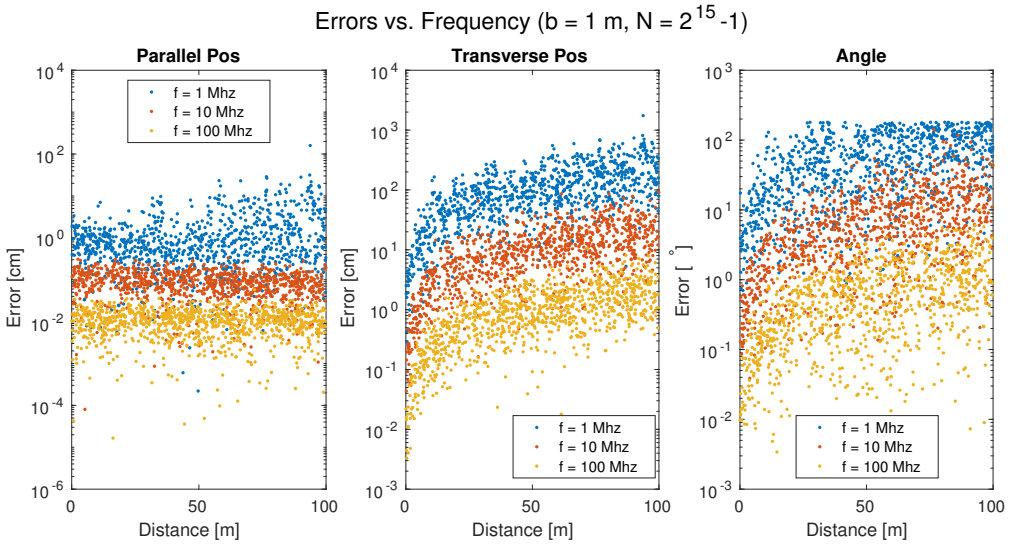


Figure A.3: Errors as a function of frequency. Adapted from [9].

APPENDIX B

Impedance Parameters

The effective relative oddmode permittivity is given by

$$\varepsilon_{r,eff,o} = [0.5(\varepsilon_r + 1) + a_0 - \varepsilon_{r,eff}]\, e^{-c_0 g^{d_0}} + \varepsilon_{r,eff} \tag{B.1}$$

where the parameters a_0, b_0, c_0 and d_0 are respectively equal to

$$a_0 = 0.7287[\varepsilon_{r,eff} - 0.5(\varepsilon_r + 1)](1 - e^{-0.179u}) \tag{B.2}$$

$$b_0 = \frac{0.747\varepsilon_r}{0.15 + \varepsilon_r} \tag{B.3}$$

$$c_0 = b_0 - (b_0 - 0.207)e^{-0.414u} \tag{B.4}$$

$$d_0 = 0.593 + 0.694e^{-0.562u} \tag{B.5}$$

with u being the ratio of trace width to dielectric height:

$$u = \frac{w}{h}. \tag{B.6}$$

The Q_{10} factor is calculated by means of the following 10 equations:

$$Q_1 = 0.8695u^{0.194} \tag{B.7}$$

$$Q_2 = 1 + 0.7519g + 0.189g^{2.31} \tag{B.8}$$

$$Q_3 = 0.1975 + \left[16.6 + \left(\frac{8.4}{g}\right)^6\right]^{-0.387} + \frac{1}{241}\ln\left[\frac{g^{10}}{1 + \left(\frac{g}{3.4}\right)^{10}}\right] \tag{B.9}$$

$$Q_4 = 2\frac{Q1}{Q2\left[e^{-g}u^{Q_3} + (2 - e^{-g})u^{-Q3}\right]} \tag{B.10}$$

$$Q_5 = 1.794 + 1.14\ln\left(1 + \frac{0.638}{g + 0.517g^{2.43}}\right) \tag{B.11}$$

$$Q_6 = 0.2305 + \frac{1}{281.3}\ln\left[\frac{g^{10}}{1 + \left(\frac{g}{5.8}\right)^{10}}\right] + \frac{1}{5.1}\ln(1 + 0.598g^{1.154}) \tag{B.12}$$

$$Q_7 = \frac{10 + 190g^2}{1 + 82.3g^3} \tag{B.13}$$

$$Q_8 = \exp\left[-6.5 - 0.95\ln(g) - \left(\frac{g}{0.15}\right)^5\right]; \tag{B.14}$$

$$Q_9 = \ln(Q_7)\left(Q_8 + \frac{1}{16.5}\right); \tag{B.15}$$

$$Q_{10} = \frac{1}{Q_2}\left(Q_2Q_4 - Q_5\exp\left[\ln(u)Q_6u^{-Q_9}\right]\right); \tag{B.16}$$

where g is given by

$$g = \frac{s}{h} \tag{B.17}$$

with s being the separation between the traces[10].

APPENDIX C

Transceiver Design Resources

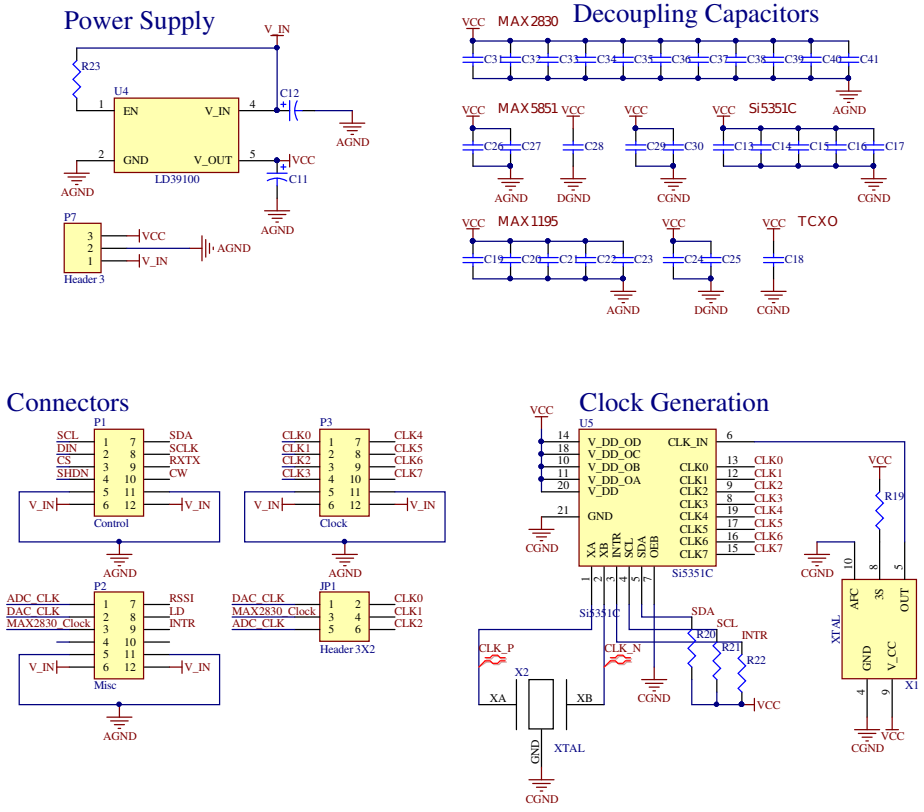


Figure C.1: Secondary sections of the transceiver schematic.

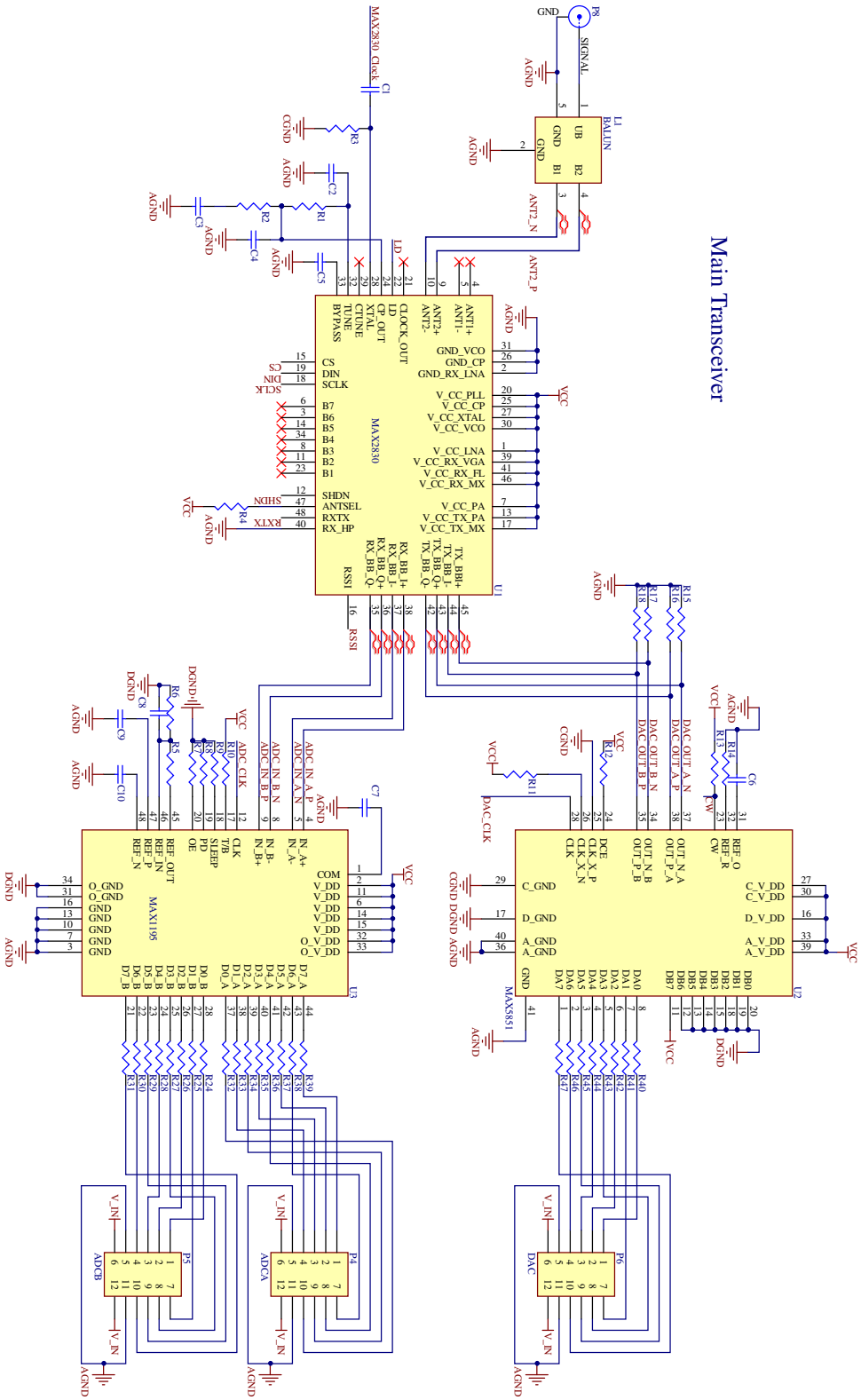


Figure C.2: Primary section of the transceiver schematic.

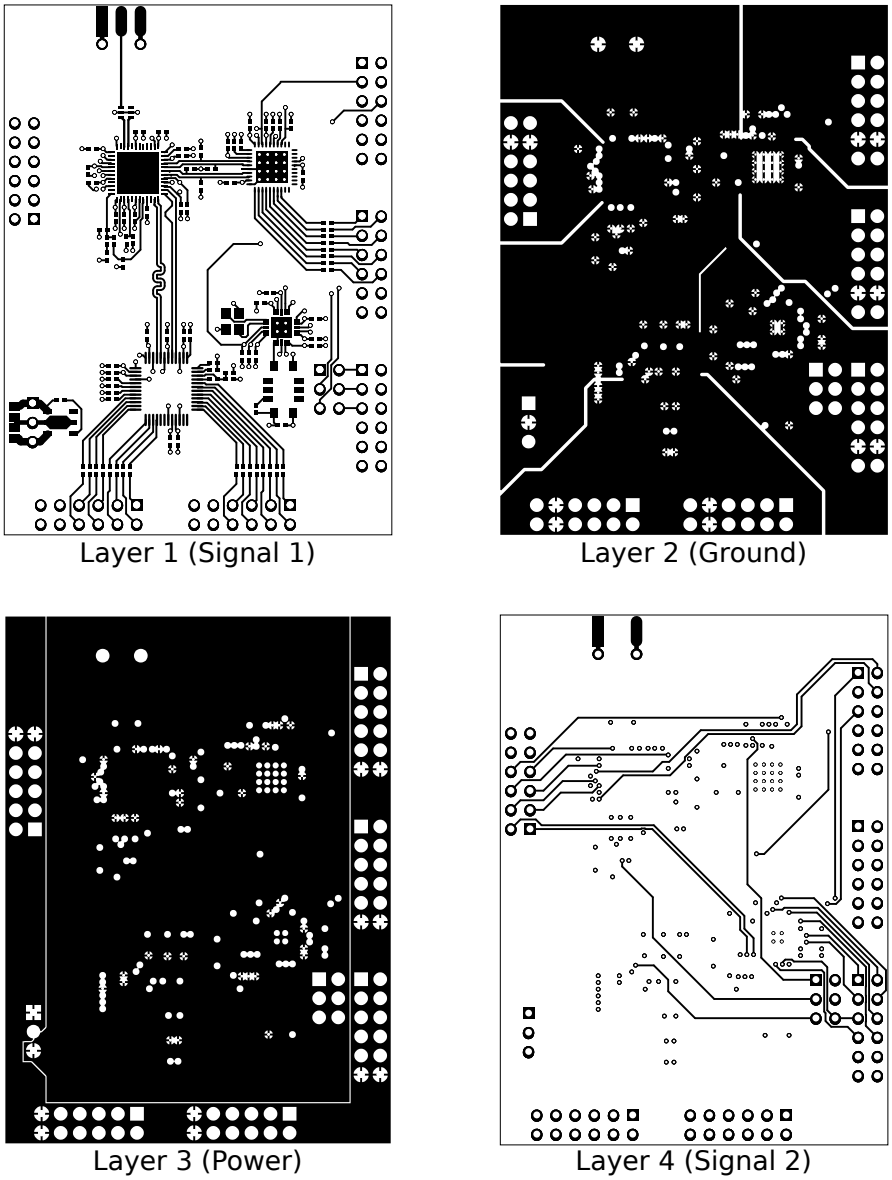


Figure C.3: The different layers of the PCB (reproduced to scale).

Name	Footprint	Value	Area
C1	C1206	100pF	MAX2830
C2	C1206	68pF	
C3	C1206	2200pF	
C4	C1206	100pF	
C5	C1206	100pF	
C6	C1206	100nF	MAX5851
C7	C1206	100nF	MAX1195
C8	C1206	100nF	
C9	C1206	100nF	
C10	C1206	100nF	
C11	C0805	4.7uF	LDO
C12	C0805	10uF	
C13	C1206	100nF	Si5351C
C14	C1206	100nF	
C15	C1206	100nF	
C16	C1206	100nF	
C17	C1206	100nF	
C18	C1206	100nF	TCXO
C19	C1206	100nF	MAX1195
C20	C1206	100nF	
C21	C1206	100nF	
C22	C1206	100nF	
C23	C1206	100nF	
C24	C1206	100nF	
C25	C1206	100nF	
C26	C1206	100nF	MAX5851
C27	C1206	100nF	
C28	C1206	100nF	
C29	C1206	100nF	
C30	C1206	100nF	
C31	C1206	100nF	MAX2830
C32	C1206	100nF	
C33	C1206	100nF	
C34	C1206	100nF	
C35	C1206	100nF	
C36	C1206	100nF	
C37	C1206	100nF	
C38	C1206	100nF	
C39	C1206	100nF	
C40	C1206	100nF	
C41	C1206	100nF	
R1	C0402	750	MAX2830
R2	R0402	1.2K	
R3	R0402	2K	
R4	R0402	10K	
R5	R0402	10k	MAX1195
R6	R0402	10k	
R7	R0402	10k	

Name	Footprint	Value	Area
R8	R0402	10k	MAX1195
R9	R0402	10k	
R10	R0402	10k	
R11	R0402	10k	MAX5851
R12	R0402	10k	
R13	R0402	1.9k	
R14	R0402	1.9k	
R15	R0402	100	
R16	R0402	100	
R17	R0402	100	
R18	R0402	100	
R19	R0402	10K	TCXO
R20	R0402	1K	Si5351C
R21	R0402	1K	
R22	R0402	1.5K	
R23	R0402	10K	LDO
R24	R0402	100	MAX1195
R25	R0402	100	
R26	R0402	100	
R27	R0402	100	
R28	R0402	100	
R29	R0402	100	
R30	R0402	100	
R31	R0402	100	
R32	R0402	100	MAX1195
R33	R0402	100	
R34	R0402	100	
R35	R0402	100	
R36	R0402	100	
R37	R0402	100	
R38	R0402	100	
R39	R0402	100	
R40	R0402	100	MAX5851
R41	R0402	100	
R42	R0402	100	
R43	R0402	100	
R44	R0402	100	
R45	R0402	100	
R46	R0402	100	
R47	R0402	100	
U1		MAX2830	
U2		MAX5851	
U3		MAX1195	
U4		XD6220 LDO	
U5		Si5351C	
X1		25MHz TCXO	
X2		27MHz XTAL	

Figure C.4: List of transceiver components.

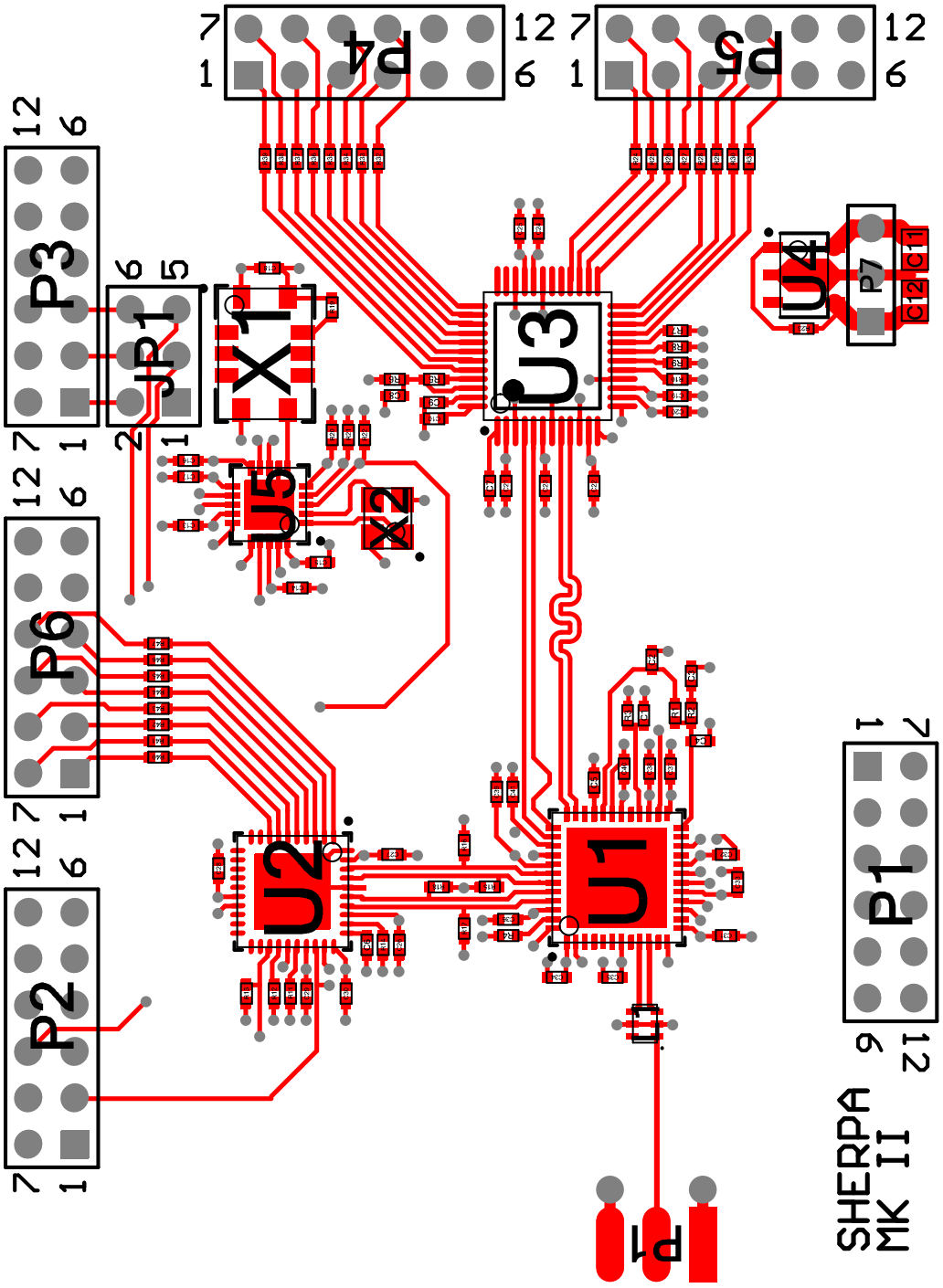
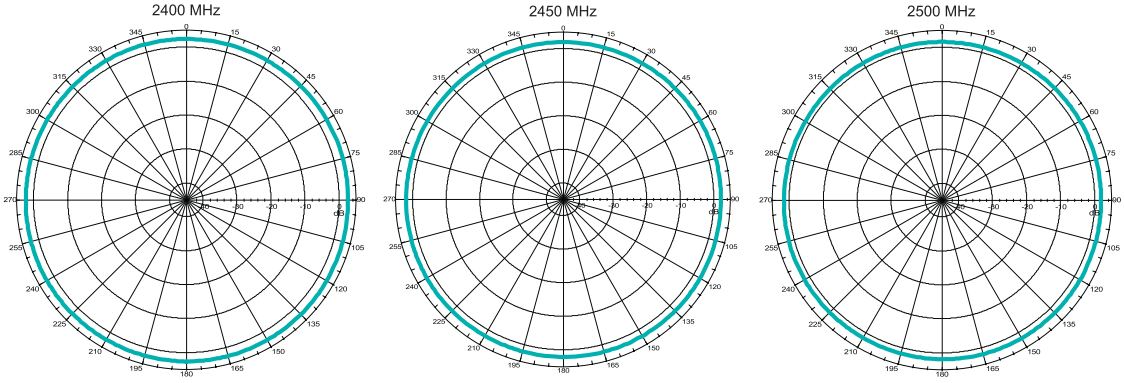


Figure C.5: Layout of the different transceiver components.

Gain Performance W1030

Horizontal Position



Vertical Position

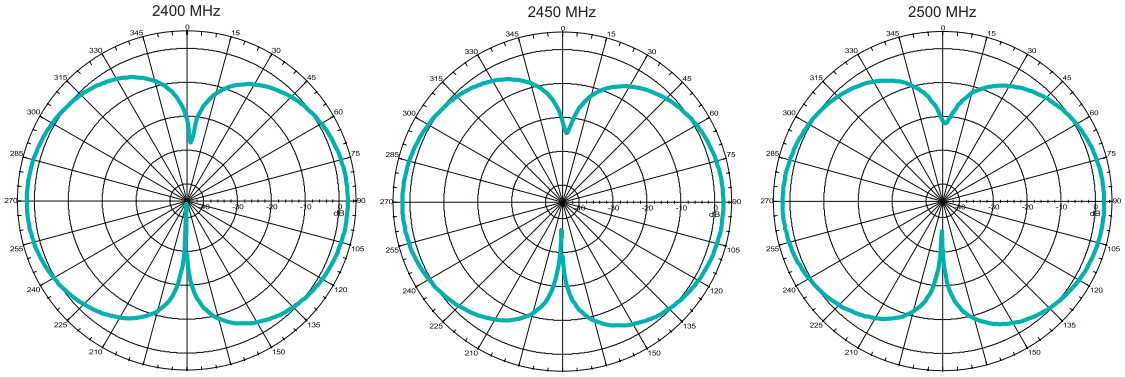


Figure C.6: Gain pattern of the antennae used for testing. Adapted from [32].

APPENDIX D

Source Code

The source code for this project is available under the GPL v3.0 license and can be found on GitHub at the following address `https://github.com/Dragonturtle/SHERPA`.

The code is split up into three different folders; `HDL` contains the VHDL code used to program the Xilinx Spartan® 6 for signal acquisition and generation; `Host` contains the C-code for the PC application that communicates with the FPGA and computes distances. Also included in this folder is the Kst2 configuration file enabling live viewing of the data processing. In order to compile the C-code the "makestuff"[21] build environment is needed to provide the FPGALink library functionality; `Simulation` contains the MATLAB® scripts used for simulation purposes;

Also included is the `Hardware` folder which holds the Altium Designer™project files describing the PCB design of the custom made transceiver.

The main sections of the code are also included in the following.

D.1 Host Application

C:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>
#include <errno.h>

#include <libfpgalink.h>

#include <math.h>

#define M_PI 3.14159265358979323846
#define INT_THRESHOLD 3000
#define FRAC_THRESHOLD 0.3

void PRNCode(int8 *array) {
  FILE* f = fopen("./PRNCode_1.txt", "r");
  int number = 0;
  int i = 0;
  while( fscanf(f, "%d\n", &number) > 0 )
  {
    array[i] = (int8)number;
    i++;
  }

  fclose(f);
  return;
}

int main(int argc, const char *argv[]) {
  const char* vp  = "1d50:602b";
  const char* ivp = "1443:0007"; //Nexys 3 ID
  const char* progConfig = "J:D0D2D3D4:../../HDL/FPGA.xsvf";

  const char *error = NULL;
  struct FLContext *handle = NULL;
  uint8 flag; FLStatus status;
  bool isCommCapable, isNeroCapable;
  uint32 requestLength, actualLength;

  FILE *fp = NULL; FILE *fp_delay = NULL;

  const int N = 32766;
  const int8 *data;
  uint8 buffer[32766];
  int8 prn1[N+1];
  int8 bufferI1[32766] = {0}, bufferQ1[32766] = {0};
  int8 bufferI2[32766] = {0}, bufferQ2[32766] = {0};
  double pulseform1[32766*3] = {0}, pulseform2[32766*3] = {0};

  double meanI1 = 0, meanI2 = 0, meanQ1 = 0, meanQ2 = 0;
  double xcorrI1 = 0, xcorrQ1 = 0;
  double meanK1  = 0, meanK2 = 0;
  double k1r = 0, k2r = 0;
  double k1f = 0, k2f = 0;
  double k1 = 0, k2 = 0;

  int timeVal = 0, iteration = 0, lockCounter = 0 ;
  int bufferOffset = 0, sampleOffset = 0;
  bool lock = false;


```

```
62
63    printf("\n\n");
64    printf("****************************************\n");
65    printf("*      S.H.E.R.P.A. Host Software      *\n");
66    printf("*                v. 0.2                *\n");
67    printf("*            (c) LAMC 2017             *\n");
68    printf("****************************************\n\n");
69
70    //Initialize FPGALink
71    status = flInitialise(0, &error);
72    if (status) goto cleanup;
73
74    //Connect to USB device
75    printf("Attempting to open connection to FPGALink device %s...\n", vp);
76    status = flOpen(vp, &handle, NULL);
77
78    if ( status ) {
79      printf("Connection failed, attempting to load standard firmware...\n");
80      status = flLoadStandardFirmware(ivp, vp, &error);
81
82      if (status == FL_SUCCESS){
83        printf("Loading complete, awaiting renumeration...\n");
84        flSleep(1000);
85        for (int i = 0 ; i < 60 ; i++){
86          status = flIsDeviceAvailable(vp, &flag, &error);
87          if (status) goto cleanup;
88          if (flag) break;
89
90          flSleep(250);
91        }
92        if ( !flag ) {
93          fprintf(stderr, "Device did not renumerate properly as %s\n", vp);
94          goto cleanup;
95        }
96
97        printf("Reattempting to open connection to FPGALink device %s...\n", vp);
98        status = flOpen(vp, &handle, &error);
99        if (status) goto cleanup;
100
101       isNeroCapable = flIsNeroCapable(handle);
102       if (isNeroCapable){
103         if (status) goto cleanup;
104         //Program FPGA
105         printf("Programming FPGA...\n");
106         status = flProgram(handle, progConfig, NULL, &error);
107
108         if (status) goto cleanup;
109         printf("Programming Succesful!\n");
110       } else {
111         printf("Device does not support NeroJTAG!\n");
112         goto cleanup;
113       }
114     }
115     else {
116       goto cleanup;
117     }
118   }
119   printf("Connection Succesful!\n");
120
121   isCommCapable = flIsCommCapable(handle, 0x01);
122   if (isCommCapable){
123     //Set up conduit
124     printf("Selecting conduit 0x01...\n");
125     status = flSelectConduit(handle, 0x01, &error);
126     if (status) goto cleanup;
127
```

```
128     printf("Searching for lock...\n");
129     fp = fopen("data.bin","wb");
130     fp_delay = fopen("delay.txt","w");
131
132     PRNCode(prn1);
133
134     while (iteration < 501){
135       status = flReadChannelAsyncSubmit(handle, 0x00, (uint32)N, buffer, &error);
136       if (status) goto cleanup;
137       status = flReadChannelAsyncAwait(handle, &data, &requestLength, &actualLength, &error);
138       if (status) goto cleanup;
139
140       for (int i = 0 ; i < N ; i++){
141         bufferI1[i] = (double)( ( (buffer[i] >> 0) & 0x01) ) * 2 - 1;
142         bufferI2[i] = (double)( ( (buffer[i] >> 1) & 0x01) ) * 2 - 1;
143         bufferQ1[i] = (double)( ( (buffer[i] >> 4) & 0x01) ) * 2 - 1;
144         bufferQ2[i] = (double)( ( (buffer[i] >> 5) & 0x01) ) * 2 - 1;
145       }
146
147       /* Uncommen this to save raw data
148       if(iteration > 1)
149         fwrite(buffer, 1, sizeof(buffer), fp);
150       iteration++;
151       /**/
152
153       if (!lock){
154         //===================================//
155         //==  Find Initial Integer Delay  ==//
156         //===================================//
157         xcorrI1 = 0; xcorrQ1 = 0;
158         for (int j = 0 ; j < 10 ; j++){
159           for (int k = 0 ; k < 32767 ; k++){
160             xcorrI1 += prn1[k] * bufferI1[(bufferOffset + k + N) % 32766];
161             xcorrQ1 += prn1[k] * bufferQ1[(bufferOffset + k + N) % 32766];
162           }
163
164           if ( sqrt(xcorrI1 * xcorrI1 + xcorrQ1 * xcorrQ1) > INT_THRESHOLD ){
165             lock = true; lockCounter = 0;
166             printf("Locked @ %d (corr: %f)\n", bufferOffset,
167             sqrt(xcorrI1 * xcorrI1 + xcorrQ1 * xcorrQ1));
168             break;
169           } else {
170             if (bufferOffset % 1000 == 0)
171             printf("%d\n", bufferOffset);
172
173             bufferOffset++;
174           }
175         }
176       } else {
177         //==============================//
178         //==  Find Fractional Delay  ==//
179         //==============================//
180         meanI1 = 0; meanI2 = 0;
181         meanQ1 = 0; meanQ2 = 0;
182         k1r = 0, k1f = 0;
183         k2r = 0, k2f = 0;
184         int prnOffset = 0, codeOffset = 0;
185         for (int k = 0 ; k < 32766*3; k++){
186           if (k + bufferOffset + prnOffset >= 32766)
187           prnOffset -= 32766;
188           if (k + codeOffset == 32767)
189           prnOffset -= 32767;
190
191           meanI1 = meanI1 * 0.9998 + 0.0002 * prn1[k + codeOffset] * bufferI1[bufferOffset +
                 k + prnOffset];
192           meanQ1 = meanQ1 * 0.9998 + 0.0002 * prn1[k + codeOffset] * bufferQ1[bufferOffset +
```

```
            k + prnOffset];
193         meanI2 = meanI2 * 0.9998 + 0.0002 * prn1[k + codeOffset] * bufferI2[bufferOffset +
                k + prnOffset];
194         meanQ2 = meanQ2 * 0.9998 + 0.0002 * prn1[k + codeOffset] * bufferQ2[bufferOffset +
                k + prnOffset];
195
196         pulseform1[k] = pulseform1[k] * 0.95 + 0.05 * sqrt(meanI1 * meanI1 + meanQ1 *
                meanQ1);
197         pulseform2[k] = pulseform2[k] * 0.95 + 0.05 * sqrt(meanI2 * meanI2 + meanQ2 *
                meanQ2);
198
199         if (pulseform1[k] > 0.2 && k1r == 0){
200           k1r = k;
201         } else if (pulseform1[k] < 0.3 && k1f == 0 && k1r != 0 && k > k1r + 30000 ){
202           k1f = k;
203         }
204
205         if (pulseform2[k] > 0.2 && k2r == 0){
206           k2r = k;
207         } else if (pulseform2[k] < 0.3 && k2f == 0 && k2r != 0 && k > k2r + 30000 ){
208           k2f = k;
209         }
210       }
211
212
213       if ( (k1r != 0 && k1f != 0) && (k2r != 0 && k2f != 0) ){
214         k1 = (k1r + k1f) / 2;
215         k2 = (k2r + k2f) / 2;
216         meanK1 = meanK1 * 0.9 + 0.1 * (k1 + sampleOffset);
217         meanK2 = meanK2 * 0.9 + 0.1 * (k2 + sampleOffset);
218
219         lockCounter = 0;
220         fprintf(fp_delay, "%d,%f,%f,%f,%f\n", timeVal,
221         (k1 + sampleOffset) / 3276700.0 / 32766.0 * 299792458.0 * 100.0,
222         (k2 + sampleOffset) / 3276700.0 / 32766.0 * 299792458.0 * 100.0,
223          meanK1             / 3276700.0 / 32766.0 * 299792458.0 * 100.0,
224          meanK2             / 3276700.0 / 32766.0 * 299792458.0 * 100.0);
225
226         if (k1 < 16383 || k2 < 16383){
227           bufferOffset -= 1;
228           sampleOffset -= 32767;
229           timeVal++;
230
231           for (int k = 0 ; k < 32766*3; k++){
232             pulseform1[k] = 0;
233             pulseform2[k] = 0;
234           }
235         }else if (k1 == 81915 && k2 == 81915){
236           bufferOffset += 1;
237           sampleOffset += 32767;
238           timeVal--;
239
240           for (int k = 0 ; k < 32766*3; k++){
241             pulseform1[k] = 0;
242             pulseform2[k] = 0;
243           }
244         }
245
246       } else {
247         lockCounter++;
248         if (lockCounter > 1000){
249           printf("%d\n", bufferOffset);
250           bufferOffset -= 2000;
251           lock = false;
252         }
253       }
```

```
254       }
255
256       timeVal++;
257       if (bufferOffset < 0){
258         bufferOffset += 32766;
259         sampleOffset += 2*32767;
260       } else if (bufferOffset >= 32766){
261         bufferOffset -= 32766;
262         sampleOffset -= 2*32767;
263       }
264     }
265
266     if (fp == NULL) {
267       printf("fopen failed, errno = %d\n", errno);
268     } else {
269       fclose(fp);
270       fclose(fp_delay);
271     }
272   } else{
273     printf("Device does not support CommFPGA!\n");
274     goto cleanup;
275   }
276
277   //Clean up and handle errors
278   cleanup:
279   if ( error ) {
280     fprintf(stderr, "%s\n", error);
281     flFreeError(error);
282   }
283   flClose(handle);
284
285   return 0;
286 }
```

code/main.c.

MATLAB:

```
%%%%%%%%%%%%%%%
%% Load Data %%
%%%%%%%%%%%%%%%
clear; clc;

fprintf('Loading Data..');
fileID = fopen('data.bin');
dataRaw = uint8(fread(fileID,inf,'uint8'));

sampleRate = 3276600;
msgLength  = 32767;
CA1 = PRNCode(1,msgLength);

fprintf('.Done!\n');

%%%%%%%%%%%%%%%%%%
%% Unfold Data %%%
%%%%%%%%%%%%%%%%%%
fprintf('Unfolding Data..');

I = zeros(length(dataRaw),4);
I(:,1) = bitand(dataRaw,1)   / 1;
I(:,2) = bitand(dataRaw,2)   / 2;
I(:,3) = bitand(dataRaw,4)   / 4;
I(:,4) = bitand(dataRaw,8)   / 8;
I = I * 2 - 1;

Q = zeros(length(dataRaw),4);
Q(:,1) = bitand(dataRaw,16)  / 16;
Q(:,2) = bitand(dataRaw,32)  / 32;
Q(:,3) = bitand(dataRaw,64)  / 64;
Q(:,4) = bitand(dataRaw,128) / 128;
Q = Q * 2 - 1; Q = -Q; % Q is inverted on the PCB

t = ((0:length(I)-1)/sampleRate)';
N = floor(length(I)/msgLength)-4;

fprintf('.Done!\n');

clear dataRaw dataUnfold fileID;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Find initial integer delay %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
fprintf('Determining Initial Integer Delay..');

CA1_S  = fft(CA1,msgLength);
A_S    = fft(I(1:msgLength,1));
B_S    = fft(Q(1:msgLength,1));

ACA1_S = A_S .* conj(CA1_S);
BCA1_S = B_S .* conj(CA1_S);
ACA1   = ifft(ACA1_S);
BCA1   = ifft(BCA1_S);

index = find(sqrt(BCA1.^2+ACA1.^2) > 2000);
%index = index(1) + msgLength-1;
startIndices = index:(msgLength-1): ...
               (index+(msgLength-1)*(N-1));
initialIntDelay = index - (msgLength-1);

fprintf('.Done!\n');

figure
plot(sqrt(BCA1.^2+ACA1.^2))
hold on; plot(sqrt(BCA1.^2+ACA1.^2),'o'); hold off;
```

```
xlim([-10 10] + startIndices(1))
xlabel('Sample #')
ylabel('Correlation (XCVR 1)')

%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Find fractional delay %%
%%%%%%%%%%%%%%%%%%%%%%%%%%
pulseformAvg1 = zeros(3*msgLength,1);
pulseformAvg2 = zeros(3*msgLength,1);
mixIQ1 = zeros(3*msgLength,1);
mixIQ2 = zeros(3*msgLength,1);
fracDelay1 = zeros(length(startIndices)-50,1);
fracDelay2 = zeros(length(startIndices)-50,1);

risingEdge1 = 0; risingEdge2 = 0;
fallingEdge1 = 0; fallingEdge2 = 0;
startOffset = 0;

ii1 = 0; ii2 = 0;
qq1 = 0; qq2 = 0;
n = 3; start = -2;
code = repmat(CA1,n,1);
for i = 1:length(startIndices)
    idx = startIndices(i) + 1 + startOffset;

    % Mixing with code
    mixI1 = code .* I(idx+start:idx+(2^15-1)*n-1+start,1);
    mixI2 = code .* I(idx+start:idx+(2^15-1)*n-1+start,2);
    mixQ1 = code .* Q(idx+start:idx+(2^15-1)*n-1+start,1);
    mixQ2 = code .* Q(idx+start:idx+(2^15-1)*n-1+start,2);

    % Averaging and combination of channels
    ii1 = 0; ii2 = 0;
    qq1 = 0; qq2 = 0;
    for j = 1:32767*3
        ii1 = ii1 * 0.9998 + 0.0002 * mixI1(j);
        ii2 = ii2 * 0.9998 + 0.0002 * mixI2(j);
        qq1 = qq1 * 0.9998 + 0.0002 * mixQ1(j);
        qq2 = qq2 * 0.9998 + 0.0002 * mixQ2(j);

        mixIQ1(j) = sqrt(ii1^2 + qq1^2);
        mixIQ2(j) = sqrt(ii2^2 + qq2^2);
    end

    % Averaging pulseforms
    pulseformAvg1 = min(pulseformAvg1 * 0.95 + 0.05 * mixIQ1,1);
    pulseformAvg2 = min(pulseformAvg2 * 0.95 + 0.05 * mixIQ2,1);

    % Finding rising and falling edges
    if (i > 50)
        risingEdge1 = find(pulseformAvg1 > 0.2);
        risingEdge2 = find(pulseformAvg2 > 0.2);
        if (length(risingEdge1) > 1)
            fracDelay1(i-50) = risingEdge1(1);
            fracDelay2(i-50) = risingEdge2(1);
            fallingEdge1 = find(pulseformAvg1 < 0.3);
            fallingEdge2 = find(pulseformAvg2 < 0.3);
            if (length(fallingEdge1) > 0)
                fallingEdge1 = ...
                    fallingEdge1(fallingEdge1 > risingEdge1(1)+30000);
                fallingEdge2 = ...
                    fallingEdge2(fallingEdge2 > risingEdge2(1)+30000);
                if (length(fallingEdge1) > 0)
                   fracDelay1(i-20) = (fallingEdge1(1) + risingEdge1(1))/2;
                   fracDelay2(i-20) = (fallingEdge2(1) + risingEdge2(1))/2;
                end
```

```
132             end
133         end
134     end
135
136     % Handling edge cases
137     if (i > 50)
138         if (risingEdge1(1) < 32766*0.25 || ...
139             risingEdge2(1) < 32766*0.25)
140             startOffset = startOffset - 1;
141             pulseformAvg1 = circshift(pulseformAvg1,32766);
142             pulseformAvg2 = circshift(pulseformAvg2,32766);
143         elseif (fallingEdge1(1) > 32766*2.75 || ...
144                 fallingEdge2(1) > 32766*2.75)
145             startOffset = startOffset + 1;
146             pulseformAvg1 = circshift(pulseformAvg1,-32766);
147             pulseformAvg2 = circshift(pulseformAvg2,-32766);
148         end
149     end
150 end
```

code/main.m.

D.2 Simulation

```
clear; clc;
%% Initialization
baseline = 1; c = 299792458;
XCVRBase = [[+1 +0 -1/sqrt(2)]; [-1 +0 -1/sqrt(2)]; ...
           [+0 +1 +1/sqrt(2)]; [+0 -1 +1/sqrt(2)]]' / 2 * baseline;

N = 10000;
distanceError = zeros(18,1);
G = zeros(18,7);
m = ones(7,1);
epsilon = 1e-5; diffOffset = zeros(7,1);
errors = zeros(N,4);
for kk = 1:N
    %% Generate Simulated Measurements
    targetPos = randn(3,1);
    targetPos = targetPos / norm(targetPos) * rand(1) * 100;
    targetRot = (rand(1,3).*[2 1 2]-[1 1/2 1]) * pi;

    R = SpinCalc('EA321toDCM',rad2deg(targetRot),1e-10,0);
    XCVRTarget = R * XCVRBase + repmat((targetPos),1,4);

    measurements = CalculateDistances(XCVRTarget, XCVRBase);
    measurements = measurements + mean(randn(1000,16)*0.01)';

    %% Perform LSQ
    for k=1:20
        estXCVRTarget = EstimateTargetPoints(m, 4, XCVRBase);
        estDistances  = CalculateDistances(estXCVRTarget, XCVRBase);
        distanceError = measurements - estDistances;
        distanceError(17) = 1 - sum(m(4:7).^2);   % norm(q) = 1
        distanceError(18) = 1 - sum(m(4:7).^2)^3; % determinant(R) = 1

        % Calculate Jacobian
        for i=1:7
            diffOffset = circshift([epsilon;0;0;0;0;0;0],i-1);
            pointsTemp =  EstimateTargetPoints(m + diffOffset, 4, XCVRBase);
            G(1:16,i)  = (CalculateDistances(pointsTemp, XCVRBase) - ...
                          estDistances) / epsilon;
        end
        G(17,4:7)    = 2 * m(4:7);                    % norm(q) = 1
        G(18,4:7)    = 6 * sum(m(4:7).^2)^2 * m(4:7); % determinant(R) = 1

        % Matrix Inversion
        dM  = (G'*G)^(-1)*G' * distanceError;
        m      = m + dM;
        m(4:7) = m(4:7) / norm(m(4:7)); % Enforce norm(q) = 1
    end
    estTargetPos = m(1:3);
    estTargetRot = GenerateRotationMatrix(m(4:7),'q');

    %% Determine Errors
    %SpinCalc is an external library developed by John Fuller in 2009
    %Basically, it converts between different rotation descriptions
    x = SpinCalc('DCMtoQ',R,1e-10,0);
    y = SpinCalc('DCMtoQ',estTargetRot,1e-10,0);
    z = quatmultiply(x,quatconj(y));

    err_pos = norm(targetPos - estTargetPos);
    err_rot = rms(wrapTo360(fliplr(SpinCalc('DCMtoEA321',estTargetRot,1e-10,0))) - ...
        wrapTo360(rad2deg(targetRot)) );

    errors(kk,:) = [norm(targetPos), ...
                    abs(dot(estTargetPos-targetPos,targetPos)/norm(targetPos))*100, ...
                    norm(cross(estTargetPos-targetPos,targetPos)/norm(targetPos))*100, ...
```

```
                    abs(wrapTo180(rad2deg(2*acos(z(1)))))]';
end

%% Plot the Error Distribution
figure;
subplot(1,3,1); plot(errors(:,1),errors(:,2),'.'); title('Parallel Pos');
xlabel('Distance [m]'); ylabel('Error [cm]'); ylim([0 0.03])
subplot(1,3,2); plot(errors(:,1),errors(:,3),'.'); title('Transverse Pos');
xlabel('Distance [m]'); ylabel('Error [cm]'); ylim([0 10])
subplot(1,3,3); plot(errors(:,1),errors(:,4),'.'); title('Angle');
xlabel('Distance [m]'); ylabel('Error [^\circ]'); ylim([0 10])
suptitle('Direct Distance Measurements (1000 Samples)')
```

code/DirectDistance.m.

```
function CA = PRNCode(PRN, num)
%% SHERPA Gold Code Generator
if nargin == 1
    n = 1023;
else
    n = num;
end
num = 15;
CA = zeros(1,n)';

fbIndices1 = [8 15];
fbIndices2 = [15, 14, 13, 12, 11, 9];

SV = [
        [2 6]; % 1
        [3 7]; % 2
        [4 8]; % 3
        [5 9]; % 4
      ];

G1 = ones(1,num);
G2 = ones(1,num);

for i = 1:n

    CA(i) = mod(sum([G1(num), G2(SV(PRN,:))]), 2);

    fb1 = mod(sum(G1(fbIndices1)), 2);
    fb2 = mod(sum(G2(fbIndices2)), 2);


    G1 = [fb1 G1(1:num-1)];
    G2 = [fb2 G2(1:num-1)];
end

CA = CA * 2 - 1;
end
```

code/PRNCode.m.

```
function distances = CalculateDistances(target, base)
    distances = zeros(16,1);

    for i = 1:4
        for j = 1:4
            distances((i-1)*4+j) = norm(target(:,i) - base(:,j));
        end
    end
end
```

code/CalculateDistances.m.

```
function estPoints = EstimateTargetPoints(m, n, offset)
    estRot = GenerateRotationMatrix([m(4) m(5) m(6) m(7)],'q');
    estPoints = repmat([m(1),m(2),m(3)]',1,n) + estRot * offset;
end
```

code/EstimateTargetPoints.m.

```
function A = GenerateRotationMatrix(m, type)
if (type == 'q')
    A = [[m(1)^2+m(2)^2-m(3)^2-m(4)^2, 2*m(1)*m(4)+2*m(2)*m(3), -2*m(1)*m(3)+2*m(2)*m(4)]; ,
        ...
        [-2*m(1)*m(4)+2*m(2)*m(3), m(1)^2-m(2)^2+m(3)^2-m(4)^2, 2*m(1)*m(2)+2*m(3)*m(4)]; ,
            ...
        [2*m(1)*m(3)+2*m(2)*m(4), -2*m(1)*m(2)+2*m(3)*m(4), m(1)^2-m(2)^2-m(3)^2+m(4)^2]];
else if (type == 'e')
    A = [[cos(m(3))*cos(m(1))-sin(m(3))*cos(m(2))*sin(m(1)), cos(m(3))*sin(m(1))+sin(m(3))*
        cos(m(2))*cos(m(1)), sin(m(3))*sin(m(2))]; ...
        [-sin(m(3))*cos(m(1))-cos(m(3))*cos(m(2))*sin(m(1)), -sin(m(3))*sin(m(1))+cos(m(3))*
            cos(m(2))*cos(m(1)), cos(m(3))*sin(m(2))]; ...
        [sin(m(2))*sin(m(1)), -sin(m(2))*cos(m(1)), cos(m(2))]];
else
    A = eye(3);
end
end
```

code/GenerateRotationMatrix.m.

```
function a = QuatConj(a)
    a(2:end) = -a(2:end);
end
```

code/QuatConj.m.

D.3 HDL

```
library ieee;

use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity top_level is
port(
  -- FX2LP interface
      ---------------------------------------------------------------------------
  fx2Clk_in      : in    std_logic;                    -- 48MHz clock from FX2LP
  fx2Addr_out    : out   std_logic_vector(1 downto 0); -- select FIFO: "00" for EP2OUT, "10"
      for EP6IN
  fx2Data_io     : inout std_logic_vector(7 downto 0); -- 8-bit data to/from FX2LP

  -- When EP2OUT selected:
  fx2Read_out    : out   std_logic;                    -- asserted (active-low) when reading
      from FX2LP
  fx2OE_out      : out   std_logic;                    -- asserted (active-low) to tell FX2LP
       to drive bus
  fx2GotData_in  : in    std_logic;                    -- asserted (active-high) when FX2LP
      has data for us

  -- When EP6IN selected:
  fx2Write_out   : out   std_logic;                    -- asserted (active-low) when writing
      to FX2LP
  fx2GotRoom_in  : in    std_logic;                    -- asserted (active-high) when FX2LP
      has room for more data from us
  fx2PktEnd_out  : out   std_logic;                    -- asserted (active-low) when a host
      read needs to be committed early

  -- Onboard peripherals
      ---------------------------------------------------------------------------
  sseg_out       : out   std_logic_vector(7 downto 0); -- seven-segment display cathodes (one
       for each segment)
  anode_out      : out   std_logic_vector(3 downto 0); -- seven-segment display anodes (one
      for each digit)
  led_out        : out   std_logic_vector(7 downto 0); -- eight LEDs
  sw_in          : in    std_logic_vector(7 downto 0); -- eight switches
  clk       : in    std_logic;

  -- External peripherals
      ---------------------------------------------------------------------------
  RX_DATA     : in    std_logic_vector(7 downto 0);
  TX_DATA     : out   std_logic_vector(7 downto 0);

  CLOCK_ADC   : in    std_logic;
  CLOCK_ADC_OUT : out   std_logic_vector(2 downto 0);

  CLOCK_DAC   : in    std_logic;
  CLOCK_DAC_OUT : out   std_logic_vector(2 downto 0);

  CONFIG_SDA    : inout std_logic;
  CONFIG_SCL    : inout std_logic;
  CONFIG_DIN    : out   std_logic;
  CONFIG_SCLK   : out   std_logic;
  CONFIG_CS   : out   std_logic;
  CONFIG_SHDN   : out   std_logic;
  CONFIG_RXTX   : out   std_logic;
  CONFIG_CW   : out   std_logic

);
end entity;

architecture structural of top_level is
```

```
53  -- Channel read/write interface
        -----------------------------------------------------------------
54  signal fx2Data   : std_logic_vector(7 downto 0);  -- data lines used when the host reads
        from a channel
55  signal fx2Valid  : std_logic;                     -- channel logic can drive this low to
        say "I don't have data ready for you"
56  signal fx2Ready  : std_logic;                     -- '1' means "on the next clock rising
        edge, put your next byte of data on f2hData"
57  --
        ------------------------------------------------------------------------------------------


58
59  -- Needed so that the fx2_interface module can drive both fx2Read_out and fx2OE_out
60  signal fx2Read   : std_logic;
61
62  -- Reset signal so host can delay startup
63  signal fx2Reset  : std_logic;
64
65  alias dac_clk   :   std_logic is CLOCK_DAC;
66  alias adc_clk   :   std_logic is CLOCK_ADC;
67
68  signal reset : std_logic := '0';
69
70  signal configClock : std_logic := '0';
71  signal configDone  : std_logic := '1'; -- Active Low
72 begin
73  CLOCK_ADC_OUT <= CLOCK_ADC & CLOCK_ADC & CLOCK_ADC;
74  CLOCK_DAC_OUT <= CLOCK_DAC & CLOCK_DAC & CLOCK_DAC;
75
76  -- CommFPGA module
77  fx2Read_out <= fx2Read;
78  fx2OE_out <= fx2Read;
79  fx2Addr_out(0) <= '0' when fx2Reset = '0' else 'Z'; -- So fx2Addr_out(1)='0' selects EP2OUT
        , fx2Addr_out(1)='1' selects EP6IN
80  reset <= '0';
81  fx2_interface : entity work.fx2_interface
82    port map(
83      clk_in         => fx2Clk_in,
84      reset_in       => '0',
85      reset_out      => fx2Reset,
86
87      -- FX2LP interface
88      fx2FifoSel_out => fx2Addr_out(1),
89      fx2Data_io     => fx2Data_io,
90      fx2Read_out    => fx2Read,
91      fx2GotData_in  => fx2GotData_in,
92      fx2Write_out   => fx2Write_out,
93      fx2GotRoom_in  => fx2GotRoom_in,
94      fx2PktEnd_out  => fx2PktEnd_out,
95
96      -- DVR interface -> Connects to application module
97      chanAddr_out   => open,
98      h2fData_out    => open,
99      h2fValid_out   => open,
100     h2fReady_in    => '0',
101     f2hData_in     => fx2Data,
102     f2hValid_in    => fx2Valid,
103     f2hReady_out   => fx2Ready
104   );
105
106 -- Transceiver link module
107 sherpa_rxtx : entity work.sherpa_rxtx
108   port map(
109     adc_clk_in   => adc_clk,
110     dac_clk_in   => dac_clk,
111     usb_clk_in   => fx2Clk_in,
```

```
      reset_in     => '0',

      -- DVR interface -> Connects to fx2_interface module
      data_out  => fx2Data,
      valid_out => fx2Valid,
      ready_in  => fx2Ready,

      -- External interface
      sseg_out     => sseg_out,
      anode_out    => anode_out,
      led_out      => led_out,
      sw_in      => sw_in,

      RX_in      => RX_DATA,
      TX_out     => TX_DATA,
      dacCW_out  => CONFIG_CW,
      RXTX_out     => CONFIG_RXTX,

      sync_clk_out => open --CLOCK_DAC_OUT1 --CLOCK_SYNC
    );

  clock_divider : process (clk) begin
      if (rising_edge(clk)) then
        configClock <= not configClock;
      end if;
    end process;

  -- Initialize the SI5351C Clock Generator Chip
  si5351c_handler : entity work.si5351c_handler
    port map (
      clock_in  => configClock,
      reset_in  => '0',
      done_out    => configDone,
      sda_inout   => CONFIG_SDA,
      scl_inout => CONFIG_SCL
    );

  -- Initialize the MAX2830 analog front end
  max2830_handler : entity work.max2830_handler
    port map (
      clock_in => configClock,
      reset_in => configDone,
      din_out =>  CONFIG_DIN,
      sclk_out => CONFIG_SCLK,
      cs_out   => CONFIG_CS,
      sw_in   => sw_in
    );

  CONFIG_SHDN <= '1';
end architecture;
```

code/top_level.vhd.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;

entity max2830_handler is
  port (
    clock_in : in std_logic;
      reset_in : in std_logic;
    sw_in   : in std_logic_vector(7 downto 0);
    din_out  : out std_logic;
    sclk_out : out std_logic;
    cs_out   : out std_logic
  );
```

```
14 end max2830_handler;
15
16 architecture Behavioral of max2830_handler is
17   type state_type is (idle,fetching,sending,finishing,updating);
18   type spi_type   is (starting, ending);
19   signal state    : state_type := fetching;
20   signal spi_state : spi_type   := starting;
21
22   signal max2830_config_addra : std_logic_vector(3 downto 0)  := "0000";
23   signal max2830_config_data  : std_logic_vector(17 downto 0);
24
25   signal spi_ena     :  std_logic  :=  '0';
26   signal spi_valid  : std_logic;
27
28   signal delay_counter : integer range 0 to 101  := 0;
29
30   signal swState_current  : std_logic_vector(7 downto 0) := "00111111";
31   signal swState_old      : std_logic_vector(7 downto 0) := "00111111";
32
33   alias clk  : std_logic is clock_in;
34   alias rst  : std_logic is reset_in;
35   alias din  : std_logic is din_out;
36   alias sclk : std_logic is sclk_out;
37   alias cs   : std_logic is cs_out;
38 begin
39
40     spi_interface : entity work.spi_master
41     generic map ( N => 18 )
42     port map (
43        sclk_i     =>   clk,
44        pclk_i     =>   clk,
45        rst_i    =>  rst,
46        spi_ssel_o =>   cs,
47        spi_sck_o  =>   sclk,
48        spi_mosi_o =>   din,
49        spi_miso_i =>  open,
50        di_req_o   =>  open,
51        di_i     =>  max2830_config_data,
52        wren_i   =>   spi_ena,
53        wr_ack_o =>   open,
54        do_valid_o =>   spi_valid,
55        do_o     =>   open
56     );
57
58   state_machine : process(clk, rst, max2830_config_addra)
59     constant delay_threshold  : integer := 100;
60     type max2839_config_data is array (15 downto 0) of std_logic_vector(17 downto 0);
61     variable max2839_config : max2839_config_data :=
62       ("010" & "1" & "1101000000"              & "0000",
63       "0" & "1" & "000110011010"             & "0001",
64       "01000000000011"                      & "0010",
65       "10011" & "001111010"                    & "0011",
66       "01100110011001"                       & "0100",
67       "0000" & "0" & "010" & "0" & "00" & "1" & "00"  & "0101",
68       "0" & "00" & "0000" & "0" & "1000" & "0" & "0"  & "0110",
69       "01" & "000000" & "010" & "010"            & "0111",
70       "1" & "1" & "0" & "0" & "00" & "001000" & "00"  & "1000",
71       "000" & "1" & "1110110101"             & "1001",
72       "0111" & "011" & "0100" & "100"            & "1010",
73       "0000000" & "00" & "00000"             & "1011",
74       "00000101" & "000000"                  & "1100",
75       "0011" & "1010" & "010010"             & "1101",
76       "000" & "0" & "0" & "10" & "0111011"       & "1110",
77       "00" & "01" & "0101000101"             & "1111");
78   begin
79     if (rst = '1') then
```

```
80        max2830_config_addra <= "0000";
81        delay_counter     <= 0;
82        state           <= fetching;
83      elsif (rising_edge(clk)) then
84        case state is
85          when fetching =>
86            delay_counter <= delay_counter + 1;
87            spi_ena <= '0';
88
89            if (delay_counter = delay_threshold) then
90              max2830_config_data  <= max2839_config(to_integer(15-unsigned(
                    max2830_config_addra)));
91            elsif (delay_counter = delay_threshold + 1) then
92              delay_counter <= 0;
93              state <= sending;
94              spi_state <= starting;
95            end if;
96
97          when sending =>
98            case spi_state is
99              when starting =>
100               spi_ena <= '1';
101               spi_state <= ending;
102             when ending =>
103               spi_ena <= '0';
104               if (spi_valid = '1') then
105                 if (to_integer(unsigned(max2830_config_addra)) = 15) then
106                   state <= finishing;
107                 else
108                   state <= fetching;
109                   max2830_config_addra <= std_logic_vector(unsigned(max2830_config_addra) +
                          1);
110                 end if;
111               end if;
112             when others => null;
113           end case;
114
115         when finishing =>
116           delay_counter <= delay_counter + 1;
117           spi_ena <= '0';
118
119           if (delay_counter = delay_threshold) then
120             delay_counter <= 0;
121             state <= idle;
122           end if;
123
124         when idle =>
125           spi_ena <= '0';
126           swState_current <= sw_in;
127           swState_old <= swState_current;
128
129           if (swState_current /= swState_old) then
130             state <= updating;
131             spi_state <= starting;
132
133             if (swState_current(7) = '1') then
134               max2830_config_data <= "00000101" & swState_current(5 downto 0) & "1100";
135             else
136               max2830_config_data <= "0000000"  & swState_current(6 downto 0) & "1011";
137             end if;
138           end if;
139
140         when updating =>
141           case spi_state is
142             when starting =>
143               spi_ena <= '1';
```

```
144               spi_state <= ending;
145             when ending =>
146               spi_ena <= '0';
147               if (spi_valid = '1') then
148                 state <= idle;
149               end if;
150           end case;
151         when others => null;
152       end case;
153     end if;
154   end process;
155 end Behavioral;
```

code/max2830_handler.vhd.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use ieee.numeric_std.all;
4
5  entity si5351c_handler is
6    port (
7      clock_in  : in std_logic;
8        reset_in  : in std_logic;
9      done_out  : out std_logic;
10     sda_inout : inout std_logic;
11     scl_inout : inout std_logic
12   );
13 end si5351c_handler;
14
15 architecture Behavioral of si5351c_handler is
16   type state_type is (idle,fetching,sending);
17   signal state : state_type := sending;
18
19   signal i2c_busy   : std_logic             :=  '0';
20   signal i2c_ena      : std_logic             := '0';
21   signal i2c_addr   : std_logic_vector(6 downto 0)  := "0000000";
22   signal i2c_rw     : std_logic             := '0';
23   signal i2c_data_wr  : std_logic_vector(7 downto 0)  := "00000000";
24   signal i2c_data_rd  : std_logic_vector(7 downto 0)  := "00000000";
25   signal busy_prev    : std_logic             := '0';
26
27   signal si5351c_config_addra : std_logic_vector(6 downto 0);
28   signal si5351c_config_data  : std_logic_vector(15 downto 0);
29
30   signal rst_n  : std_logic;
31
32   alias clk : std_logic is clock_in;
33   alias rst : std_logic is reset_in;
34   alias sda : std_logic is sda_inout;
35   alias scl : std_logic is scl_inout;
36 begin
37   rst_n <= not rst;
38   done_out <= '0' when state = idle else '1';
39
40   si5351c_config_rom : entity work.si5351c_config_rom
41     port map (
42       clka      => clk,
43       addra     => si5351c_config_addra,
44       douta     => si5351c_config_data
45     );
46
47   i2c_interface : entity work.i2c_master
48     port map (
49        clk       => clk,
50        reset_n   => rst_n,
51        ena       =>  i2c_ena,
```

```
52        addr      => i2c_addr,
53        rw        => i2c_rw,
54        data_wr   => i2c_data_wr,
55        busy      => i2c_busy,
56        data_rd   => i2c_data_rd,
57        ack_error => open,
58        sda       => sda,
59        scl       => scl
60      );
61
62    state_machine : process(clk, rst, i2c_busy, si5351c_config_addra)
63      variable busy_counter     : integer range 0 to 2    := 0;
64      variable delay_counter    : integer range 0 to 1001 := 0;
65      constant delay_threshold  : integer             := 1000;
66    begin
67      if (rst = '1') then
68        si5351c_config_addra <= "0000000";
69        busy_counter      := 0;
70        delay_counter     := 0;
71        state           <= sending;
72      elsif (rising_edge(clk)) then
73        case state is
74          when fetching =>
75            delay_counter := delay_counter + 1;
76
77            if (delay_counter = delay_threshold) then
78              si5351c_config_addra <= std_logic_vector(unsigned(si5351c_config_addra) + 1);
79            elsif (delay_counter = delay_threshold + 1) then
80              delay_counter := 0;
81              state <= sending;
82            end if;
83          when sending =>
84            busy_prev <= i2c_busy;
85            if(busy_prev = '0' and i2c_busy = '1') then
86              busy_counter := busy_counter + 1;
87            end if;
88            case busy_counter is
89              when 0 =>
90                i2c_ena <= '1';
91                i2c_addr <= "1100000";
92                i2c_rw <= '0';
93                i2c_data_wr <= si5351c_config_data(15 downto 8);
94              when  1 =>
95                i2c_data_wr <= si5351c_config_data(7 downto 0);
96              when 2 =>
97                i2c_ena <= '0';
98                if(i2c_busy = '0') then
99                 busy_counter := 0;
100
101                if (to_integer(unsigned(si5351c_config_addra)) = 69) then
102                 state <= idle;
103                else
104                 state <= fetching;
105                end if;
106               end if;
107             when others => null;
108           end case;
109         when idle => null;
110         when others => null;
111       end case;
112     end if;
113   end process;
114 end Behavioral;
```

code/si5351c_handler.vhd.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity sherpa_rxtx is
  port(
    reset_in     : in  std_logic;

    -- Clock interface
        -----------------------------------------------------------------------------
    usb_clk_in   : in   std_logic;
    adc_clk_in   : in   std_logic;
    dac_clk_in   : in   std_logic;
    sync_clk_out : out  std_logic;

    -- DVR interface
        -----------------------------------------------------------------------------
    data_out  : out std_logic_vector(7 downto 0);  -- data lines used when the host reads
        from a channel
    valid_out : out std_logic;                     -- channel logic can drive this low to say
         "I don't have data ready for you"
    ready_in  : in  std_logic;                     -- '1' means "on the next clock rising
        edge, put your next byte of data on f2hData"

    -- Peripheral interface
        -----------------------------------------------------------------------
    sseg_out       : out   std_logic_vector(7 downto 0); -- seven-segment display cathodes (
        one for each segment)
    anode_out      : out   std_logic_vector(3 downto 0); -- seven-segment display anodes (one
         for each digit)
    led_out        : out   std_logic_vector(7 downto 0); -- eight LEDs
    sw_in         : in    std_logic_vector(7 downto 0);  -- eight switches

    RX_in       : in    std_logic_vector(7 downto 0);
    TX_out      : out   std_logic_vector(7 downto 0) := "00000000";
    dacCW_out   : out   std_logic;
    RXTX_out      : out   std_logic
  );
end entity;

architecture rtl of sherpa_rxtx is
  alias rst : std_logic is reset_in;

  -- Flags for display on the 7-seg decimal points
  signal flags               : std_logic_vector(3 downto 0);

  -- USB FIFO
  signal fifoCount           : std_logic_vector(14 downto 0);
  signal fifoInputData       : std_logic_vector(7 downto 0);  -- producer: data
  signal fifoInputValid      : std_logic;                     --           valid flag
  signal fifoInputReady      : std_logic;                     --           ready flag
  signal fifoOutputData      : std_logic_vector(7 downto 0);  -- consumer: data
  signal fifoOutputValid     : std_logic;                     --           valid flag
  signal fifoOutputReady     : std_logic;                     --           ready flag

  -- Control states
  type   activityState_type is (initialize, idle, active);
  type   functionState_type is (transmit, receive);

  signal activityState : activityState_type := initialize;
  signal functionState : functionState_type := receive;

  -- Data transfer
  signal transmitDataReady  : std_logic;
  signal receiveDataReady   : std_logic;
```

```
58    signal sync_clk         : std_logic;
59
60    signal dataCount  :  std_logic_vector(7 downto 0)  := (others => '0');
61    signal dataVector :   std_logic_vector(98 downto 0) := (others => '1');
62    signal dataBit    : std_logic := '0';
63
64    -- Data compression
65    signal shiftValue : std_logic_vector(7 downto 0)  := (others => '0');
66    signal shiftValid : std_logic := '0';
67
68    signal adc_data       : std_logic;
69    signal syncFlag     : std_logic := '0';
70
71    signal counter  : std_logic_vector(15 downto 0) := (others => '0');
72
73    signal dacControl : std_logic_vector(7 downto 0) := "01101000";
74    signal prnSignal  : std_logic_vector(7 downto 0) := (others => '0');
75
76    signal SSCount    :  std_logic_vector(15 downto 0) := (others => '0');
77  begin
78
79    ---------------------------------------
80    --- Generate synchronization clock ---
81    ---------------------------------------
82    clock_syncer : entity work.clock_syncer
83    port map (
84      dac_clk_in  => dac_clk_in,
85      adc_clk_in  => adc_clk_in,
86      reset_in    => rst,
87      sync_out  => sync_clk
88    );
89    sync_clk_out <= sync_clk;
90
91    ------------------------------------------------------
92    --- Toggle between RX and TX at specified intervals ---
93    ------------------------------------------------------
94    transmitDataReady <= '1' when activityState = active and functionState = transmit else '0';
95    receiveDataReady  <= '1' when activityState = active and functionState = receive  else '0';
96    dataBit <= dataVector(to_integer(unsigned(dataCount)));
97    TX_out <= dacControl when activityState = initialize else prnSignal;
98    RXTX_out <= '1' when activityState = active and functionState = transmit else '0';
99
100   functionState <= transmit when sw_in(7) = '1' else receive;
101
102   state_machine : process(rst, sync_clk)
103   begin
104     if (rst = '1') then
105       dataCount   <= (others => '0');
106       activityState <= initialize;
107 --      functionState  <= initialize;
108       syncFlag    <= '0';
109     elsif ( rising_edge(sync_clk) ) then
110       case activityState is
111         when initialize =>
112           if (to_integer(unsigned(dataCount)) = 2) then
113             dataCount <= (others => '0');
114             activityState <= idle;
115             --functionState <= transmit;
116             dacCW_out <= '1';
117           else
118             dacCW_out <= '0';
119             dataCount <= std_logic_vector(unsigned(dataCount) + 1);
120           end if;
121         when active =>
122           syncFlag <= not syncFlag;
123           if (to_integer(unsigned(dataCount)) >= 98) then
```

```
124             dataCount <= (others => '0');
125             --activityState <= idle;
126           else
127             dataCount <= std_logic_vector(unsigned(dataCount) + 1);
128           end if;
129         when idle =>
130           if (functionState = transmit) then
131             --functionState <= receive;
132           else
133             --functionState <= transmit;
134           end if;
135           activityState <= active;
136       end case;
137     end if;
138   end process;
139
140
141   -------------------------
142   --- Generate PRN code ---
143   -------------------------
144   prn_code_generator : entity work.prn_code_generator
145   generic map (
146     idx1  =>  1,
147     idx2  =>  5
148   ) port map (
149     enable_in =>  transmitDataReady,
150     data_in =>  dataBit,
151     clock_in  => dac_clk_in,
152     I_out     => prnSignal,
153     Q_out     => open
154   );
155
156   -------------------------
157   --- Write to USB FIFO ---
158   -------------------------
159   fifoInputValid  <= '1' when receiveDataReady = '1' else '0';
160   fifoOutputReady   <= '1' when ready_in = '1' else '0';
161   valid_out <= '0' when fifoOutputValid = '0' else '1';
162   data_out <= fifoOutputData;
163   read_fifo : entity work.fifo_wrapper
164     port map(
165       wr_clk_in       => adc_clk_in,
166       rd_clk_in        => usb_clk_in,
167       depth_out       => fifoCount,
168
169       -- Production end
170       inputData_in    => RX_in, --fifoInputData,
171       inputValid_in   => fifoInputValid,
172       inputReady_out  => fifoInputReady,
173
174       -- Consumption end
175       outputData_out  => fifoOutputData,
176       outputValid_out => fifoOutputValid,
177       outputReady_in  => fifoOutputReady
178     );
179
180   -------------------------
181   --- Handle LED Display ---
182   -------------------------
183   SSCount(6 downto 0) <= sw_in(6 downto 0);
184   SSCount(8) <= '1' when functionState = transmit else '0';
185   flags <= '0' & sync_clk & '0' & sync_clk;
186   seven_seg : entity work.seven_seg
187     port map(
188       clk_in     => adc_clk_in,
189       data_in    => SSCount,
```

```
      dots_in    => flags,
      segs_out   => sseg_out,
      anodes_out => anode_out
    );
end architecture;
```

code/sherpa_rxtx.vhd.

```
library ieee ;
    use ieee.std_logic_1164.all ;
   use ieee.numeric_std.all;

entity prn_code_generator is
  generic (
   idx1     : natural    := 1;
   idx2     : natural    := 5
  ) ;
  port (
   enable_in  : in  std_logic ;
   data_in    : in  std_logic ;
    clock_in  :  in  std_logic ;
    I_out   :  out std_logic_vector(7 downto 0) ;
   Q_out    :  out std_logic_vector(7 downto 0) := "00000000"
  ) ;
end entity ;

architecture arch of prn_code_generator is
    alias ena : std_logic is enable_in;
    alias clk : std_logic is clock_in;
begin
   ----------------------------
   -- PRN Generation Process --
   ----------------------------
    prncode : process( clk, ena )
    variable G1      : bit_vector(14 downto 0) := "111111111111111";
    variable G2      : bit_vector(14 downto 0) := "111111111111111";
    variable data_q   : bit := '0';
    variable G1_q      : bit := '0';
    variable G2_q      : bit := '0';
    variable counter  : natural range 0 to 32767 := 0 ;
   begin
        if( ena = '0' ) then
            I_out   <= "10000000" ; -- 0
        Q_out   <= "10000000" ; -- 0
            G1      := "111111111111111";
            G2      := "111111111111111";
        data_q  := '0';
        G1_q    := '0';
        G2_q    := '0';
        counter   :=  0;
        elsif( rising_edge( clk ) ) then
        -------------------------
        -- PRN Code Generation --
        -------------------------
        data_q := (G1(14) xor G2(idx1) xor G2(idx2) xor to_bit(data_in));

        if ( data_q = '1' ) then
          I_out <= "11111111"; --  127
        else
          I_out <= "00000000"; -- -128
        end if;

--        G1_q := G1(7) xor G1(4) xor G1(6) xor G1(14);
--        G2_q := G2(1) xor G2(2) xor G2(3) xor G2(5) xor G2(7) xor G2(9) xor G2(10) xor G2
    (11) xor G2(13) xor G2(14);
        G1_q := G1(7) xor G1(14);
```

```
58          G2_q := G2(8) xor G2(10) xor G2(11) xor G2(12) xor G2(13) xor G2(14);
59
60          G1(14 downto 1) := G1(13 downto 0);-- sll 1;
61          G2(14 downto 1) := G2(13 downto 0);-- sll 1;
62
63          G1(0) := G1_q;
64          G2(0) := G2_q;
65
66          end if ;
67      end process ;
68  end architecture;
```

code/prn_code_generator.vhd.

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use ieee.numeric_std.all;
4
5   entity clock_syncer is
6       Port ( dac_clk_in    : in  STD_LOGIC;
7              adc_clk_in    : in  STD_LOGIC;
8              reset_in    : in  STD_LOGIC;
9              sync_out      : out STD_LOGIC);
10  end clock_syncer;
11
12  architecture Behavioral of clock_syncer is
13    signal sync    : std_logic := '0';
14    signal init    : std_logic := '0';
15    signal counter : std_logic_vector(12 downto 0) := (others => '1');
16
17    alias rst : std_logic is reset_in;
18  begin
19    sync_out <= sync;
20
21    -- This solution is clearly NOT ideal, but it does work.
22    syncer : process (rst, dac_clk_in, adc_clk_in)
23    begin
24      if ( rst = '1' ) then
25        sync <= '0';
26        init <= '0';
27        counter <= (others => '1');
28      elsif (rising_edge(adc_clk_in)) then
29        if ( dac_clk_in = '1' ) then
30          sync <= '1';
31          init <= '1';
32          counter <= (others => '0');
33        elsif (to_integer(unsigned(counter)) < 8000) then
34          sync <= '1';
35          if (init = '1') then
36            counter <= std_logic_vector(unsigned(counter) + 1);
37          end if;
38        else
39          sync <= '0';
40        end if;
41      end if;
42    end process;
43  end Behavioral;
```

code/clock_syncer.vhd.

Bibliography

[1] *2.4GHz to 2.5GHz 802.11g/b RF Transceiver, PA, and Rx/Tx/Antenna Diversity Switch*. MAX2830. Rev. 2. Maxim Integrated. March 2011 (cited on pages 36, 47, 51, 64).

[2] Barth Netterfield et al. *Kst - Visualize your data*. `https://kst-plot.kde.org/`. Accessed: 2017-06-14. 2017 (cited on page 59).

[3] *An overview of oscillator jitter*. Technical Note 35. Rev. B. Statek Corporation. April 2007 (cited on page 25).

[4] Richard. Aster, Brian. Borchers, and Clifford H. Thurber. *Parameter estimation and inverse problems*. eng. Volume 90. Elsevier, 2005. ISBN: 0120656043 (cited on pages 16, 17).

[5] Michael Bailey. *General Layout Guidelines for RF and Mixed-Signal PCBs*. `https://www.maximintegrated.com/en/app-notes/index.mvp/id/5100`. Accessed: 2017-03-06. 2016 (cited on page 33).

[6] Mathias Benn. "Vision Based Navigation Sensors for Spacecraft Rendezvous and Docking". eng. 2011 (cited on page 10).

[7] Kai Borre et al. *A Software-Defined GPS and Galileo Receiver*. eng. Birkhäuser Verlag GmbH, 2007. ISBN: 9780817643904 (cited on pages 9, 10, 18, 20, 24, 26, 29, 30, 48, 53).

[8] D.K. Cheng. *Field and Wave Electromagnetics*. Pearson new international edition. Pearson, 2014. ISBN: 9781292026565 (cited on page 31).

[9] Lukas Christensen. *Synthesis in Earth and Space Physics and Engineering*. Unpublished report. Technical University of Denmark, January 20, 2017 (cited on pages 9, 12, 40, 83, 84).

[10] IPC Association Connecting Electronics Industries. Controlled Impedance Task Group (D-21c). *Design Guide for High-speed Controlled Impedance Circuit Boards*. IPC standard. IPC, 2004. ISBN: 9781580982443 (cited on pages 31, 32, 86).

[11] Ju-Young Du et al. "Relative Navigation for Formation Flying of Spacecraft". und. In: (2001) (cited on page 10).

[12] *Dual, 8-Bit, 40Msps, 3V, Low-Power ADC with Internal Reference and Parallel Outputs*. MAX1195. Rev. 0. Maxim Integrated. April 2002 (cited on pages 51, 64).

[13] *Dual, 8-Bit, 80 Msps, Current-Output DAC*. MAX5851. Rev. 0. Maxim Integrated. April 2004 (cited on pages 48, 51).

[14] Fairwaves. *UmTRX v2.2*. http://shop.fairwaves.co/accessories-and-diy/UmTRX-transceiver-for-OpenBTS-and-Osmocom-OpenBSC. Accessed: 2017-03-15. 2017 (cited on page 37).

[15] P. Fortescue, G. Swinerd, and J. Stark. *Spacecraft Systems Engineering*. Wiley, 2011. ISBN: 9781119978367 (cited on pages 10, 15, 27–29, 42, 77).

[16] Great Scott Gadgets. *HackRF One*. https://greatscottgadgets.com/hackrf/. Accessed: 2017-03-15. 2016 (cited on page 37).

[17] David Money Harris. *E11 Lecture 7: Gold Codes*. eng. Accessed: 2017-06-07. 2014. URL: %5Curl%7B%22http://pages.hmc.edu/harris/class/e11/lect7.pdf%22%7D (cited on pages 17, 19).

[18] *I²C-Programmable any-frequency CMOS clock generator + VCXO*. Si5351C. Rev. 1. Silicon Labs. April 2015 (cited on pages 49, 68).

[19] New Wave Instruments. "Linear Feedback Shift Registers". eng. In: (2005). Accessed: 2017-06-08. URL: %5Curl%7B%22http://www.newwaveinstruments.com/resources/articles/m_sequence_linear_feedback_shift_register_lfsr.htm%22%7D (cited on pages 18, 19).

[20] *Low Power IEEE 802.15.4/Proprietary GFSK/FSK Zero-IF 2.4 GHz Transceiver IC*. ADF7242. Rev. 0. Analog Devices. February 2017 (cited on page 35).

[21] Chris McClelland. *Makestuff GitHub Page*. https://github.com/makestuff. Accessed: 2017-06-13. 2017 (cited on pages 55, 93).

[22] Lime Microsystems. *LimeSDR: Flexible, Next-generation, Open Source Software Defined Radio*. https://www.crowdsupply.com/lime-micro/limesdr. Accessed: 2017-03-15. 2016 (cited on page 37).

[23] *Multi-band Multi-standard Transceiver with Integrated Dual DACs and ADCs*. LMS6002D. Rev. 1.1.0. Lime Microsystems. December 2012 (cited on page 36).

[24] Shree K. Nayar and Mohit Gupta. "Diffuse structured light". eng. In: *2012 Ieee International Conference on Computational Photography, Iccp 2012* (2012), page 6215216. DOI: 10.1109/ICCPhot.2012.6215216 (cited on pages 10, 11).

[25] Nuand. *bladeRF*. https://www.nuand.com/. Accessed: 2017-03-15. 2016 (cited on page 37).

[26] Giovanni B. Palmerini. "Relative Navigation in Autonomous Spacecraft Formations". eng. In: *Ieee Aerospace Conference* 2016- (2016), page 7500944. ISSN: 1095323x. DOI: 10.1109/AERO.2016.7500944 (cited on page 10).

[27] *Precise SMD Temperature Compensated Crystal Oscillators*. 7N-25.000MBP-T. TXC Corporation (cited on page 49).

[28] Ettus Research. *USRP B200*. https://www.ettus.com/product/details/UB200-KIT. Accessed: 2017-03-15. 2016 (cited on page 37).

[29] *RF Agile Transceiver*. AD9361. Rev. F. Analog Devices. November 2016 (cited on page 36).

[30] *Single chip 2.4 GHz Transceiver*. nRF2401. Rev. 1.1. Nordic Semiconductor. June 2004 (cited on page 35).

[31] Eric W. Weisstein. *Euler Parameters. From MathWorld–A Wolfram Web Resource*. Accessed: 2017-06-07. 2017. URL: %5Curl%7Bhttp://mathworld.wolfram.com/EulerParameters.html%7D (cited on page 15).

[32] *Wireless External Antenna for 2.4 GHz Applications*. W1030. Rev. B. Pulse Electronics. February 2014 (cited on pages 64, 92).

{2.7182818284}ϑφερτυθιοπσδφγηξκλ

DTU Space
National Space Institute
Technical University of Denmark

Elektrovej, building 327
DK-2800 Kgs. Lyngby
Tlf. (+45) 4525 9500
Fax (+45) 4525 9575

www.space.dtu.dk