Contents

List of Tables

List of Figures

Chapter 1

Introduction

In telecommunications, a communication system is a system consisting of a transmitter and a receiver enabling the possibility of transmitting information from a transmitter to a receiver. The signal is passed through some form of radio communications channel, altering the transmitted signal in one or multiple ways. These alterations can be more or less predictable by the receiver depending on the cause.

For a generic, wireless communication system, the radio communications channels introduces both additive and multiplicative noise. Additive noise typically originates from the thermal and/or shot-noise introduced in the electrical components themselves, but can also come from other sources such as cosmic rays or interfering transmissions. Multiplicative noise originates from the physical environment where the radio system is located. Effects such as reflections, absorption, scattering, diffraction and refractions are all multiplicative noise depending on the physical environment.

Noise makes the received signal harder to detect. Most radio systems have a goal of transferring information from the transmitter to the receiver as accurately with as few errors as possible. To achieve this goal, the radio system should be designed to mitigate some of the noise. Additive noise is typically random and inherent in the radio system and cannot easily be dealt with. Multiplicative noise on the other hand, can often be suppressed to some extent by clever transmitter and receiver design. It is common to further split multiplicative processes into path loss, multipath fading and shadow fading. [1, 2] Where fading is the variation in the attenuation in received signal strength due to environmental factors. Path loss is the reduction of signal strength due to the physical distance between transmitter and receiver.

1.1 Channel emulation and DSP algorithms

When designing any radio system, it is common to start with simulations and modelling. However, simulations are never perfect. After a radio system prototype has been built, it is common practice to verify that simulations match the realized system performance. Both for simulated components, but also for simulated DSP algorithms, of which there can be many.

Many DSP algorithms are often used to compensate for the effects of bad channel fading. Examples of such algorithms include beamforming to increase SNR, optimal detection and channel equalization algorithms. The behaviour of such algorithms can often be deeply dependent on how the channel looks. Programmers almost always have to make some assumptions about the channel behaviour when writing the implementation of DSP algorithms. It must be very clear what will happen if those assumptions are broken. In the best case it can lead to reduced performance, in the worst case it can break the whole radio link. Also, radio hardware is often different from the hardware platform the algorithm was developed on, which can lead to unforeseen behaviour.

A radio channel for a hand-held device in a big city will look very different from the radio channel observed by an aeroplane or drone in a rural environment. It as also virtually impossible to test a radio system in all possible channels to ensure a robust system which always behaves as expected.

This is where channel emulation comes in. A channel emulator replaces the physical radio channel buy a emulated one allowing for real-time checking of a radio system. This becomes more important as a radio systems' complexity increases. An emulated channel can allow an engineer to easily check the realized radio system without going out in the field to perform tests. The main purposes of a good channel emulator is to enable easier debugging of the radio system, verify system performance, reduce development costs and time-to-market.

1.2 Thesis scope

This thesis covers the process documenting the main challenges when generating and implementing channel models suitable to find errors in phased array radios with up to 16 antennas. These radio channels should be implemented and realized in real-time on FPGAs if possible. More specifically, using four Xilinx Zynq-7000 SoC ZC702[3] development boards and eight Analog Devices AD9361[4] transceivers as radio front-ends. The implemented system should be scalable up to 64 antenna radios. The radio models should comply with the system specifications stated in the next section. The completed channel emulator must be able to cope with both very long, high delay spread and very short urban environment channels.

1.3 Specifications

Table 1.1: Specifications of radio channels that must be emulated.

Specification	Value
Carrier frequency, f_c	2.4 - 6.0 GHz
Channel bandwidth, B	20 MHz
Maximum Delay Spread, T_m	1.5 ms
Maximum distance between TX and RX	$>$ 200 km
Relative speed between TX and RX	0 - Mach 2

1.4 Commercial products

There exists several commercial channel emulators. Examples are PROPSIM from Keysight and PXIe-564xR from National Instruments. Both have a price on request, but are expected to be very expensive. The solution from National Instruments is unable to meet the specifications with regards to maximum delay spread and relative speed. [5]

Chapter 2

Theory

An introduction to how radio communication channels can be modelled and the challenges involved are presented in this chapter. The principles of phased array radios and how they can outperform single-antenna systems are shown. Some of the strengths and limitations of FPGA implementations are also explained.

2.1 The Radio channel

A generic communication system is shown in Figure 2.1. With N tx-antennas and M rx-antennas.

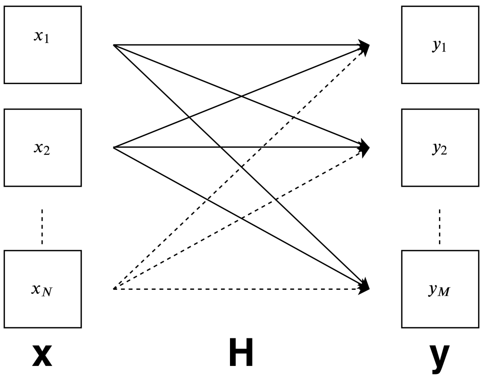


Figure 2.1: Generic communication system with N tx- and M rx-antennas.

The signals transmitted on each x_i element is passed through a radio channel h_{ij} before being received by a rx antenna element y_j.

In vector notation it can be written as Equation 2.1.

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n} \tag{2.1}$$

Where $\mathbf{n}$ is the $1 \times M$ additive noise vector. It is common to use Gaussian noise with noise power σ_n. The radio channel matrix $\mathbf{H}$ environment dependent multiplicative noise source which radio systems commonly tries to deal with. In other words it is the $\mathbf{H}$ that must be emulated by a channel emulator. How to find a suitable $\mathbf{H}$ is discussed in Chapter 2.4.

2.2 Phased array radios

One of the limiting factor in any communication system is the SNR. Signal to noise ratio can be improved by having a directional antenna. That is, a directional antenna with a radiation pattern stronger than a isotropic antenna in the direction of the received signal for a receiver. Alternatively in the direction of the receiving radio in the transmitter case. A single antennas radiation pattern is dependent on the antenna geometry. Directional antennas can also reduce interference from other transmitters using the same frequency band.

While modifying antenna diagrams works well for stationary systems, its use is limited in dynamic or moving systems. Motors can be used to physically move the antenna, but motors are slow, bulky and power-consuming. Often a better option is to use multiple antennas, or antenna elements. Each antenna element is multiplied by a complex weight. The antenna elements' weights effectively shifts their phase and/or modify their amplitude. All products are then summed. The summation of the weighted elements, together with their physical placement within the array, effectively generates a new antenna with a aggregated radiation diagram. This a called a phased array.

Such an array can be formulated as Equation (2.2). Where $\mathbf{x} = [x_1, x_2, \dots x_N]^T$ are the outputs of each phased array antenna element and $\mathbf{w} = [w_1, w_2, \dots w_N]^T$ their respective weights. y is the scalar output of the phased array.

$$y = \mathbf{x}^T\mathbf{w} \tag{2.2}$$

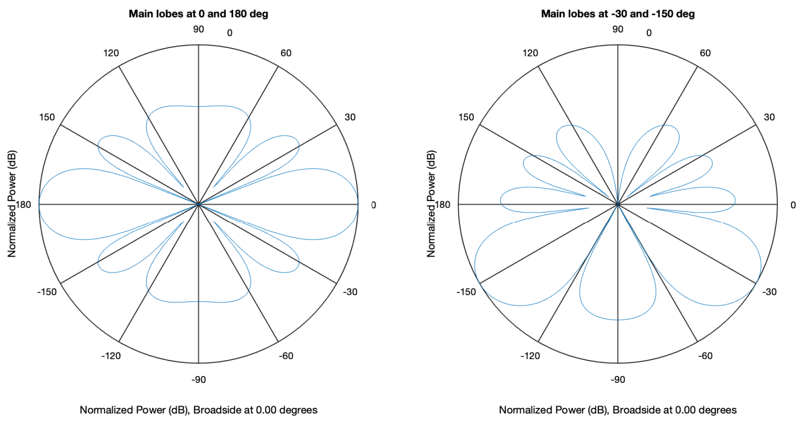


Figure 2.2: Example of active beamforming with 16 antennas placed on a line $\lambda/2$ apart.

A huge advantage of phased arrays over single-antenna systems is that different radiation patterns can be achieved by modifying $\mathbf{w}$. This can be done in real-time, while the system is active. An example of this is given in Figure 2.2. Here 16 antennas is put in a straight line along the vertical axis $\lambda/2$ apart. Where λ is the wavelength of the carrier frequency, f_c. By modifying $\mathbf{w}$ the main lobe is moved from $0°$ to $-30°$. Due to the physical placement of the antenna elements, the radiation pattern is mirrored around the horizontal axis. By setting correct weights directions of destructive interference can also be selected. This feature can be used to cancel out interference sources.

Finding optimal weights

There exists multiple algorithms for choosing w, depending on the goal of the system. They all have their separates strengths and weaknesses. Choosing the correct algorithm for the correct application can be essential to have a reliable radio link.

Assume that there exists multiple transmitters and that just the information from one of the is desired. In other words, the goal is to turn the main lobe in the direction of the desired transmitter and cancel out all other interferers. The received signal can be expressed as Equation (2.3).

$$\mathbf{x} = \sum_{i=0}^{N} s_i \mathbf{h}_i + \mathbf{n} \tag{2.3}$$

Where s_i is transmitted signal i and h_i is the ith column in the channel matrix $\mathbf{H}$. $\mathbf{n}$ is the Gaussian noise vector. By denoting the desired signal by d and undesired signals by u Equation (2.4) is obtained.

$$\mathbf{x} = s_d\mathbf{h}_d + \sum_{u=0}^{U} s_u\mathbf{h}_u + \mathbf{n} \tag{2.4}$$

Where U is of size $1 \times (N-1)$. The goal is to maximize the $s_d\mathbf{h}_d$ term and minimize the others such that the array output y only contains the desired signal. This is often not possible and a second best solution is to maximize the SINR, signal to interference noise ratio.

It can be shown that an optimal solution is a Wiener filter. The Wiener solution is

$$\mathbf{w}_{\text{opt}} = \mathbf{R}_{xx}^{-1}\mathbf{h}_d \tag{2.5}$$

Where $\mathbf{R}_{xx} = E\{\mathbf{x}\mathbf{x}^H\}$ is the channel correlation matrix and H denoted the complex conjugate transpose operation, also known as Hermitian. From Equation (2.4) it is clear that $\mathbf{R}_{xx}$ can be formulated as

$$\mathbf{R}_{xx} = \mathbf{h}_d\mathbf{h}_d^H + \sum_{u=0}^{U} \mathbf{h}_u\mathbf{h}_u^H + \sigma^2\mathbf{I} \tag{2.6}$$

When writing the Identity matrix as $\mathbf{I}$ and noise variance as σ^2. The Weiner solution assumes that all signals and propagation channels are uncorrelated and that all signals have unit variance ($E\{|s_i]^2\} = 1$).[2]

Optimal weights in a phased array

In a free space environment with no scatterers the incoming signal can be viewed as plane waves coming from the direction of the transmitter. That is, assuming that the receiver is in the far-field of the transmitter. The phased array consists of M elements in a straight line spaced a distance d apart, as illustrated in Figure (2.3). It is also assumed that the plane waves have constant amplitude over the array. This implies that the incoming wave from an angle θ is identical at each antenna element, except for a phase skew, ϕ. This assumption only holds if $d \ll 2\pi\lambda$. If the signal

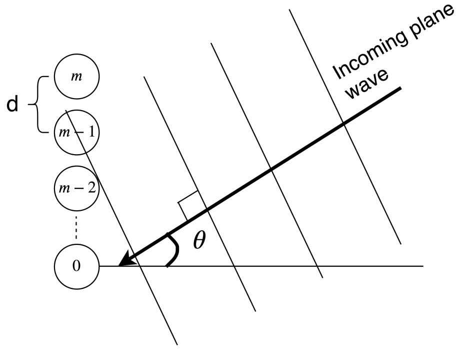


Figure 2.3: Incoming plane wave to a uniform linear phased array.

By using the 0'th element as reference all the other elements will receive the signal phase shifted relative to the reference. The result is commonly referred to as a steering vector, $\mathbf{a}$. As the steering vector determines the desired direction to steer the main lobe of the phased array. The steering vector is given by

$$\mathbf{a}(\theta) = [a_0(\theta), a_1(\theta), ..., a_m(\theta), ..., a_N(\theta)]^T \tag{2.7}$$

where the m'th element is given by

$$a_m(\theta) = e^{-jkmd\sin\theta} \tag{2.8}$$

where $k = \frac{2\pi}{\lambda}$ and j the imaginary number. This model is easily extended to three dimensions by defining the position of each antenna element by a positing vector $\mathbf{d}$ and the plane waves' direction by a unit vector $\hat{\mathbf{r}}$. The phase of the m'th element is then given by Equation

$$a_m(\theta) = e^{-jk\hat{\mathbf{r}}\cdot\mathbf{d}_m} \tag{2.9}$$

By the result of Equation (2.9), inserting Equation (2.7) into Equation (2.2) the optimal radiation diagram can be fond as the magnitude response of the phased array output with optimal weights (Equation 2.10).

$$|y| = |\mathbf{a}(\theta)^T \cdot \mathbf{w}_{\text{opt}}|, \quad \theta \in [0, 2\pi] \tag{2.10}$$

Equation (2.10) can be used to investigate how the phased array will behave in different scenarios in a free space environment. It is assumed that $d \leq \frac{\lambda}{2}$. Some of the main points to notice are as follows:

- With only a desired signal the main lobe will be focused on the desired signals.
- Without noise, N antennas, $N - 1$ interferers, and 1 desired signal. All interferers can be cancelled out completely.
- A system with N antennas have N degrees of freedom. If the numbers of interferers is greater of equal to N, all interference can not be cancelled out completely. A trade-off will be made.
- In the presence of noise, the optimal solution may not be to completely cancel out all interferers. A trade-off to obtain the highest SINR is made.
- If the assumption of $d \leq \frac{\lambda}{2}$ is broken the signal is effectively undersampled in spatial resolution. The Nyquist sampling rate is broken and the algorithm receives an aliased signal. This can cause aliased lobes which are unsuitable to both enhance desired signals or cancel interfering signals.

It can be summarized as follows. A phased array with N antennas can completely remove $N - 1$ interfering signal sources in the absence of noise. With the presence of noise and optimal solution which maximizes the SINR is chosen. The Wiener solution is one of several solution to this problem, commonly found in radio equipment because it is easy and cheap to implement.REF

2.3 FIR filters

The causal finite impulse response, or FIR filter is given by the difference equation

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] \tag{2.11}$$

Where b_n is the filter's impulse response, and $y[n]$ and $x[n]$ the filter's output and input, respectively. By causal is meant that the filter does only can depend on previous and current samples,not future ones. As the name implies, the filter has finite response. Mathematically this can be expressed as

$$h[n] = \begin{cases} b_n, & n = 0, 1, \ldots, M \\ 0, & \text{otherwise} \end{cases} \tag{2.12}$$

By taking the z-transform of Equation 2.11 it is trivial to show that the filter is all-zero, except for a pole at $z = 0$. That implies that the FIR filter is inherently BIBO-stable, independent of the filter coefficients. This is as opposed to IIR-filters which can be BIBO-unstable, depending on the input. Although IIR-filters can achieve higher performance through feedback-loops for a given amount of computation/digital logic, FIR-filters are still widely used in communication systems due to their mathematically guaranteed stability.

A filter with $M \to \infty$ and $b_n \in \mathbb{R}$ could in theory produce any filter response. Except, of course, having an infinitely long filter breaks the assumption of having a finite impulse response in the first place. In practice will the filter coefficients b_n also have to be quantized because of memory limitations. A better statement would be that a sufficiently long FIR-filter with filter coefficients of acceptable dynamic range, can approximate any filter response.

A general FIR filter structure is given in Figure 2.4. A visual representation of Equation (2.11) for any b_k. The structure is known as a normal direct form FIR filter.

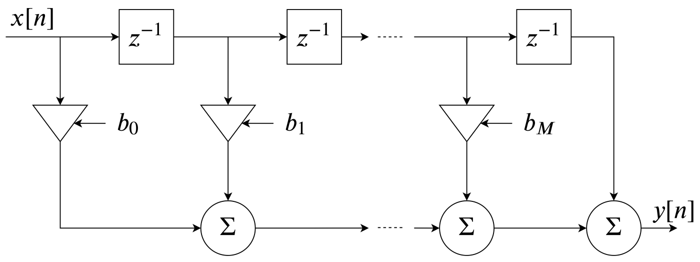


Figure 2.4: Normal direct form FIR filter structure.

Such a FIR filter can be used to model as SISO (Single Input Single Output) communication system. That is, a communication system with one transmit and one receive antenna. How to expand this model to allow for angular spread is discussed further in chapter 2.4.

2.3.1 FIR filter optimizations

FIR filters are often used as band limiting filters in communication system. Low pass or high pass filters for instance. There exists more efficient filter structures for such use cases as coefficients are symmetrical. That is $b_i = b_{M-i}$. Examples of such FIR filter structures are Cascade form and Linear-phase form. As a general communication channel impulse response does not act as a pure band-limiting filter, there are very few to these optimizations that are relevant for the channel emulation. There does however exist efficient ways to implement FIR filters in FPGAs, which can be utilized.

2.4 Channel modelling

A single radio channel can be modelled as a FIR filter by combining Equation (2.1) in the scalar case ($M = 1$) and Equation (2.11) into Equation (2.13).

$$y[t] = h[t] * x[t] + n[t] \tag{2.13}$$

Where $*$ is the convolution operator, $h[t]$ the channel's impulse response, $x[t]$ the transmitted and $y[t]$ the received signal. $n[t]$ is additive noise.

The interesting part is the channel's impulse response. In the case of multipath fading a channel impulse response may look like in Figure 2.5. Where the arrows are instantaneous tap power, i.e. the fir filter tap magnitudes. The red line represents the average energy which often is modelled to decay as an exponential function for each multipath component.

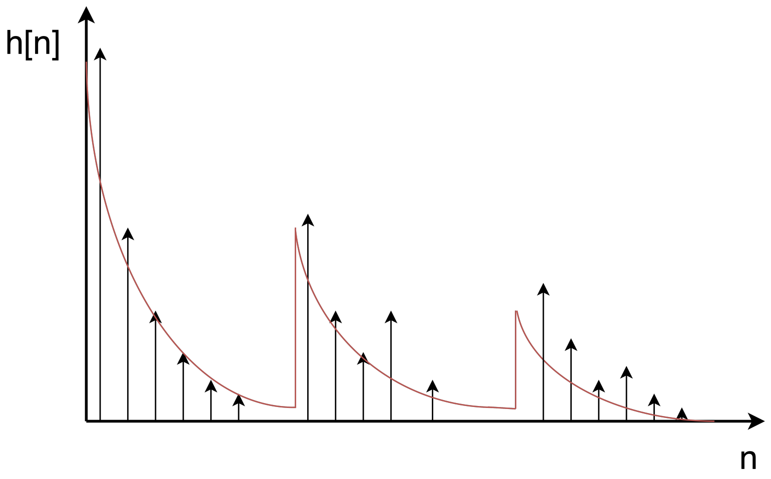


Figure 2.5: Channel impulse response $h[n]$ as function of relative delay n.

However, there is a big limitation for this model rendering it unsuitable for phased-array radios. It implicitly assumes that multipath components are only functions of time and not angle of arrival. This defeats the sole purpose of using a phased array.

To extend the model it is clear that the filter response should be a function of θ as well as time. Due to hardware limitations, the antenna diagram of the transmitter needs to be known then computing the channel response in order to compensate for it. Assume that the channel is drawn from a probability density with some PDF (Probability Density Function)

$$f(t, \theta) \tag{2.14}$$

The shape of the function will depend on the simulated environment put as an input to the channel emulator. The creation of different channels is beyond the scope of this thesis. Since $h(t, \theta)$ is directly proportional to $f(t, \theta)$ the total output of the channel emulator can be defined as

$$\mathbf{y}(t) = \mathbf{H}(t, \theta)(\mathbf{x}(t) \odot \mathbf{w}_{\text{tx}}^{-1}(t)) + \mathbf{n}(t) \tag{2.15}$$

Where $\mathbf{w}_{\text{tx}}^{-1}(t)$ is the inverse of the transmit antenna weighs. This is necessary due to a hardware of using multiple FPGAs, further discussed in Chapter 3.2. $\odot$ represents the element wise multiplication of two vectors. The operation of Equation (2.15) has effectively converted the phased array into a SIMO system. Where the transmit antenna is treated as a single antenna with a radiation diagram set by the transmit weights $\mathbf{w}_{\text{tx}}(t)$. The receives is still allowed to change its weights at will, one of the algorithms channel emulators often can be used to test.

It is natural that the weights depend on time as the optimal weights will change as the channel changes with time. For how long a channel is considered stationary is called the coherence time, T_c. The coherence time itself is proportional to the maximum doppler shift, f_m, which is given in the specification.

2.5 FPGAs

Field-programmable gate arrays or FPGAs are integrates circuits containing digital logic designed to be reprogrammable. FPGAs allows for prototyping or producing fast, highly-specialized digital circuits. These circuits are generally used in areas where high speed and accurate timing are required. In communication systems for instance.

FPGAs are normally much cheaper than their ASIC equivalent. At least unless the production volume is substantial. REF That is why FPGAs are used in specialized, small volume produced equipment.

FPGAs are almost always programmed using a Hardware Description Language (HDL) such as VHDL or Verilog. HDLs differ from programming languages as their output cannot be run on a CPU, but their output is itself used to create digital logic circuits. A hardware description language can therefore be used to build a CPU, but the CPU has to be programmed in a programming language.

An FPGAs programmable logic (PL) is divided into logic blocks.[1] There can also exist slightly different logic blocks within the same FPGA. An example of a SLICEL REF logic

[1]Logic blocks have different names and structure depending on the FPGA manufacturer.

block used in Xilinx' FPGAS is given in Figure 2.6.

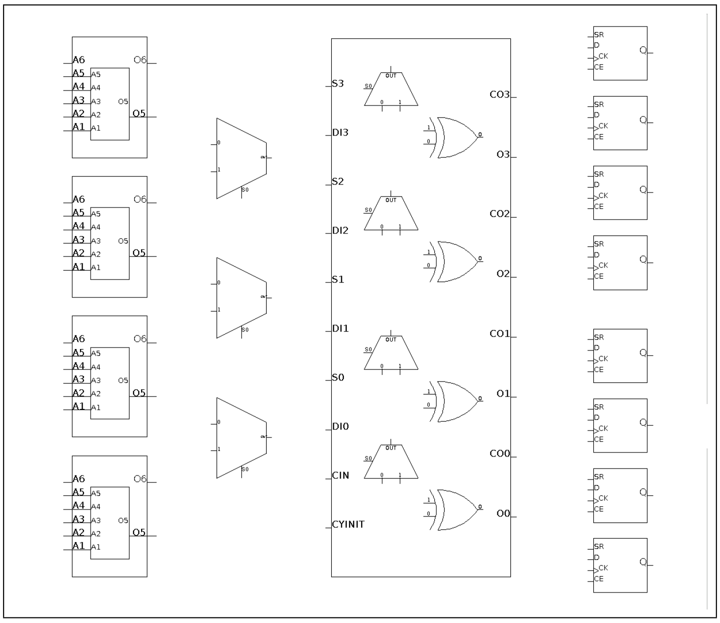


Figure 2.6: SLICEL logic block from Xilinx FPGA REF

From the left it contains four lookup tables (LUTs) with can be programmed to perform and arbitrary boolean function. Next is three multiplexers (MUXes). The big block is a four bit Carry-lookahead adder made from MUXes and XOR-gates. The last row are Flip-flops for storing bits and reducing the critical path by utilizing multiple clock cycles. It is important to understand that there can exist tens of thousand of logic blocks in a FPGA fabric and that these can be connected to create more complex logic.

Modern FPGAs also contain more units such as hardware DSP-blocks, SRAM and hardware multipliers. These allow for a faster implementation of common DSP functions. FIR-filters for instance. It is possible to implement multipliers using LUTs, but they will never achieve the performance of hardware multipliers. Some FPGAs, such as the Xilinx Zync-7000 SoC series, contain two ARM Cortex A57 CPU cores with several peripherals. This is referred to as the PS, or processing system. REF Using a processing system allows the FPGA to run software code for easier communication with the Programmable Logic. It is common to run a lightweight Linux distribution as an operating system in the PS and then use drivers to communicate with hardware modules in the PL.

Not all FPGAs have an PS implemented. As an alternative a soft-CPU programmed in the PL can be used. This has many drawbacks. They are slower, consume more power and occupy valuable logic-blocks that could be used for specialized programmable circuity.

Fixed-point numbers

To explain fixed-point numbers a brief introduction of regular floating point numbers is needed. Floating point numbers are an approximation of real numbers with a trade-off between range and precision. REF Almost all programming languages have support for floating point numbers. REF Since many problems in computer science are easiest to solve using real numbers, floating point arithmetic is widely used. A floating point number is given as:

$$s \cdot b^e \tag{2.16}$$

where s is the significand, b the base and e the exponent. Where all of the components (s, b and e) are integers and $b \geq 2$. Floating point numbers are essentially numbers written in scientific notation. This allows both very small and very large numbers to be represented in the same number of bits, effectively introducing a floating decimal point. Thus the name, floating-point numbers. There are many quirks regarding the details of floating-point numbers. For further reading see the IEEE-754 standard. REF

There is however one major drawback of floating point arithmetic compared to integer arithmetic. It requires vastly more complex electronic circuits to implement. This added complexity generally makes calculations with floating point numbers slower that calculations using integer numbers. In CPUs floating points calculations are often hardware accelerated using a floating-point unit or FPU for short.[2]

Since FPGAs traditionally does not have hardware FPUs, it is common practice to use fixed-point arithmetic. Contrary to floating-point arithmetic with a floating decimal point, fixed-point arithmetic uses a fixed stationary decimal point. That is, it is predetermined how many bits that will be dedicated to the integer and fractional part of the number.

[2]Not always true for small microcontrollers.

Chapter 3

Method

3.1 Channel emulator, a system overview

There are multiple challenges to overcome when moving from a theoretical simulated channel model to a implementation. Foremost that the hardware itself my affect the model introducing errors or artifacts into the model. Since the channel is modelled with FIR filters, these effects can be classified into two different artifacts. Phase- and amplitude changing artifacts. Phase artifacts may generally occur when due to imperfect sampling and/or clocking while amplitude artifacts may originate from imperfect amplifiers, ADCs and DACs. In this chapter a set-up for realizing a channel emulator with the available hardware is purposed. A measurement set-up for characterizing the hardware's performance is also purposed.

The hardware set-up are multiple Xilinx Zynq-7000 [3] boards, each with two Analog Devices AD-FMCOMMS2-EBZ [6] development boards connected to them. The set-up is shown in Figure 3.1. In this master projects four such boards were used as a proof of concept. The FMCOMMS2 boards each contain a AD9361 RF IC, supporting two RX- and two TX-channels in full duplex mode. This gives each FPGA four radio channels to emulate. [4]

It should be noted that the FMCOMMS2 boards contain a Johanson Technology's 2450BL15B050E rated for operation in the 2.4-2.5 GHz range, optimized for operation at 2.45 GHz. This is a hardware limitation makes measurements outside this range unreliable. Because of this limitation all measurements are done at 2.45 GHz unless stated otherwise.[6]

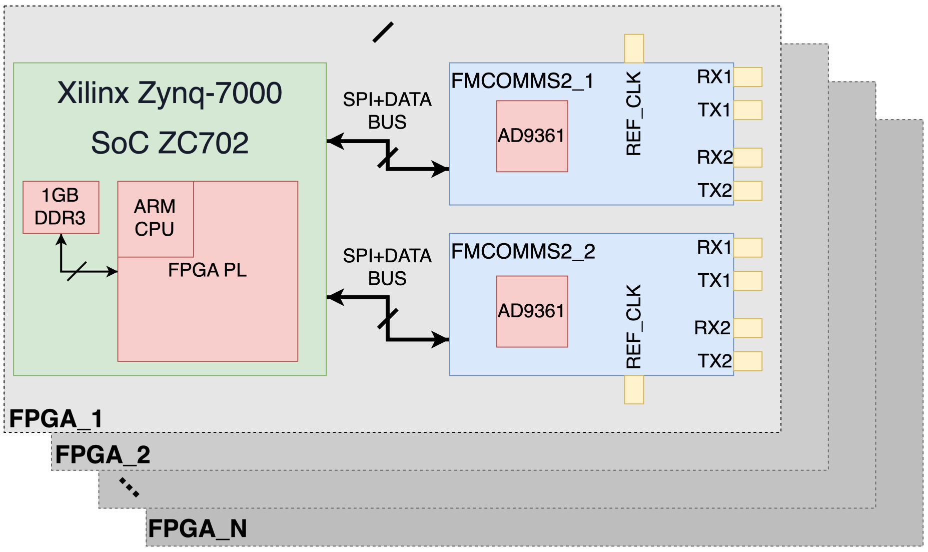


Figure 3.1: FPGA and Hardware set-up for an arbitrarily large antenna array.

The configuration of the AD9361s are not trivial to set up as there are literally several thousands of configuration registers to set in order for the to operate as desired and meeting specifications. The ease the task of the programmer somewhat Analog Devices has written a HDL-block that interfaces with the AD9361 chip. The AD9361s are configured through a HDL-block located in the FPGA. This block is again connected to the ARM CPU in the fpga and has to be configured through an API or Linux driver. In this project all ARM chips run linux and is therefore controlled through a linux driver. The a HDL modification is needed because the default driver does not allow for a pass-through of the signal from RX to TX as is required in a channel emulator. Rather the chip is designed to send and receive data, thus the HDL block has direct access to the 1 GB of DDR3 RAM. The driver is modified to just start and stop the pass-through of data in addition to controlling the Automatic Gain Control of the AD9361s. This is a necessary step which will be explained in greater detail in Chapter 3.3.

As shown in Figure all FMCOMMS2-boards have an REF_CLK input. This reference clock is used in order to keep multiple boards synchronized. It can vary between 10-80 MHz, but for the best performance a clock faster than 30 MHz is recommended. [4] The AD9361 does contain an internal reference clock, but it can normally not be shared between multiple boards in this configuration. The clock frequency must be specified in the Linux Device Tree.[7]

One can easily imagine that the cables coming from the reference source are of different length. This will create a relative delay between each boards timing, thus the will never be *phase aligned* without proper calibration. As long as they don't drift relative to each other and thus stay *phase coherent*, this will not be an issue. If the channels cannot be configured to be *phase coherent* one basically have created an antenna array where the antenna elements are moving relative to each other in space. This will of course not be a working system.

3.1.1 A closer look on the AD9361

A block diagram of the AD9361 chip is shown in Figure 3.2. The $XTALN$ and $XTALP$ pins refer to the REF_CLK pin in Figure 3.1. There are multiple RF input ports to each of the channels, A, B and C. The purpose of this is to support communication across multiple bands. The wideband LNA connected to port A supports the whole frequency range of the chip. Since only one band per input is required for channel emulation, port A is used. The same goes for the output TX channel. Port A is also used here.

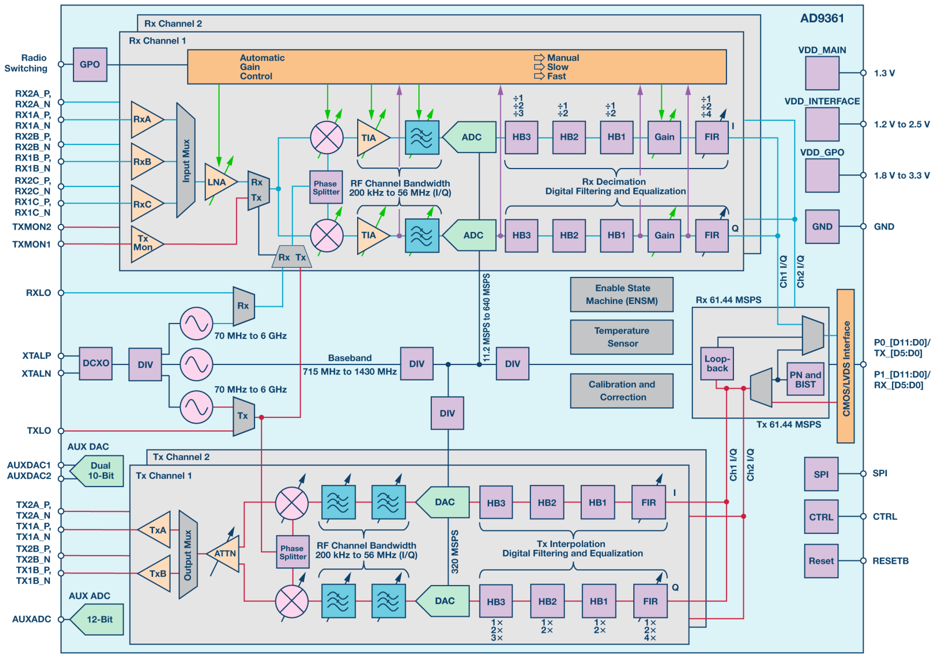


Figure 3.2: Block diagram of the AD9361 chip. Courtesy of Analog Devices REF

The REF_CLK is connected to three different PLL synthesizers according to the schematic (This may be implemented in Silicon, but that is Analog Devices' intellectual property and the schematic is therefore assumed to be correct). This can pose a problem because phase coherency is required by the system. More on this later. Another issue is LO leakage between the TX and RX side of the chip. Placing them at the same or similar frequencies may cause frequency intermodulation of the LO leakage which in turn distorts the TX signal.

In this project the ADCs and DACs are run at their maximum frequencies, 56 MHz and 61.44 MHz respectively, even with a maximum specified channel bandwidth of just 20

MHz to get the maximum oversampling possible. This will help the channel emulator to avoid timing misalignment between antenna elements. Note that this is the output sampling frequency of the signal processing chain. In practice the RX is sampled at 640 MSPS and the the TX ad 320 MSPS. The output is filtered and decimated. Since the channel emulator always will be connected to mains power, power consumption was not taken into consideration in the design. All ADCs and DACs are 12-bit, giving a high dynamic range for the signal.

The channels does support FIR filters internally for channel equalization. These are left as all-pass filter and the processing is rather done in the FPGA. The only reason for this design choice is arbitrary filter length (only limited by the FPGA resources available) and more control over the design when keeping everything in the FPGA signal chain. There is certainly room for better recourse utilization in later design revisions.

3.1.2 FPGA Signal path design

The signal chain is signal chain is shown in Figure 3.3. Since the in- and output of the AD9361 Core data contain real and imaginary parts (IQ-data), all blocks utilize complex numbers, increasing resource usage. In the figure it is assumed that the incoming signal is 20 MHz wide centred around $f_c = 2.45$ GHz.

First the signal is mixed down with a f_c with a frequency 15.36 MHz lower than f_c. This is a design choice made to avoid intermodulation products between RX and TX. The frequency 15.36 is chosen because it is a decimation of exactly $1/4$ from the ADCs sampling frequency. It is also well outside the channel bandwidth of ± 10 MHz, allowing less resources to be used for the IF filter. The goal is to leave as much resources as possible for the radio channel, **H**, after all.

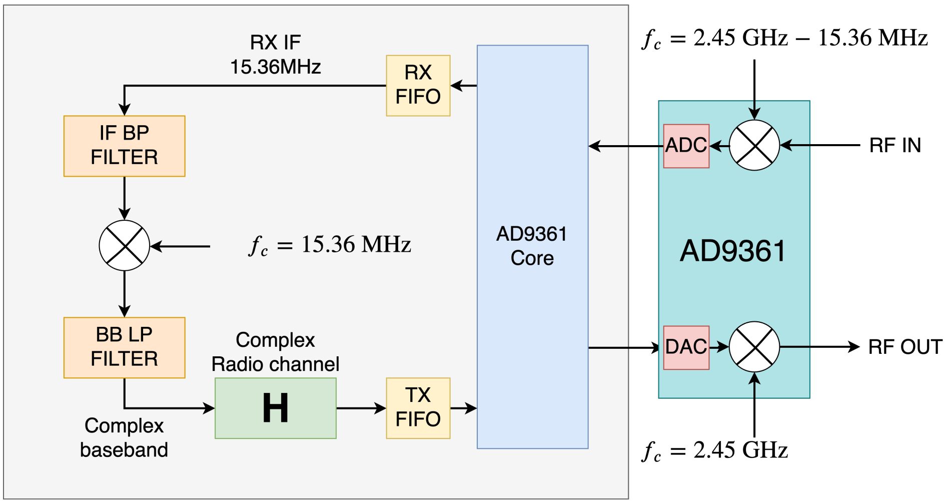


Figure 3.3: Hardware signal path for one channel.

After the first down-mixing the output is sampled and stored in the RX FIFO. An RX FIFO may seem unnecessary when the result is not written to DDR3 memory. It should be, but the driver requires it to function properly. Signal is fed through a intermediate frequency bandpass (IF BP) to remove mirrored frequency. Then the signal is digitally mixed down to baseband around $f = 0$ Hz.

The filters are designed in Matlab to use 16 bit data bus with 12 bit quantization to match the FPGA's internals. The coefficients can then be loaded into a custom made FIR filter block inside the FPGA. This however is not a good idea. FIR filter are very commonly used in FPGAs, and because of this some parts of the FPGA is optimized for FIR filters. To use these parts it is easiest to utilize the FIR filter builder given by Xilinx, the filter builder also gives an option for real-time change of coefficients which is needed in the channel emulator.

The IF band-pass filter is given in Figure 3.4. The filters length of 35 coefficients is not arbitrarily chosen. In simulations it gives a relatively flat frequency response in the pass-band ($[-0.2, 0.8]$ dB in 15.36 ± 10 MHz).

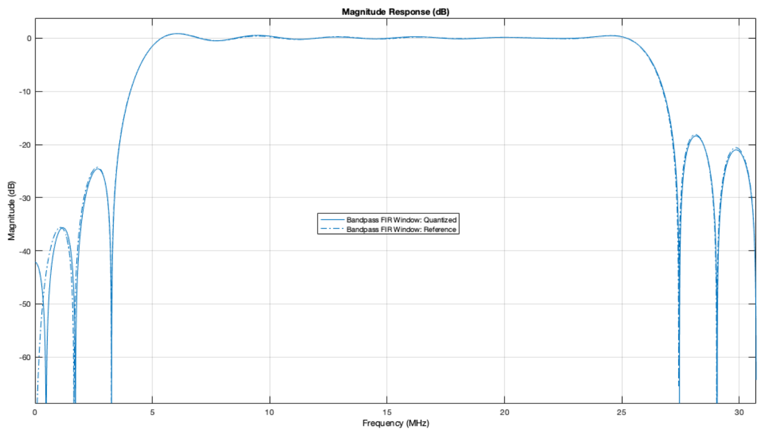


Figure 3.4: The 35 tap intermediate frequency band-pass FIR filter used in the channel emulator.

In addition it has very high dampening (> 80 dB) around 0 Hz and 30.72 MHz where LO leakage could occur depending if the RX is set to f_c plus or minus 15.36 MHz. From Figure 3.4 it is shown that the quantized filter is not predicted to do as well as the floating-point simulation at DC level. Only a dampening of -40 dB is predicted. At 30.72 MHz it is predicted to be more or less identical to the floating-point. Which is good as the RX-frequency is set lower than the TX and thus it is the dampening at 30.72 that is relevant for the systems performance.

The base-band low pass filter is given in Figure 3.5. It is a low-pass filter with a bandwidth of 10 MHz, creating a complex base-band bandwidth of 20 MHz. A raised cosine filter is used for its well-proven performance in real communication systems (a good trade-off between filter length and stop-band performance).

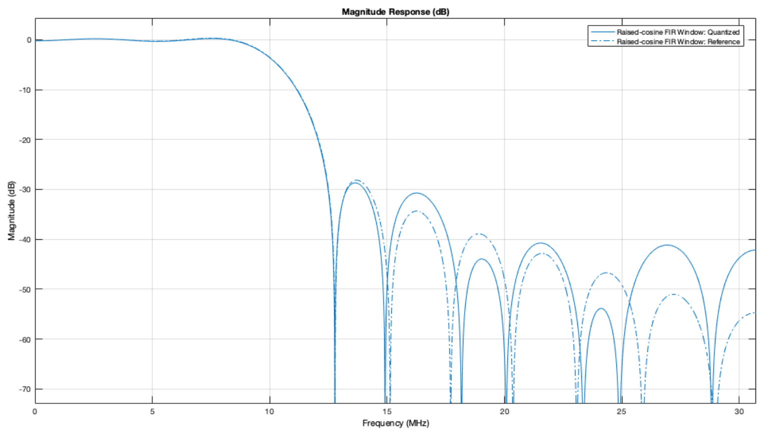


Figure 3.5: The 23 tap baseband low-pass FIR filter used in the channel emulator.

3.2 Limitations of using more than one FPGA

Creating a four antenna elements channel emulator where everything is kept within and connected to a single FPGA would be a challenge. Creating a 16 or more antenna elements channel emulator split across multiple FPGAs is much more difficult. In this section some of these challenges will be presented.

3.2.1 Keeping oscillators in Sync

A frequency synthesizer can generate a high frequency signal by multiplying a low frequency and keeping the high frequency signal locked to the phase of the low frequency signal with a PLL. In an idealistic world one could create multiple high frequency signals locked to the same low frequency signal and the would all have the same high frequency and be phase coherent. Such a system is shown in Figure 3.6.

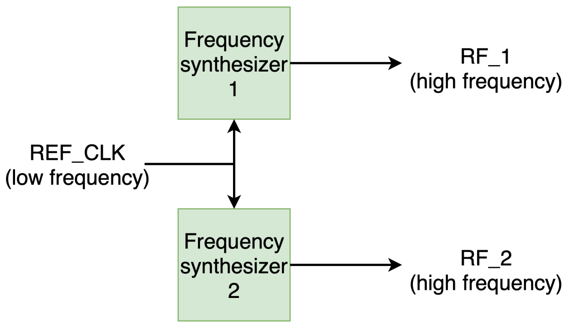


Figure 3.6: Generating multiple high frequencies from the same low frequency.

Unfortunately the world is not ideal and two frequency synthesizers with the same input low frequency may output a slightly different high frequency signal. That is RF_1 and RF_2 has a slightly different frequency. If this occurs, two or more antenna elements would be receiving and/or transmitting at slightly different carrier frequencies.

An obvious solution to solve all synchronization issues is to use one synthesizer as reference for all the FMCOMMS2 boards. An obvious initial attempt may be to use the same synthesizer for both TX and RX channel as shown in Figure 3.7.

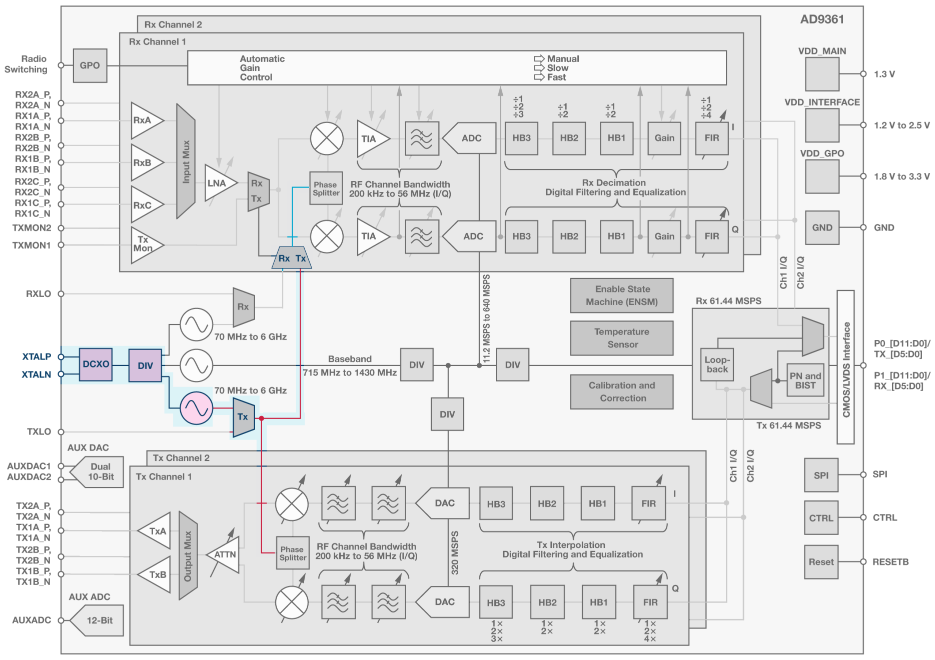


Figure 3.7: Using one synthesizer for both RX and TX.

Unfortunately the configuration shown in Figure 3.7 is invalid. There is no way for the user to access the RX/TX-mux in the RX channel, neither through the driver nor through setting registers manually. In fact, this mux is used internally for RSSI power measurements only according to an Analog Devices employee. One may ask if this mux cannot be used, why include it in the schematic?

Even if the RX/TX-mux was accessible, using the single synthesizer would not solve synchronization issues. The only achievement is to move the synchronization issue one step up the hierarchy. There is certainly no way to get this signal out of the chip and thus multi-chip synchronization would be impossible.

A different approach could be to use an external RF reference like shown in Figure 3.8. It is certainly possible to do that. And it would solve the synchronization between multiple boards elegantly. There are a couple of quirks worth mentioning with this approach.

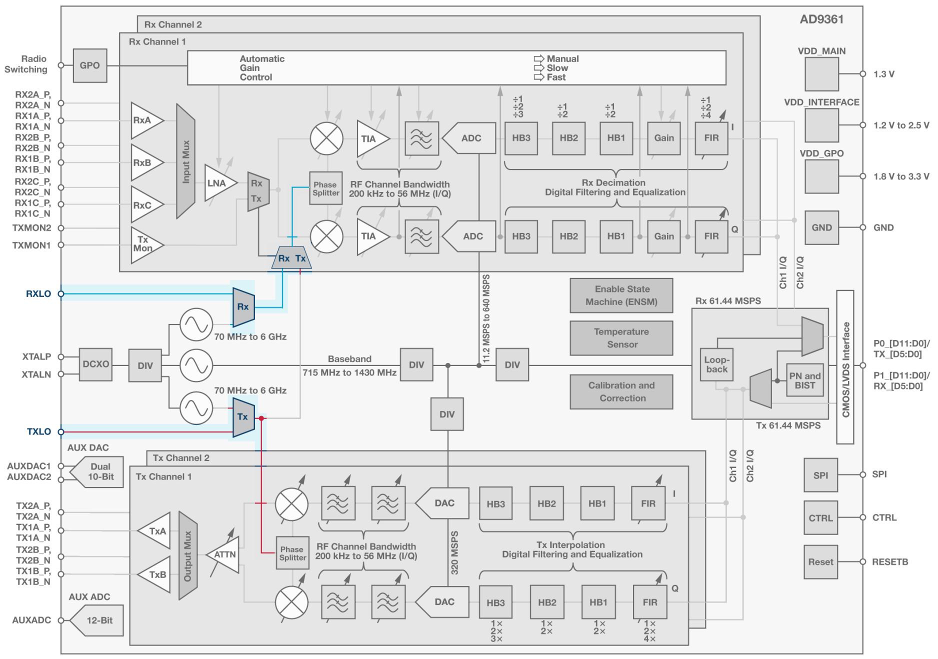


Figure 3.8: Using oan external RF reference for both RX and TX.

First, if the external RXLO and TXLO are too similar in frequency the same LO leakage as described earlier may occur causing undesired intermodulation products. Second, while Analog Devices have managed to show an unusable MUX on their schematic, they have forgotten to add the divide-by-two frequency dividers that exists on both RXLO and TXLO inputs. This implies that a channel emulator operating at $f_c = 6.0$ GHz requires a reference frequency of 12.0 GHz. A relatively high frequency which can be expensive to generate with good performance (low phase noise being most important).

It is therefore tempting to use the same reference signal for both RXLO and TXLO. This will eliminate any problems with intermodulation products, even with LO leakage. Keeping in mind to use similarly length of cables to keep them relatively similar in phase. It should be noted that even though the RXLO and TXLO inputs exist on the FMCOMMS2-boards, they only exist as test points without SMA connectors such that external connectors must be soldered on in order to use them.

3.2.2 Sharing data between FPGAs

One major disadvantage, as mentioned in Chapter 2.4, is the lack of inter-FPGA communication links. This renders the channel emulator unable to simulate MIMO systems. It is limited to simulate SIMO systems, that is a system where the input antenna diagram is know and set, then to produce outputs for all the receiving antenna elements. Since one receiving antenna element at most can know the input of three other antenna elements (four channels within one FPGA), this also limits the possible antenna diagrams (number of lobes) that the receiver can use. This may or may not matter, depending one the receivers ability to produce RX-lobes.

It is possible to calculate how much bandwidth that is required by an inter-FPGA communications link in order to allow for full MIMO operation. With a maximum delay spread from the specifications given at 1.5 ms and a sampling frequency of 56 MSPS, it gives a maximum channel filter length of 84000 coefficients. All of which are 12 bits long. That is around 0.96 Mbits of data. Since the channel is operating at complex baseband, the filter coefficients can be complex too. Doubling this requirement to approximately 1.92 Mbits of data. One must not forget that there exist 4 such complex channels per FPGA, 7.68 Mbits of data that is.

As described in 2.4 the Coherence time (T_c) is proportional to the maximum Doppler shift (f_m). A common approximation is 3.1.

$$T_c \approx \frac{1}{f_m} \tag{3.1}$$

To meet the specified maximum relative velocity difference between RX and TX of Mach 2 and at the same time maximum operating frequency of 6.0 GHz, $f_m = \frac{v}{c} f_c \approx \frac{680}{3 \cdot 10^8} 6 \cdot 10^9 = 13.6$ KHz. This is assuming air density at sea level. With $f_m = 13.6$ KHz the channel must be expected to change (in other words, the Coherence time) every 73.5μs.

This implies that in order to achieve a fully supported MIMO channel emulator with there specifications and without optimizations, each FPGA have to transmit 7.68 Mbits of data every 73.5μs. This relates to about 102 gigabit per second. A transfer speed which is unmanageable by must modern communication systems. If one in addition factor in that each FPGA has to transmit this to all the other FPGAs the number becomes ridiculous and is clearly not the way to go. An approximation is needed.

There are multiple optimizations that can be implemented. The first is to only include filter taps that are in the dynamic range of the channel emulator and/or the devices under test. Looking back at Figure 2.5 if only the three taps with most energy is detectable by the system, there is no need to to do multiplications by zero for the others. Thus there is less information to be transmitted. One simply has to implement a circuit for ”no operation” in the FPGA to ignore the taps that are irrelevant.

Another option is to design a system that can fulfil all the specifications, but not simultaneously. A third, the one that is simplest is to just leave out inter-FPGA communications all together with the limitations this impose.

3.3 Analog front ends and AGCs

The AD9361 have Automatic gain control (AGC) implemented as shown in Figure 3.2. This is a good idea when building a communication system in order to achieve the highest possible SNR without distortion and thus reducing BER. This will also try to adapt the input signal such that the dynamic range of the ADCs are utilized to its maximal extent.

Automatic gain control is a good idea for a communication system. It would also be a good idea for a channel emulator, given that the gain could be instantly known and compensated for in the channel model. Unfortunately this is not the case without major modifications. The AGC setting is controlled by the Linux driver and can be polled from the driver in software, but it will not be timing perfect. That is, it will take some time before compensation can occur. A possible solution could be to have a buffer in the FPGA that was long enough such that it could get a message about change of AGC setting, then measure in the samples exactly where the energy increases or decreases and thus add compensation at just the correct sample.

3.4 Test set-up

The test set-up is given in Figure 3.9. The arrows show connections made. It does not mean that all connections were are all the time. Some rearranging has to be done depending on number of input/output channels, available cables etc.

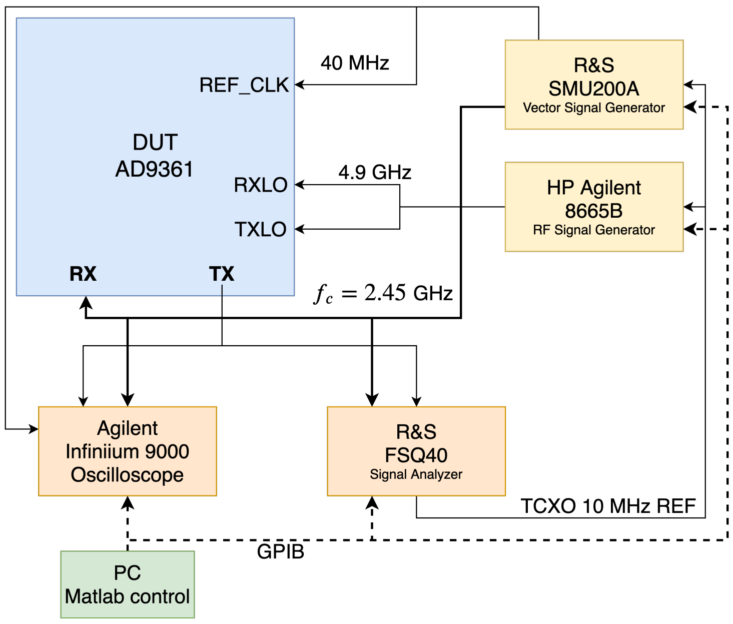


Figure 3.9: Test set-up

Matlab scripts are used to automate some of the testing. The instruments are controlled over GPIB and GPIB-over-USB. It should be emphasized that the measurement set-up Unfortunately only allows for a single channel measurement due to lack of RF inputs in the measurement equipment.

3.4.1 Oscillator drift: clock sync between signal generators

Oscillator drift is a minor point that should be considered when doing phase measurements. From Figure 3.9 it is clear that all signals generated to the input of the DUT is referenced to the internal temperature compensated crystal oscillator (TCXO) of the R&S FSQ 40. Signals internally in the other instruments are synthesized and phase locked to this reference signal. Again, the same challenge illustrated in Figure 3.6. Only this time between separate instruments, as opposed to within the same IC.

Assume that one wants to measure phase coherency between outputs of the DUT. Also assume that these originate or are clocked in some way from separate instruments. The wrong question to ask is if signals generated in two separate instruments drift relative to each other. They always will drift to some extent, depending on the quality of the hardware. A rather better question to ask is: ”Do the outputs drift *more* relative to each other than the inherent clock drift between the instruments?”.

Chapter 4

Testresults

This chapter is about different measurements done to the system with the system set-up given in Figure 3.9 and their results. The results are discussed in Chapter 5. All the tests are performed with **H** as an one-tap channel. All tests, except 4.0.1 were clocked by the external 4.9 GHz RF signal from the HP Agilent signal generator with the IF filter and digital down mixer removed. There is an image of the channel emulator in the Appendix.

4.0.1 Follow signal through FPGA pipeline

This test is to tap the digital signal processing pipeline through the FPGA and verify that the pipeline is functioning as expected. It is a basic test to check the actual performance versus the double-precision implementation in Matlab. The signal is tapped at multiple stages in the FPGA. The tap points are shown in Figure 4.1. The Output results are shown in Figure 4.2 in descending order from the start to the end of pipeline. The AGC is active and set in ”slow-attack” mode. The y-axes are dimensionless digital values proportional to the signal energy.

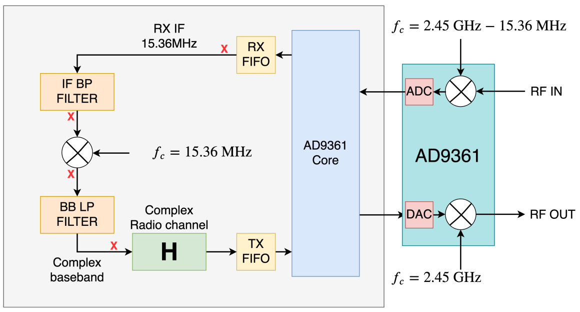


Figure 4.1: Hardware signal path for one channel with tap points indicated.

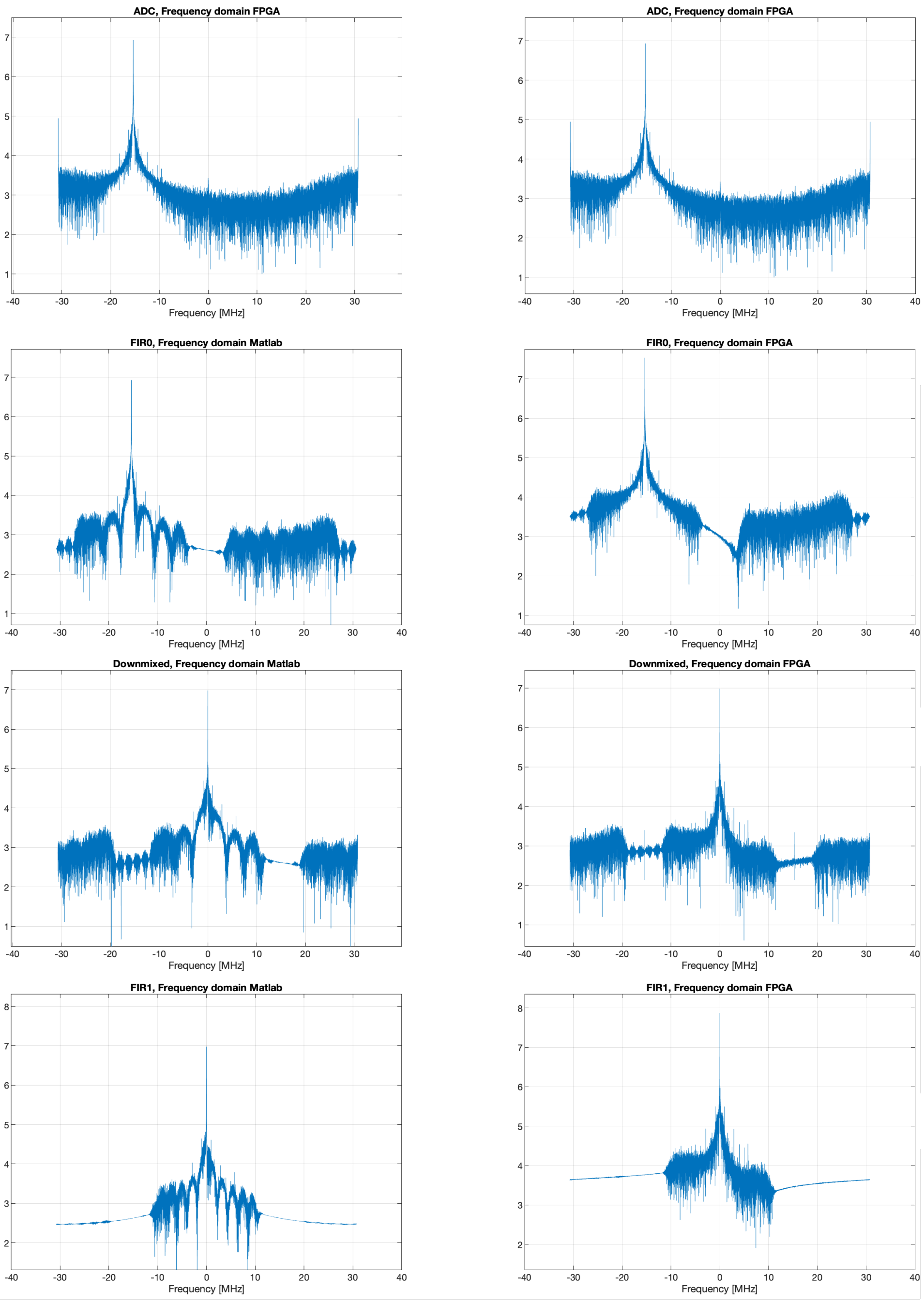
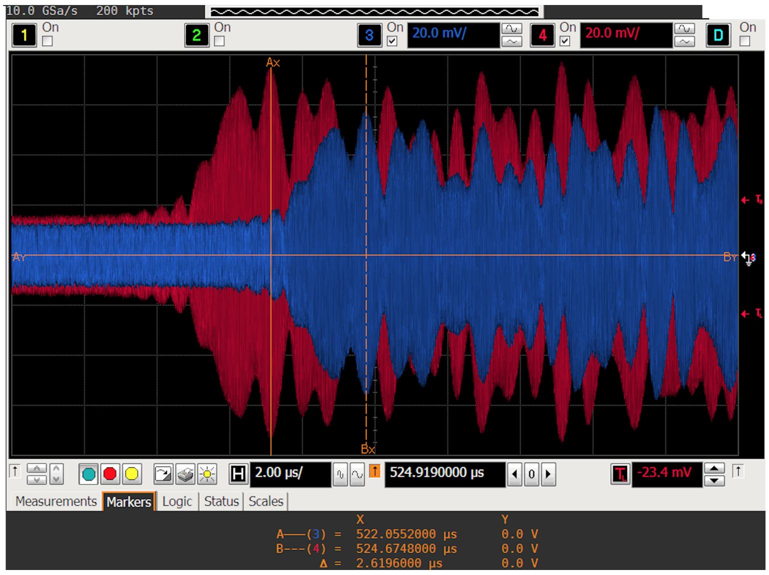


Figure 4.2: Sanity check through pipeline. Y-axes are dimensionless and proportional to signal energy.

4.0.2 Delay through the channel emulator.

As stated in the title this is a measurement of the delay through the system as timing and delay can be relevant if the system is to be expanded. The input signal to the RX is 16QAM modulated at a 2.45 GHz carrier.



```
              X                      Y
A——(3) =  522.0552000 µs      0.0 V
B---(4) =  524.6748000 µs      0.0 V
     Δ =  2.6196000 µs       0.0 V
```

Figure 4.3: Delay of signal through system with single tap all pass **H**.

4.0.3 Measurements of LNA distortion

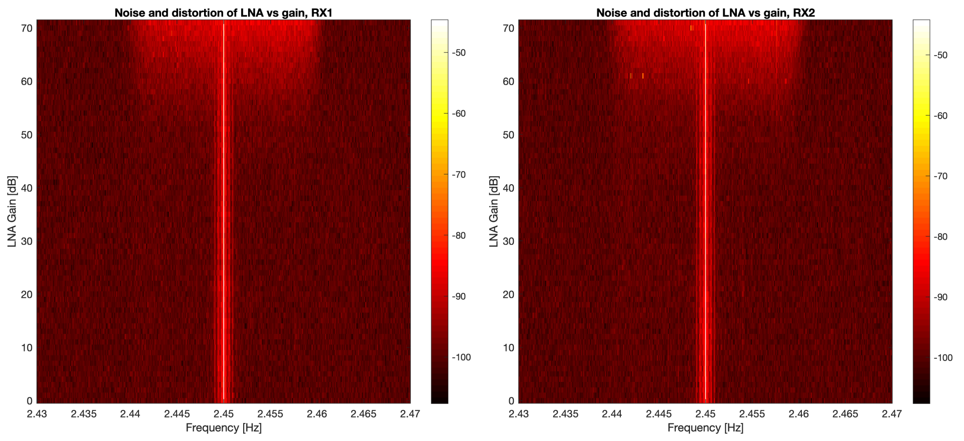


Figure 4.4: The distortion of LNAs for different gain settings. CW at 2.45 GHz.

The AGC have a tendency to overcompensate the gain of the LNA distorting the output. Because of this it is interesting to measure the LNAs performance over gain and frequency ranges. The RX test signal was a CW at 2.45 GHz at -75 dBm power level. Any higher and the ADCs would saturate and distort the output anyway. This would of course give an invalid measurement of the LNAs performance. Both channels are measured to find possible differences.

4.0.4 Measurements of LNA Phase vs gain

In order to have a working system the relative phase between RX and TX should not be dependent of the LNAs gain setting. If it is, it must be compensated for. This is especially important in a system where the AGC is constantly changing the gain setting of the LNA. A measurement of the phase at different gain setting is given in Figure 4.5. As test signal a CW at 2.45 GHz at -10 dBm power level is used.

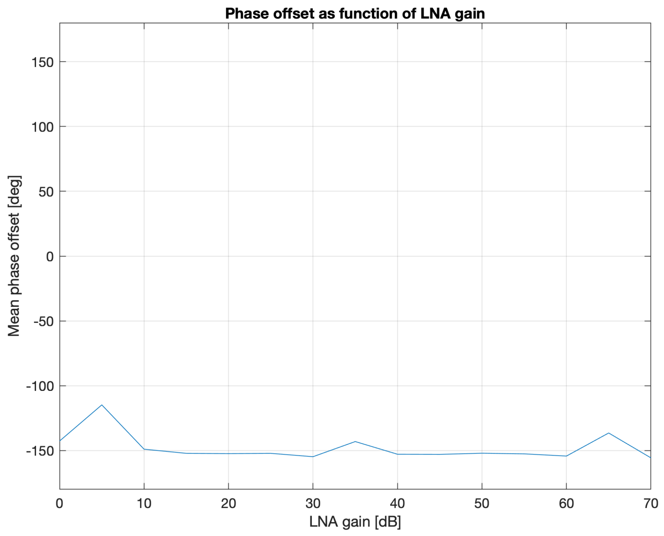


Figure 4.5: The phase relative phase between RX and TX w.r.t. LNA gain. CW at 2.45 GHz.

4.0.5 Phase change width respect to frequency

The same argument as in the previous test. The relative phase between RX and TX should not depend on frequency. The test result is given in 4.6. The test signal is identical to the previous test. LNA is fixed at -20 dBm. All points are averaged phase of between 1071 and 1220 measurements. (The variation is due to timing between oscilloscope and computer).

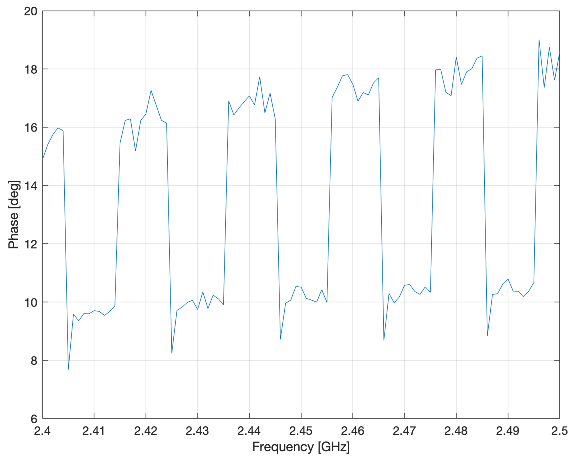


Figure 4.6: The phase relative phase between RX and TX w.r.t. frequency.

4.0.6 Measurements of phase drift

This test is designed to test if the internal synthesizer is adding any significant phase noise to the system. The 40 MHz REF_CLK and test signal (same as above) are clocked from the same TCXO reference oscillator. 100 000 measurements were done at the input and output. If the system internal synthesizer does indeed at phase noise the output should have a significantly different standard deviation.

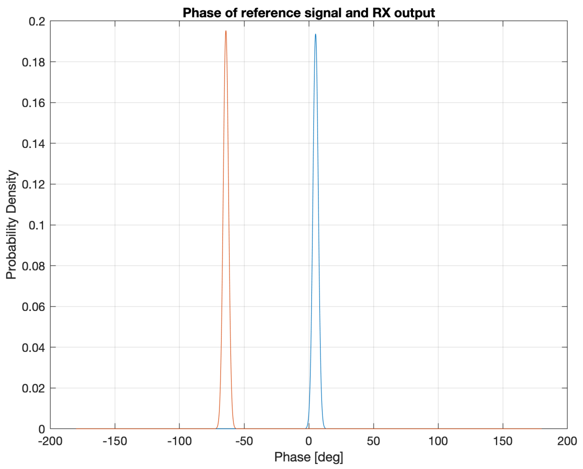


Figure 4.7: Phase of output signal and reference signal.

4.0.7 Resource usage

Table 4.1: Post-implementation resource utilization in one FPGA with who FMCOMMS2 boards attached and 1 tap channel.

Resource	Used	Available	Utilization %
LUT	14538	53200	27.33
LUTRAM	2725	17400	15.66
FF	25377	106400	23.85
BRAM	130.5	140	93.21
DSP	166	220	75.45
IO	94	200	47.00
BUFG	5	32	15.63

In addition the for each extra tap added to $\mathbf{H}$, four dsp slices are required. Since the number of DSP slices available are fairly low and the sampling frequency is relatively slow, the Xilinx synthesizer may be able to swap some of these DSP blocks for other resources.

Table 4.2: Post-implementation timings. All requirements are met.

	Worst Negative Slack	Worst Positive Slack	Worst Pulse Width Slack
Slack	0.132 ns	0.018 ns	0.264 ns

Chapter 5

Discussion

There are many challenges to this project in order to create a fully working channel emulator that meets specifications. The first stage is to verify the system works as expected. This was done in 4.0.1. The first IF filter works as expected and the ripples in the pass-band is very subdued compared to Matlab. Actually it seems to perform better in the FPGA with quantized coefficients. It is also observed that the filter adds some gain that should not be there according to simulations. This is clearly an artefact of the quantized filter coefficients.

The mixer works just as well in the FPGA with just four coefficients (the minimum needed to satisfy the Nyquist rate and very resource effective) as it does with a high resolution sine wave in Matlab.

The baseband filter is fundamentally the same as the IF filter. The FPGA filters performs worse than the double-precision filter which is to be expected due to quantization. The double-precession filter achieves a stop-band dampening of around $7 - 2.75 = 4.25$ relative to carrier, while the FPGA implementation achieves $7.8 - 3.8 = 4$ relative to carrier.

The delay through the system, as shown in 4.0.2, is around 2.62μs, compared to the theoretical 1.05μs for 59 filter taps, sampled at 56 MHz. This implies that the FPGA interface and/or AD9361 chip itself is adding quite a bit of delay through the system.

The AGCs in the system has caused quite a few problems through the project. It was therefore necessary to quantify how they behave. From 4.0.3 it is clear that the LNAs perform well until a gain of around 50 dB. Afterwards is starts to add noise to the signal. The noise floor at the FSQ40s settings were around -108 dBm such that any rise in noise floor level was easily visible. One observes that the output noise is filtered strongly by the FIR filters in the pipeline. However in the pass-band there is a lot of noise energy when the LNAs gain settings are high. This can especially be a problem for the channel emulator as the AGCs seem to prefer high settings rather than low, depending on the mode. Since the AGCs use RSSI to calculate settings, sending in a strong (-10 dBm) CW, will still cause the AGCs to push the LNAs to its maximum gain of 71 dB distorting the output. In a multi-antenna input this will be an even greater challenge as the RX-inputs will receive different amounts of power, causing some TX outputs to be much more noisy than others. This is not a very realistic behaviour and undesired in a channel emulator. A solution to this is, as partly mentioned in the Method chapter, a digital compensation circuit for this. When connecting the transmitter to the channel emulator by cable it is very unlikely that an LNA gain above 50 dB is necessary anyway.

As shown in 4.0.4 the phase seems to be relatively constant over the LNA gain. The deviation is likely do to measurement errors. For the phase over frequency (4.0.5) on the other hand, some ware strange behaviour were observed. The phase is increasing slightly with respect to frequency which is to be excepted, but the square-wave on top is not. It has a period of around 10^{-6} seconds. Since the phase changes about every 10 MHz, it can be that is is affected by the FIR filters. However, FIR filter have linear phase and should not have this behaviour. It can also be that Analog Devices uses chopping amplifiers in their design with a slightly different phase for different frequencies. This is very hard to test as this is Analog Devices IP.

As shown in 4.0.6 the phase in and out of the system as different, which is to be expected. The number of observations were $n = 100000$ and the standard deviations for receive σ_r and transmit σ_t were $\sigma_r = 2.0593$ and $\sigma_t = 2.0423$. The hypothesis $\sigma_r < \sigma_t = 2.0423$ fails at all significance levels for n this large. One can therefore not conclude that the internal synthesizer of the AD9361 makes the phase less coherent.

The resource usage of the system is generally high as 75% of the DSPs are used at only a single tap channel. A majority of the resource usage comes from Analog Devices' own block to interface with the chips. To improve upon this somewhat, one could consider to move some of the filtering into the AD9361s so free up some resources.

There are some issues when creating a system for one FPGA with four channels. These mainly consider calibration. Phase-aligning all channels across all AGC-settings and frequencies certainly is possible but it is a lot of work depending on the accuracy required. This will highly depend on how picky the radio system under test is, but it can almost certainly be solved.

There are some more, rather serious issues when porting this system to multiple FPGAs. The most severe issue is keeping everything synchronized. The internal frequency synthesizers will operate at slightly different frequencies even with the same REF_CLK (< 1 Hz). This was tested, but is rather difficult to graph in any meaningful way. Therefore a video is attached that confirms this observation. A result of different frequencies are that the antenna elements will be operating at slightly different carriers

At first glance antenna elements operating at different frequencies would seem totally system breaking. However, this is not necessarily true. take for instance a relative frequency offset of 1 Hz. A 1 Hz relative frequency offset can be seen as a 1 Hz doppler shift for that particular antenna element. Any system capable of utilizing a channel bandwidth of 20 MHz, will have to be able to deal with such small doppler shifts due to channel dispersion anyway. Thus, a small frequency offset in the RF front ends is acceptable.

In the ADCs and DACs a frequency offset is not at all acceptable. Having a relative frequency offset in the sampling will break all the theory discussed in Chapter 2.2 because the steering vector $\mathbf{a}(\theta)$ will only exist at a single point in time. This is again equivalent for the antenna elements to be moving relative to each other in space. This will be system-breaking as you do not at all want to simulate antenna deformations. That is not the goal of this project.

Having a closer look at Figure 3.2 does not help. It is clear that even if all the RF front-ends are in sync by RXLO and TXLO, is it not possible with the hardware to synchronize the ADCs and DACs across multiple FPGAs. Every chip will use its own synthesizer for the baseband frequency and these will drift relative to each other.

5.1 Further improvements

All hope is not out. It turns out that the AD9361 does have a SYNC pin to achieve the baseband synchronization. However, this pin is grounded and not accessible (BGA-package) on the FMCOMMS2 boards. They can be accessed in the FMCOMMS5 boards, making this project possible in theory. In order to fit the required filter lengths some form of compression where only a few taps are used must be implemented. Also a module to read coefficients from RAM im real-time must be implemented. This is can be difficult as the timing requirements are strict even when the RAM is fast. It is also clear that changing so many coefficients so often, will chew though the RAM address space very fast. A suggestion is to only give a few megabytes of RAM to the linux distribution leaving everything else for coefficient storage. It should also be implemented some form of control application such that a user can control and load coefficients into the FPGAs easily and efficiently.

Chapter 6

Conclusion

Creating a good channel emulator with the given hardware was a hard project. With the given specifications, maybe impossible. That said, following this rabbit hole has enlightened a whole range of potential problems and challenges with such a channel emulator. First it is the calibration that needs to be done on every single channel for every single board. That is a lot of work. But it can be done. It is simply not possible to meet the specifications with the given hardware. The biggest issue maybe the requirement for inter-FPGA communication, depending on the desired performance. Inter-FPGA communication at the data rates given by the specifications is simply infeasible. Moreover, the synchronization of the hardware is unlikely to by good enough to do phase-array channel emulator. The synchronization problem can most easily be solved by switching the different hardware.

There is probably a good reason why the commercially available products does not support channel update rates this fast. It is also clear that if a better compression solution is used, the system can be realized with limited performance. Furthermore, if the doppler shift specifications are lowered to support for example helicopters instead of supersonic jet planes, the design task will be easier.

Bibliography

[1] Simon R. Saunders and Alejandro Aragón-Zavala. *Antennas and Propagation for Wireless Communication Systems*. Jogn Wiley & Sons, Inc., 2007.

[2] Simon Haykin and Michael Moher. *Communication Systems*. Jogn Wiley & Sons, Inc., 2010.

[3] Xilinx. *Zynq-7000 SoC*, 6 2018. `https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf`.

[4] Analog Devices. *RF Agile Transceiver AD9361*. `https://www.analog.com/media/en/technical-documentation/data-sheets/AD9361.pdf`.

[5] National Instruments. *MIMO Channel Emulator for PXIe-564xR*, 7 2018. `https://forums.ni.com/t5/Example-Programs/MIMO-Channel-Emulator-for-PXIe-564xR/ta-p/3820490?profile.language=en`.

[6] Analog Devices. *AD-FMCOMMS2-EBZ User Guide*. `https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz`.

[7] Analog Devices. *AD-FMCOMMS2-EBZ User Guide*. `https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz/software/linux/zynq_tips_tricks`.

Appendix

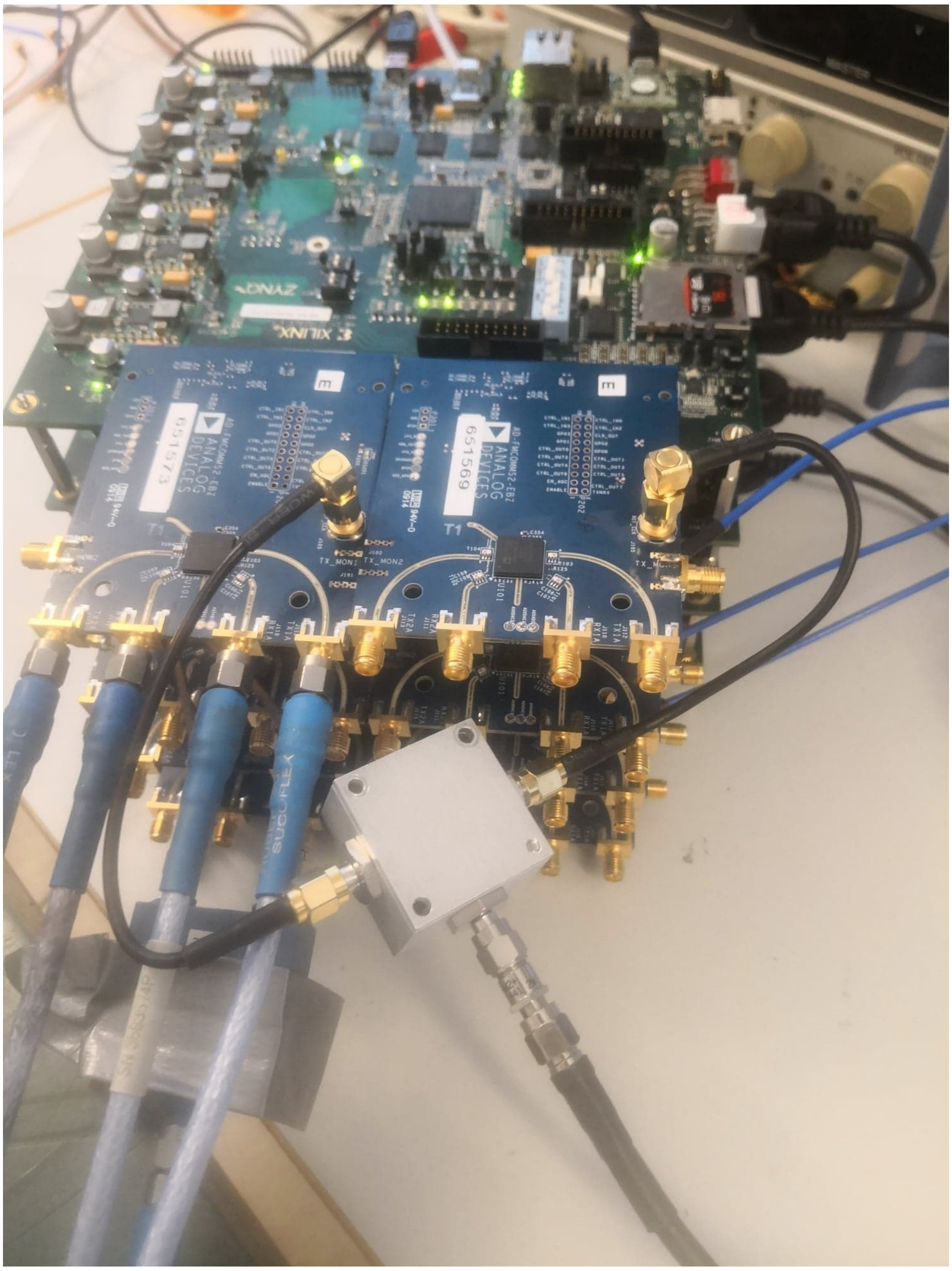


Figure 6.1: Appendix: Image of the channel emulator.

Programming source code used in the project is available at `https://www.github.com/cartfjord/MasterThesisCode`