Intermediate Frequency Data Aquisition Device (*IF_DAD*)

Team name: sddec23-20

Group Members: Nathan Ayers, Matthew Caron, Michael Levin, and Rodrigo Romero

Advisor: Dr. Mohammad Tayeb Al Qaseer

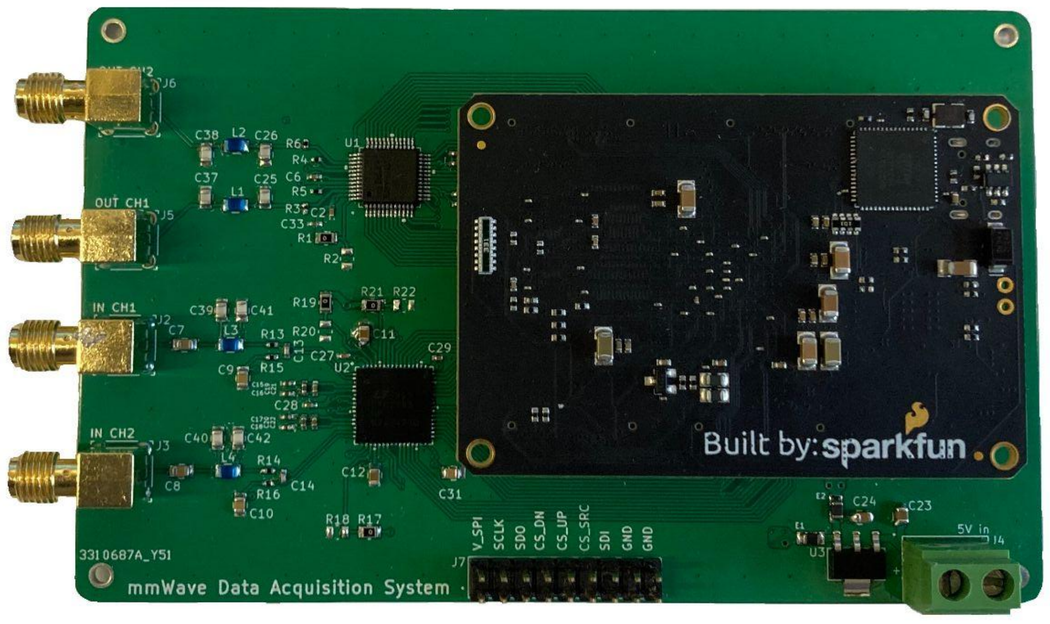


Table of Contents

Introduction

In the world of mmWave imaging it is crucial to have instruments that are able to collect phase and magnitude information from a reference signal. From the phase and magnitude information collected you can calculate data such as reflection coefficients, impedance measurements, and attenuation at various mmWave frequencies.

This project set out to do just that. While utilizing data acquisition components such as: an ADC, a DAC, and an FPGA. Our project did signal processing on output and input signals to calculate such relevant data. This processing allows devices to capture phase and magnitude of an intermediate frequency signal.

This device also has the potential to be connected to an RF board that sends and receives mmWaves in Ka band: 26.5 GHz - 40 GHz. These waves then interact with materials and will result in reflections which create corresponding attenuation and phase shifts that the IF board can read and send to a PC to be plotted in an easy to use user interface.

Device Abstract of Signal Processing

This device calculates the phase shift and attenuation by utilizing the inner products of the output signal and input signal to calculate the phase and magnitude of the received signal. What this essentially means is this calculates how "much" the input signal is "aligned" with the output signal. For example if the output signal is…

$$fout(t) = cos(wt)$$

Then the received IF signal will be a signal of the same frequency but phase shifted by θ and attenuated by some constant A. So the received signal could look like…

$$fin(t) = Acos(wt + \theta)$$

And if you find a signal fout⊥(t) that is 90∘ out of phase from fout(t) such as..

$$hilbert(fout(t)) = fout\perp(t) = sin(wt)$$

Finally you can take the inner product of $fin(t)$ with $fout(t)$, and $fout\perp(t)$ respectively to calculate a complex number whose magnitude is A/√2 and phase is θ
The "real" part of this complex number is found by taking the inner product of $fin(t)$ and $fout(t)$. The derivation can be seen below…

$$< fout(t),\ fin(t) > \ = \int cos(wt) \cdot Acos(wt + \theta)\, dt$$
$$< fout(t),\ fin(t) > \ = A\int cos(wt) \cdot [cos(wt)\, cos(\theta) - sin(wt)\, sin(\theta)]dt$$
$$< fout(t),\ fin(t) > \ = A\int cos(wt) \cdot [cos(wt)\, cos(\theta)]dt$$
$$< fout(t),\ fin(t) > \ = A/2\, cos(\theta)$$

Conversely to find the "imaginary" component you take the inner product of $fin(t)$, and $fout\perp(t)$

$$< fout\perp(t),\ fin(t) > \ = \int sin(wt) \cdot Acos(wt + \theta)\, dt$$
$$< fout\perp(t),\ fin(t) > \ = A\int sin(wt) \cdot [cos(wt)\, cos(\theta) - sin(wt)\, sin(\theta)]dt$$
$$< fout\perp(t),\ fin(t) > \ = A\int sin(wt) \cdot [sin(wt)\, sin(\theta)]dt$$
$$< fout\perp(t),\ fin(t) > \ = A/2\, sin(\theta)$$

These real and imaginary coefficients form a complex number that gives you phase and magnitude information about the received IF signal with respect to the IF output signal, and these coefficients will be different for various materials the mmWaves reflect off of.

Project Design

Making a data acquisition device necessarily involves a complex, interconnected design. For our purposes, data needs to be created in a digital environment to be sent through a digital to analog converter (DAC). This analog signal on its own is not sufficient as the DAC can't produce millimeter wave signals. It must be manipulated in such a way that our antenna can properly transmit a high enough frequency signal. The antenna has to collect that signal, manipulate it for suitable analog to digital conversion (ADC), and analyze accordingly. Through this section, we will describe in more detail the process and expectations for such a device.

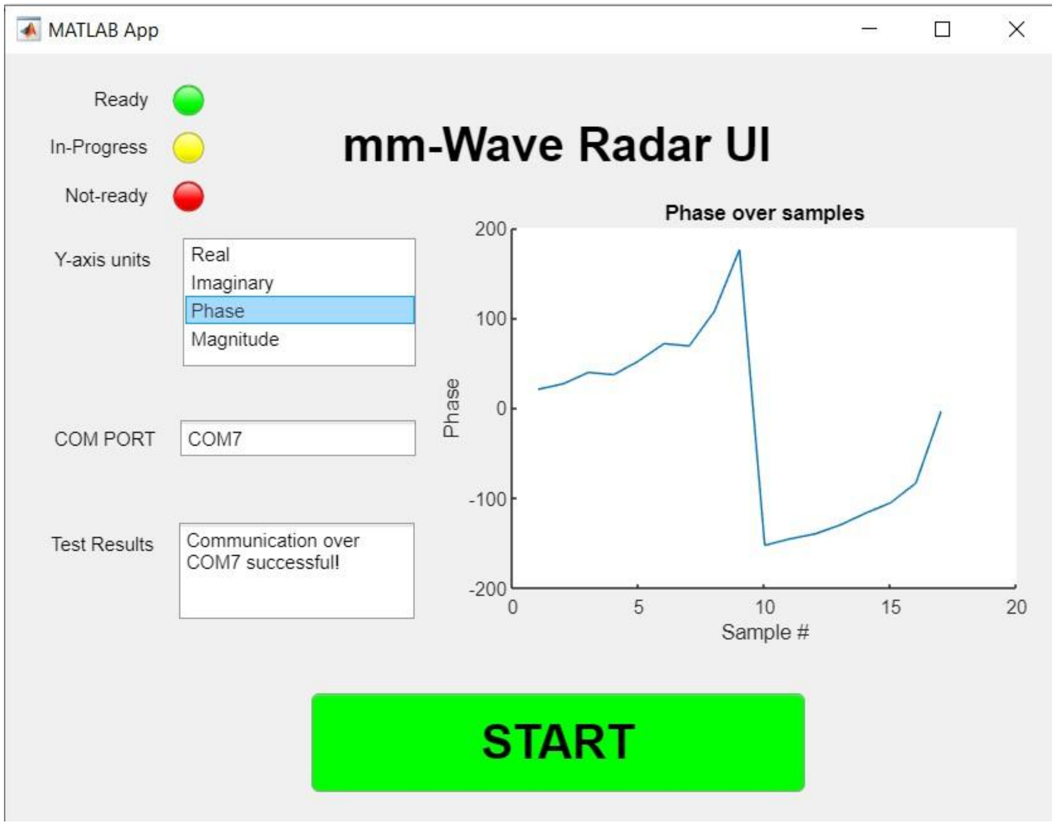
Design Approach

There are three main functions that need to be completed for the device to function as intended. The computer interface has to initialize and trigger the signal created in the FPGA as well as collect the data that comes from the FPGA. The Digital Signal Processing (DSP) needs to take the raw data from the ADC and convert it into something that the UI program can easily display and analyze. The backbone of the device are the hardware components such as the FPGA, ADC, DAC, Filters, and up/down converters which all need to be interconnected and synchronized to adequately move and manipulate the signal.

Computer Interface

For this component, we used a Matlab app as our GUI for the user to input the port on which the USB is connected, the desired analysis on the data, and to trigger the program with the start button. It also needs to communicate all of the relevant information such as the status of the program and whether or not it was successful. Internally, the program uses the inputted port to connect to the FPGA through USB and creates the variable which will trigger an interrupt to create and send the signal when the start button is pressed.

It will use the same port with which it is connected to collect the real and imaginary values produced by the DSP. It then produces a plot of the real values, imaginary values, the phase of these numbers, or their magnitude depending on the user input.



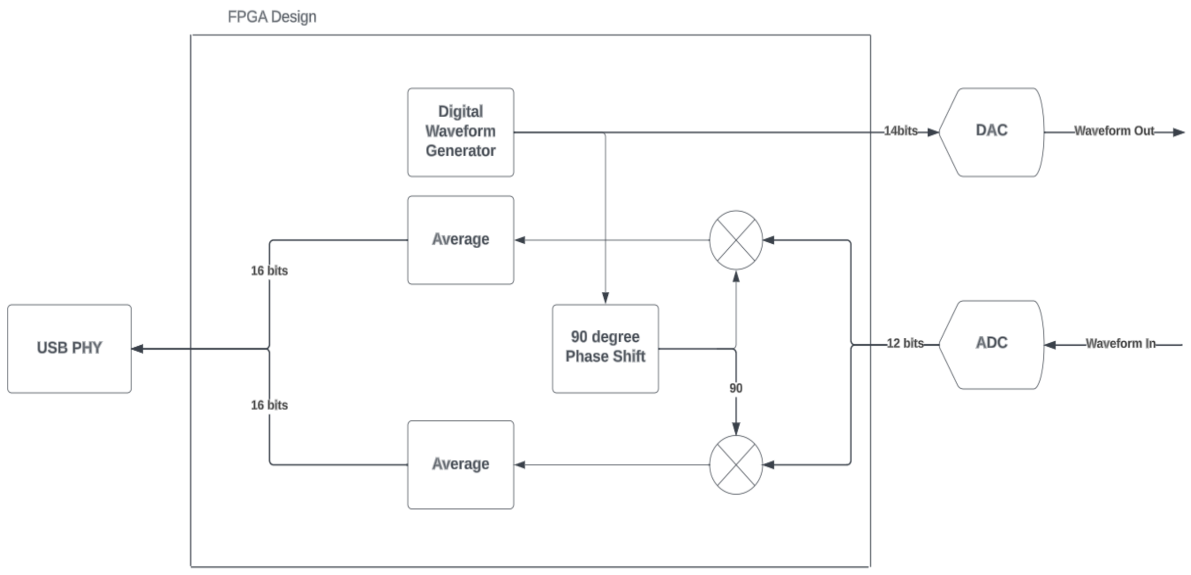
Y-axis units: Real / Imaginary / [✓ Phase] / Magnitude

COM PORT: COM7

Test Results: Communication over COM7 successful!

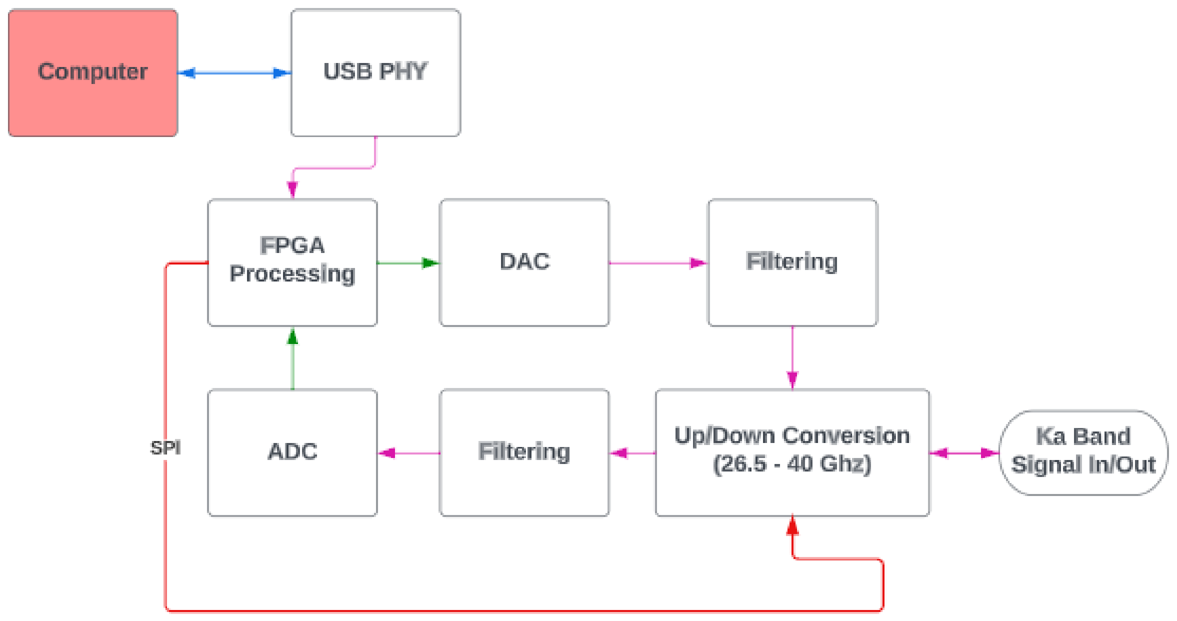
Digital Signal Processing (DSP)

Within the FPGA, there is a preprogrammed sine function which is sent out through the DAC. After the hardware transmits and receives the signal, the ADC produces raw, binary data which can then be processed. The original output signal is multiplied by the raw data to produce the real value. It is also shifted 90 degrees and again multiplied by the raw data to produce the imaginary values. These two values are outputted from the FPGA in a real then imaginary format for all the samples that were received.



Hardware

The signal when transmitted from the DAC, is a sine wave which is IF (10 MHz) which is not sufficient to penetrate the desired samples. To get the needed millimeter wavelength, the signal has to go through an up converter which converts the signal into RF (26.5 - 40 GHz). It's sent through the antenna which bounces off the sample where some of the signal is picked up by the antenna. To properly collect the signal it must be converted back to IF. The ADC can finally convert the signal into the raw data to be analyzed.



Design Requirements

Much of the criteria for this project can be seen through the design methodology. For the hardware, we expected that the signal would be able to transfer the signal through each of the components cleanly with minimal noise. It should accurately convert the signal from IF to RF and vice versa so that the DSP can appropriately manipulate the received data. The raw data must be properly converted using the original signal and its 90 degree phase. The accuracy of the convolution between the two signals determines the accuracy of the real and imaginary values so it has to be synchronized precisely. To do this, we will need a 100 MHz clock which can trigger the sample reception. Lastly, the data must be organized clearly in a file that the Matlab program can read.

Implementation Details

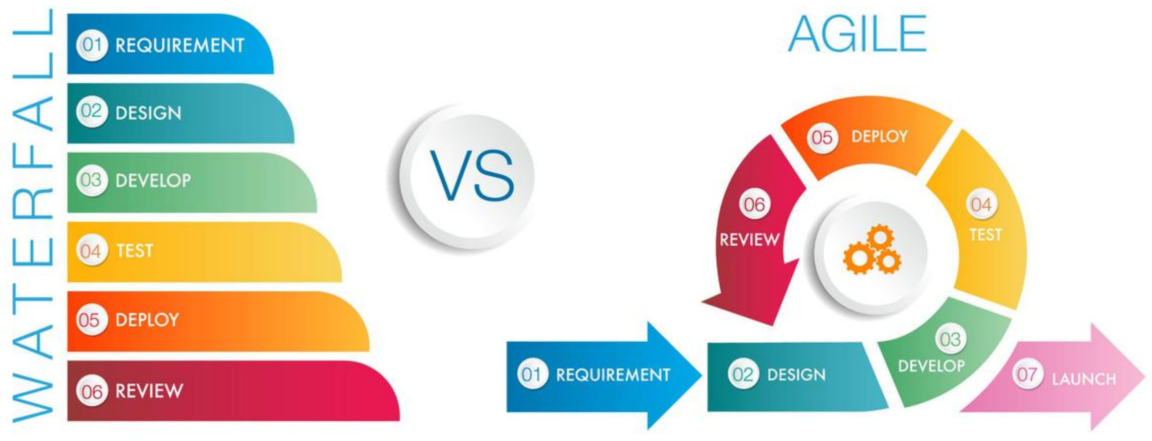
Technical Details

Restated from Project Design, four modules compose this project; first, the hardware consists of a PCB with an ADC and DAC, four precision SMA coaxial cables, and features a connection to

the FPGA development board. The FPGA development board is the Alchitry AU featuring the Xilinx Artix 7 FPGA. The interface between the ADC/DAC and the development board is crucial to the design due to a need to retain signal integrity.
Another module is the Serial Peripheral Interface (SPI), which configures registers from the FPGA to the mm-wave radar portion of the circuit, which is utilized to program the up and down converters. The code itself will be written in Verilog using Vivado. Also utilizing Vivado is the DSP block, which sends real and imaginary signal components to the UI; this module is written in a mix of Verilog and C via the Microblaze processor.

Lastly, the UI is written using MATLAB's App Designer Suite, allowing for blocks to be placed to form a GUI, a very simple process; the original User Interface was to be written for an FTDI chip with the D3XX. Unfortunately, the UI designer struggled with this version of FTDI software, so a last-minute change was made to use MATLAB as the UI vias a serial port in the FTDI chip.

Workflow Methodologies



Waterfall

This project embodied the waterfall methodology because of the perceived sequential order of implementation. We interpereted the order as first PCB, then FPGA (SPI/DSP), and finally the User Interface. The waterfall methodology dictates that each step depends on the previous step's output, meaning that the way the DACs or ADCs or placed and routed to the FPGA development board will affect what pins the FPGA programmer communicates with. With that being said, the FPGA programmer is still responsible for preparing to implement their work in advance, even if they do not know what pins they will use. The tiers of the waterfall methodology are typically listed as Requirements, Design, Implementation, Verification, and Deployment; it is a good rule of thumb to follow the step in front of you by one step, like a D flip-flop. So, if the PCB is in the Verification phase, the FPGA programming shall be in the implementation phase.

The waterfall method is only perfect if the group communicates well despite their independence and maintains high self-accountability. It is more difficult to hold others accountable without a constant authority figure who does not exist in a project of this magnitude. Another issue that can arise from the waterfall method is deadline creep, where one missed deadline can push other deadlines back, and that delay propagates through the entire project.

Our group struggled with many side effects of the waterfall method; sometimes, we needed more accountability for our contributions. We were also guilty of letting our deadlines come and go. We may have set overly optimistic goals, but we could have done better to manage those deadlines passing us by. Some of us never developed a clear picture of what the project was intended to be, and that is typically an issue; the project needs to have a very clear, well-defined pathway.

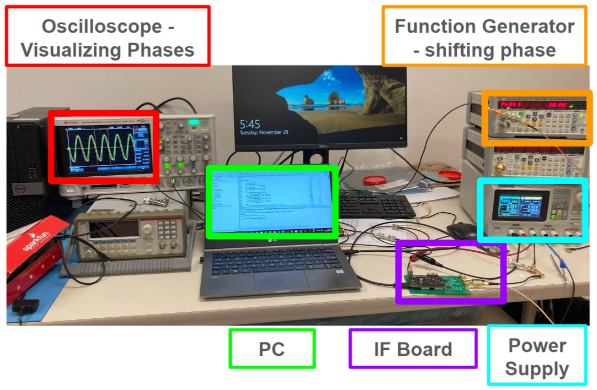
Agile

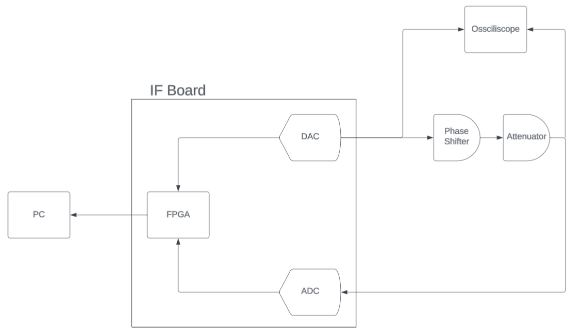
The alternative to the waterfall methodology is Agile, a continuous cycle of collaboration through planning, execution, and evaluation at the end of each cycle. It focuses on breaking up objectives into digestible tasks that can be planned and completed in as little as a few days. While a waterfall methodology may have only one chance to evaluate a participant's contributions over the entire project, the Agile methodology offers the opportunity to critique a participant at least once weekly.

Agile is a great system for many leading-edge companies, but it requires much effort to set up and organize multiple weekly meetings. Our solution to improve our outcomes would be to create a more vigilant waterfall with weekly conversations regarding status, roadblocks, project needs, and more.

Testing Process and Results

To create a standardized testing procedure we created a 10 MHz signal from the IF board then fed the output into the reference port of a function generator. This allowed us to utilize the function generator to precisely control the phase of the signal received to the IF board. This output was then fed into an attenuator to control the amplitude of the output signal. A block diagram and photo can be seen below to explain our setup.

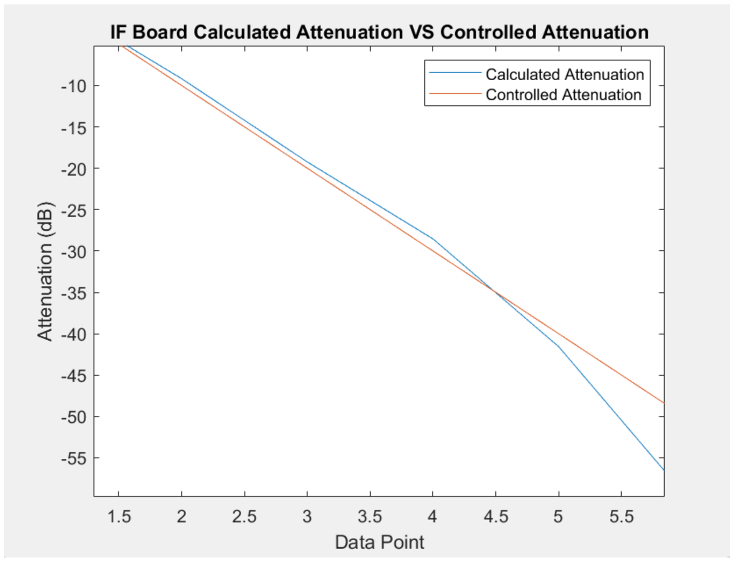

Oscilloscope - Visualizing Phases
Function Generator - shifting phase
PC
IF Board
Power Supply



We then ran the phase shift test and attenuation test separately to confirm we were getting the correct results

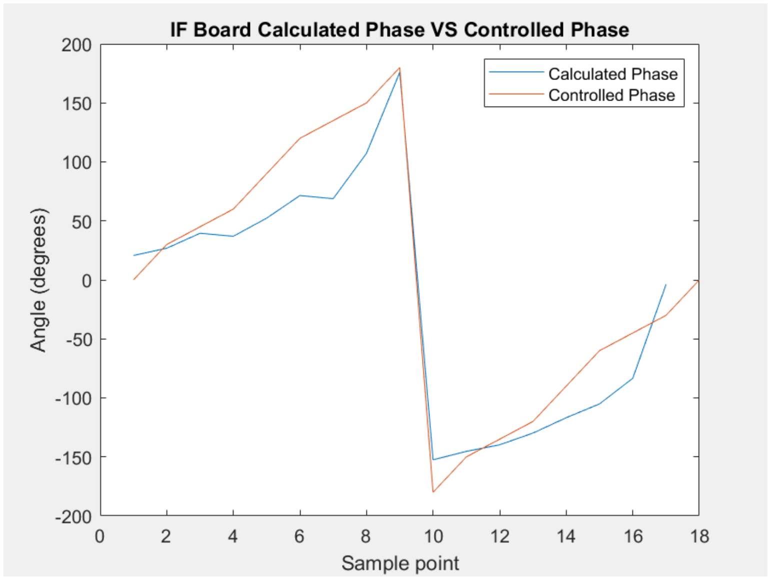
First we ran the attenuation test. This test was done with a 0 - 50 dB attenuator; the signal was fed through this attenuator and back into the IF board. We did not do any phase shifting of the signal for this test.

This test was swept from 0 dB to -50 dB with -10 dB increments. The results for this test can be seen below.



The orange trace is the controlled attenuation meaning this is the actual attenuation and the blue trace is the calculated attenuation by the IF board. From this data the IF board is able to calculate the attenuation well up until -40 db since the plot starts diverging.

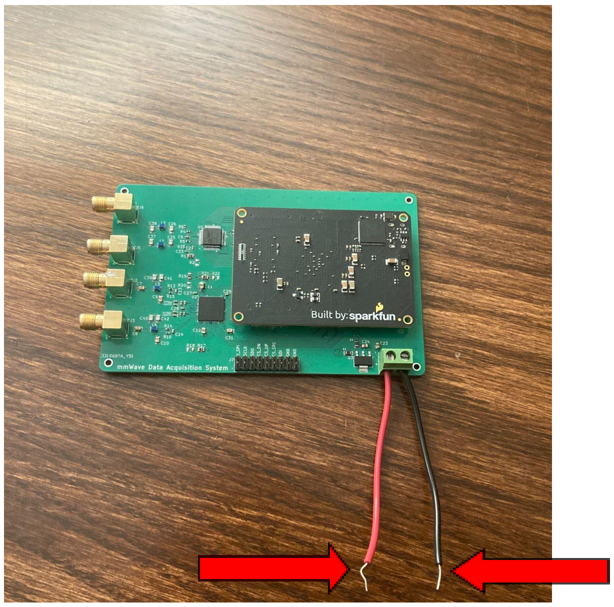
The next step we took was testing the phase shifting like was written before we utilized the function generator to shift the relative phase of the IF signal from 0° to 180° then -180° to 0°. The plot of this test is shown below.



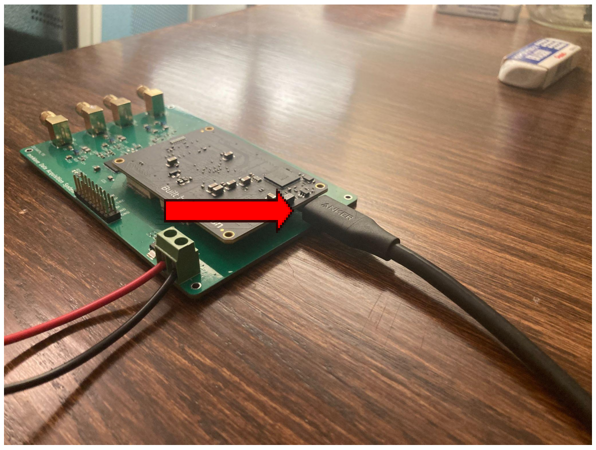
As the plot shows from this test the controlled phase shift from the function generator is the orange trace and blue trace is the calculated phase output from the IF board. From this plot we can tell that the prototype is obviously able to track the phase shift well.

Operation Manual

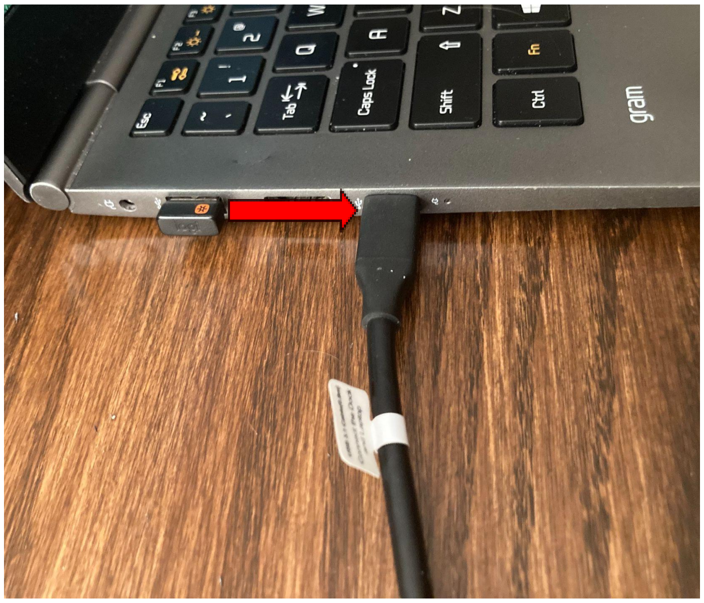
First connect the board to an external 5V power supply with the red and black wires as seen below



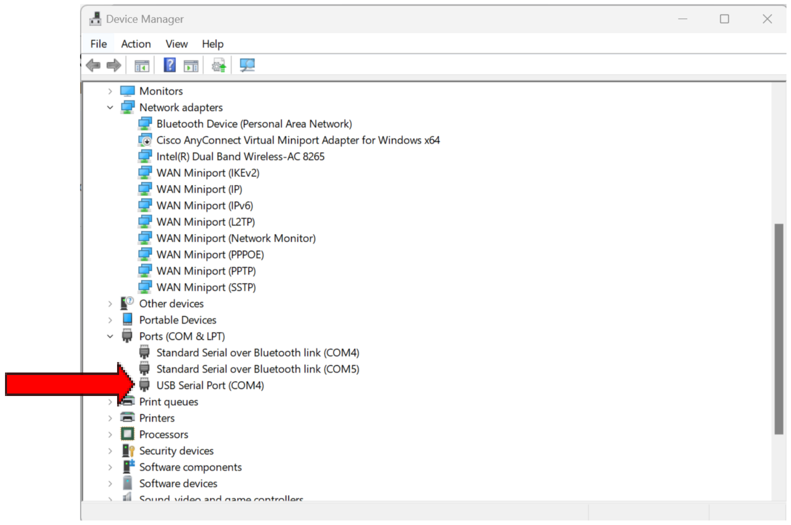
Then connect the USB-C cable on the lower side of the FPGA as shown below



Then connect the other side of the USB cable to the USB input to your PC



Then open up your device manager on your computer to find which serial communication port the device is connected to

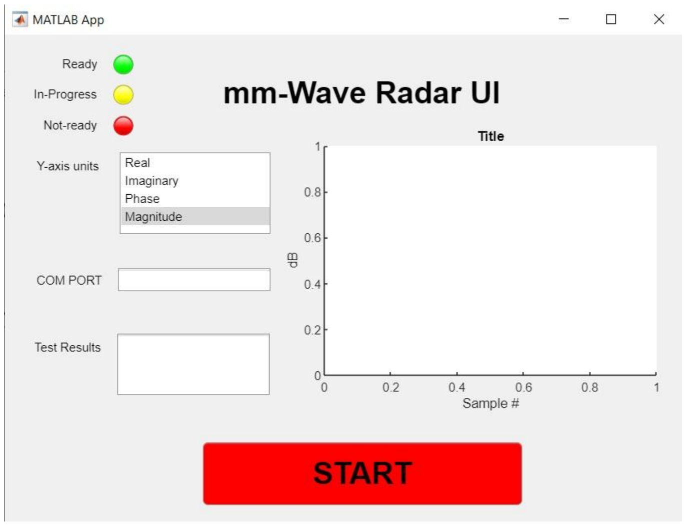


Device Manager

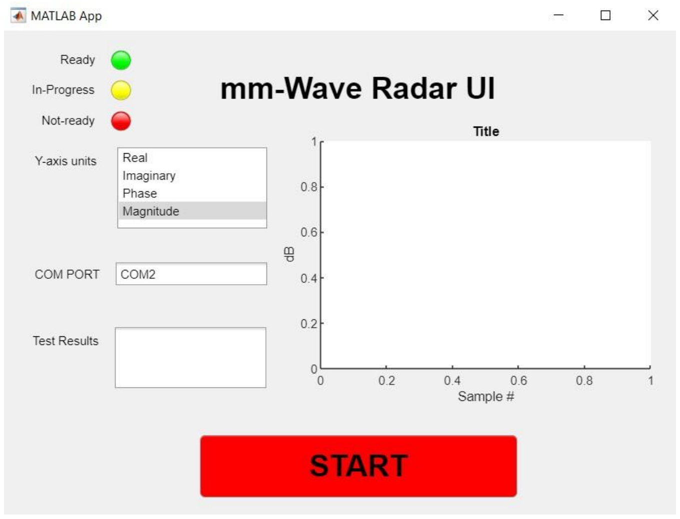
File Action View Help

- Monitors
- Network adapters
  - Bluetooth Device (Personal Area Network)
  - Cisco AnyConnect Virtual Miniport Adapter for Windows x64
  - Intel(R) Dual Band Wireless-AC 8265
  - WAN Miniport (IKEv2)
  - WAN Miniport (IP)
  - WAN Miniport (IPv6)
  - WAN Miniport (L2TP)
  - WAN Miniport (Network Monitor)
  - WAN Miniport (PPPOE)
  - WAN Miniport (PPTP)
  - WAN Miniport (SSTP)
- Other devices
- Portable Devices
- Ports (COM & LPT)
  - Standard Serial over Bluetooth link (COM4)
  - Standard Serial over Bluetooth link (COM5)
  - USB Serial Port (COM4)
- Print queues
- Printers
- Processors
- Security devices
- Software components
- Software devices

Once the connection is established, the mm-Wave Radar UI screen opens up.



In this screen, the green, yellow, and red denote the status of the process represented in the ‘start’ button.

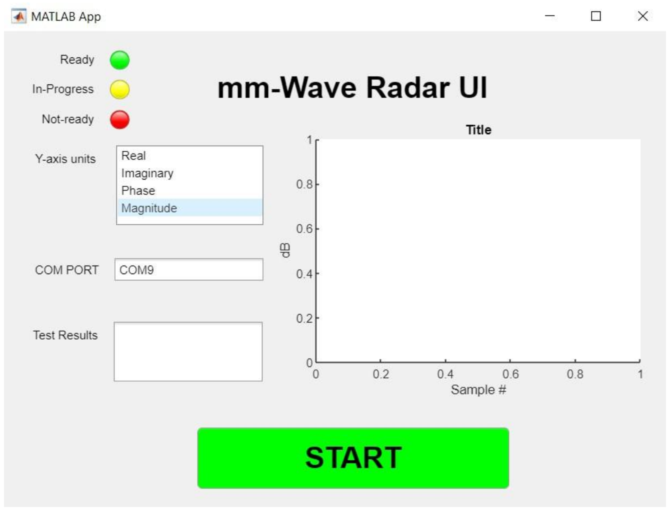


COM PORT COM2

The ‘Y-axis units’ selection window allows the user to plot different parameters of collected data, which are plotted in the right side of the screen.

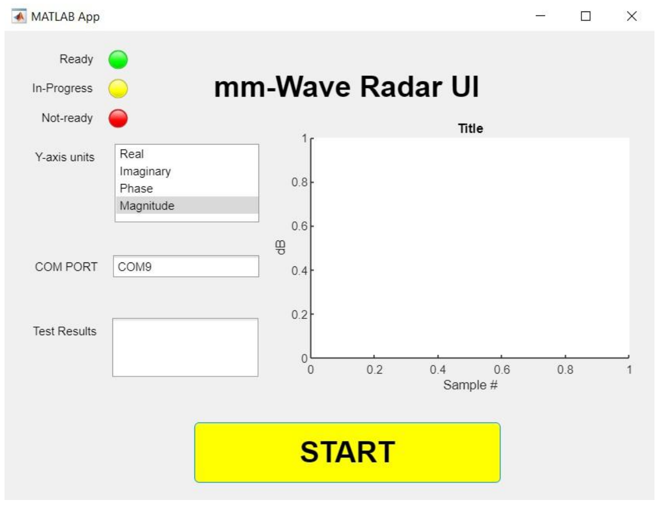
The window ‘Test Results’ gives a comment on the processed data.

The following screen is an example of established connection and shows that the system is ready to process data.

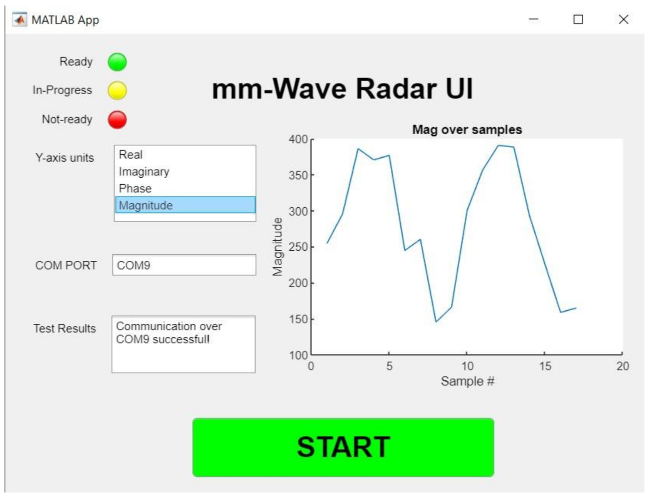


COM PORT COM9

In this case the ‘magnitude’ has been selected to be processed; and the screen shows that the system is ‘In-progress’.

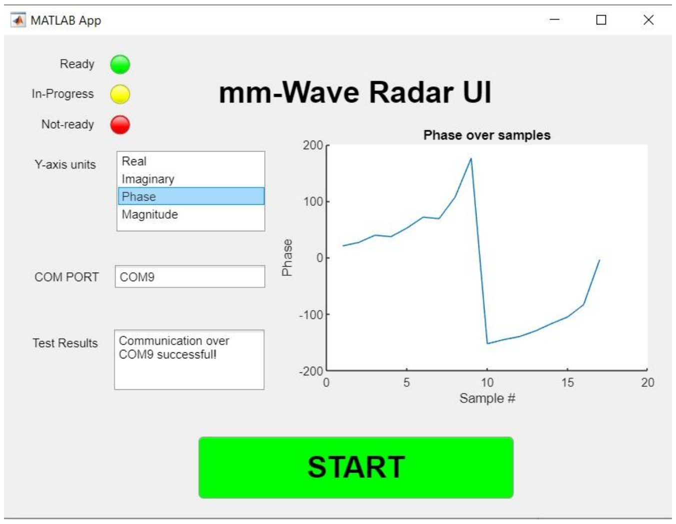


COM PORT COM9



COM9

Communication over COM9 successful!



COM9

Communication over COM9 successful!

GUI Setup (Ideal)

In a perfect world where the FPGA has been programmed to wait for a RX signal, also known as a handshake, the GUI would simply send a signal out of its TX port to the FPGA which would initiate a sequence, starting the IF circuit. The GUI had been implemented for this utility where the user only had to enter the correct COM port and the START button would turn green, the button sequence can only initiate and transmit over the serial port if the button is green. There are edge cases where the button is green and the user enters an incorrect but available serial port, this will be harmless as the Serial read and write functions will simply timeout. If the COM port is set to the correct Com port expect time for communication to vary depending on the amount of samples being sent from the FPGA, and the amount of time it takes for FPGA to sample and output the data, therefore, the timeout variable will be set to a larger than default (10 sec) amount.

GUI Setup (Actual)

We were not able to read Serial from the GUI with SPI, so the method we have to use if we want to utilize the GUI is to start the FPGA and the GUI at similar times and ensure that the DSP output to the GUI RX happens before the Serial Read function in MATLAB times out. This is obviously not a sustainable option but we implemented this to test the GUI and the DSP together, and as you can see in the last two pictures of the GUI above, it is receiving the correct DSP data, proving that DSP, HW, and UI are functional and interact properly.

Conclusion

The project successfully met its objectives by designing and implementing a functional IF circuit for the mm-wave imaging radar system. The comprehensive testing validated the circuit's capabilities, including accurate attenuation calculation and effective tracking of phase shifts. The utilization of FPGA technology and a user-friendly MATLAB-based UI demonstrated the project's technical proficiency. The system is now ready to collect phase and magnitude information from a reference signal.
Future work includes the implementation of SPI to communicate the FPGA with upconverter and downconverter, as well as the quantification of the measured error compared to the theoretical results.