S-BAND QPSK TRANSMITTER FOR POCKETQUBE SATELLITES

by

Jay Deorukhkar
A Thesis
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Master of Science
Computer Engineering

Committee:

_____	Dr. Peter W. Pachowicz, Thesis Director
_____	Dr. Jens-Peter Kaps, Committee Member
_____	Dr. Brian L. Mark, Committee Member
_____	Dr. Monson Hayes, Department Chair
_____	Dr. Kenneth S. Ball, Dean, College of Engineering and Computing

Date: _____ Fall Semester 2021
George Mason University
Fairfax, VA

S-Band QPSK Transmitter for PocketQube Satellites

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Jay Deorukhkar
Bachelor of Science
George Mason University, 2019

Director: Dr. Peter W. Pachowicz, Associate Professor
Department of Electrical and Computer Engineering

Fall Semester 2021
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to my family who has selflessly supported me in my entire academic career.

Acknowledgments

This thesis was possible because of the guidance and mentorship of Dr. Peter Pachowicz. From the conception of the idea to its finish, his advice helped me to stay on track throughout my thesis and my Masters journey.

I would also like to thank Dr. Jens-Peter Kaps and Dr. Brian Mark for being part of my committee and providing valuable inputs on crafting my thesis.

Table of Contents

Page

List of Tables vii
List of Figures viii
Abstract x
1 Introduction 1
2 Background 4
2.1 IQ Modulation 4
2.2 Phase Shift Keying 5
2.3 Pulse Shaping Filter 6
3 Functionality 9
3.1 QPSK Modulator 9
3.1.1 Splitter 9
3.1.2 Symbol Map 10
3.1.3 Pulse Shaping 11
4 Design 13
4.1 Proposed Architecture 13
4.1.1 MachXO2-1200HC FPGA 13
4.1.2 DAC121S101 DAC 14
4.1.3 ADF4351 VCO 14
4.1.4 AD8346 IQ Modulator 15
4.2 Testing Architecture 15
4.2.1 Artix-7 FPGA 15
4.3 Alternate Architecture 16
4.3.1 MSP430-FR2355 MCU 17
4.3.2 ADL5375 IQ Modulator 18
5 Implementation 19
5.1 Simulations 19
5.1.1 Pulse Shaping 19
5.1.2 QPSK Structure 23

5.2 Hardware 27
5.2.1 Proposed Architecture 28
5.2.2 Testing Architecture 29
5.2.3 Alternate Architecture 29
5.3 Software 30
5.3.1 Testing Architecture 32
5.3.2 Proposed Architecture 41
5.3.3 Alternate Architecture 42
6 Testing 49
6.1 Output Plots 49
6.1.1 Proposed Architecture 49
6.1.2 Testing Architecture 51
6.1.3 Alternate Architecture 52
7 Analysis 57
7.1 End-to-End System 57
8 Conclusion 62
Bibliography 64

List of Tables

Table		Page
1.1	Details for PocketQubes launched	3
5.1	Resource utilization for first iteration of software on Basys 3 FPGA	36
5.2	Timing report for first iteration of software on Basys 3 FPGA	36
5.3	Resource utilization for second iteration of software on Basys 3 FPGA	38
5.4	Timing report for second iteration of software on Basys 3 FPGA	39
5.5	Resource utilization for third iteration of software on Basys 3 FPGA	40
5.6	Resource utilization of software on TinyFPGA	43
5.7	Timing report of software on TinyFPGA	44

List of Figures

Figure		Page
1.1	List of modulation modes and frequency bands in SatNOGS	2
2.1	Structure of an IQ Modulator	4
2.2	Bits to symbols mapping for BPSK and QPSK modulation	6
2.3	Impulse and Frequency Response of filters with windowing functions	7
3.1	Full structure of a QPSK Modulator	9
3.2	Hardware implementation of a splitter	10
3.3	Hardware implementation for symbol mapping	11
3.4	Hardware implementation for pulse shaping	12
4.1	Proposed QPSK architecture	14
4.2	Testing architecture for QPSK	16
4.3	Alternate QPSK Architecture	17
5.1	Time and frequency domain of the input signal	20
5.2	Time and frequency domain of the output from windowed filtering	21
5.3	Impulse and frequency response of RC and RRC filters	22
5.4	Time and frequency domain of the output from RC and RRC filtering	23
5.5	Transmitter side of the QPSK baseband structure	24
5.6	Receiver side of the QPSK baseband structure	25
5.7	Time and frequency domain of the QPSK baseband output	26
5.8	Transmitter side for QPSK passband structure	27
5.9	Receiver side for QPSK passband structure	28
5.10	Time and frequency domain of the QPSK passband output	29
5.11	Hardware used in the proposed architecture	30
5.12	Hardware used in the testing architecture	31
5.13	Hardware used in the alternate architecture	32
5.14	Block diagram for first iteration of software on Basys 3 FPGA	33
5.15	Output waveform for first iteration of software on Basys 3 FPGA	35
5.16	Block diagram for second iteration of software on Basys 3 FPGA	37

5.17 Output waveform of software on TinyFPGA 42
5.18 Software flow chart for the MSP430 47
5.19 Memory usage of software on MSP430 48
6.1 Frequency spectrum of the carrier signal from the ADF4351 VCO 50
6.2 Time domain of the baseband signal from the TinyFPGA 51
6.3 Frequency domain of the baseband signal from the TinyFPGA 52
6.4 Input symbol decoding of the RF signal from the AD8346 IQ modulator . . 53
6.5 Random symbol decoding of RF signal from the AD8346 IQ modulator . . 54
6.6 Time domain of the baseband signal from the Basys 3 FPGA 54
6.7 Frequency domain of the baseband signal from the Basys 3 FPGA 55
6.8 Input symbol decoding of the RF signal from the AD8346 IQ modulator . . 55
6.9 Random symbol decoding of RF signal from the AD8346 IQ modulator . . 56
7.1 End-to-end setup of QPSK modulation chain 58
7.2 Flow graph for QPSK demodulator 59
7.3 Time domain and frequency domain plots for QPSK demodulator 60

Abstract

S-BAND QPSK TRANSMITTER FOR POCKETQUBE SATELLITES

Jay Deorukhkar

George Mason University, 2021

Thesis Director: Dr. Peter W. Pachowicz

The small satellite field has become popular amongst academia, amateur satellite (AMSAT) community, and commercial businesses due to the miniaturization of components and smaller form factors. Specifically, the PocketQube structure has gained attraction for its size and affordability of launch fees. However, the size constraint makes it difficult to generate power and limits the transmit power for downlink. Therefore, efficient data modulation is key to providing high data downlink rates. Also, the typical VHF and UHF frequency spectrum used for satellites is getting congested. Hence, the higher frequency bands such as S-band and X-band are gaining attraction and offer higher data bandwidth. To address both issues, an architecture to implement QPSK modulation for S-band operation is proposed. The design is focused on low-power PocketQubes and the implementation is targeted for academia and the AMSAT community.

Chapter 1: Introduction

The recent growth in small satellites has enabled academia, amateurs, and the commercial sector to design and launch payloads for specific applications. This can include complex payloads for earth observation, ship tracking, scientific experiments, interplanetary missions, and various others. The technology has also evolved to allow for smaller form factors ranging from multi-unit CubeSats (10 cm cube) to PocketQubes (5 cm cube) and even smaller.

The PocketQube form-factor is particularly challenging to work with, primarily due to its size. This smaller satellite bus results in a smaller surface area for solar cells, consequently generating less power. This puts a limit on the payload and subsystems since they will need to be designed with tighter power constraints. For the transmitters, the transmit power will also be limited for data downlink. This combination of small size and limited power drives the need for efficient data modulation structures, providing high data downlink rates for complex payloads.

Along with the challenges of the transmitter, the frequency spectrum used for satellites is getting congested and noisy. Typically, satellites use the VHF and UHF bands for uplink and downlink with ground stations. However, with the growth of small satellites and the need for higher data rates, more missions are shifting towards the use of higher frequency bands. Particularly, the S-band and X-band frequencies are much less crowded and offer higher bandwidth compared to VHF and UHF bands.

For the amateur satellite (AMSAT) community, there is a popular open-source ground station platform called SatNOGS. The idea is to allow amateurs to build inexpensive ground stations, collect data from satellites within their database, and upload the data for everyone to access. Currently, there are 585 satellites with 1139 satellite transmitters in their database [1]. They also provide various statistics, including a list of modulation modes and frequency

bands used on the satellites. Figure 1.1 shows the top 15 modes and top 5 frequency bands used.

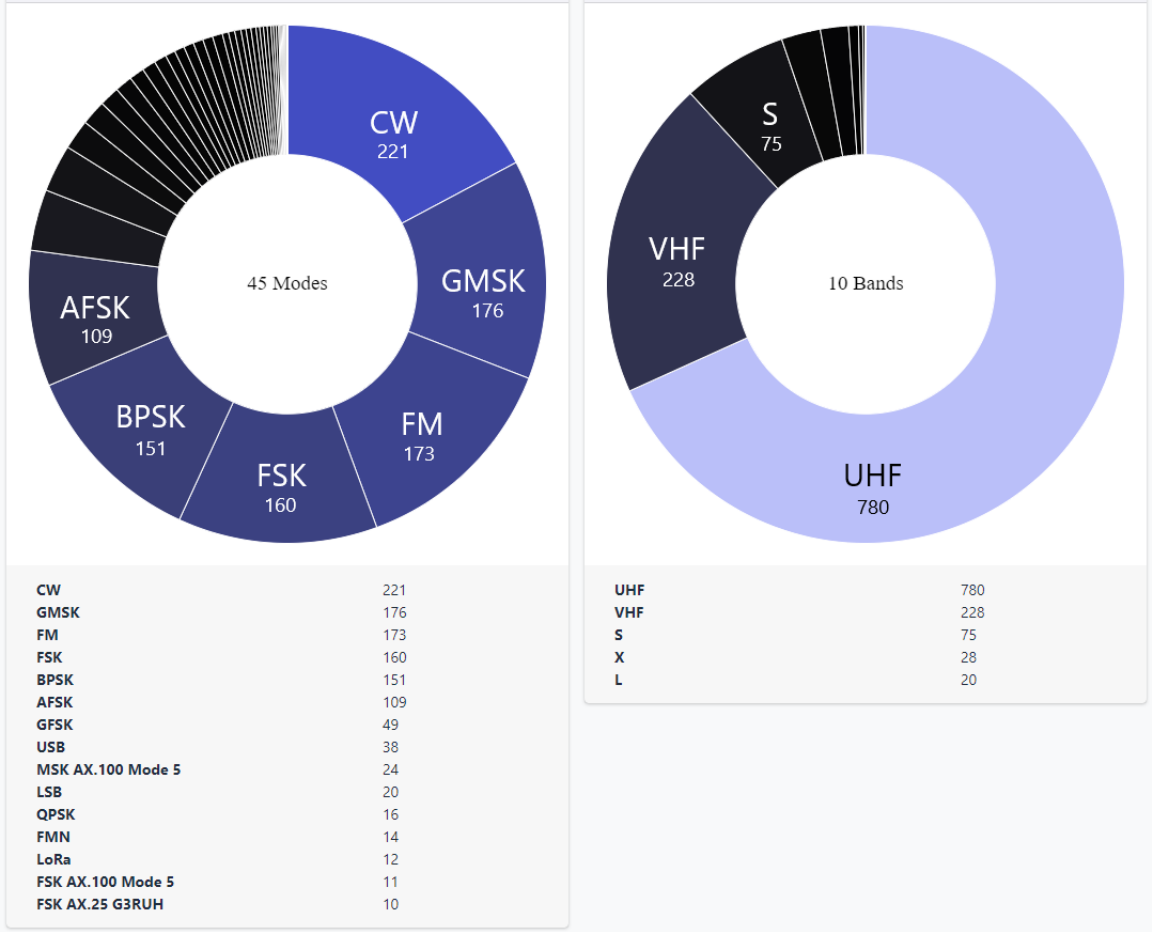

Figure 1.1: List of modulation modes and frequency bands in SatNOGS

The list of satellites in SatNOGS encompasses all form factors within the database. If we want to narrow it down to PocketQubes, we can use a well-known satellite repository called Gunters Space Page. This website has updated information on satellite and spacecraft

launches. From this website, we can get a list of launched PocketQubes [2]. By cross-searching each of these satellites with the SatNOGS database, table 1.1 lists the details about each satellite and its onboard transmitters.

Table 1.1: Details for PocketQubes launched

Size	Satellite	Org	Date	Launcher	Transmitters
1P	DIY 1 (ArduiQube)	DIYsatellite Group	22.03.2021	Soyuz-2-1a Fregat	CW (437.125 MHz), RTTY FSK (437.125 MHz)
	FossaSat 1	Fossa Systems	06.12.2019	Electron KS	BPSK 1k2 (436.7 MHz), LoRa (436.7 MHz)
	SMOG 1	BME University	22.03.2021	Soyuz-2-1a Fregat	CW (437.345 MHz), GMSK 1k25/2k5/5k/12k5 (437.345 MHz)
	SMOG-P (MO 105, Magyar-OSCAR 105)	BME University	06.12.2019	Electron KS	GMSK 12k5 (437.150 MHz)
	TRSI-Sat (TRSI 1)	ACME AtronOmatic / MyRadar	06.12.2019	Electron KS	FM (437.057 MHz), MFSK (437.075 MHz), BPSK 1k2 (437.075 MHz)
	Wren	STADOKO UG	21.11.2013	Dnepr	-
1.5P	S50SAT (Eagle 2, MO 76, Morehead-OSCAR 76)	Morehead State University	21.11.2013	Dnepr	GFSK 9k6 (437.507 MHz)
2P	ATL 1 (MO 106, Magyar-OSCAR 106)	ATL Ltd.	06.12.2019	Electron KS	GMSK 12k5 (437.175 MHz)
2.5P	Beakersat 1 (Eagle 1, SWEsat, MagPocketQube) → T-LogoQube	Morehead State University, Sonoma State University	21.11.2013	Dnepr	-
	QubeScout S1	University of Maryland (UMBC)	21.11.2013	Dnepr	-
3P	NOOR 1A (Unicorn 2B)	Stara Space	06.12.2019	Electron KS	-
	NOOR 1B (Unicorn 2C)	Stara Space	06.12.2019	Electron KS	-
6P	STECCO	GAUSS	22.03.2021	Soyuz-2-1a Fregat	FSK 9k6 (435.8 MHz)

At the time of writing this thesis, a few PocketQubes have used BPSK at 1,200 baud however, none have used QPSK or any of its variants for their modulation scheme. Also, note that the frequencies used are only in the UHF band. This masters thesis will provide an architecture to implement QPSK modulation for S-band operation on PocketQubes satellites. The following sections will discuss the functionality, design, implementation, testing, and analysis of the proposed architecture. This is primarily directed for use by academia and the AMSAT community.

Chapter 2: Background

2.1 IQ Modulation

IQ modulation is the process of converting baseband signals to RF signals using two sinusoids that are in quadrature. One of these sinusoids is the in-phase component with no phase shift while the other is the quadrature component with a 90 degree phase shift [3]. These are typically represented by sine and cosine waves with the same carrier frequency. Using these two components, the modulation process is shown in figure 2.1.

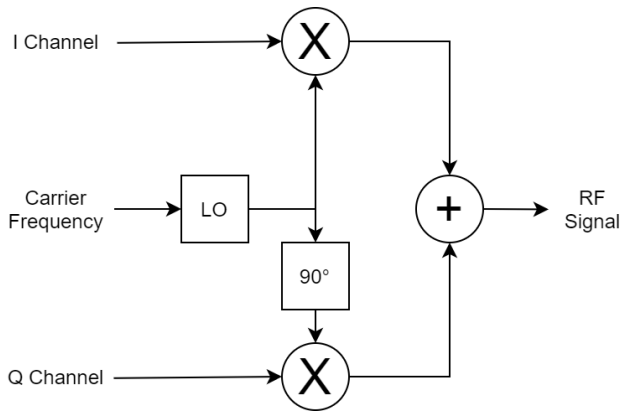

Figure 2.1: Structure of an IQ Modulator

The local oscillator generates a sinusoid signal at the carrier frequency that is used by the in-phase channel without any phase shift and the quadrature channel with a 90 degree phase shift. The baseband signal is split between the in-phase and quadrature channels, where it is multiplied by the two sinusoids and upconverted to the carrier frequency. The

resulting signals from the two channels are added together to form the IQ modulated RF signal [4].

The advantage of this method is that the baseband signal encodes the data only in its amplitude level while keeping its frequency and phase the same. Based on the variation in amplitude, any form of modulation (amplitude, frequency, and phase) can be achieved in the resulting RF signal [4]. Since any signal can be fully represented by its IQ components, the same process but flipped can be used to demodulate the RF signal.

2.2 Phase Shift Keying

A form of modulation where the data is represented by changes in the phase of a signal is known as Phase Shift Keying (PSK). As seen previously, by varying the amplitude baseband signal to specific levels, the resulting summation of the IQ channels after upconversion will result in a phase modulated RF signal. Depending on the number of bits used per amplitude level, a finite number of phase shifts can be generated by the output. Each phase shift is known as a symbol that is mapped to certain bits. In the case of Binary Phase Shift Keying (BPSK), two phases can be used for one data bit (0 and 180 degrees) [5]. In Quadrature Phase Shift Keying (QPSK), four phases can be used for two data bits (45, 135, 225, and 315 degrees). The mapping between symbols and bits is shown in the IQ constellation diagram in figure 2.2.

Each axis is used to show the symbols within the IQ components of the RF signal. In this case, the benefit of using QPSK over BPSK is with fixed bandwidth, QPSK can achieve twice the data rate over BPSK. This is due to BPSK being limited to encoding one bit for the in-phase channel, whereas QPSK can encode two bits by using both the in-phase and quadrature channels [4]. Also, with the added symbols in QPSK, the distance between symbols is still the same as BPSK, ensuring the same level of noise immunity as BPSK.

There are variations for QPSK such as Offset QPSK (OQPSK) that have certain benefits and Differential Phase Shift Keying (DPSK) to remove phase ambiguity at the receiver. By

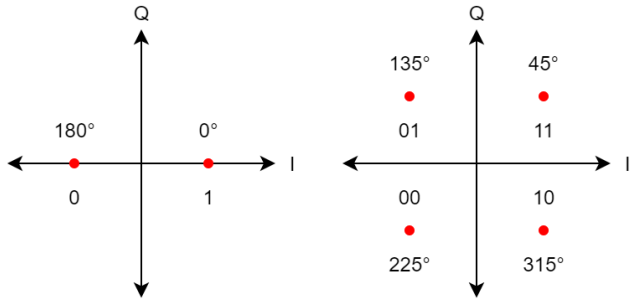

Figure 2.2: Bits to symbols mapping for BPSK and QPSK modulation

increasing the number of bits per symbol, higher-order PSK such as 8-PSK can be created for increased data throughput at the cost of receiver complexity.

2.3 Pulse Shaping Filter

To generate a baseband signal, the incoming data bits must be mapped to symbols with specific amplitude levels and fixed time intervals. Once the bits are mapped to symbols, the resulting baseband signal will contain a train of rectangular pulses with varying amplitude levels and instant transition jumps between symbols.

The issue is with the sharp transitions between symbols in the time domain which correspond to infinite bandwidth in the frequency domain. Therefore, a pulse shaping filter is needed to smooth the sharp transitions in the time domain, resulting in a band-limited signal in the frequency domain. Such filters are typically implemented digitally as Finite Impulse Response (FIR) filters.

The filter coefficients define the filter behavior (such as low-pass or high-pass), while the filter length and delay are fixed due to the impulse response being truncated by windowing functions [6]. The rectangular window is the simplest however, other windowing functions

offer better stopband attenuation. A comparison of a few windowing functions is shown in figure 2.3.

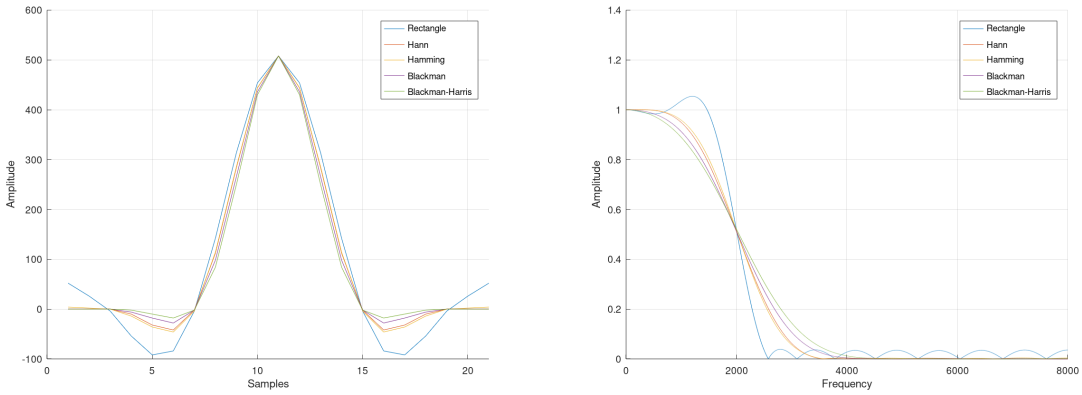

Figure 2.3: Impulse and Frequency Response of filters with windowing functions

Although certain windowing functions have better performance than the rectangular window, the impulse response of the pulses have tails that may interfere with neighboring pulses in a pulse train resulting in Inter-Symbol Interference (ISI). Therefore, an optimal pulse shape is one from the class of Nyquist pulses, such as a Raised Cosine (RC) pulse [7]. These pulses prevent ISI by ensuring that the pulse tails cross zero amplitude for every symbol period. Therefore, when the next pulse is transmitted, the energy of the previous pulse's tail is at zero during the sampling interval of the current pulse.

Typically along with the pulse shape filter at the transmitter, the receiver also needs to do some level of filtering to isolate the signal energy from the noise. Therefore, a more optimal pulse shape filter is one that has a relationship between the transmit and receive filter. In this case, a Square Root Raised Cosine (SRRC or just RRC) pulse can be used at the transmitter and at the receiver [7]. This pulse shape itself is not a Nyquist pulse, however, filtering at the transmitter and receiver with the same RRC pulse has the net

effect of an RC pulse where the received signal will not experience ISI. This operation of filtering at the receiver with the same RRC pulse as the transmitter is known as matched-filtering, where the receiver convolves the signal with the flipped version of the pulse shape. The advantage of matched-filtering is it maximizes the Signal-to-Noise Ratio (SNR) at the receiver.

Chapter 3: Functionality

3.1 QPSK Modulator

The entire QPSK modulation chain is shown in figure 3.1. It starts with the binary input data being split into even and odd bits for the IQ channels. Then, the bits are mapped to symbols based on the desired modulation scheme. Next, the symbols are passed into the pulse shaping filter to generate the baseband signals. Lastly, both signals are modulated using an IQ modulator to generate the RF signal at a specific carrier frequency [8].

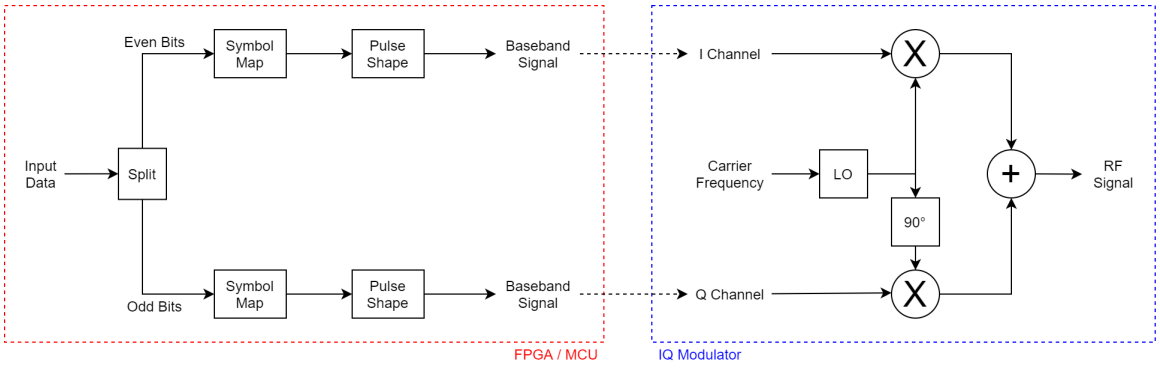

Figure 3.1: Full structure of a QPSK Modulator

3.1.1 Splitter

The splitter is used to separate the incoming data stream into two channels. The format of the split depends on the type of modulation and can affect the layout of the symbol map.

QPSK alternates each bit between the two channels to have the even bits pass to the I channel and the odd bits pass to the Q channel.

The splitting operation can be done in hardware or software depending on the implementation. In the case of software, this can be implemented as an if-statement. The decision can be a variable that alternates its state between two buffers for storing the data.

For a hardware implementation, the splitter can be viewed as a 1-to-2 Demultiplexer (Demux) as shown in figure 3.2. The input data is passed to one of the two channels based on the select signal. This signal could be the clock source that is used by the input data, where the value alternates every rising or falling edge of the clock signal. Another potential solution would be to feed the same input data to both channels but alternate the write enable pulse for the data buffers in each channel.

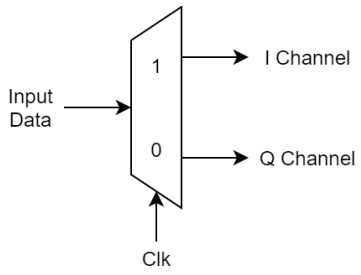

Figure 3.2: Hardware implementation of a splitter

3.1.2 Symbol Map

Symbol mapping is the stage where the incoming bits are mapped to specific symbols in the form of amplitude levels. The number of levels and their values depend on the modulation scheme used by the transmitter. For QPSK, the Non-Return-to-Zero (NRZ) line code is used to map binary one to positive one and binary zero to negative one.

There are various ways to implement the mapping of bits to symbols in both software

and hardware. Similar to the splitter, the mapping can be done using an if-statement, where it checks for a binary zero to convert it to a negative one while the binary one would stay the same.

The symbol mapping can be implemented in hardware as a 2-to-1 Multiplexer (Mux) as shown in figure 3.3, where the two inputs are positive one and negative one. The select signal is the binary input data that corresponds to the input levels. If the select value is zero, the negative one input is passed to the output and the positive one is passed if the select is one. An efficient alternative is to use a 2-bit signed integer where the binary input is inverted and concatenated with a binary one.

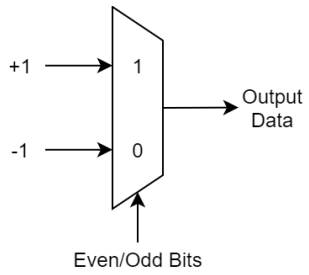

Figure 3.3: Hardware implementation for symbol mapping

3.1.3 Pulse Shaping

The pulse shaping operation is used to generate analog pulses with a specific time interval corresponding to each symbol. This is typically implemented as an FIR filter where the coefficients are the values from the impulse response of the desired filter shape. Before the symbol values can be passed to the FIR filter, they must be upsampled to a specific length for the filter to output a proper pulse shape. The upsampling operation is essentially having a fixed number of zeros between each symbol value.

An FIR filtering operation in its direct form is equivalent to multiple Multiply-And-Accumulate (MAC) operations. In software, this can be implemented using three arrays (an input buffer, output buffer, and coefficients buffer) and a for-loop. During each loop, the current symbol is multiplied by a coefficient value and added with the previous calculation.

In the case of hardware, the implementation consists of three storage units where two of them are Random Access Memories (RAMs for input and output buffers) and one is Read-Only Memory (ROM for coefficients buffer) as shown in figure 3.4. The filtering operation can be done either in parallel using multiple MAC and input delay blocks or sequentially using a single MAC and delay block.

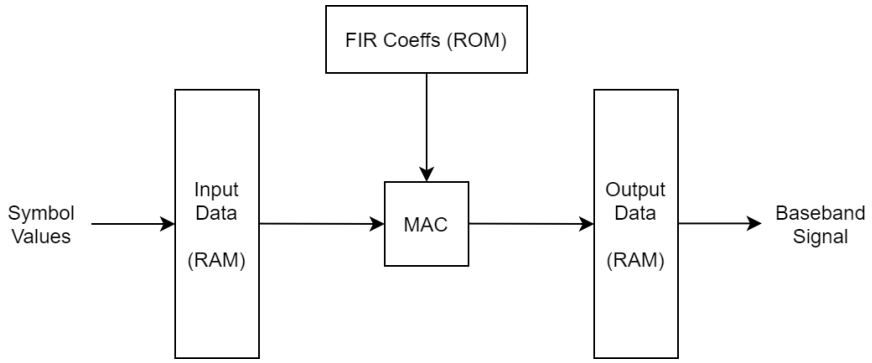

Figure 3.4: Hardware implementation for pulse shaping

Chapter 4: Design

4.1 Proposed Architecture

The proposed architecture for QPSK modulation is shown in figure 4.1. The input data is received by the MachXO2 Field Programmable Gate Array (FPGA) over Serial Peripheral Interface (SPI). The FPGA splits the incoming data into two channels, maps it to the appropriate symbols, and applies pulse shape filtering. The resulting baseband signals in each channel are sent out through a modified SPI interface where the two outputs correspond to a differential pair output. The first output is the positive signal (same as the baseband signal) and the second is the negative signal (an inverted version of the baseband signal).

Both set of SPI outputs are fed into the two PMOD-DA2 2-channel Digital-to-Analog Converter (DAC). This converts the differential digital baseband signals into differential analog baseband signals. These are connected to the differential inputs of the AD8348 IQ modulator. Additionally, the FPGA also controls the ADF4351 Voltage Controlled Oscillator (VCO) over SPI which sets the oscillator frequency to the desired carrier frequency. The VCO outputs the oscillator signal in a differential pair, which is passed into the IQ modulator. The result of the IQ modulator is the RF signal at the upconverted carrier frequency.

4.1.1 MachXO2-1200HC FPGA

This is a small-footprint and low-power FPGA with 1,200 logic cells, 10 kbits DRAM, 64 kbits block RAM, 64 kbits of flash memory, and clock speeds from 2.08 MHz to 133 MHz. The TinyFPGA board is a breakout board for this chip, allowing for fast prototyping. The main advantages of this chip are the 3.3 V operating voltage and 32-pin QFN chip package, making it a small power efficient chip for small satellite applications.

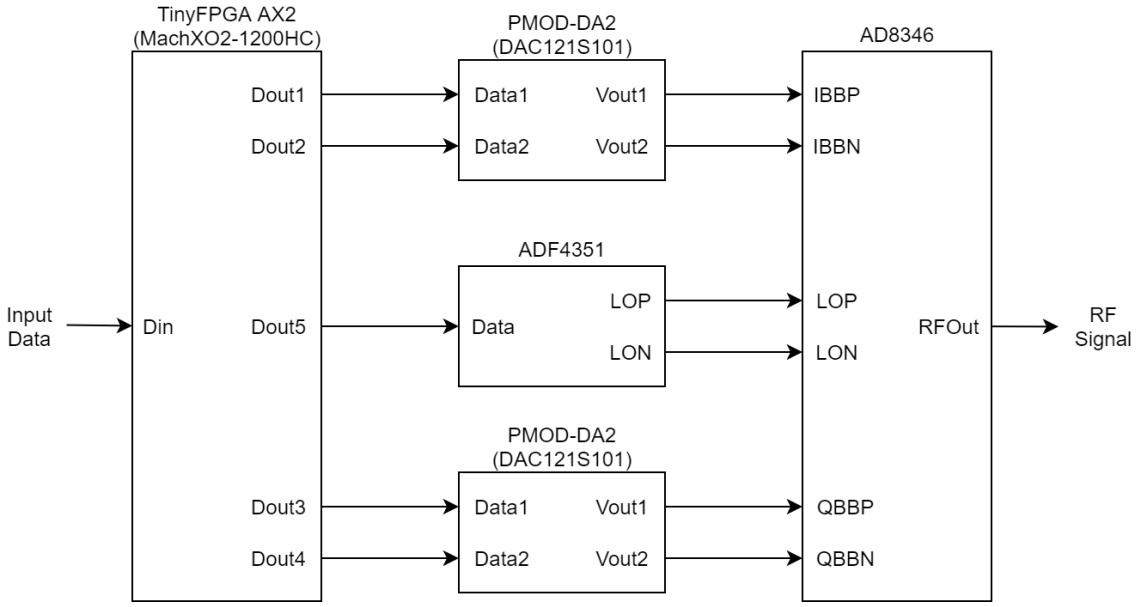

Figure 4.1: Proposed QPSK architecture

4.1.2 DAC121S101 DAC

Within each PMOD-DA2 board is the DAC121S101 12-bit 2-channel DAC. This 6-pin QFP chip has an operating voltage of 3.3 V and draws between 0.64 to 0.78 mW over a range of operating temperatures. The DAC has a modified SPI interface, where each channel has independent MOSI lines allowing for parallel outputs. For the SCK input clock, it can accept up to 30 MHz as the maximum output rate. The PMOD-DA2 board makes it convenient to test the chip by directly attaching it to the Basys 3 FGPA board.

4.1.3 ADF4351 VCO

The ADF4351 uses an internal Phase-Locked Loop (PLL) synthesizer along with a VCO to generate an oscillator signal and maintain the desired frequency. This 32-pin QFN chip can operate at 3.3 V and has a frequency range between 2.2 GHz and 4.4 GHz with

dividers to generate a minimum of 35 MHz. For stability in space environments, the VCO needs a Temperature Compensated Crystal Oscillator (TCXO) or Oven Controlled Crystal Oscillator (OCXO) with appropriate loop filters to remove harmonics generated by the crystal.

4.1.4 AD8346 IQ Modulator

As a Zero Intermediate Frequency (ZIF) modulator where the signal is directly converted to the RF frequency, the AD8346 is a small-footprint IQ modulator suitable for the PocketQube form factor. It can operate at 3.3 V with a low current draw of 45 mA for low-power applications. The generated RF signal can range from 0.8 GHz to 2.5GHz, which is ideal for the intended S-band frequency of 2.4 GHz. The 16-pin QFP chip has inputs that are differential pairs in order to reduce externally generated noise. The voltage bounds on the inputs are from 1.7v to 0.7v with a 1.2v bias. The single-ended RF output can be matched to a 50-ohm load, such as an S-band patch antenna.

4.2 Testing Architecture

For the testing architecture, the entire structure is the same as the proposed architecture except for the FPGA. In this case, the input data is received by the Artix-7 FPGA over SPI as shown in figure 4.2. The rest of the QPSK modulation chain is the exact same, resulting in the same RF signal as before.

4.2.1 Artix-7 FPGA

This chip is a very capable FPGA with 33,280 logic cells, 1,800 kbits block RAM, 90 DSP slices, a vast array of internal peripherals, and clock speeds of 100 MHz to 450 MHz. It is used on the Basys 3 development board for convenient testing of hardware logic. Also, a broad range of external peripherals can be directly plugged into the board (like the PMOD-DA2 DACs). However, the chip itself is not ideal for use on small satellites due to the 238-pin BGA footprint and the excess resources that are unused in this design.

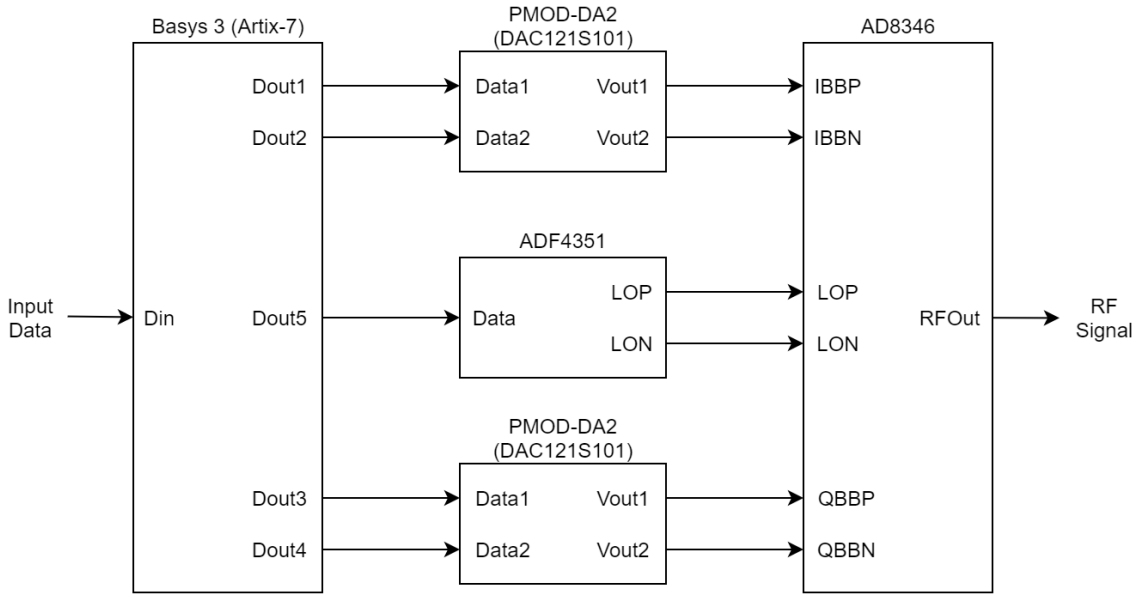

Figure 4.2: Testing architecture for QPSK

4.3 Alternate Architecture

An alternate architecture was considered for a simpler approach that did not use FPGAs but instead used a Microcontroller (MCU) with integrated DACs. In figure 4.3, the input data is received by the MSP430-FR2355 MCU over SPI. The incoming data is split into two channels, mapped to symbols, and pulse shape filtered by the MCU.

In this case, the MCU has internal 4-channel DACs which can convert the filtered baseband signal to the equivalent analog signal. The differential baseband signals are fed into the differential inputs of the IQ modulator. Along with this, the MCU has internal SPI peripherals for controlling the VCO oscillator's carrier frequency. The VCO output signal is connected the same as before to the IQ modulator. In this architecture, the ADL5375 IQ modulator is a different chip but has similar functionality and pinouts as the previous IQ modulator. The resulting RF signal is also the same as before.

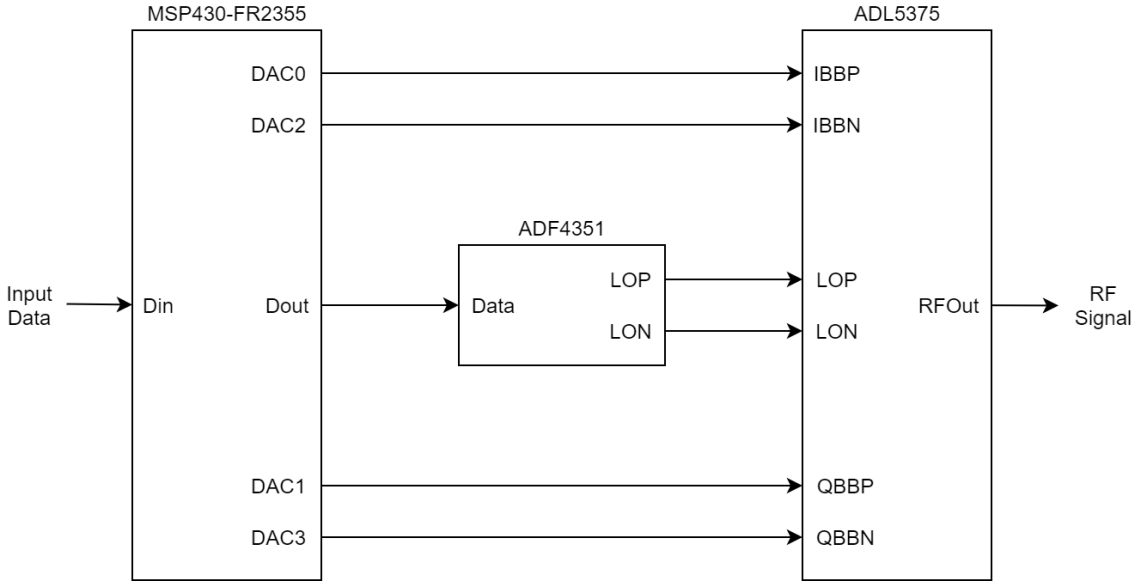

Figure 4.3: Alternate QPSK Architecture

4.3.1 MSP430-FR2355 MCU

The MSP430 is a popular 16-bit MCU with very low-power consumption and on-chip Ferroelectric RAM (FRAM). It is considered a legacy MCU for use in small satellites due to its flight history. The major advantage of FRAM is its nonmagnetic structure, which makes it radiation resistant. This can prevent Single-Event Upsets (SEUs) due to memory corruption (like bit flips), making it robust for space applications. This specific MSP430 can operate at clock speeds between 1 MHz to 24 MHz, has a 4-channel 12-bit DAC, and multiple SPI peripherals available. Although the chip has hardware multipliers, it lacks Digital Signal Processing (DSP) specific hardware and Direct Memory Access (DMA) which heavily impacts its performance when FIR filtering for the pulse shape.

4.3.2 ADL5375 IQ Modulator

The ADL5375 is another ZIF broadband IQ Modulator that directly converts the signal to RF frequency. It has a 5v operating voltage with a current draw of 203 mA and an output frequency range from 400 MHz to 6 GHz. Depending on the specific chip, the input voltage range can be from 0.25v to 0.75v with 0.5v bias (05 chip version) or 1.25v to 1.75v with 1.5v bias (15 chip version). Along with the high current draw and bus voltage, this 24-pin QFN chip also generates a discernible amount of heat making this an impractical and inefficient option for small satellite use.

Chapter 5: Implementation

5.1 Simulations

The following simulations demonstrate the operation of the pulse shape filter and the QPSK modulation chain. The pulse shape filtering involves testing various types of FIR Filters. For the filters that use windowing functions, the simulation was conducted in GNU Octave using the remez function along with the respective windowing functions [7]. With the RC and RRC filters, MATLAB was used due to the available rcosdesign function. The IQ modulator functionality and the demonstration of the entire QPSK chain are simulated using GNU Radio.

5.1.1 Pulse Shaping

When designing filters, various approaches can yield the desired filter shape. Earlier in the background section, figure 2.3 shows different windowing functions and their resulting passband, stopband, and cutoff frequency.

Window Based

These filters are based on the Parks-McClellan optimal FIR filter design method where the passband cutoff is 1.5 kHz, stopband cutoff is 2.5 kHz, stopband attenuation is 60 dB, and the desired filter length is 21 [6]. This length is determined by a formula that calculates the optimal length based on the passband cutoff, stopband cutoff, stopband attenuation, and sampling rate as shown in the reference. After generating the filter coefficients, the different windowing functions can be applied by multiplying the coefficients with the window weights.

Since the filter will be implemented on an FPGA, the floating-point coefficients need to be scaled and quantized into fixed point integers. This is done by multiplying them by

the number of bits used to store them in memory. After scaling, the floating-point values are rounded to store them as integers. In this case, the coefficients are scaled by 10 bits to store them as signed 10-bit integers. Therefore, the range of values is between 511 and -512 as shown in the impulse response of all the filters.

As an example, a binary input signal is generated with zeros and ones corresponding to a frequency of 1 kHz. This is evident in the frequency domain, where there are spikes at 1 kHz and the odd harmonics of 3 kHz due to sharp jumps between zero and one. It represents the signal after splitting into the even and odd bits. It is passed into the symbol mapper to apply NRZ coding, where binary ones become positive ones and binary zeros become negative ones. Then, the symbols are upsampled before entering the pulse shaping filter. The upsampling is set to 8 for padding each symbol with zeros. The resulting waveform is shown in figure 5.1.

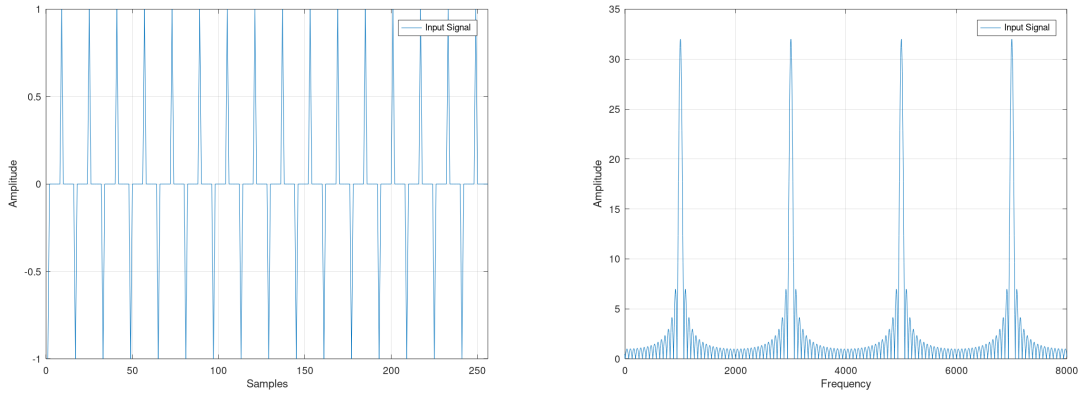

Figure 5.1: Time and frequency domain of the input signal

Filtering the input signal with each of the filters results in the output shown in figure 5.2. The output signals exhibit very similar time domain responses. In the frequency domain, the output responses are only marginally different.

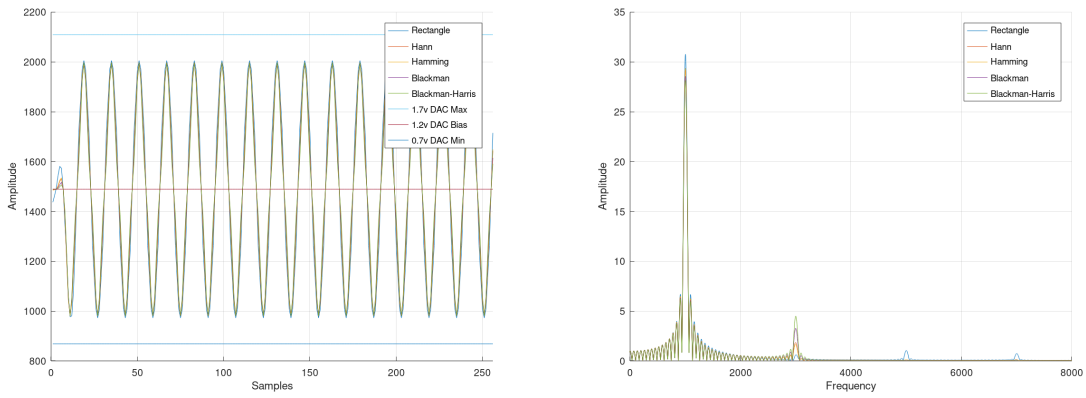

Figure 5.2: Time and frequency domain of the output from windowed filtering

After pulse shape filtering, the baseband signals are passed into the DAC to generate the equivalent analog signals. Notice in the time domain plot there are three flat lines. These represent the voltage bounds for the input of the DAC. The output of the pulse shape filter must be bounded between 1.7v and 0.7v with a bias of 1.2v. To ensure the output fits within these bounds, the baseband signals are shifted up from zero to the bias and the filter coefficients are scaled based on the output signal.

Typically, the output range of an FIR filter can be characterized by the sum of the absolute value of the coefficients. This would define the absolute maximum and minimum values for the output from the FIR filter, assuming a bounded input is fed into it. However, this characterization cannot be applied due to the upsampling of the input values. Therefore, another approach is to run multiple iterations of the FIR filter with random inputs. Based on 1000 iterations, the output was verified to stay between the DAC input bounds using the scaled filter coefficients.

Nyquist Pulse

The window-based filters are effective at removing out-of-band frequencies however, they do not solve the problem of ISI. As discussed earlier, ISI is the interference from previous and future pulse shape tails affecting the current pulse shape. This issue can be mitigated by filtering at the transmitter and matched-filtering at the receiver with an RRC pulse [7]. Figure 5.3 shows the impulse response and frequency response of an RC and RRC pulse. These are based on a few parameters such as the samples per symbol set to 8 and filter span of 5. Also, the roll-off factor is set to 0.35 which determines the extra bandwidth used by the filter. The rest of the scaling and quantizing operations are the same as before.

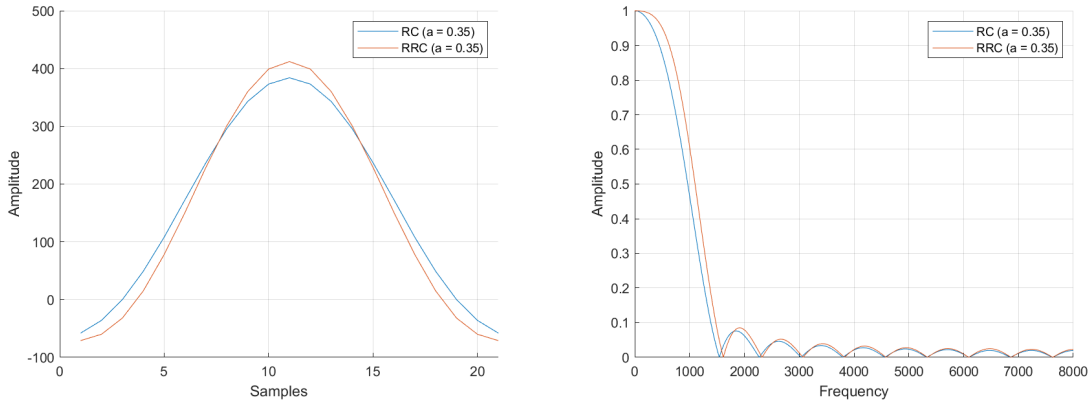

Figure 5.3: Impulse and frequency response of RC and RRC filters

By using the same NRZ coded and upsampled input signal, the output signal is shown in figure 5.4. The time domain result shows a slightly higher RRC filter output. This distinction can also be seen in the frequency domain as the RRC filter is taller than the RC filter. This is expected as the RC filter cutoff is before the RRC filter as seen in the frequency response, hence the larger attenuation.

As seen previously, the three flat lines correspond to the maximum, minimum, and

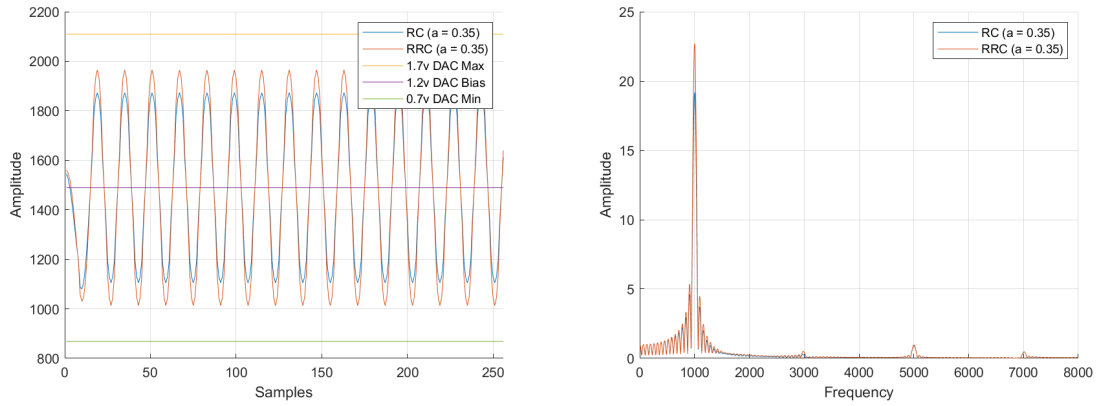

Figure 5.4: Time and frequency domain of the output from RC and RRC filtering

bias levels of the DAC input. The same techniques for shifting the output and scaling the filter coefficients were used. Likewise, the random input approach was executed with 1000 iterations for both filters, verifying that the response is within the DAC bounds.

5.1.2 QPSK Structure

To understand how the entire QPSK structure works, the following simulations were conducted. They cover the baseband version and the upconverted passband version with different blocks on the transmitter and receiver sides. In both cases, an ideal channel was assumed where no channel impairments or noise was added. For the baseband structure, the receiver performs symbol synchronization, while the passband signal assumes perfect synchronization between the transmitter and receiver.

Baseband Structure

For the baseband transmitter, figure 5.5 starts with the random input source to generate a stream of binary values. To visualize and compare this with the final output stream, the

binary values are converted to float values. The next step for the input stream is to take each binary value (which is a byte wide) and package it into one bit chunks so that each bit in the output is a data value. These values are passed into the constellation modulator that generates the QPSK symbols based on the constellation object. The block also does RRC filtering using the samples per symbol and excess bandwidth parameters, which are set to 8 and 0.35 respectively. The output from the block is a complex modulated baseband signal that is passed to the receiver. The throttle block is for the simulation to control the number of samples being processed [9].

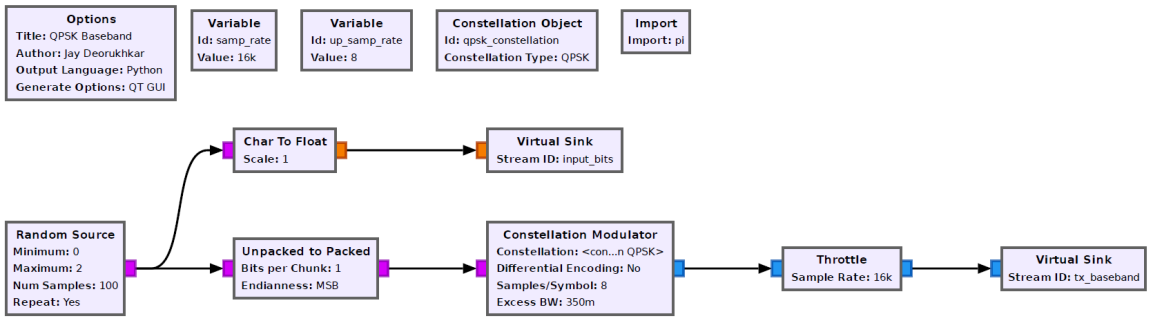

Figure 5.5: Transmitter side of the QPSK baseband structure

The receiver side shown in figure 5.6 passes the modulated baseband signal into an Automatic Gain Control (AGC) block. This will maintain the output signal amplitude between positive and negative one by scaling the input signal. Next, the signal goes to the symbol sync block which does four things. It matched-filters the input signal for optimal SNR, estimates and tracks the symbol rate based on a close initial estimate, does timing synchronization for sampling at the correct times, and decimates the signal to generate one sample per symbol. Most of the parameters are kept as default except for the Timing Error Detector (TED) that is set to "Gardner", the samples per symbol set to 8, and a loop bandwidth of 0.0628 [10]. The synchronized complex baseband signal is passed

into the constellation decoder block that uses the same QPSK constellation object as the transmitter. The output is a stream of bytes that correspond to the symbol values between zero and three. These values are unpacked into chunks of two bits and concatenated so that all the bits in the output are data bits. Finally, the output stream is converted to floats to compare with the input stream [9].

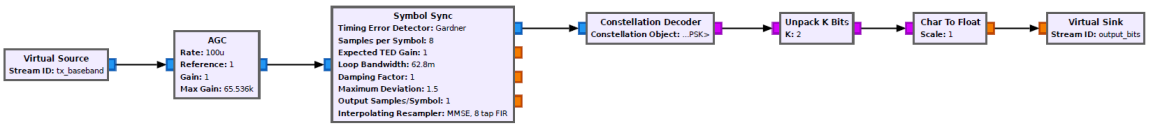

Figure 5.6: Receiver side of the QPSK baseband structure

Looking at the time domain plot, the output stream and the input stream have the same data values as shown by the overlapping lines in figure 5.7. This confirms that the transmitter correctly propagates the symbols to the receiver. Originally, the output stream was delayed by some number of samples due to the pulse shape filtering (half the filter length), the matched-filtering, and the synchronization loops. This was corrected by using a delay block (which is not shown) on the input stream to align it with the output stream. For the frequency domain, a similar conclusion can be drawn as the output spectrum accurately overlaps the input spectrum.

Passband Structure

The passband structure takes a different approach from the baseband structure to understand more about the IQ modulation process. Figure 5.8 shows the same random input source being used at the start for the input data. The binary data is converted into float values, multiplied by 2 and subtracted by 1, which accomplishes the task of NRZ coding. The stream of positive and negative ones are de-interleaved to alternate the values between

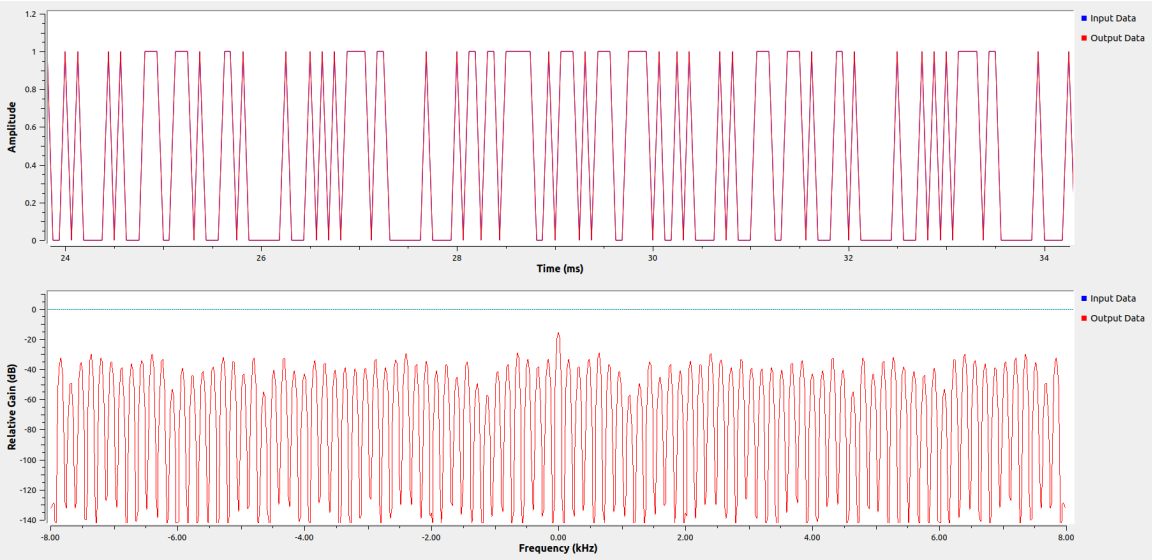

Figure 5.7: Time and frequency domain of the QPSK baseband output

the IQ channels. In both branches, the values are pulse shape filtered with the same RRC coefficients generated in the earlier simulation section. Next, the pulse shaped baseband signal is upconverted to the carrier frequency of 4 kHz with multiplication. The input to both multipliers is a complex cosine signal where the real and imaginary values are converted to float by splitting them into two channels. The real channel is fed to the in-phase multiplier while the imaginary channel is fed to the quadrature multiplier. Lastly, both channels are added together and the RF signal is sent to the receiver side.

Most of the blocks in the receiver are the same as the transmitter but in the reverse order as seen in figure 5.9. At the start, the RF signal is passed into two multipliers for downconversion to baseband. The same complex cosine signal with 4 kHz carrier frequency is converted to float and passed to both multipliers. The resulting signals are low-pass filtered with a cutoff frequency of 1 kHz and a Hamming window. Next, the baseband signal is matched-filtered with the same RRC filter coefficients and decimated by 8 to have

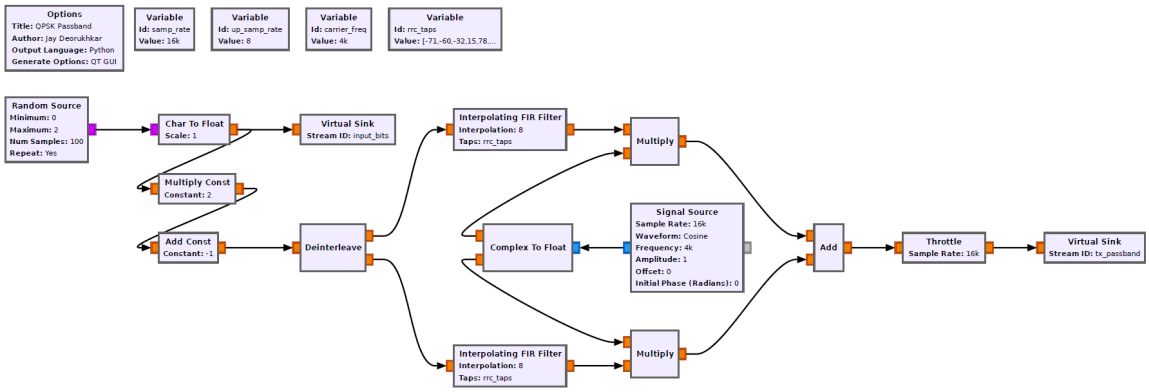

Figure 5.8: Transmitter side for QPSK passband structure

one sample per symbol. At this point, the signal is ready to be sampled and quantized by the threshold block. With the threshold set to zero, the block will output a positive one for values greater than zero and a zero for values less than zero. This stream of binary values is interleaved to form a single stream of binary values. Finally, the values are converted to float to visualize with the input stream.

From figure 5.10, the output data stream is the same as the input data stream as seen in the time domain plot. Also, the input stream was delayed by a different number of samples, which is justified by the different passband simulation structure as compared to the baseband version. Likewise, the frequency domain shows the two streams precisely overlapping each other.

5.2 Hardware

In the following sections, the hardware used in the three architectures is shown. The annotations highlight the corresponding components from the architecture diagrams along with the main inputs and outputs.

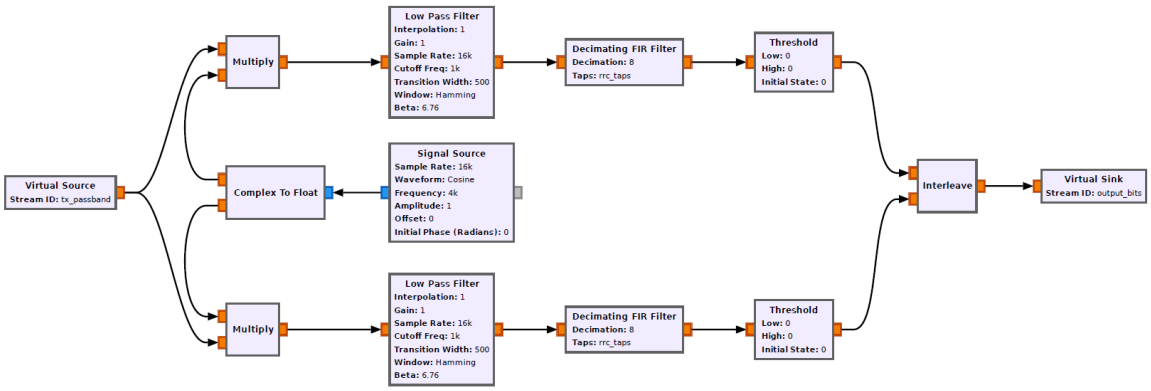

Figure 5.9: Receiver side for QPSK passband structure

5.2.1 Proposed Architecture

In this architecture, we start with the input data entering the TinyFPGA over SPI as shown in figure 5.11. Next, the data is sent to the two PMOD-DA2 DAC boards over a modified SPI interface, where two MOSI lines are used for the two independent output channels. In this case, the input SPI side powers the TinyFPGA and DAC boards. Then, the four analog baseband signals (in differential pairs) are passed into the AD8346 IQ modulator along with the local oscillator (LO) signal from the ADF4351 VCO. The IQ modulator evaluation board is set up as a single-ended input for the LO signal, whereas typically it would be a differential pair as well. Lastly, the RF signal is generated as a single-ended 50 ohm output.

The board on the bottom left is an evaluation board for the ADF4351 VCO as well as the ADL5375 IQ modulator (which is covered in the later section). The boxed portion of the board is the VCO and the positive LO output is connected to the AD8346 IQ modulator. Note that the architecture shows the ADF4351 being connected to the FPGA over SPI however, it is controlled over USB with a specific program in this setup. This allows for

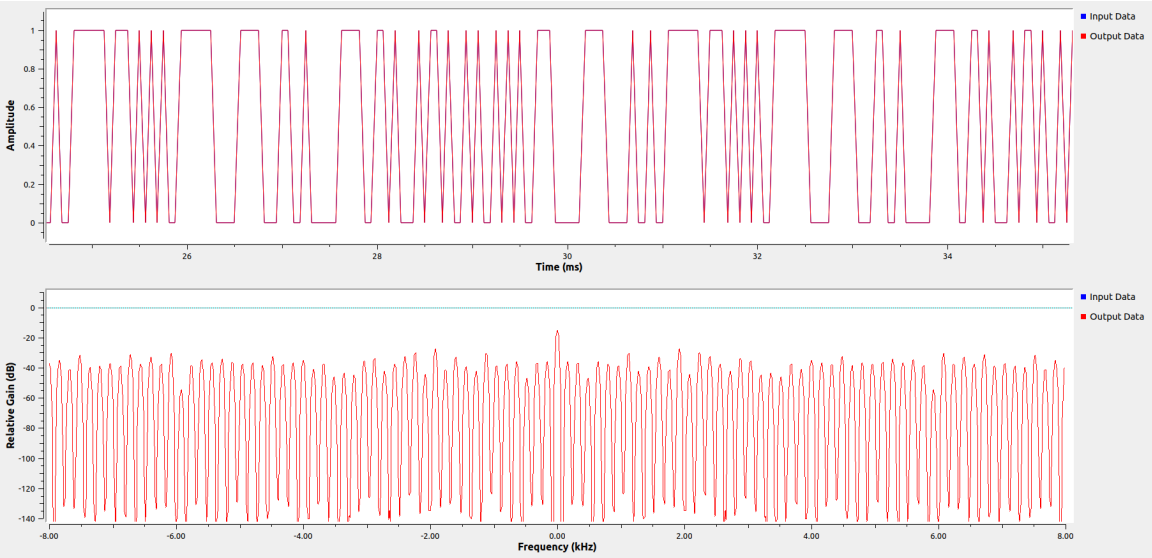

Figure 5.10: Time and frequency domain of the QPSK passband output

setting the carrier frequency and other functions on the computer with ease.

5.2.2 Testing Architecture

The testing architecture in figure 5.12 is very similar to the previous one, except the TinyFPGA is replaced by the Basys 3 board. It is also powered and controlled over USB, hence the DACs are also powered by the FPGA board. The interfaces and rest of the chain are identical to the earlier architecture.

5.2.3 Alternate Architecture

For the alternate architecture, the setup shown in figure 5.13 is fairly different from the previous two architectures. Firstly, the input data passes through SPI into the MSP430 MCU (which is powered and programmed over USB). Since this particular MSP430 has a 4-channel integrated DAC, the output is the baseband analog signal which goes into the

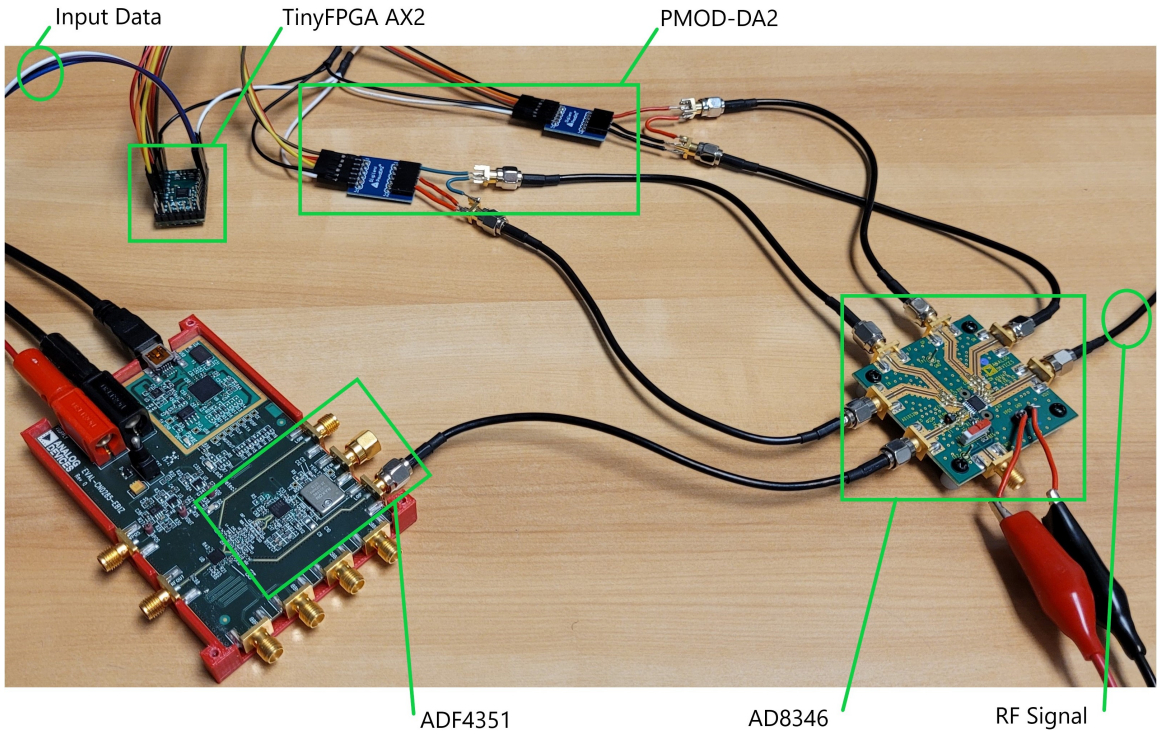

Figure 5.11: Hardware used in the proposed architecture

ADL5375 IQ modulator to generate the output RF signal. The chip itself can be seen in the top portion of the evaluation board along with the ADF4351 VCO at the bottom. The differential LO outputs from the VCO are internally connected to the IQ modulator through a series of loop filters for reducing the oscillator harmonics.

5.3 Software

The software that is running on the FPGA and MCU boards within each architecture is discussed in the sections below. The goal of the software is to implement functionality for generating the baseband signals as seen in the functionality section. One thing to note

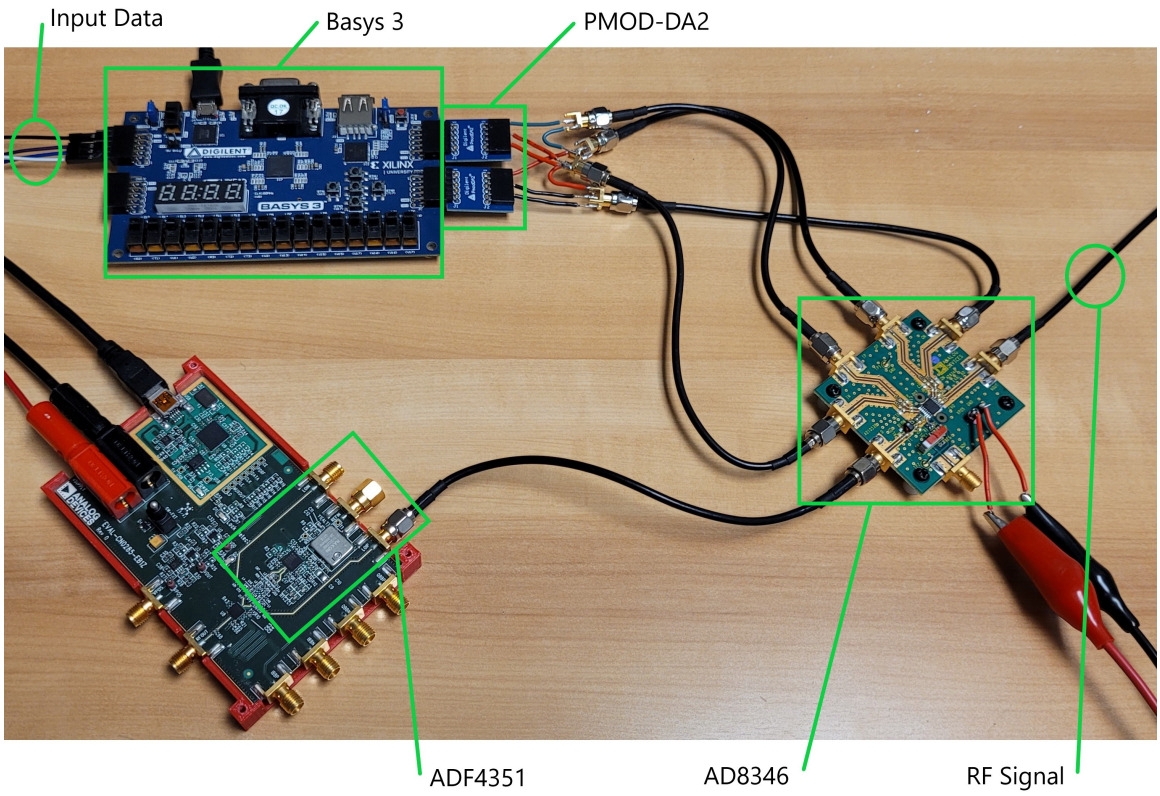

Figure 5.12: Hardware used in the testing architecture

for the FPGAs, the Very-High-Speed-Integrated-Circuit Hardware Description Language (VHDL) code was written as generically as possible for the logic elements and memory inferring to be transferable between the Basys 3 board and the TinyFPGA board. With modifications to the pinouts and clock speed settings, using the same VHDL code helped save time when testing both architectures.

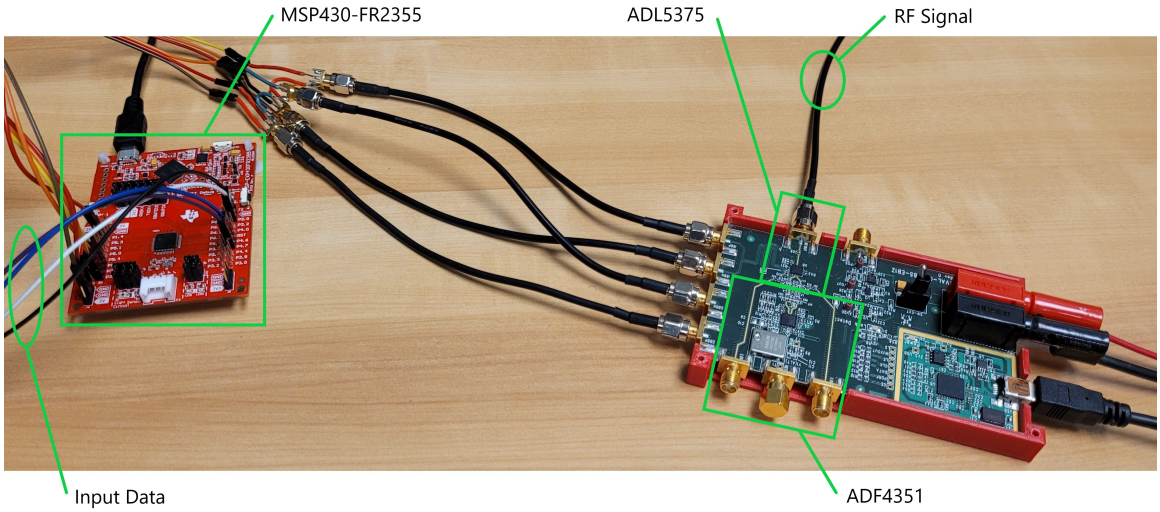

Figure 5.13: Hardware used in the alternate architecture

5.3.1 Testing Architecture

Starting with the testing architecture, the Basys 3 board had three iterations of software where each one is more optimized with lower resource utilization:

1st Iteration

With the first iteration, the idea was to build logic blocks that performed each function independently. This is shown in figure 5.14 where the input SPI block takes the input data and passes it to FIR filter blocks in each channel. Then, the filtered data is sent to the output SPI blocks for the DACs to generate the baseband signal. The FIR filter and SPI output blocks in each channel are identical as they were instantiated using the same VHDL code entities in the top-level file. This results in duplicate logic hardware such as input/output buffers which significantly increases resource utilization.

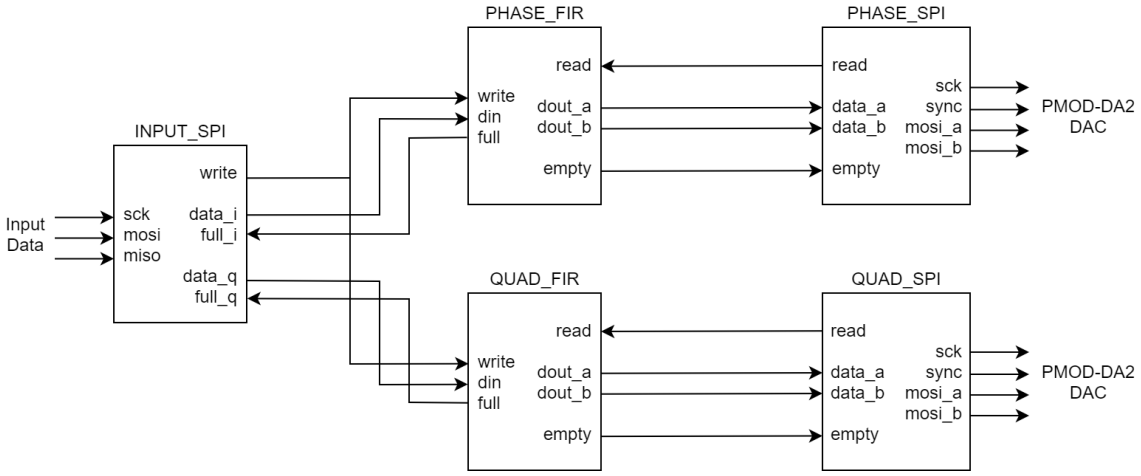

Figure 5.14: Block diagram for first iteration of software on Basys 3 FPGA

For the input SPI block, the SCK and MOSI inputs are double-flopped to prevent meta-stability issues. Anytime synchronous signals from an external device are sampled by an FPGA, the inputs should pass through two flip-flops to synchronize the signals with the FPGA clock. For the MISO output, it uses the full flags from the FIR filter blocks and passes one of them to the input device. In this case, the input device operates in master SPI mode with the MISO signal functioning as an on/off switch, indicating when to start/stop sending data. Another function of this block is to split the incoming data stream into two channels. This can be done with a Mux however, the Mux alone will induce an offset of one bit between the two channels as it flips between them. Additionally, a delay block is used on one of the channels to synchronize the data writes in both channels.

The FIR filter blocks are the most resource intensive blocks due to the filtering logic and data buffers. The filter coefficients are the same 21-element RRC coefficients seen previously and are stored as signed 10-bit integers in a ROM memory block. The input and output buffers are 32-element circular buffers where there is a single input buffer with two output

buffers for the positive and negative differential outputs. Both the length of the FIR filter and input/output buffers are declared as a generic value and can be adjusted. For the input data, the values are NRZ coded and upsampled by 8 before being stored in the input buffer as 2-bit signed integers. The upsampling rate is also a generic value that can be adjusted depending on the FIR filter requirements. To minimize resource usage, the FIR filtering operation is done on a single cycle basis, where each multiply and addition operation takes a clock cycle. Since the filter length is small the performance impact is not as significant, as opposed to the notable increase of resource utilization with a parallel filter structure. Lastly, the 12-bit signed integer results are stored in differential output buffers as 16-bit values. The leading 4 bits are used by the DACs for specific functions and are set to zeros. These differential outputs in each channel are sent bit by bit to the output SPI block.

In both channels, the SPI output block passes the filtered data to the DACs by generating a clock and chip-select signal. There are two clocks created by this block, a data clock and an SPI clock. Both are generic values that can be changed however, they are initially set to 8 kHz and 500 kHz respectively. The data clock can be viewed as the output sample rate, where a new value is sent to the DAC for updating the output waveform whereas the SPI clock is used to send each bit of the new value. Therefore, the differential outputs per channel are sent to the DAC through this SPI block to generate the baseband output signal.

The simulation waveform shown in figure 5.15 shows the input and output SPI signals as well as the expected baseband signals from the DACs. For this simulation, the main clock frequency is set to 100 MHz which is the intended operating frequency of the Basys 3 board. First, the testbench file transmits two data bytes (which are 0x33) as seen at the top. The output SPI signals in the middle show the filtered values being sent back to the testbench file. Lastly, the baseband signals at the bottom correspond to the output data received from the SPI output. The two markers in the middle highlight the maximum and minimum values of the wave, which are shown on the left sidebar. These values are verified with the RRC filter simulation to be within the bounds of the IQ modulator input. The markers also show the time between the two symbols at the bottom. In this case, the

data rate was set to 8 kHz with an upsampling rate of 8, hence the time interval is 1 ms corresponding to 1 kHz symbol rate.

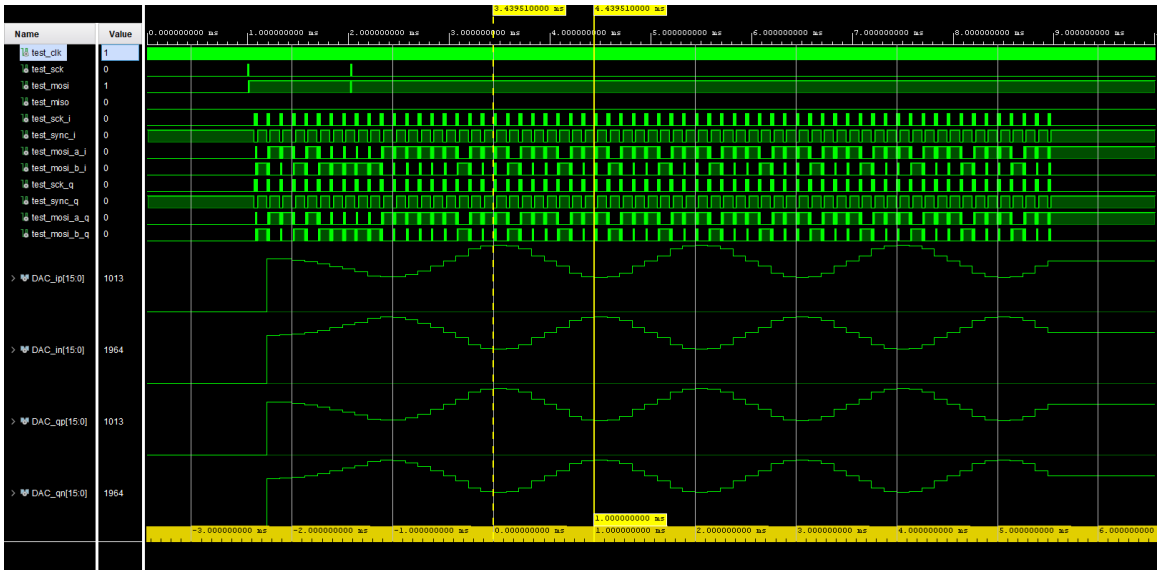

Figure 5.15: Output waveform for first iteration of software on Basys 3 FPGA

The resource utilization for this software iteration is expected to be the highest due to the FIR filter and SPI output blocks being duplicated for both channels. Table 5.1 shows the post-implementation resource utilization for each block file. The FIR filter blocks take the most resources (as expected) while the SPI input and output blocks take much less. The memory implemented in this case is Distributed RAM using Look-Up Tables (LUTs) due to the small buffer sizes and faster speed compared to block ram. One key point is that all the registers are clock-edge driven to ensure the internal signals are properly synchronized to clock edges. This generally results in no latches being used in the implementation, which is highly desirable due to their asynchronous nature causing signals to potentially miss clock edges.

Looking at the timing report in table 5.2, it shows that there are no failing endpoints

Table 5.1: Resource utilization for first iteration of software on Basys 3 FPGA

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Bonded IOB (106)	BUFGCTRL (32)
N top	345	220	119	305	40	12	1
INPHASE_FIR (fir_buf)	149	73	50	129	20	0	0
INPHASE_SPI (spi_output)	20	32	13	20	0	0	0
INPUT_SPI (spi_input)	8	11	5	8	0	0	0
QUADRAT_FIR (fir_buf_0)	149	72	46	129	20	0	0
QUADRAT_SPI (spi_output_1)	21	32	13	21	0	0	0

and the timing constraints are met. The positive value for the Worst Negative Slack (WNS) means that the time it takes for the longest path in the implemented logic still fits within a clock period. Initially when designing the FIR filter block, the multiply and add logic was entirely parallel. This resulted in a large negative value for the WNS, indicating that the filtering logic cannot fit within the clock period. Hence, this single-cycle filtering approach offers a more desirable WNS time.

Table 5.2: Timing report for first iteration of software on Basys 3 FPGA

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.662 ns	Worst Hold Slack (WHS):	0.088 ns	Worst Pulse Width Slack (WPWS):	3.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	974	Total Number of Endpoints:	974	Total Number of Endpoints:	301

2nd Iteration

This iteration is designed to reduce the resources used in the first iteration. To achieve this, the duplicate FIR filter and SPI output blocks in both channels are merged into single blocks as shown in figure 5.16. Therefore, most of the logic elements that were duplicated are now reduced however, certain elements are still the same (such as the buffers). The SPI input block is the same as the previous iteration and the overall functionality is also unchanged.

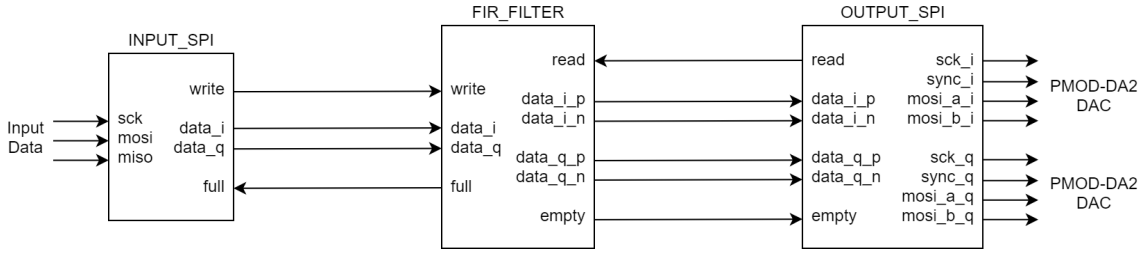

Figure 5.16: Block diagram for second iteration of software on Basys 3 FPGA

With the same SPI input block, the FIR filter block is re-designed to contain the logic for both channels. The input/output signals and data buffers are the same however, the control logic is now shared between the buffers. The filtering operation is also the same and each filter cycle does the multiply and addition for both channels. The result is the differential output for each channel, which will be passed to the SPI output block.

Likewise, the SPI output blocks are combined to form a single block with the same input/output signals. Also, both channels share the same 8 kHz data clock and 500 kHz SPI clock. For the SCK and SYNC outputs, they are independent signals that go to each DAC separately. However, an alternative could have only one set of SCK and SYNC pins while externally connecting them to both DACs.

In this case, the simulation waveform is identical to figure 5.15. The baseband output

signal expected from the DAC is within the IQ modulator bounds and the symbol rate is 1 kHz.

After the optimizations, the benefits can be seen in table 5.3. The resource utilization shows a significant reduction in the LUTs and registers used in this implementation. The number of LUTs is reduced by 78, registers by 73, and the overall slices by 31. Since this optimization was focused on the logic elements of the FIR filter and SPI output blocks, the data buffers remain the same, hence the number of LUT memory elements is the same as before.

Table 5.3: Resource utilization for second iteration of software on Basys 3 FPGA

Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Bonded IOB (106)	BUFGCTRL (32)
top	267	147	88	227	40	12	1
FIR_FILTER (filter)	242	102	76	202	40	0	0
SPI_INPUT (input)	6	11	3	6	0	0	0
SPI_OUTPUT (output)	19	34	11	19	0	0	0

Similar to the previous implementation, the timing report in table 5.4 shows no failing endpoints with the timing constraints being met. Although the WNS is reduced by a very small amount, the Worst Hold Slack (WHS) has a noticeable increase. This gives a slightly larger margin when modifying the design and working to optimize it further.

3rd Iteration

Lastly, this implementation optimizes a step further by reducing the output buffers in the FIR filter block. The block diagram is the same as the previous iteration as shown in figure 5.16. Apart from the FIR filter block, the SPI input and SPI output blocks are unaltered.

In all the past iterations, the FIR filter block needs one input buffer and two output buffers per channel. The two output buffers are for storing the differential output of the

Table 5.4: Timing report for second iteration of software on Basys 3 FPGA

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	1.655 ns	Worst Hold Slack (WHS):	0.127 ns	Worst Pulse Width Slack (WPWS):	3.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	823	Total Number of Endpoints:	823	Total Number of Endpoints:	228

baseband signals. However, since the output of the FIR filter is a signed value between the positive and negative bounds, the negative part of the differential signal can be generated by inverting the positive part. Then, both positive and negative parts can be shifted to the desired bias voltage to conform to the input requirements of the IQ modulator.

The resulting simulation is analogous to the first iteration as seen in figure 5.15. Since the main functionality is still the same, the differential baseband signals have a 1 kHz symbol rate as they are passed into the IQ modulator from the DACs.

From the resource utilization shown in table 5.5, there are a few changes compared to the previous iterations. First, the LUTs have decreased by 14 and registers by 10 whereas the overall slices increased by 1. Although additional 12 Muxes were added due to the changes in the FIR filter block, the LUT memory elements decreased by 16. Overall, this optimization results in a net benefit of reduced resource utilization.

In this iteration, the timing report is the same as table 5.4 from the second iteration. This would mean that the removal of the two output buffers does not affect the critical path timing.

Optimizations and Structure

Other optimizations were considered but not implemented due to the cost outweighing the performance. Typically with FIR filters, the first half of the coefficients are the same as the

Table 5.5: Resource utilization for third iteration of software on Basys 3 FPGA

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Slice (8150)	LUT as Logic (20800)	LUT as Memory (9600)	Bonded IOB (106)	BUFGCTRL (32)
N top	253	137	8	4	89	229	24	12	1
FIR_FILTER (filter)	228	92	8	4	76	204	24	0	0
SPI_INPUT (input)	6	11	0	0	3	6	0	0	0
SPI_OUTPUT (output)	19	34	0	0	11	19	0	0	0

second half. Therefore, the buffers can be reduced so only the first half of the coefficients are stored. In this case with 21 coefficients, the benefits would only be 11 LUT memory elements at the cost of additional logic elements such as an up-down counter for the index. Therefore, this optimization is not worth implementing for small filter sizes but may offer some resource savings for much larger FIR filters. Another potential optimization was using block RAMs instead of LUT memory elements for the buffers. As mentioned previously, the LUT memory elements are preferred due to the small buffer sizes and faster speeds. However, another reason is for compatibility between different FPGAs as implementing block RAM usually involves writing FPGA-specific code. Therefore, changing the main logic to incorporate block RAM may result in more logic elements being used.

Regarding the structure of the VHDL code, a typical state machine structure with controller and datapath was considered. However, the code needed to be highly parallel as the state machine would become very complex. Another consideration was to use the Advanced eXtensible Interface (AXI) for connecting blocks. This was not implemented due to the extra logic overhead per block, which would substantially impact the overall resource utilization.

5.3.2 Proposed Architecture

For the proposed architecture, the software for the TinyFPGA adopts the same generic VHDL code from the third iteration of the testing architecture. The first two iterations were not implemented as the designs exceeded the available resources on the TinyFPGA. Likewise, the block diagram is the same as figure 5.16.

The base-level code with SPI input, FIR filter, and SPI output blocks are kept mostly the same although the pinouts are completely different. However, the top-level file was modified to have the clock initialization code, which would generate the main clock signal using the internal PLL oscillator. In order for the timing constraints to be met, the maximum clock rate that can be set is 53.2 MHz to ensure a positive WNS. Additionally, the SPI output block was slightly modified so the data clock and SPI clock use the 53.2 MHz main clock rate. This was done by declaring it as a generic value that can be changed. Apart from this, the overall functionality is identical to the previous architecture.

Looking at the simulation waveform in figure 5.17, all the SPI signals are identical to the previous architecture. The baseband signal amplitudes are also within the expected bounds of the IQ modulator. With the main clock frequency set to 53.2 MHz, the expected symbol rate is still 1 kHz. However, the symbol period is not exactly 1 ms due to the fractional portion of the main clock period being truncated.

The key difference from the previous architecture is the post-implementation resource utilization as shown in table 5.6. FPGA chips usually differ in the number of LUTs, registers, and other hardware as well as the grouping of this hardware (known as SLICEs). Although less than half of the available registers, SLICEs, and LUTs are being used, an additional 11 registers, 133 SLICEs, and 188 LUTs are used in this implementation as compared to the previous architecture. Additionally, there are 24 more LUTs being used as distributed memory alongside 3 block RAMs. The TinyFPGA inferred these block RAMs despite the code being unchanged, albeit with no noticeable difference in functionality.

The timing report shows zero unrouted paths, hence all the paths in the design meet the timing constraints. Table 5.7 shows both the worst slack (which is equivalent to the WNS)

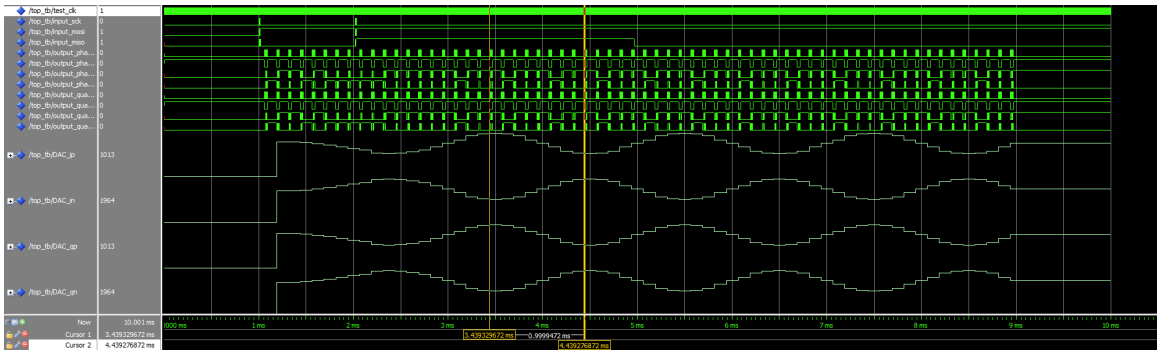

Figure 5.17: Output waveform of software on TinyFPGA

and the worst hold slack as being positive values. This indicates that the implementation has small available margins for additional functionality without decreasing the main clock frequency.

5.3.3 Alternate Architecture

In the case of the alternate architecture, the software is written in C and is similar in certain aspects to the VHDL code. Although the hardware is entirely different, the same overall functionality is implemented where the hardware blocks are now software tasks that run on the MSP430. The flow chart of the software is shown in figure 5.18.

Once the MSP430 boots up in the main function, it first initializes the pins and configures the SPI and DAC peripherals, and enables interrupts. This allows for the Interrupt Service Routine (ISR) for the SPI and DAC peripherals to be invoked depending on their trigger condition. Continuing with the infinite while-loop, the first if-statement checks if the input buffers have data to be filtered and if the output buffers have space. Assuming these conditions are met, the FIR filtering operation takes place in a for-loop using the FIR coefficients and input data for both channels. The final results are written to the output

Table 5.6: Resource utilization of software on TinyFPGA

```
Design Summary
   Number of registers:    148 out of  1346 (11%)
      PFU registers:          148 out of  1280 (12%)
      PIO registers:            0 out of    66 (0%)
   Number of SLICEs:       222 out of   640 (35%)
      SLICEs as Logic/ROM:    198 out of   640 (31%)
      SLICEs as RAM:           24 out of   480 (5%)
      SLICEs as Carry:         91 out of   640 (14%)
   Number of LUT4s:        441 out of  1280 (34%)
      Number used as logic LUTs:        211
      Number used as distributed RAM:    48
      Number used as ripple logic:      182
      Number used as shift registers:     0
   Number of PIO sites used: 11 + 4(JTAG) out of 22 (68%)
   Number of block RAMs:  3 out of 7 (43%)
   Number of GSRs:        0 out of 1 (0%)
```

buffers. If the condition was not met, the filtering operation would be omitted and the execution will continue. The last if-statement checks the input buffers and raises the MISO signal if they are full or lowers it otherwise. This indicates to the input device whether to send more SPI data packets or to wait.

As the while-loop covers the filtering task, the other two subset diagrams on the right side show the ISR for the SPI input task and DAC output task:

The SPI ISR task is triggered when a complete byte is received in the SPI peripheral's shift register. Then, the values from the shift register are written to the input buffers. However, the binary values are converted to NRZ coded values between positive and negative one, split into odd and even bits, and upsampled by 8. After this, the input buffers are checked again for full capacity and the MISO signal is changed accordingly.

For the DAC ISR, the trigger is based on a timer that raises a flag every 1 ms. This corresponds to the DAC ISR being called at a rate of 1 kHz and the output updating at that rate. The values from the output buffers are first inverted to generate the negative part of the differential signal. Both are then scaled to the intended bias value corresponding to the bias voltage of the IQ modulator input.

Table 5.7: Timing report of software on TinyFPGA

Cost Table Summary

Level/ Cost [ncd]		Number Unrouted	Worst Slack	Timing Score	Worst Slack(hold)	Timing Score(hold)	Run Time	NCD Status
5_1	*	0	1.570	0	0.282	0	06	Completed

The memory layout and usage are shown in figure 5.19, where less than half of the total available resources are being used. As mentioned in the design section, placing as much of the code in FRAM make the software more resistant to radiation-related errors. Due to this, all the buffers and variables are placed in FRAM while only the stack is present in the RAM. This is due to the extensive number of read/write operations taking place in the stack, which would reduce the lifetime of the FRAM. Also, errors during execution can be fixed by power cycling the MCU, hence retaining the FRAM data while resetting the stack in the RAM.

Code Structure

Similar to the final FPGA code, there are two input/output buffers corresponding to the IQ channels and one FIR coefficients buffer. All the buffers are all declared as integer arrays of specific bit length. The FIR filter coefficients are the same RRC filter values from the previous architectures. Initially, the length of the arrays was chosen to be bit-aligned to the 8-bit length of the read/write index variables. This allowed for the read/write variables to roll over the 256 value and back to zero, essentially implementing a circular buffer without the additional if-statement index checks. However, this caused issues where random values were populated in the buffers and resulted in the non-deterministic behavior of the software. Instead, explicit checks of the index using if-statements were employed, resulting in reliable

and stable functionality at the output.

From the perspective of functionality, the filtering operation results in the same output as the aforementioned architectures while being implemented differently. Since the MSP430-FR2355 does not contain any hardware DSP units for filtering, the first approach was to use multiplication for calculating the FIR filtering results. However, the multiply/divide operators are resource intensive for large-scale computations even though the operands are integers.

Therefore, the second approach was to use the hardware multipliers unit present on the MSP430-FR2355. Although this gave a significant computation boost, it exposed another issue with memory read/write operations. With the lack of Direct Memory Access (DMA) hardware on this specific MSP430, any read/write operations would involve processor cycles whereas a DMA unit would manage the memory while the processor can work on filter calculations.

The problems did not affect the overall functionality but instead slow down the DAC output rate significantly. Hence, other potential solutions were attempted where the MCU clock speed was increased to the max rate. The MSP430-FR2355 runs at 1 MHz by default but can be increased up to 24 MHz. This can be done using the in-built Digitally-Controlled Oscillator (DCO) and Frequency Locked Loop (FLL) to generate the 24 MHz clock however, the final implementation should use a more reliable and temperature stable 24 MHz crystal. The result of this modification yielded a significant increase in the DAC output rate however, it was still under the desired 1 kHz output rate.

These cascading issues were solved by the final approach which replaces the multiplication operation with if-statements. Essentially, the multiplication operation only multiplies the FIR filter coefficient with the NRZ coded input data. Since the data can only be a positive one, negative one, or zero, the multiplication result can be explicitly coded using if-statements. Depending on the value of the input data, the corresponding result can be substituted into the rest of the calculation per iteration of the filter.

Overall, this method works very well in this case where the input data is NRZ coded and

only three states need to be checked. However, if the symbol mapping was not NRZ coding, the if-statements would need to be expanded to cover all possible cases. This method was also considered for the FPGA implementation however, the if-statements would result in more resources being used as opposed to the direct multiplication approach.

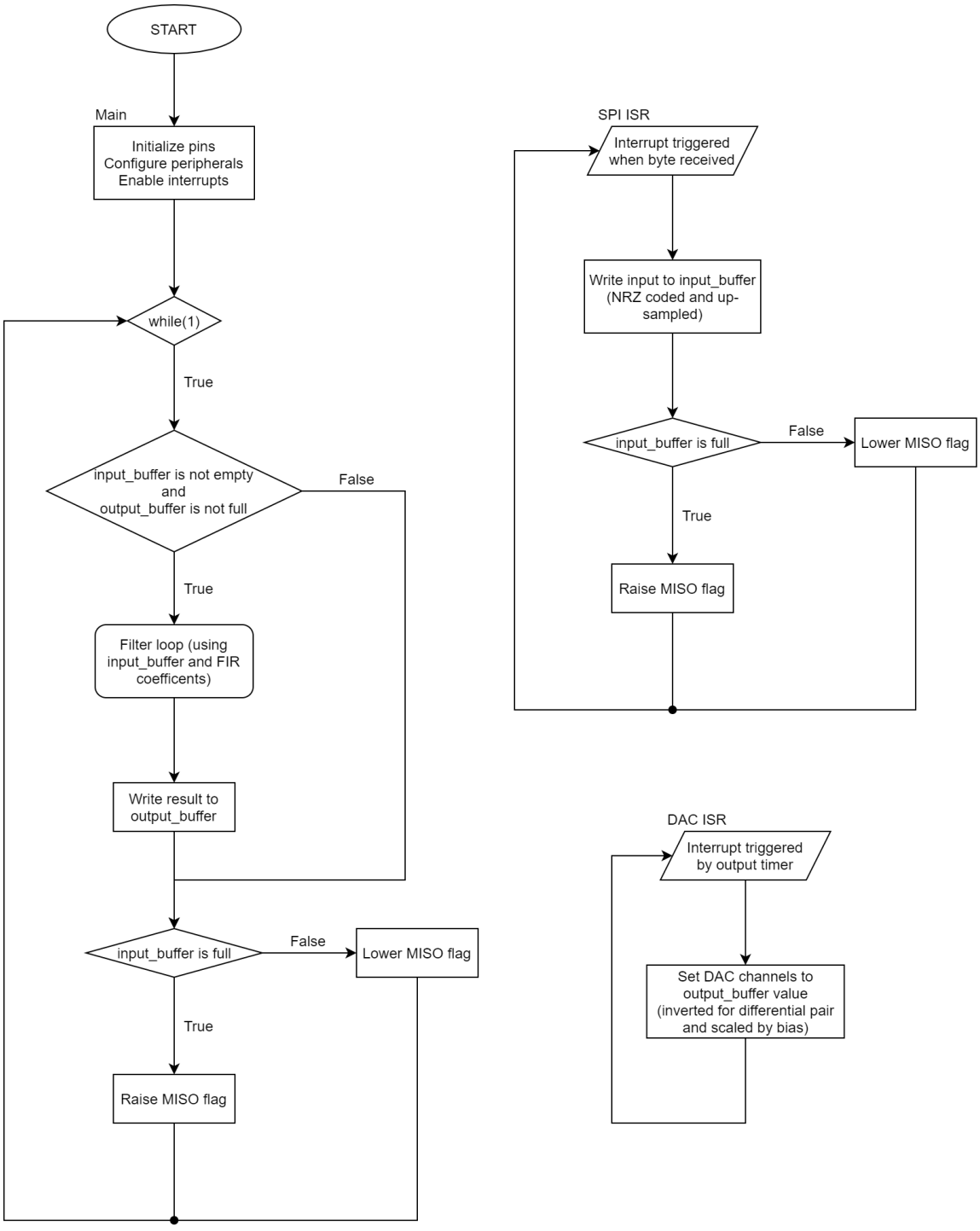

Figure 5.18: Software flow chart for the MSP430

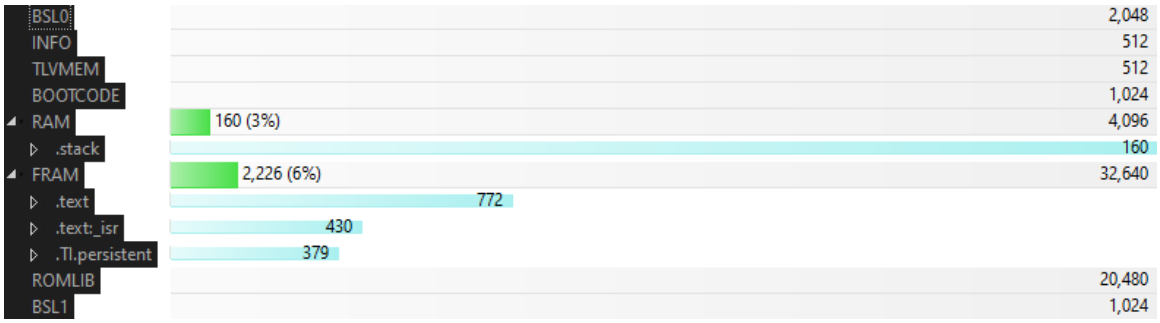

Figure 5.19: Memory usage of software on MSP430

Chapter 6: Testing

6.1 Output Plots

The ensuing sections show the output at various stages for each architecture. The setup is the same as the figures in the hardware implementation section. For the baseband signal plots, the output was connected to an oscilloscope for visualizing the time and frequency domain. With the RF signals, they were connected to a Vector Signal Analyzer (VSA). This particular instrument can show the frequency spectrum of a signal up to 6.5 GHz with 40 MHz bandwidth. It also has a symbol decoder feature, where the incoming signal is demodulated and the symbols are shown in different formats.

6.1.1 Proposed Architecture

Starting with the ADF4351 VCO, the output carrier signal is shown in figure 6.1. The sharp peak is at roughly 2.4 GHz with an amplitude of -24.62 dBm. The frequency is not exactly at 2.4 GHz due to the clock calibration and temperature changes within the chip. This carrier signal is connected to the LO positive input of the AD8346 IQ modulator.

Next, the time domain of the baseband signal from the FPGA is shown in figure 6.2. The top two traces correspond to the in-phase channel positive and negative outputs while the bottom two are the quadrature channel positive and negative outputs. The input data is the same as the simulation (0x33) and the symbol interval is also 1 ms (1 kHz symbol rate). The 509.97 Hz shown at the bottom right is the waveform's frequency. The slight offset from the expected 500 Hz is due to the inaccuracy of the internal FPGA clock, which can be improved by using an external crystal. Notice the highlighted markers are at the maximum and minimum amplitude points. These correspond to 1.52 V and 800 mV, which are both within the bounds of the IQ modulator.

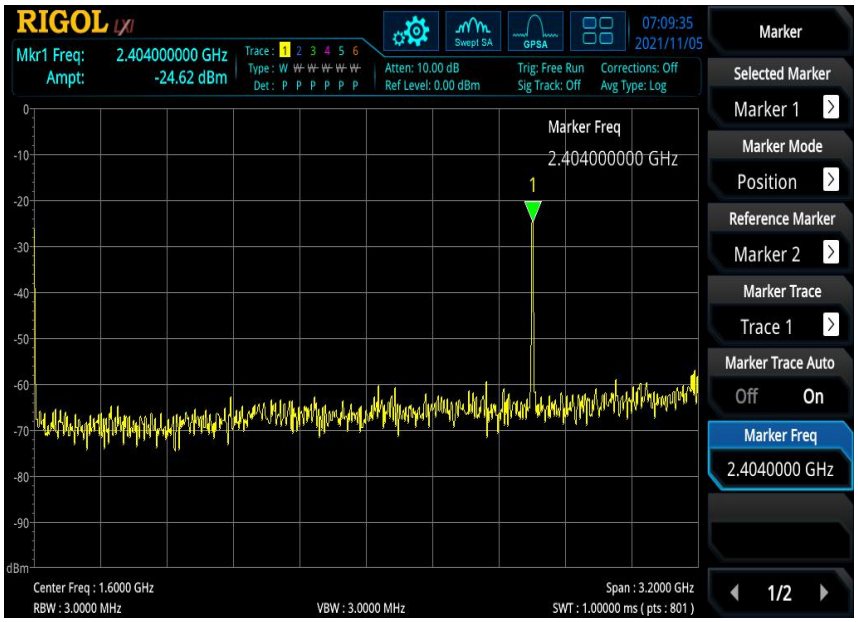

Figure 6.1: Frequency spectrum of the carrier signal from the ADF4351 VCO

In figure 6.3, the frequency domain of the in-phase positive signal is shown. The markers indicate the first two peaks which are at 500 Hz and 1500 Hz. Although the symbol interval is at 1 ms, the frequency is at 500 Hz. Hence, the pulse shape filter passes the 500 Hz signal with -21.5 dB while attenuating the 1.5 kHz signal to -39.7 dB.

Moving on to the VSA, the symbol decoding of the RF signal is shown in figure 6.4. The top left plot shows the symbol constellation where the received signal goes between two symbol points. This matches the plot on the right which shows the decoded binary values (0x33). The yellow highlighted bits are indicating the burst-search feature of the VSA to find the intended bit sequence. The plot in the bottom left shows the frequency domain of the RF signal while the bottom right plot shows the time domain of the decoded baseband signal. Notice the right sidebar shows the measurement interval of 50 bits, the points per symbol set to 16, and the symbol rate set to 1.02 kHz. Ideally, these would be set to 8

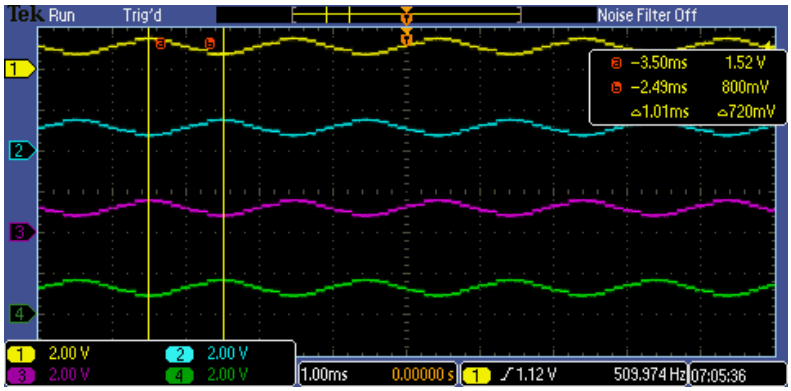

Figure 6.2: Time domain of the baseband signal from the TinyFPGA

points per symbol and 500 Hz however, these cannot be set due to the minimum symbol rate of 1 kHz of the VSA. Also, the rate matches the slight offset from the time domain plot, where the 1.02 kHz symbol rate is two times the 509.97 Hz frequency.

Lastly, figure 6.5 shows the output when the input data contains random bits. The constellation diagram shows four clusters of symbols while the decoded values are all random bits. The main plot to observe is the time domain plot of the baseband signal. This is known as an eye diagram due to the overlaid signals forming an eye shape. The middle of the eye represents the optimal sampling point (highest SNR) for the receiver. Whether the eye is open or closed is determined by the timing errors due to clock jitter and noise-like interference added to the RF signal.

6.1.2 Testing Architecture

Identical to the earlier implementation, the same carrier signal shown in figure 6.1 is passed into the LO positive input of the AD8346 IQ modulator.

Looking at the baseband signal, the time domain plot from figure 6.6 shows the same symbol interval of 1 ms and similar voltage level bounds of 1.52 V and 720 mV. However, the

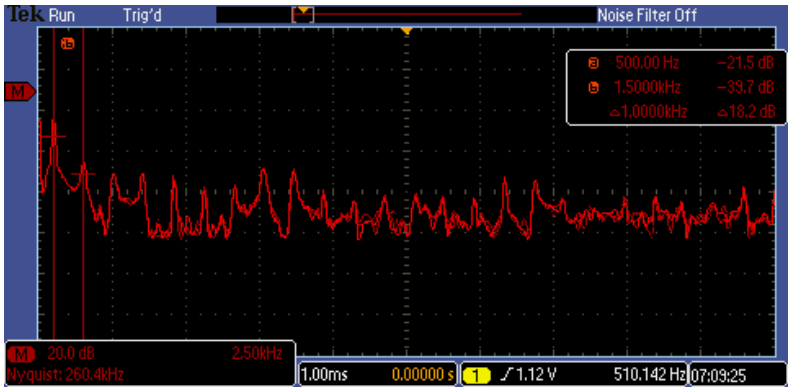

Figure 6.3: Frequency domain of the baseband signal from the TinyFPGA

frequency of the waveform is at 500.00 Hz as opposed to the slight offset from the previous architecture. This is due to the Basys 3 board having an external crystal that provides a much more stable clock.

With the frequency domain plot in figure 6.7, it is also identical to the earlier architecture with the same peaks of 500 Hz and 1500 Hz. In this case, the 500 Hz passband signal is -18.4 dB while the attenuation at 1500 Hz is -33.7 dB.

The VSA symbol decoding plot in figure 6.8 is nearly identical to the earlier implementation. The only difference is the symbol rate of 1 kHz due to the 500 Hz waveform frequency.

With random input bits, the same constellation is shown in figure 6.9. The eye diagram shows a similar eye shape where the middle of the eye is open, indicating the ideal sampling point for the receiver.

6.1.3 Alternate Architecture

In this architecture, the same carrier signal is used as shown in figure 6.1. This signal is fed to the onboard ADL5375 IQ modulator, where the LO positive and negative inputs

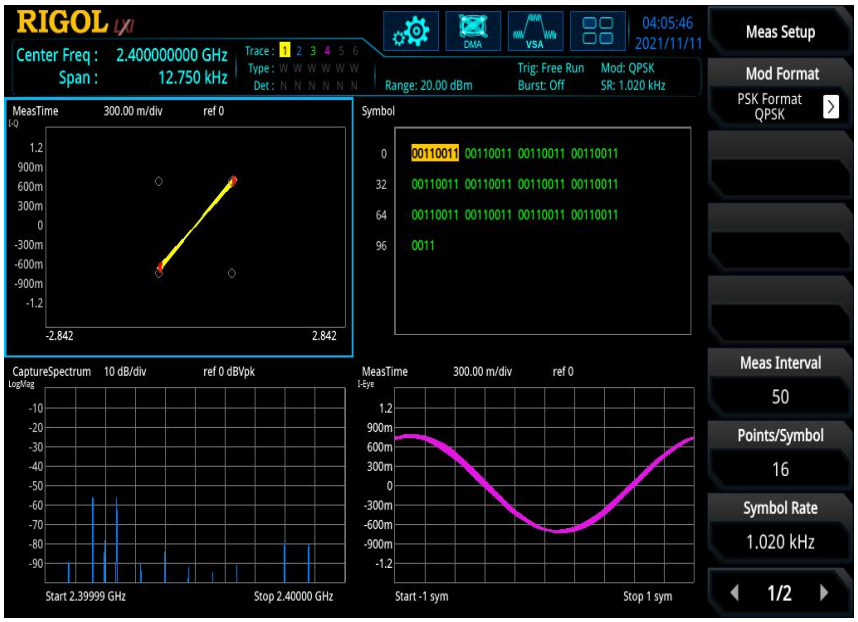

Figure 6.4: Input symbol decoding of the RF signal from the AD8346 IQ modulator

are already connected through loop filters. All of the subsequent plots for the baseband time domain, baseband frequency domain, VSA input symbol decoding, and VSA random symbol decoding are identical to figure 6.6, 6.7, 6.8, and 6.9 respectively from the previous architecture. Although a completely different IQ modulator was used, there were no noticeable differences in the output of each plot. This would imply that the IQ modulators function independently from the baseband and carrier signals as expected.

Figure 6.5: Random symbol decoding of RF signal from the AD8346 IQ modulator

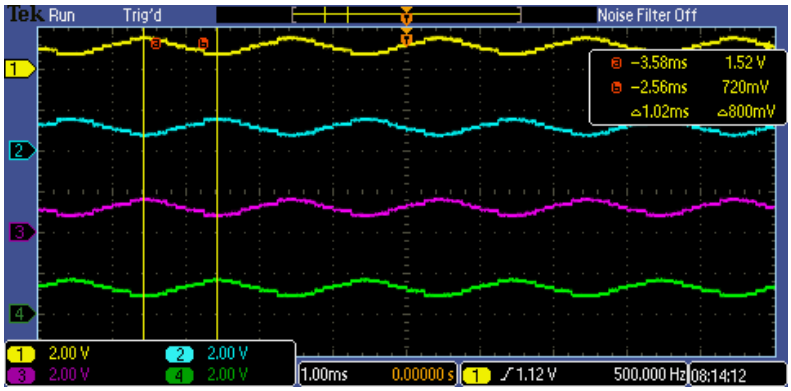

Figure 6.6: Time domain of the baseband signal from the Basys 3 FPGA

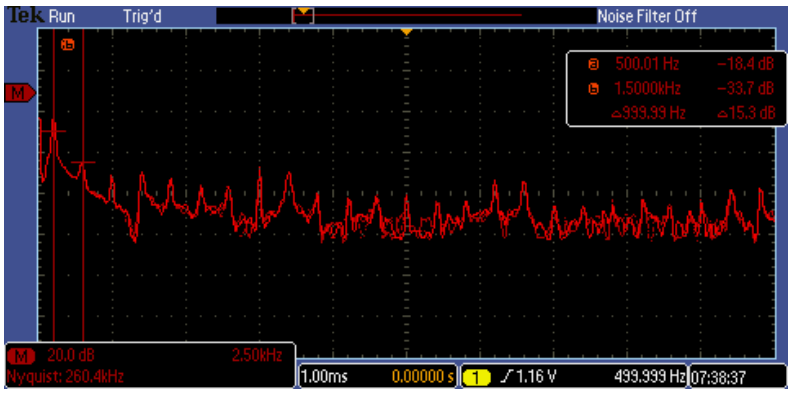

Figure 6.7: Frequency domain of the baseband signal from the Basys 3 FPGA

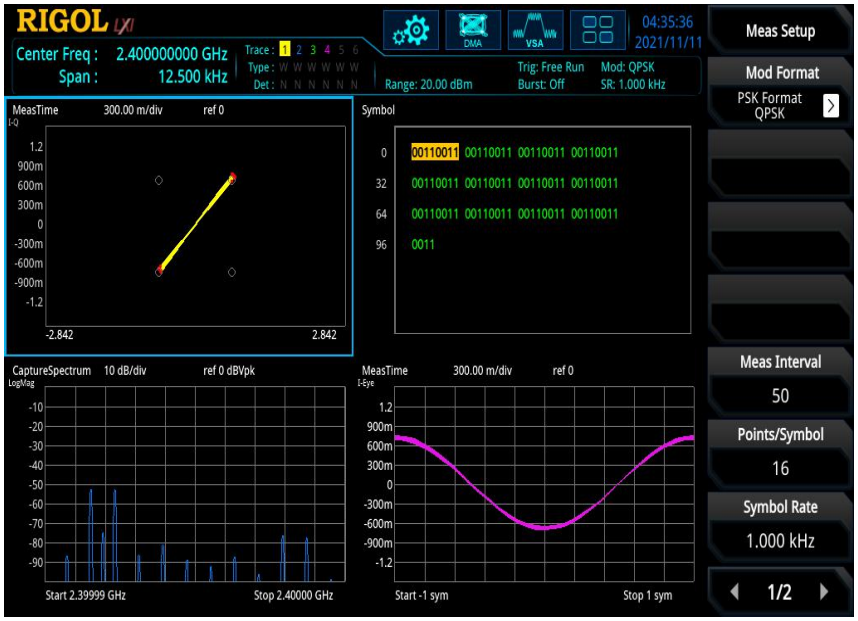

Figure 6.8: Input symbol decoding of the RF signal from the AD8346 IQ modulator

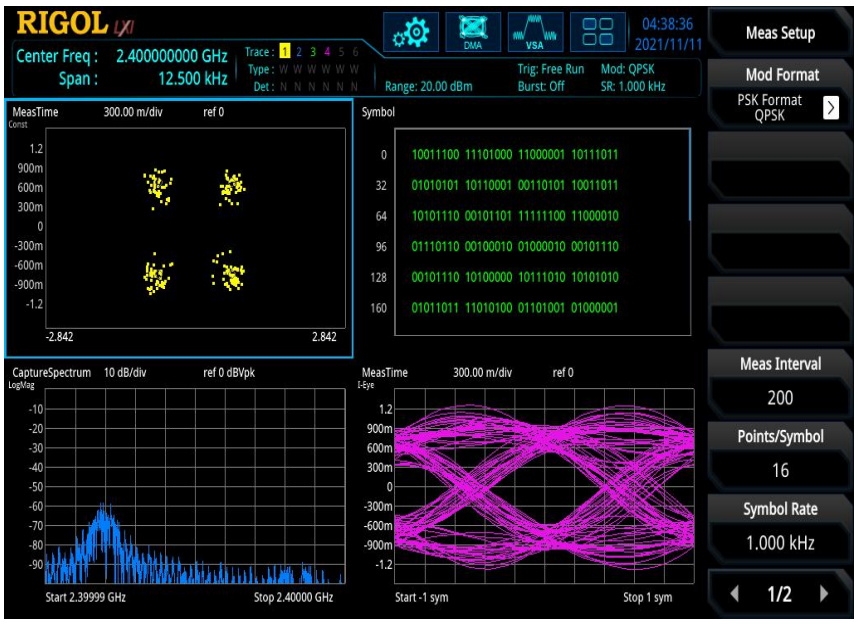

Figure 6.9: Random symbol decoding of RF signal from the AD8346 IQ modulator

Chapter 7: Analysis

7.1 End-to-End System

The following section covers an end-to-end setup for the entire QPSK modulation chain. It starts from a transmitter sending known data through the proposed architecture to the receiver. The RF signal is then received using a LimeSDR-USB Software-Defined Radio (SDR) and demodulated with GNU Radio to recover the baseband signal. This would mimic a satellite-to-ground communication system where the payload on the satellite would send data to its transmitter, which transmits the RF signal to the ground station receiver for recovering the payload data.

The payload is a Raspberry Pi 3B+ (RPI) which sends fixed data values (0x33) over SPI. It is configured to function as the bus master and runs the SPI clock at 500 kHz while the data rate is roughly 1 kHz. With the MISO signal behaving as an on/off switch, the RPI starts sending data when the signal is low and stops when the signal is high. This allows for flow control at the receiver end as the input data buffers reach full capacity.

Implementing the proposed architecture, the receiver is the TinyFPGA AX2 which functions as a slave device on the SPI input bus. The same code discussed earlier is executed on the FPGA and the output is the filtered baseband differential signals from the IQ channels. These signals are passed into the PMOD-DA2 DACs through a modified SPI bus at the same 500 kHz SPI clock rate and 1 kHz data rate.

Next, the DACs receive the digital baseband signals and convert them to analog signals. The outputs are updated as soon as a new SPI data packet is received, hence the output rate is also 1 kHz. Both sets of differential signals are connected to the appropriate positive and negative inputs of the AD8346 IQ modulator.

For the ADF4351 VCO, the board is controlled over USB and powered using a power

supply. The carrier frequency is set to 2.4 GHz and LO positive output is connected to the IQ modulator LO input. Although the VCO provides differential outputs, the IQ modulator board is configured in single-ended mode and only the positive signal is needed.

Lastly, the IQ modulator is also powered by the same power supply and generates the resulting RF signal. This is connected to the RX input of the LimeSDR-USB, which is set to use the high-band specific channel for inputs greater than 1.5 GHz. Once the signal is sampled and demodulated by the GNU Radio software, the baseband signal is recovered at the receiver end of the system. The entire setup is shown in figure 7.1.

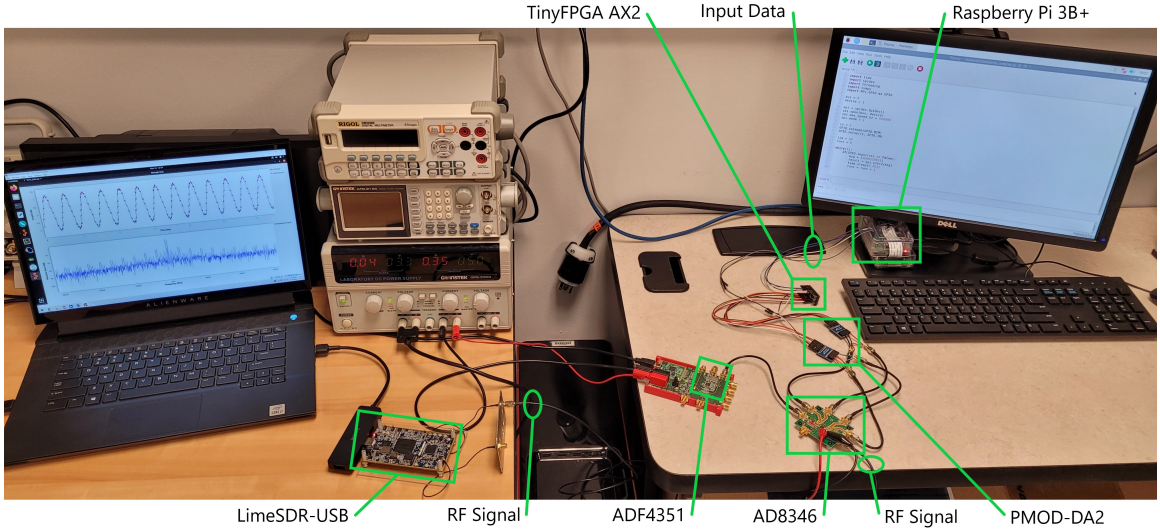

Figure 7.1: End-to-end setup of QPSK modulation chain

Looking at the GNU Radio flowgraph in figure 7.2, it largely resembles the receiver side of the QPSK baseband structure from the implementation section. Starting with the left side, the osmocom source block is used to interface with different SDRs. In this case, it is configured to work with the LimeSDR-USB with the frequency set to 2.4 GHz, bandwidth being automatically controlled, and sample rate of 200 kHz. The block generates

IQ samples at the specified rate, which are passed into an AGC block. This block along with the symbol sync block is identical to the receiver side of the QPSK baseband structure from the implementation section. However, the flowgraph differs as the output is passed into a Costas loop block.

The QPSK baseband simulation from the implementation section assumed that the baseband signal is perfectly centered at zero frequency. However, this is not the case with a real hardware implementation where the transmitter oscillator may have a slight offset when upconverting to the 2.4 GHz carrier frequency. Similarly at the receiver, the oscillator in the LimeSDR-USB can have a different offset from the carrier frequency, resulting in the baseband signal being frequency shifted. The Costas loop is used to adjust the frequency shift back to zero frequency. Lastly, the signal is viewed in the time domain and frequency domain using the sink blocks [9].

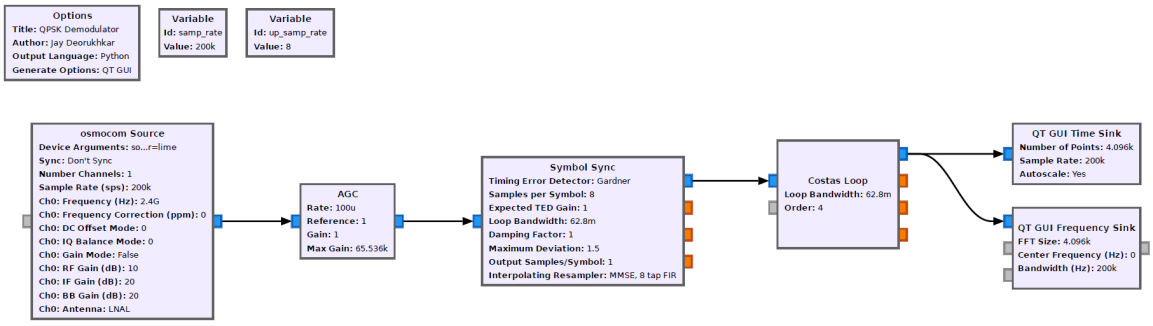

Figure 7.2: Flow graph for QPSK demodulator

From figure 7.3, the top plot shows the time domain of the demodulated signal while the bottom plot shows the frequency domain. The time domain plot has both IQ traces that are overlapping each other. Once the IQ channels are re-interleaved with odd and even bits, the result is the same as the transmitted data. In the frequency domain plot, the distinct spike corresponds to 1 kHz as expected.

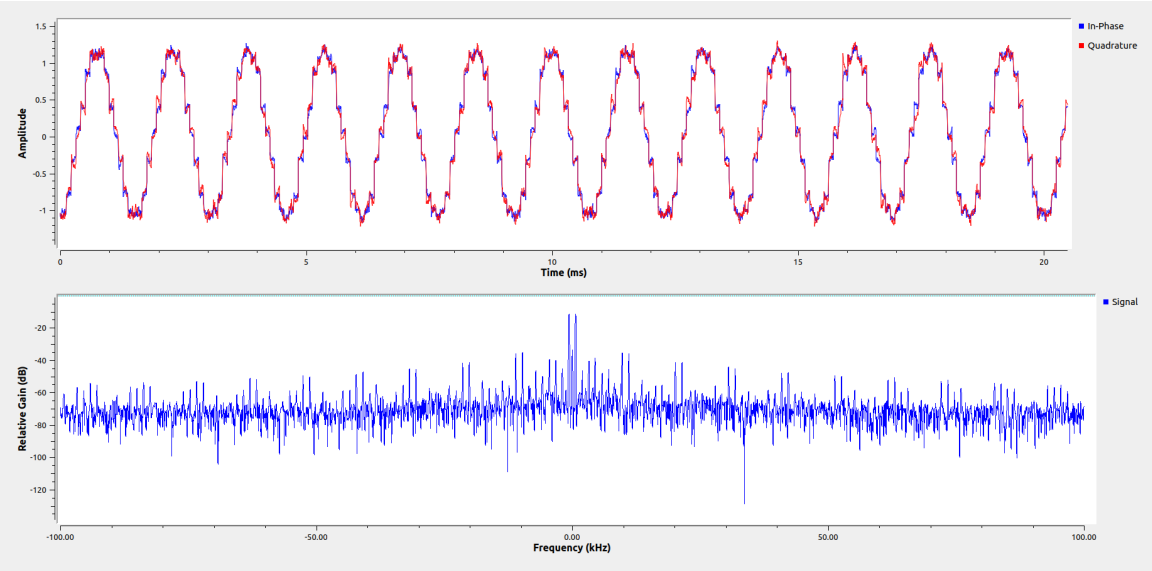

Figure 7.3: Time domain and frequency domain plots for QPSK demodulator

Setup Considerations

For the SPI input device, the current setup is to have the RPI as the bus master while the FPGA is the slave device. The FPGA uses the MISO signal to enable/disable the data transmit from the RPI. However, a typical SPI implementation would allow the RPI to transmit continuously while the FPGA would send ACK/NACK bytes over the MISO signal. An ACK would indicate the byte being transmitted is successfully received by the FPGA. A NACK would be sent to indicate the FPGA buffers are full and that the RPI should keep re-transmitting the last byte until acknowledged by the FPGA. The flip side of this implementation would be if the FPGA was the bus master and the RPI was the slave device. In this case, the FPGA would send a byte to request the data from the RPI. When the buffers are full, the FPGA can simply turn off the SPI transmission to prevent new data from being requested. Note that having the FPGA as the bus master would require it

to generate the SCK clock signal, resulting in more resources being used.

When performing carrier synchronization, the Costas loop method is known to have a phase ambiguity when decoding the data symbols. This can be solved by validating the output symbols with a known synchronization header. If the symbols are inconsistent with the header, then the flow graph can be re-run until the correct header sequence is detected. Otherwise, the received data can be corrected manually by shifting the symbols and adjusting the data bits. Another approach to solve this issue is using differential encoding. This is where the symbols are encoded based on changes in the data bits, as opposed to the symbols corresponding to the data bits themselves [9].

Chapter 8: Conclusion

With the growing number of small satellites being launched, the PocketQube form factor has gained attention for its reduced launch fees and relatively inexpensive building cost with the trade-off of power and size. With more complex missions taking place, the need for higher data rates using better modulation schemes becomes apparent. This growth has also increased the congestion at the typical VHF/UHF bands and provokes the use of higher frequency bands for better signal strength and higher bandwidth.

To address these issues, this thesis proposes a QPSK transmitter architecture for S-band operation on PocketQubes satellites. This architecture alongside the testing and alternate architectures is shown in detail starting with the overall functionality of the QPSK structure. Next, the design of each architecture was shown with the reasoning for selecting the specific components. Then, the implementation stage covers the simulation, hardware, and software aspects of the architectures. This is followed by the testing of each architecture which verifies the intended behavior of the transmitter. Lastly, the analysis portion brings the proposed architecture into an end-to-end test where data is sent through the entire chain from the transmitter side to the received end.

Although the testing and analysis portions show this architecture being a viable option for use on PocketQube satellites, additional testing needs to be conducted. This would include building the final board and undergoing functional and environmental testing. While the output plots show the expected symbols being received while transmitting at 2.4 GHz, these were under the assumption of no channel impairments which will inevitably cause symbol errors. Additionally, these tests would also reveal the performance bounds of the architecture. With the end-to-end link being tested at a 1 kHz data rate, the upper and lower bound may vary from this initial baseline.

To build on this, other variations can be considered in place of the QPSK structure.

This includes offset-QPSK, 8-PSK, or even higher-order modulation schemes. These can be achieved in the FPGA case by building a generic symbol mapping block where the mapping between the input data and output symbols can be modified. Similarly in the MCU case, it will involve changing the current mapping function to be the desired symbol mapping. In both cases, the underlying hardware may need to be changed as well if the MCU cannot handle the extra processing or if the FPGA does not have enough hardware resources.

In future implementations, the idea of software-defined modulation could be incorporated, where the modulation can adapt to a changing communication channel to optimize the data rate. Implementing this feature would require additional hardware and accompanying software which far exceeds this architecture. However, with enough hardware resources and processing capability, the modulation scheme could be reconfigured in software to maximize performance. This would result in better link reliability and efficient use of the channel.

Bibliography

Bibliography

[1] "Statistics," Libre Space Foundation, 2021. [Online]. Available: https://db.satnogs.org/stats

[2] G. D. Krebs, "Pocketqube," Gunter's Space Page, 2021. [Online]. Available: https://space.skyrocket.de/doc_sat/pocketqub.htm

[3] S. H. Bruce Hemp, "Low power iq modulator for digital communications," Analog Devices, 2021. [Online]. Available: https://www.analog.com/en/technical-articles/low-power-iq-modulator-for-digital-communications.html

[4] "Understanding i/q signals and quadrature modulation," All About Circuits, 2021. [Online]. Available: https://www.allaboutcircuits.com/textbook/radio-frequency-analysis-design/radio-frequency-demodulation/understanding-i-q-signals-and-quadrature-modulation/

[5] B. H. Weston Sapia, "A bpsk modulator for 2ghz to 12ghz," Analog Devices, 2021. [Online]. Available: https://www.analog.com/ru/technical-articles/bpsk-modulator-for-2ghz-to-12ghz.html

[6] T. Youngblood, "Practical fir filter design: Part 1 - design with octave or matlab," All About Circuits, 2021. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/design-of-fir-filters-design-octave-matlab/

[7] A. G. K. C. Richard Johnson Jr, William A. Sethares, *Software Receiver Design*. Cambridge University Press, 2011.

[8] M. Viswanathan, "Qpsk modulation demodulation (matlab and python)," GaussianWaves, 2021. [Online]. Available: https://www.gaussianwaves.com/2010/10/qpsk-modulation-and-demodulation-2/

[9] "Guided tutorial psk demodulation," GNU Radio, 2021. [Online]. Available: https://wiki.gnuradio.org/index.php/Guided_Tutorial_PSK_Demodulation

[10] "Symbol sync," GNU Radio, 2021. [Online]. Available: https://wiki.gnuradio.org/index.php/Symbol_Sync

Curriculum Vitae

Jay Deorukhkar received his Bachelors of Science in Computer Engineering from George Mason University in Spring 2019. In Summer of 2020, he interned at HawkEye 360. While pursing his Masters degree in Computer Engineering, he worked as a graduate teaching assistant for various courses. After graduation, he will start his PhD at George Mason University in Spring 2022.