The University of Oxford

Engineering Science

Fourth Year Project

PiCom: A Digital Communication Test Bed Based on Raspberry Pi

Layout of title page – Must have project title, name, college

---

Candidate

Cameron Eadie

Exeter College

Supervisor

Justin Coon

Trinity Term, 2018

FINAL HONOUR SCHOOL OF ENG

UNIVERSITY OF OXFORD

DECLARATION OF AUTHORSHIP

You should complete this certificate. It should be bound into your fourth year project report, immediately after your title page. Three copies of the report should be submitted to the Chairman of examiners for your Honour School, c/o Clerk of the Schools, examination Schools, High Street, Oxford.

Name (in capitals): ……

College (in capitals): …… **Supervisor:** ……

Title of project (in capitals): ……..……

Page count (excluding risk and COSHH assessments): ……

Please tick to confirm the following:

I have read and understood the University's disciplinary regulations concerning conduct in examinations and, in particular, the regulations on plagiarism (*The University Student Handbook. The Proctors' and Assessors' Memorandum, Section 8.8*; available at https://www.ox.ac.uk/students/academic/student-handbook) ☐

I have read and understood the Education Committee's information and guidance on academic good practice and plagiarism at https://www.ox.ac.uk/students/academic/guidance/skills. ☐

The project report I am submitting is entirely my own work except where otherwise indicated. ☐

It has not been submitted, either partially or in full, for another Honour School or qualification of this University (except where the Special Regulations for the subject permit this), or for a qualification at any other institution. ☐

I have clearly indicated the presence of all material I have quoted from other sources, including any diagrams, charts, tables or graphs. ☐

I have clearly indicated the presence of all paraphrased material with appropriate references. ☐

I have acknowledged appropriately any assistance I have received in addition to that provided by my supervisor. ☐

I have not copied from the work of any other candidate. ☐

I have not used the services of any agency providing specimen, model or ghostwritten work in the preparation of this project report. (See also section 2.4 of Statute XI on University Discipline under which members of the University are prohibited from providing material of this nature for candidates in examinations at this University or elsewhere: http://www.admin.ox.ac.uk/statutes/352-051a.shtml.) ☐

The project report does not exceed 50 pages (including all diagrams, photographs, references and appendices). ☐

I agree to retain an electronic copy of this work until the publication of my final examination result, except where submission in hand-written format is permitted. ☐

I agree to make any such electronic copy available to the examiners should it be necessary to confirm my word count or to check for plagiarism. ☐

Candidate's signature: …….. **Date:** ……..

Abstract

RED - Important information to check/change

BLUE - Sections and parts that still need writing/editing

GREEN - Formatting of sections and layout of images/figures

Here we shall have our abstract.

Write Abstract - Do I need an abstract? If so, write this last.

USE ”CLEARPAGE” (INCLUDE FIGURES) AND ”NEWPAGE” TO MAKE SURE SECTIONS ARE AS THEY SHOULD BE

Read back through the documentation, help pages, my own notes on how 4YP should be written and presented

Contents

Todo list

Chapter 1

Introduction

This is what is going on over here.

Figure out how the Intro Chapter will be formatted

1.1 Motivation

Still need to write this whole section

Modern digital communication systems are built upon a solid foundation of modulation and coding theory. Over the years, researchers have successfully developed numerous schemes using pen and paper along with computer models. Any such scheme ultimately must be tested on a suitable hardware/software platform to prove their usefulness in practice. Standard 'software-defined radio' test beds can cost thousands of pounds. Although these test beds provide users with advanced development tools, much of their functionality is superfluous to requirement.

A Raspberry Pi is a simple, affordable ARM-based computer module that is capable of interfacing with external peripheral devices through a bank of IO ports. It is also programmable (using Python), and as such has found many uses by hobbyists and electronics/computer engineers in recent years. The purpose of this project is to develop a basic digital communication test bed using two Raspberry Pi modules (one transmitter and one receiver). The test bed will be affordable and the interested student will need to work to a budget to ensure a successful outcome. The project will require a considerable amount of Python programming as well as knowledge of, and a keen interest in, digital communication theory and techniques.

1.2 Background - Literature Review

Explanation of the existing literature [1].

$$\mathbf{F}(t) = (\mathbf{m}(t) \cdot \nabla)\mathbf{E}(t) \tag{1.1}$$

where $\mathbf{E}(t)$ is the electric field in $150\,\mathrm{cT}$.

1.3 Contributions

```
import numpy as np

def incmatrix(genl1,genl2):
M = "Hello" # To become the incidence matrix
return M
```

Listing 1.1: Hello

Chapter 2

The Raspberry Pi and the Test Bed

Talk about the RPi and test bed.

2.1 Fundamentals

The Raspberry Pi is a simple, affordable ARM-based computing module. It has, which can interface

It is run using a Linux-based operating system called Raspbian which is available for download from the Raspberry Pi official website [**web˙Raspbian**].

Write Raspberry Pi Fundamentals

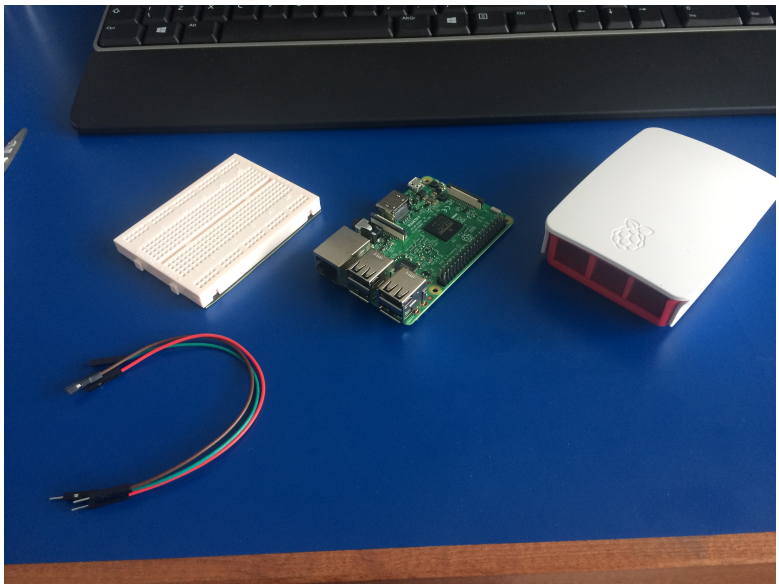


Figure 2.1: The Raspberry Pi

2.1.1 Setting up the Raspberry Pi

The Raspberry Pi can be interfaced with in a number of ways, but first it needs to be set up with its operating system on the SD card [2]. Raspbian can be accessed via the command line - typing in commands - or through an X Window similar to Windows OS. Either of these options can be used by connecting a screen, keyboard and mouse to the Pi, but this is not always available, so it is useful to have Secure Shell (SSH) set up for the command line, and Virtual Network Computing (VNC) for the X Window. Secure Shell is necessary regardless as it is used in the test bed execution. When actually using both Raspberry Pis in the test bed, both of them will be running in command line mode as this saves resources, although during development, VNC or a screen using the X Window makes it much easier to test and edit code. In the test bed, the Transmitter Pi will be connected to a screen and peripherals in command line mode, and then it will start the Receiver via SSH, which will run headless (without a screen or control peripherals) and receive the data, process and output it, and store information about the run in a log file which can be accessed later or again through SSH.

Once the operating system is written to the SD card, SSH can be enabled by creating an empty file 'ssh' in the main portion of the card before putting it in the Raspberry Pi (for setup using a screen and peripherals, this can be turned on in the configuration settings). Now an Ethernet cable can be connected from a computer to the Pi and it can be logged into. A program such as PuTTY for Windows can be used, with "pi@raspberrypi.local" as the host name so the IP address isn't needed. The standard login is "username: pi" and "password: raspberry".

Inline code reference should look like inline code

The command "sudo raspi-config" accesses the configuration settings where the file system can be expanded to the whole SD card, and other utilities such as Wi-Fi and VNC can be turned on. Again, all utilities except for SSH should be turned off when using the final test bed to improve performance of the devices. Once the Pi is connected to Wi-Fi and the IP address is known, it can be accessed remotely via the free RealVNC VNC Viewer **??** and the X Window can be started with the command "startx" if it isn't already set to open the X Window on startup in the configuration settings.

MISSING REFERENCE - https://www.realvnc.com/en/connect/download/viewer/

This access can be made permanent by setting the Pi to have a static IP address or with a free RealVNC account which lets it be accessed independent of the IP address.

With full control of the Pi, all that is needed is to download all of the libraries and software used by the test bed and clone the GitHub repository with its code. The Python version used is 3.x.x.

Find python version of Pis

and as the Raspberry Pi has Python 2 and 3, pip - the Python package manager - is called as pip3 for python3. Any dependencies not in this list give clear instructions how to be downloaded when this is run line by line.

Create new format (and inline for above) called ”shell”

Include matplotlib if used in the end - ALSO INCLUDE REFERENCES TO ALL LIBRARIES MENTIONED

```
1   sudo apt-get update
2
3   sudo apt-get install vpnc
4   sudo apt-get install libffi6 libffi-dev python-dev
5   sudo pip3 install cryptography paramiko
6
7   sudo apt-get install libatlas-base-dev
8   sudo pip3 install numpy imageio
9   sudo apt-get install pigpio
10
11  sudo apt-get install git
12  cd "<Path for the code e.g. /pi/home/Documents/>"
13  git clone https://github.com/CamEadie/4YP_PiCom
14
15  sudo apt-get update && sudo apt-get upgrade && sudo apt-get dist-upgrade
16
17  # Used in Transmit_Binary_Data(): import RPi.GPIO as GPIO
18  # ... This library is already included in the Raspberry Pi
19  # ... Only for OOK transmission which uses Python not C
20  # Used in Check_Input_Masks(): from RPiSim.GPIO import GPIO
21  # ... pip GPIOSimulator (or pip3 if necessary)
22  # ... Only works in Windows, simulates Raspberry Pi pins (use like RPi.GPIO commands)
```

Listing 2.1: Libraries and Packages Required for the Test Bed

Do I need to change GitHub to an anonymous account if only candidate number is on cover page

Included are the following, as well as their dependencies:

- VPNC - VPN software used to access the Oxford VPN in order to use the OWL Wi-Fi network
- Paramiko - Python library used for SSH to start the receiver

- NumPy - Python library used for easier array manipulation as well as interface with images
- imageio - Python library used to read and save image files as NumPy arrays
- pigpio - C library used to interface with the GPIO pins
- Git - Version control software which also allows for the cloning of the project code to any device

2.2 Test Bed Architecture

Mention BMC numbering scheme and full name for it

The test bed comprises the two Raspberry Pis and a number of chips to provide the functions required for more advanced modulation schemes. This is built up as three arrangements with increasing complexity. The first is the Pis connected together by two wires, a serial data line and a clock line (Figure 2.2). This arrangement is similar to that used for an I^2C bus, however the code written for this form of communication doesn't rely on any available modes of serial interfacing because it needs to be extensible to the parallel communication in the next arrangements. The second arrangement is used for Pulse Amplitude Modulation schemes. It uses a single parallel Digital Analogue Converter (DAC) connected to the transmitter which transmits to a parallel Analogue Digital Converter connected to the receiver, allowing for multi-level signals to be transmitted between the two devices. The final arrangement extends the set-up to two DACs and two ADCs. These signals are then multiplied by a sine wave and a cosine wave respectively, and can be separated due to the orthogonality of the two signals at the receiver. This arrangement is used for Quadrature Amplitude Modulation as well as Orthogonal Frequency Division Multiplexing.

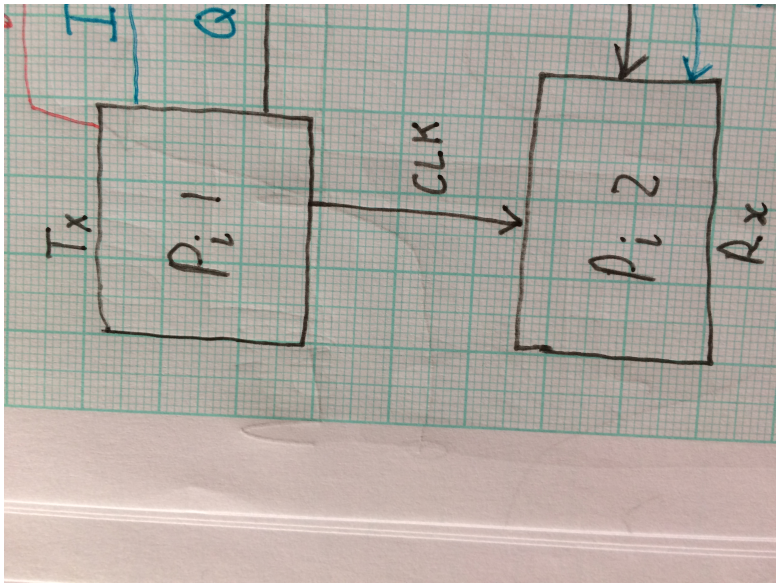


Figure 2.2: Test Bed Set-up For On-Off Keying

2.2.1 Digital Analogue Converter

The Digital Analogue Converter used is the AD5424, an 8-bit CMOS current output DAC with an easy interface to microcontrollers. It has a 17 ns write cycle and a maximum update rate of 20.4 MSPS. The converter is set up with a non-inverting operational amplifier to produce a full-range voltage output.

The Read/Write ($R/\overline{W}$) pin is pulled low permanently so the chip is in write mode; read back of the parallel digital outputs is not required. The Chip Select pin ($\overline{CS}$) needs a falling edge and a rising edge to complete a write cycle, where the rising edge loads new parallel data, so this is connected to the clock pin of the Raspberry Pi. Layout of the connections to the Pi are in the Figure 2.3.

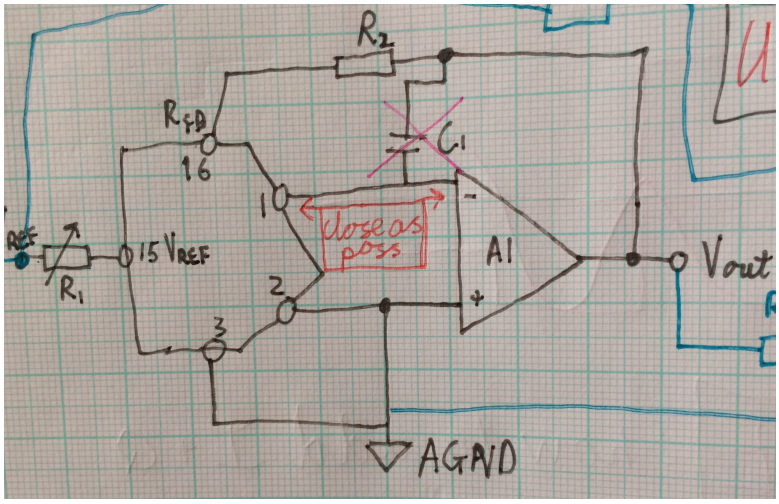


Figure 2.3: Layout of the Digital Analogue Converter

2.2.2 Analogue Digital Converter

The Analogue Digital Converter used is the ZN448,

Layout of Digital Analogue Converter

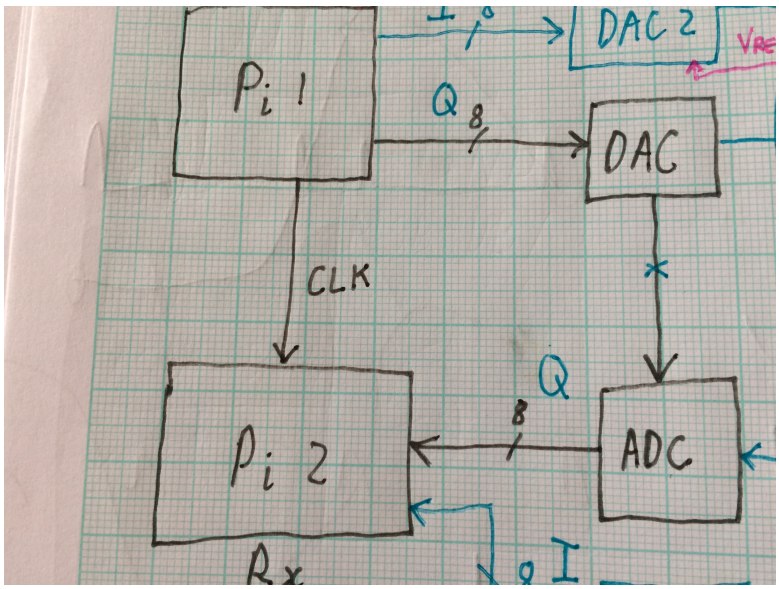


Figure 2.4: Layout for Pulse Amplitude Modulation

2.2.3 Multiplier

Depending whether it's doable, include this section without multipliers on transmitter, pseudo ground on output

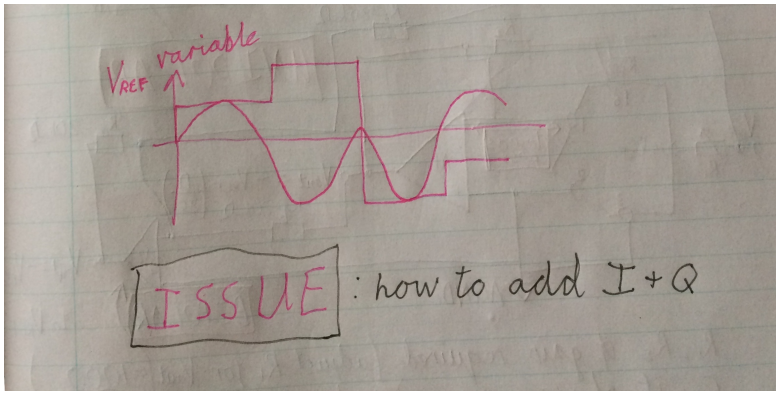


Figure 2.5: Four Quadrant Multiplication of input sinusoid using DAC

The Pi is not able to output negative voltages. Thus, Pulse Amplitude Modulation uses the range 0 to Vmax not -Vmax to Vmax. Similarly, QAM uses a grid all in the positive quadrant (0,1,2,3 not -3,-1,1,3). The same "negative" effect is still achieved by using the "GROUND" of the multipliers (which are fully differential) as Vmax/2 using a voltage divider. This means the sine and cos will be inverted for the values 0 and 1 in the same way they would be for -3 and -1 and the transmitted signal

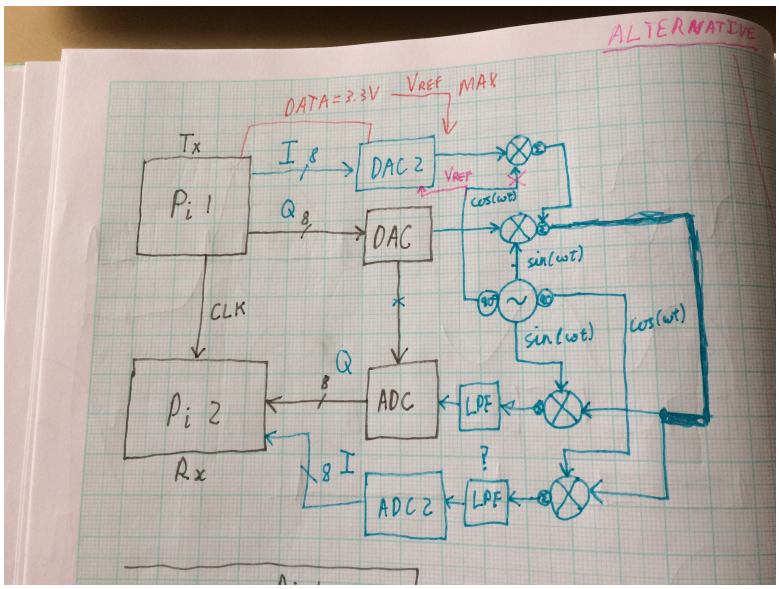


Figure 2.6: Layout for Quadrature Amplitude Modulation and Potentially Orthogonal Frequency Division Multiplexing

2.2.4 Parts Used

Need a diagramatic representation as well as an actual picture

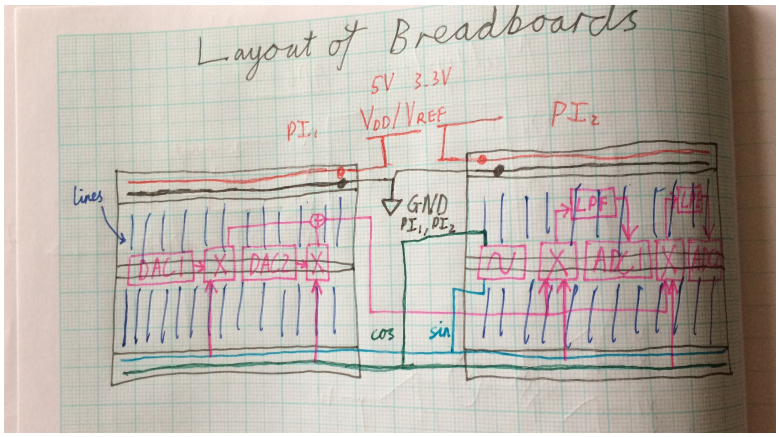


Figure 2.7: What the Test Bed Actually Looks Like in its Final Configuration

Include a section on pricing of the parts and comparison to actually used test beds

It is worth noting that there is a large variety of available options for each component of the test bed. Each possibility has certain advantages and disadvantages, and a lot of the options are not suitable due to the power requirements or ease of interfacing with the Raspberry Pi. As a result of this, the parts used in this project were the most suitable parts which could be found and successfully sourced. However, there may be more suitable chips available given more time or experience, and being aware of this would be useful if this project were to be extended and/or replicated. All parts could be replaced with minimal adjustment to the set-up and code.

Changes which would be made with hindsight, if components with the required qualities could be found, are as follows:

Continue to add to this list as you write the Architecture section - remove redundant information already discussed in earlier sections

- A number of chips used are surface mounted, requiring difficult soldering to solder pads, This would be useful if they were to be used on a printed circuit board for a final design, but on a prototyping breadboard, dual in-line packages would have been easier to use where available
- The Digital Analogue Converter was chosen for its easy interfacing with a microcontroller, but a voltage-output device would remove the need to use additional operational amplifiers at the output. It also had some issues with correct values and it is unclear if it was the implementation or the chip itself.
- Analogue Digital Converter
- The Quadrature Sinusoid Generator uses an oscillator chip which outputs 90^o out-of-phase square waves, and the used chip was the only simple one which did this. Ideally the outputs would already be sinusoidal (one quadrature chip or a sine generator and phase shifter) so that low pass filters with fixed frequency response could be omitted to make changing the carrier frequency purely software-dependent.

  Make sure I am consistent with use of Quadrature Sinusoid Generator vs Oscillator in report
- The multiplier is designed to operate around $10\,\mathrm{V}$, and so has a built in $10\,\mathrm{V}$ normalisation in the multiple which attenuates the signal and required re-amplification before transmission. A similar chip designed for lower voltages would be ideal.

  Fact check this
- Low Pass Filters

2.3 Programming

The Raspberry Pi is used for its low cost, ease of use, and the fact that it has programmable Input/Output (I/O) pins. The I/O pins can be programmed using different libraries in Python and C. The standard GPIO library which comes installed with Raspbian is RPi.GPIO for Python **??**.

Add bib reference to RPi.GPIO https://pypi.python.org/pypi/RPi.GPIO

This is used for the On-Off Keying part of the communications test bed. The Python library is slow however, and so a C library is used for the pin-level manipulation for all modulation schemes requiring multi-level outputs through Digital Analogue Converters. This is done both for the improved speed performance of the C library, and the capability of this library to output to multiple pins at once. Section 3.2.2 goes into a detailed investigation of the differences between these options.

CHECK - Do I need to change this from my name to an anonymous account if only candidate number is on cover page

All of the code and the report for the project are maintained on GitHub, and may be found at `https://github.com/CamEadie/4YP_PiCom`.

The transmitter and receiver code for all modulation schemes considered works from a single final version of the test bed. This consists of the Python transmitter *PiComTx_5_DAC.py* and receiver *PiComRx_5_DAC.py* files, and the executables compiled from C code for the transmitter *PiTransmit_3* and receiver *PiReceive*. The body of the main Python codes for each the transmitter and receiver is split into OOK and Advanced Modulation Schemes. On-Off Keying (Section 2.3.1) is the simplest form of clocked communication possible, and acts as a proof of concept for the Raspberry Pis as a test bed. It is implemented using a native Python list which stores '1's and '0's to represent the binary stream, and these are output using the native RPi.GPIO library. The OOK was added into the final code of the Advanced Modulation Schemes (Section 2.3.2) from previous versions retrospectively, as all of the Advances Schemes are implemented in the same code. This was done both to make it easier to conduct all tests through one program, and to adhere to the idea of Software Defined Radio being as change-independent as possible (Section 1.2).

Rephrase this and make sure the section referenced is as consistent with this comment as possible

The Advanced Schemes considered are 4-level Pulse Amplitude Modulation, 256-level Pulse Amplitude Modulation (used more for setting up the DAC and ADC, as differentiating between levels this precisely is not viable), 16-Quadrature Amplitude Modulation and Orthogonal Frequency Division

Multiplexing. The code also improves the data manipulation, includes image handling so images can be transmitted allowing for visualisation of the error rate etc. of the transmission, and implements a separate compiled C module for the actual transmitting and receiving of data.

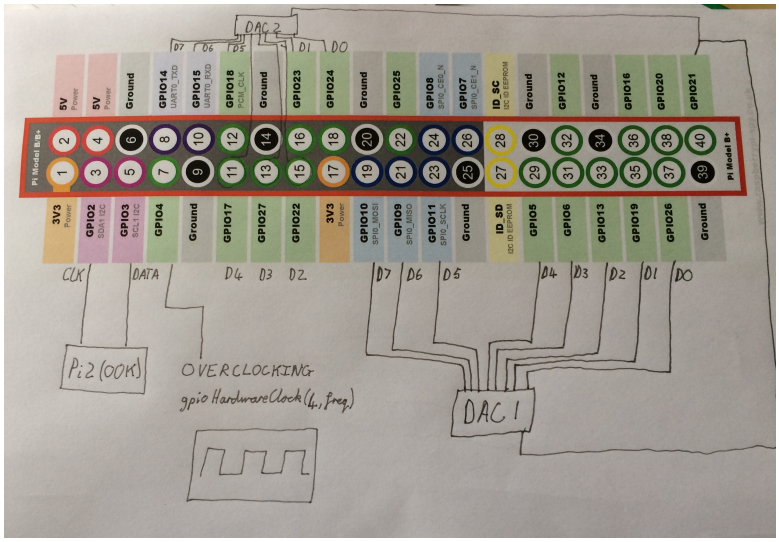


Figure 2.8: Pin Diagram of Connections to Raspberry Pi (reference to image source) - These are alterable fairly easily in the code so changes can be made with little overhead

2.3.1 On-Off Keying

On-Off Keying is a modulation scheme based on using V_{max} as '1' and $0\,\mathrm{V}$ as '0'. The transmit section of the test bed loops through the data list, outputting each value followed by a clock pulse using the sleep function. The speed and accuracy of the *GPIO.output()* and *sleep()* functions is covered in Section 3. The receiver uses the function *GPIO.wait_for_edge()* on rising edges of the clock pin to trigger reading of the data pin, and .

Earlier versions of this modulation scheme included a function *Prep_Binary_Data()* which added initial and final padding to the data of '1's to prevent missing timing of the beginning and end of transmission. The function also added a padding bit to the data when either value had been repeated a certain number of times (for example a '1' if '0' had been repeated 5 times in a row). This function was removed when *Encode_Error_Correction()* was included, as forward error correction was seen as a better way of avoiding these and other errors without including padding bits which could be difficult to remove if errors did occur in the code transmission. The channel coding used is syndrome decoding and is discussed more in Section 4.3.

2.3.1.1 Starting the Receiver

The receiver is started with the library Paramiko **??** which is used to make an SSH connection, and then to execute a command on the device which it connects to.

Missing reference - Paramiko

The host names of the transmitter and the receiver were changed to *raspberrypi1* and *raspberrypi2* to differentiate them, and their passwords changed to *rasPass1* and *rasPass2*. The devices are connected by an Ethernet cable so no connection to the internet is required, and so the transmitter uses the local host name of the receiver *raspberrypi2.local*.

The command to start the program is:

```
1  command = "sudo python3 " + \
2      "/home/pi/Documents/4YP_PiCom/4YP_PiCom_Receiver/PiComRx_5_DAC.py" + \
3      " " +str(mask_length) + " " +str(TRANSMISSION_TYPE)
```

Listing 2.2: Command Line to Start the Receiver

The super user call *sudo* is necessary for control of the GPIO pins in the receiver code. The command line argument *mask_length* is required by the receiver, and ensures that the amount of data received is the amount expected, and allows for the checking of deletions in the transmission. The other argument *TRANSMISSION_TYPE* is not required, but if passed it overwrites the transmission type in the receiver code to ensure that it is expecting the same modulation scheme that the transmitter is using.

The SSH connection is closed as soon as the command is executed so that both Raspberry Pis are not expending resources during transmission, and any readout to *stdout*, the standard output of the program under the connection is ignored. All logging of the execution of the receiver is instead appended to a Python list *LOGS*, and this variable is written to a file *LOGS.txt* at the end of execution. This includes all of the trivial to calculate results of the post-transmission analysis such as the number of bits/bytes received (whether there was any data lost). The transmitter also implements another SSH connection after transmission to read the *LOGS.txt* file to the *stdout* of the connection using the command *cat* so that the user doesn't need to change the screen (if only one screen available) between Raspberry Pis every time to see whether the transmission was successful.

2.3.2 Advanced Modulation Schemes

PAM - as mentioned in section 2.2.3 0-3

256 PAM is similar to 4PAM and is not included for space and because it's not viable in this setup especially with the problem mentioned in Section 3.3.

Make sure the right section does reference the fact that the DAC doesn't work exactly as it's supposed to and how I dealt with it

OFDM is also conceptualised here but due to the same problem, the continuous value issue mentioned above would need to be fixed before it could be implemented. OFDM could be used with blablabla... NumPy fftn, ifftn

Here we will definitely talk about the transmitter generally.

Here we will definitely talk about the receiver generally enough to separate the flow charts by a page.

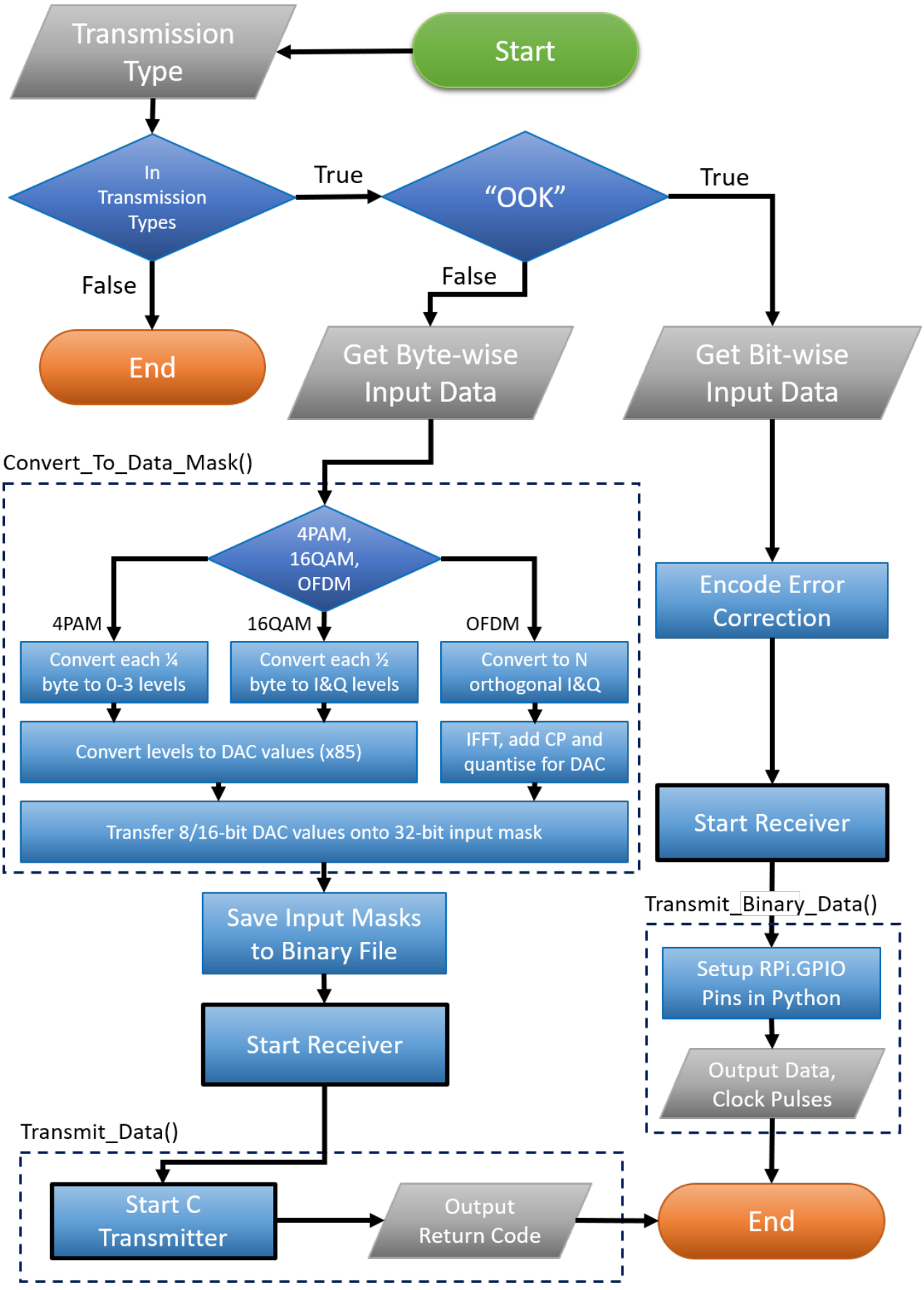


Figure 2.9: Flow Chart for Transmitter Code

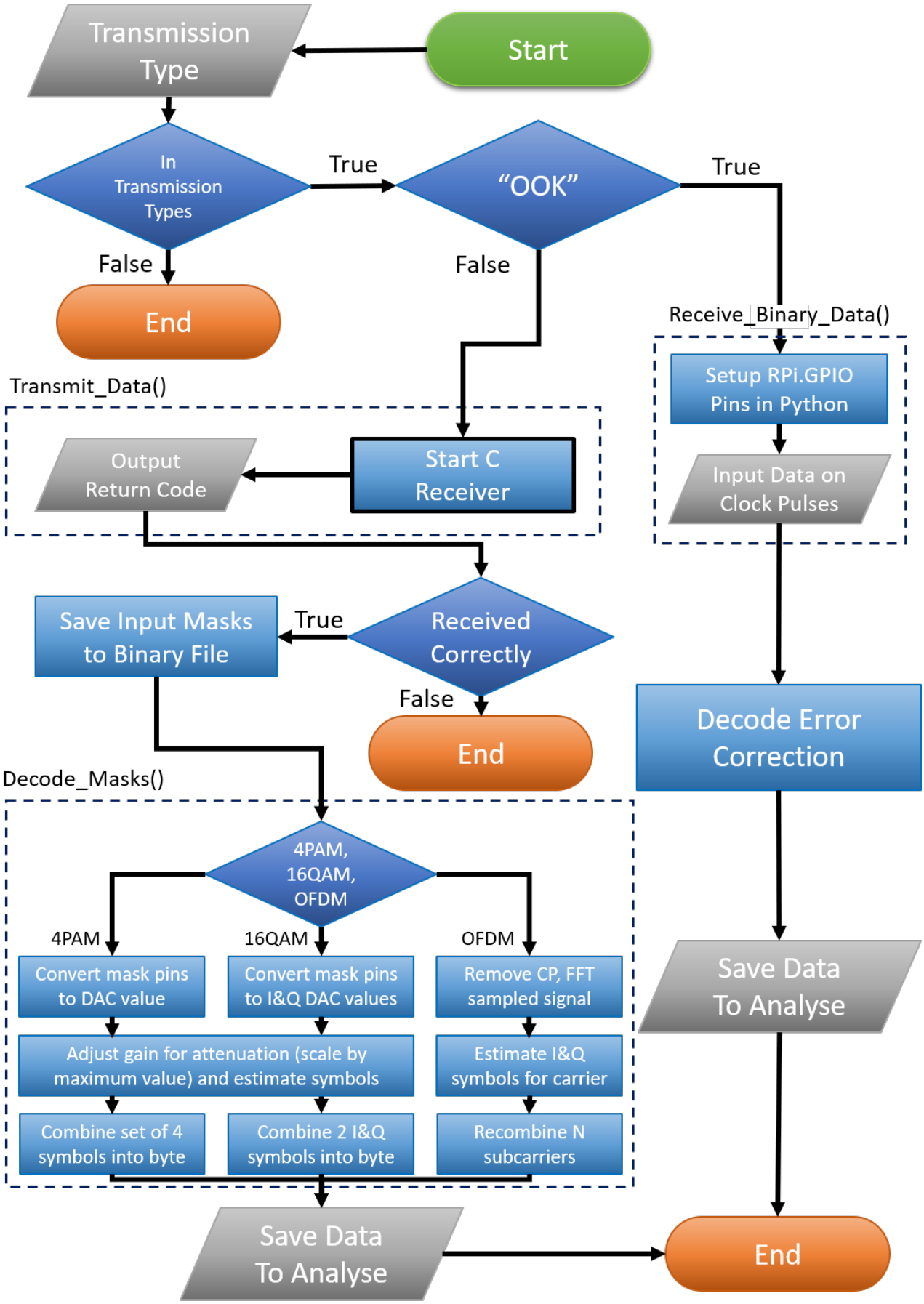


Figure 2.10: Flow Chart for Receiver Code

2.3.2.1 Data and Image Handling

NumPy and imageio.

2.3.2.2 C Transmitter and Receiver

Receiver: in Section 3.2 the options of data mask in on trigger vs read data then in on clock were compared and BLA was chosen as the more effective method.

Which method is more effective

/* PiTransmit now works independent of the choice of DAC pins. * It reads in the bit-mask (bits to set) from a binary file for speed, * which is calculated and saved in the Python before transmission, * and it also reads in inversion mask (bits to clear) so the calculation * isn't done during transmission. * This also means the Transmit program doesn't need to know how many * DAC's there are so works for all transmission schemes. */

The pins selected are based on the DAC1 choice from Figure 2.8 but could be used for any set of possible output pins.

Add a comment about the range of 2 to 28 not 0 to 32 in bank writing

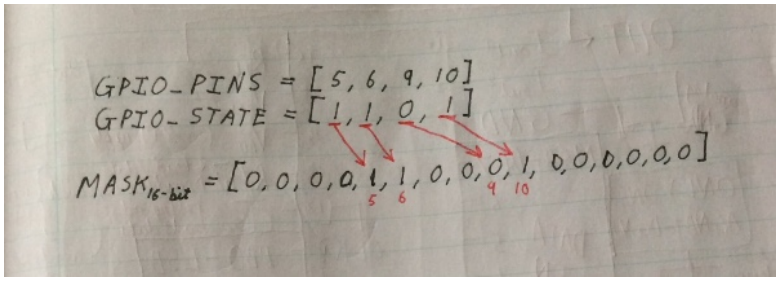


Figure 2.11: Explanation of the concept of mask expansion

Chapter 3

Electronic Testing

Assessing the capabilities of the test bed is split into three main sections. The first and second sections deal with the electrical and computational characteristics respectively of the Raspberry Pi, and the third deals with the components used in the test bed. The electrical section pertains to the GPIO pins and their physical capabilities, whereas the computational section considers how fast the pin values can be changed and read by the underlying code, and how most efficiently to achieve this. The final section discusses limitations to the methods discussed in the above sections, which are posed by the components' specifications.

3.1 Electrical Characteristics of the Raspberry Pi

The Raspberry Pi 3B has 40 pins, including eight ground pins, two $3.3\,\mathrm{V}$ power pins and two $5\,\mathrm{V}$ power pins. 26 of the remaining pins (BCM pins 2 to 27) are free to be used as General Purpose Input/Output (GPIO). The GPIO pins are powered by the same $3.3\,\mathrm{V}$ rail as the power pins at the same voltage, and so there is a maximum current that can be drawn from all of these pins as well as each GPIO individually.

The Embedded Linux Wiki [3] claims that the $5\,\mathrm{V}$ pins can provide a maximum current equal to, "The USB [power cable] input current (usually $1\,\mathrm{A}$) minus any current draw from the rest of the board." It also provides the maximum current to be drawn from all $3.3\,\mathrm{V}$ power pins as $50\,\mathrm{mA}$ (and this would apply to the power pins and the GPIO pins combined). However that specification was actually a design value for the original Pi according to Gert Van Loo, the hardware engineer of the first Pi's boards, designed to supply $3\,\mathrm{mA}$ for each of 17 pins for $\approx 51\,\mathrm{mA}$ total . There is no current-limiting on

the pins so they will attempt to drive the current pulled they stops working, however multiple sources including Gert Van Loo suggest that the maximum current that should be drawn from any one pin for safe operation is $16\,\mu A$ as this is what the electronics on each pad are rated for [4].

The maximum current which can be drawn is a useful detail. This is both in order to ensure that none of the attached components draw too much current, as well as to decide, along with the input impedances of the pins, whether or not the GPIO pins of one Pi can be connected directly to another without a protective resistor between them. The input and output impedances of the pins in various set-up modes are investigated in Table 3.1. All values were measured on a multi-meter and using the pin GPIO4 as pins 2 and 3 have permanent internal pull-up resistors whereas all of the rest have software-controllable pull-up or pull-down resistors.

Input/Output Mode	Impedance to Ground ($k\Omega$)
Input	48.93
Input with Pull Up Resistor	NEED TO CHECK
Input with Pull Down Resistor	NEED TO CHECK
Output	NEED TO CHECK
Raspberry Pi ON (No Mode)	NEED TO CHECK
Raspberry Pi OFF	53 550

Table 3.1: Table of GPIO Pin Impedances for Different Operating Modes

These impedances show that even the lowest input impedance of $48.93\,k\Omega$ will only draw $67.44\,\mu A$ of current from a $3.3\,V$ GPIO pin, significantly low enough that these pins can be connected directly together.

3.2 Computational Characteristics of the Raspberry Pi

Write Computing and Components Sections

time.time() in for loop - 3.32 us time.time() in while loop with i++ - 4.02 us = slower time.time() sequential - 2.81 us therefore for loop time - 0.51us

100Hz (excluding the 10ms) sleep(1/freq) - 104.13 us sleep(1/freq-timeDif) - 90.08 us == REMOVES DELAY OF 15us - GPIO AND A BIT SAME without sleep(1/freq) (just GPIO) - 5.18 us SAME without GPIO.output (just sleep(1/freq)) - 97.18 us == SLEEP IS THE INEFFICIENT FUNC-

TION LOSING ME TIME You can hard-code in the 104e-6 value to get offset of 5us instead of 104us but this is not ideal - need to compare to coding in c...

3.2.1 Maximum Frequency

Comparison of transmitter bank vs individual write and receiver read (callback) and read vars on clock pulse vs bank read on clock pulse – Also mention

3.2.2 Comparing Python and C

3.3 Characterising Components of the Test Bed

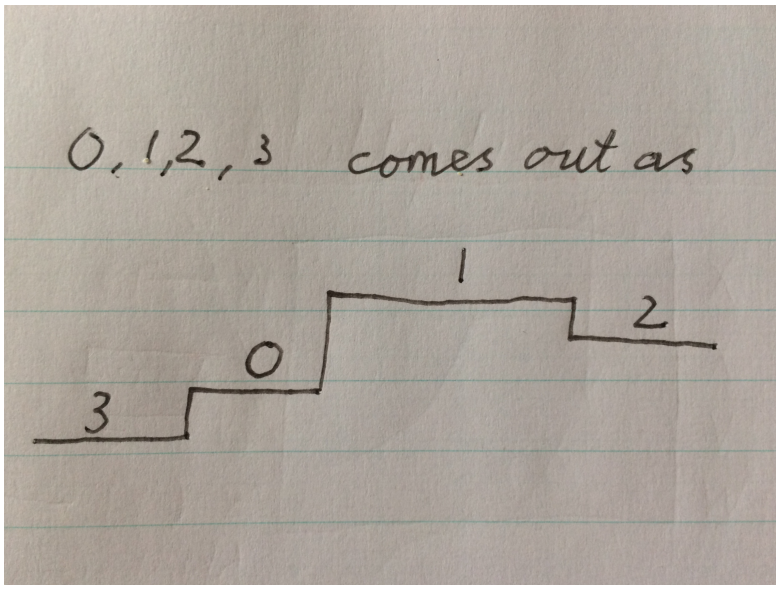


Figure 3.1: Non-continuous Output for Continuous Input Values of the DAC

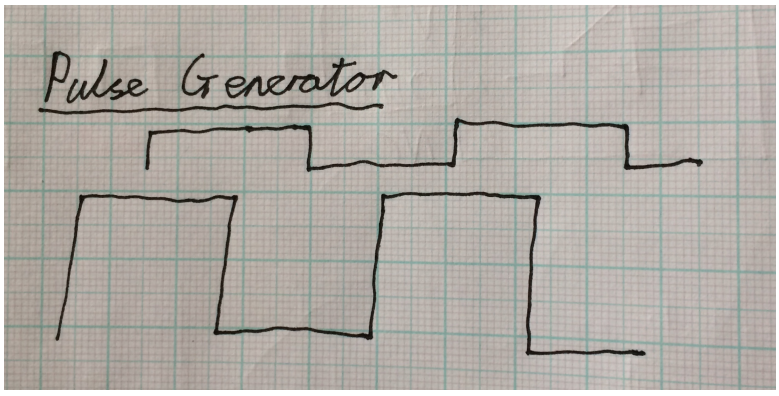


Figure 3.2: Output of pulse generator

Electrical Components:

- Analogue Digital Converter

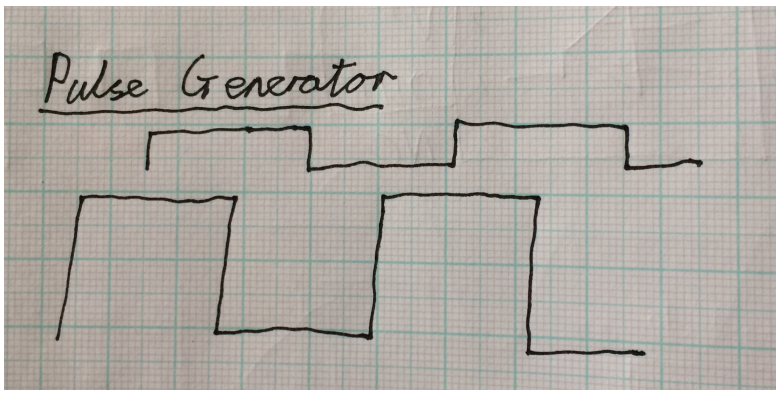


Figure 3.3: Output of pulse generator once filtered to be sine waves

- Digital Analogue Converter
- Quadrature Sinusoid Generator
- Multiplier/Mixer
- Low Pass Filters

3.3.1 Overclocking Components

The *pigpio* library has access to the hardware clocks of the Raspberry Pi. Specifically on the Pis used, it has the ability to set a hardware clock which is not reserved for system use to a specified frequency between 4.7 kHz and 250 MHz on pin 4, although the library documentation suggests that frequencies above 30 MHz are unlikely to work. There are certain components such as the Analogue Digital Converter and the pulse generator which work using an internal clock set by an external resistor, but which may be overclocked by an external clocking signal, and this functionality may be used. This would allow the frequencies of these devices to be defined in software with less reliance on external physical components. The pulse generator isn't as good an example as it still uses external filtering to generate sinusoidal outputs, but this is particularly relevant for the ADC as it requires eight clock pulses per conversion, and setting a hardware clock for (at least) eight times the transmission frequency is one way of ensuring this in software.

??.

Missing reference above and – Check rise time and max freq of this function below

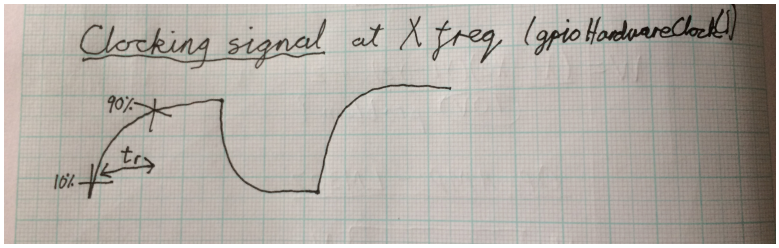


Figure 3.4: This is a high frequency view of the hardware clock for interest and rise-time (max frequency achievable actually)

Chapter 4

Communications Testing

s

Mention the fact that we had to deal with deletion channel vs error channel, clocked vs unclocked Justin's project - http://diwine-project.eu/public/publications.html

Testing Communications and shizniz.

Write Comms Testing

Easily Attainable: Construct a basic wired unidirectional communication test bed complete with a transmitter and a receiver. These units should be synchronised and an appropriate line code (i.e., baseband modulation scheme) should be exploited to convey test data from one device to another.

Medium Complexity: Characterise the performance of the test bed, identifying bandwidth limitations, noise characteristics, and reliability for different modulation and coding schemes. Test specific state-of-the-art modulation techniques recently published in the research literature. (These will be identified by the supervisor).

Advanced: Develop design enhancements that will enable the test bed to be extended to wireless scenarios, including RF and optical wireless systems. Implement these modifications if the budget permits.

4.1 SNR for Different Modulation Schemes

Make sure SNR computation is fully understood before capturing values

4.2 Error Rate

4.3 Channel Coding

Chapter 5

Conclusion

Here we shall have our conclusion.

Write Conclusion

Interesting stuff this has been. Summary, what has been achieved, recommendations for future work.

Make sure each figure is referenced explicitly in text and has a title

Bibliography

[1] Neil Dhir, Adam Roman Kosiorek, and Ingmar Posner. “Bayesian Delay Embeddings for Dynamical Systems”. In: *NIPS Timeseries Workshop*. 2017.

[2] James Mackenzie. *Headless Raspberry Pi Setup*. [Online: Accessed ...] Jan. 2, 2017. URL: https://hackernoon.com/raspberry-pi-headless-install-462ccabd75d0.

[3] Embedded Linux Wiki. *RPi Low-Level Peripherals*. [Online: Accessed ...] Jan. 19, 2017. URL: https://elinux.org/RPi_Low-level_peripherals.

[4] Raspberry Pi Stack Exchange. *What are the Electrical Specifications of GPIO pins?* [Online: Accessed ...] Jan. 11, 2017. URL: https://raspberrypi.stackexchange.com/questions/60218/what-are-the-electrical-specifications-of-gpio-pins/60219#60219.

Format for web pages should be:

Creator’s surname, creator’s first name. Title. Date of publication. Name of Institution associated with site. View date. ¡http://address/filename¿.

Appendix A

Risk Assessments

A.1 General Risk Assessment

Department of Engineering Science **4YP Risk Assessment** UNIVERSITY OF OXFORD

Description of 4YP task or aspect being risk assessed here: *(Read the Guidance Notes before completing this form)* PiCom: A Digital Communications Test Bed Based on Raspberry Pi – Use of Raspberry pi and wireless breadboard		4YP Project Number: 11410
Site, Building & Room Number: Thom Building, Electronics Lab, 5th Floor	Approx size of equipment/apparatus used or built (in metres): Height: …0.03……. Width…0.08….. Length……0.15…….	Photo provided? YES/NO
Assessment undertaken by: Cameron Eadie	Signed: Eadie	Date: 02/11/17
Assessment Supervisor: Justin Coon	Signed: Justin P Coon	Date: 2·11·17

Assessing the Risk*

You can do this for each hazard as follows:

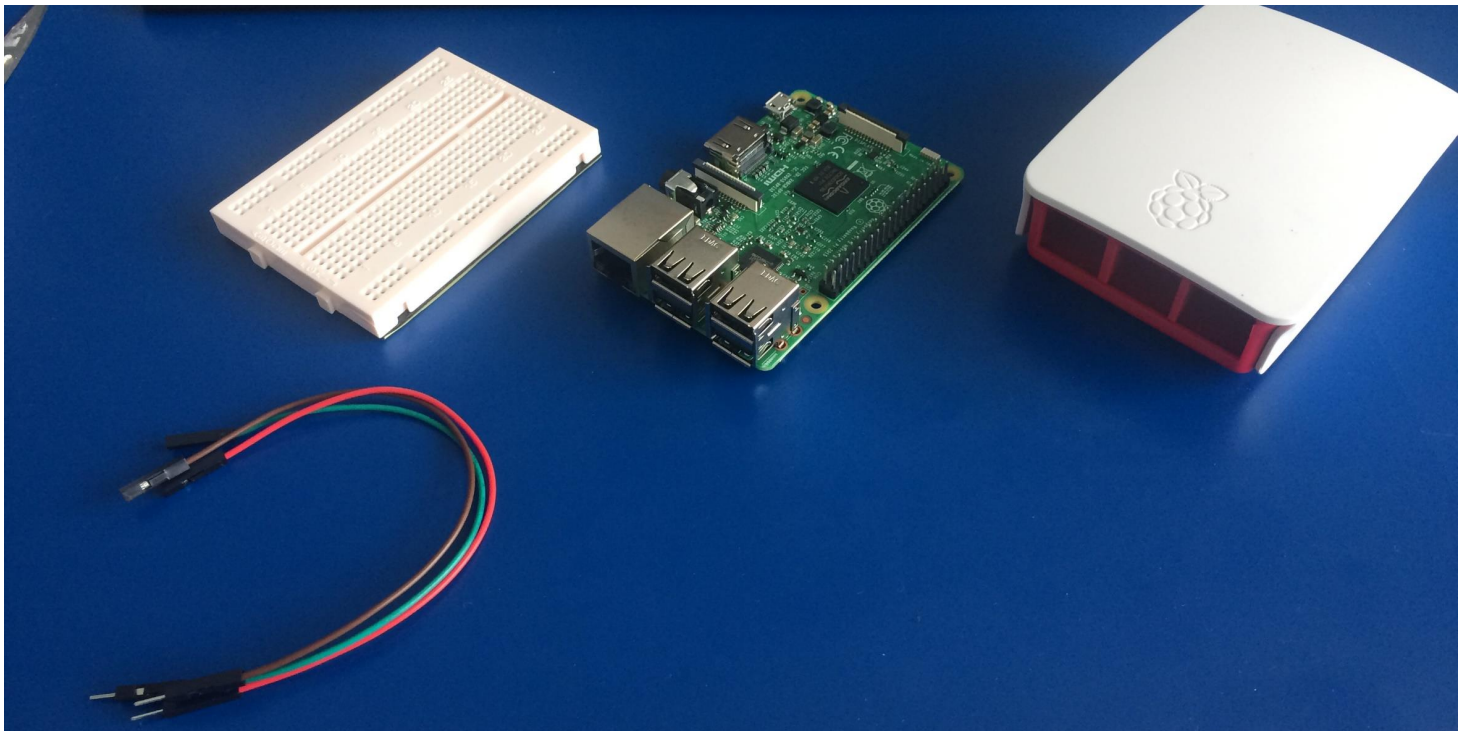
- Consequences: Decide how severe the outcome for each hazard would be if something went wrong (i.e. what are the Consequences?) Death would be "Severe", a minor cut to a finger could be regarded as "Insignificant".
- Likelihood: How likely are these Consequences to actually happen? Highly likely? Remotely likely, or somewhere in between?
- Risk Rating: Start at the left of the coloured Matrix. On your chosen Consequences row, read across until you are in the correct Likelihood column for the hazard in question. For example, an outcome with Severe consequences but with a Low probability of actually happening equates to a Medium risk overall. In this case "Medium" is what should be written in the Risk.

RISK MATRIX

CONSEQUENCES	LIKELIHOOD (or probability)			
	High	**Medium**	**Low**	**Remote**
Severe	**High**	**High**	**Medium**	**Low**
Moderate	**High**	**Medium**	**Medium/Low**	**Effectively Zero**
Insignificant	**Medium/Low**	**Low**	**Low**	**Effectively Zero**
Negligible	**Effectively Zero**	**Effectively Zero**	**Effectively Zero**	**Effectively Zero**

Hazard *(potential for harm)*	**Persons at Risk**	**Risk Controls In Place** *(existing safety precautions)*	**Risk***	**Future Actions identified to Reduce Risks** *(but not in place yet)*
Electrical shock from 3.3V powered I/O pins or open circuit board on the Raspberry Pi	Student using the Raspberry Pi	• Use Raspberry Pi within its case whenever possible and avoid contact with I/O pins • Remember to call GPIO.cleanup() function in python code to turn off any active I/O pins used by a program. Call this in the 'finally' section of a try-except block so that it always executes before program exit • Do not connect I/O pins directly together – use resistors to prevent potential short circuit • Subject power suppy of Raspberry Pi to Portable Appliance Test (PAT) at regular intervals	**Low**	

Hazard *(potential for harm)*	**Persons at Risk**	**Risk Controls In Place** *(existing safety precautions)*	**Risk***	**Future Actions identified to Reduce Risks** *(but not in place yet)*
Electrical shock constructing and prototyping electronics on wireless breadboard	Student prototyping electronics to connect to Raspberry Pi	• Never build or rearrange electronics on wireless breadboard while Raspberry Pi is powered and I/O pins are connected to the board • Ensure that both Pi's are grounded together • Don't touch electronics while I/O pin connections are active	**Low**	



A.2 Computer Risk Assessment

Department of Engineering Science

UNIVERSITY OF OXFORD

Supplementary Questions for 4th Year Project Students

Risk Factor	Answer	Things to Consider	Record details here
Has the checklist covered all the problems that may arise from working with the VDU?	☑ Yes ☐ No		
Are you free from experiencing any fatigue, stress, discomfort or other symptoms which you attribute to working with the VDU or work environment?	☑ Yes ☐ No	Any aches, pains or sensory loss (tingling or pins and needles) in your neck, back shoulders or upper limbs. Do you experience restricted joint movement, impaired finger movements, grip or other disability, temporary or permanently	
Do you take adequate breaks when working at the VDU?	☑ Yes ☐ No	Periods of two minutes looking away from the screen taken every 20 minutes and longer periods every 2 hours Natural breaks for taking a drink and moving around the office answering the phone etc.	
How many hours per day do you spend working with this computer?	☐ 1-2 ☑ 3-4 ☐ 5-7 ☐ 8 or more		
How many days per week do you spend working with this computer?	☐ 1-2 ☑ 3-5 ☐ 6-7		
Please describe your computer usage pattern	Use of laptop for a number of hours a day, most days, in department or at home		