The University of Oxford

Engineering Science

Fourth Year Project

PiCom: A Digital Communication Test Bed Based on Raspberry Pi

---

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

Here we shall have the abstract.

Contents

Chapter 1

Introduction

This is about the project in general, what it does/did, and then about the report, how its structured and what it explains. Not sure how much of the ”Conclusion-like” stuff should also be in here.

1.1 Motivation

Why is there a need for this, discuss the excessive cost of extant Software Defined Radio test beds, the low-cost alternative that is the Raspberry Pi. Use (PARAPHRASE) parts of Justin's description of the project as that sums it up nicely.

The section should do as you have organised: provide motivation for the project, define the project, give the background. Remember your interrogatives: why, what, how. This section relates mostly to why and what. The 'how' is the subject of much of the rest of the report.

> 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, what exists on the market and has this kind of project been attempted before to some degree by others.

Good. I expect to see plenty of citations to related work here. This will include not only Pi-based systems such as the paper I sent to you a while ago, but also NI kit, WARP demos, etc.

You will also need to lay out the basic communication-related tasks that you will focus on. I.e., explain the need to develop this test bed for investigating modulation and coding schemes. The degree of detail you give here is a matter of preference and flow.

1.3 Contributions

My contributions to the existing literature, what difference I have made (in the third person...) - Write once I have finished writing up tests and results.

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 General Purpose Input/Output (GPIO) pins which it can use to interface with stuff.

It is run using a Linux-based operating system called Raspbian which is available for download from the Raspberry Pi official website [1].

The main chip on the board is the Broadcom BCM2835. The Pi's pins can either be numbered as pin 1-40 going through them sequentially in layout, this is the BOARD numbering scheme, or they can be referred to by their Broadcom pin numbers with the BCM numbering scheme. The second scheme numbers the available GPIO pins in the range 2-27 (Pins 0 and 1 are reserved for system use).

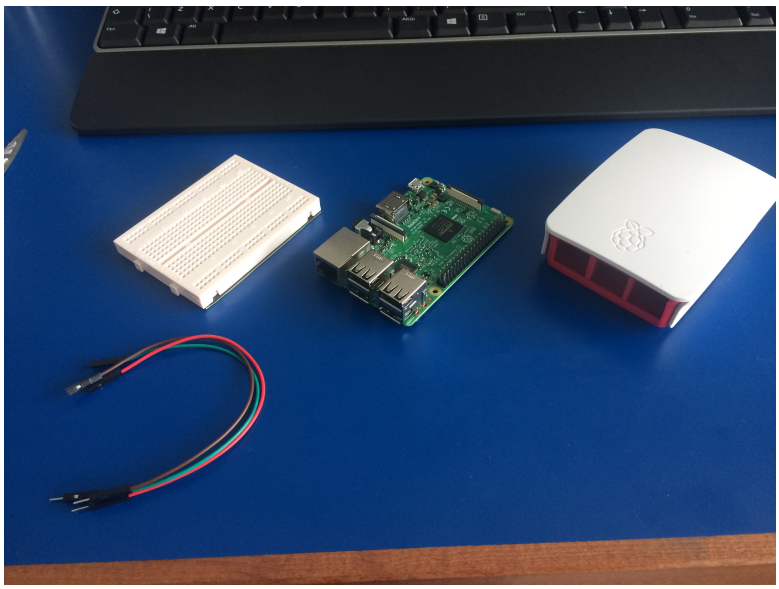


Figure 2.1: The Raspberry Pi

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 [3] for Windows can be used, with *pi@raspberrypi.local* as the host name so the IP address is not needed. The standard login is *username: pi* and *password: raspberry*. 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 [4] and the X Window can be started with the command *startx* if it is not already set to open the X Window on startup in the configuration settings. 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 3.6.4. 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 as to how they are downloaded when this list is run line by line.

IF I NEED TO AND HAVE TIME

```
sudo apt-get update

sudo apt-get install vpnc
sudo apt-get install libffi6 libffi-dev python-dev
sudo pip3 install cryptography paramiko

sudo apt-get install libatlas-base-dev
sudo pip3 install numpy imageio
sudo apt-get install pigpio

sudo apt-get install git
cd "<Path for the code e.g. /pi/home/Documents/>"
git clone https://github.com/CamEadie/4YP_PiCom

sudo apt-get update && sudo apt-get upgrade && sudo apt-get dist-upgrade

# Used in Transmit_Binary_Data(): import RPi.GPIO as GPIO
# ... This library is already included in the Raspberry Pi
# ... Only for OOK transmission which uses Python not C
# Used in Check_Input_Masks(): from RPiSim.GPIO import GPIO
# ... 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

Included are the following, as well as their dependencies:

- VPNC - VPN software used to access Oxford VPN in order to use the OWL Wi-Fi network [5]
- Paramiko - Python library used for SSH to start the receiver [6]
- NumPy - Python library used for easier array manipulation as well as interface with images [7]
- imageio - Python library used to read and save image files as NumPy arrays [8]
- pigpio - C library used to interface with the GPIO pins [9]
- Git - Version control software which also allows for the cloning of the project code to any device [10]

2.2 Test Bed Architecture

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 does not 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.

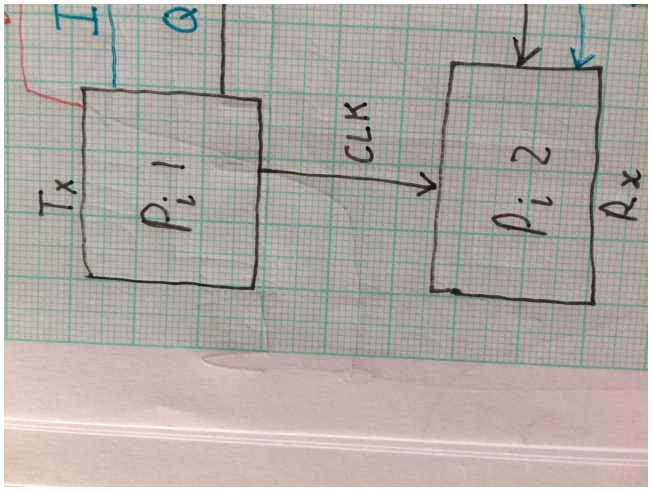


Figure 2.2: Test Bed Set-up For On-Off Keying

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. There are possible configurations in the data sheet using an inverting operational amplifier to produce the output, however due to the single supply and difficulties finding low-power operational amplifiers which provide rail-to-rail output for a 5V single supply, this was avoided where possible. 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.

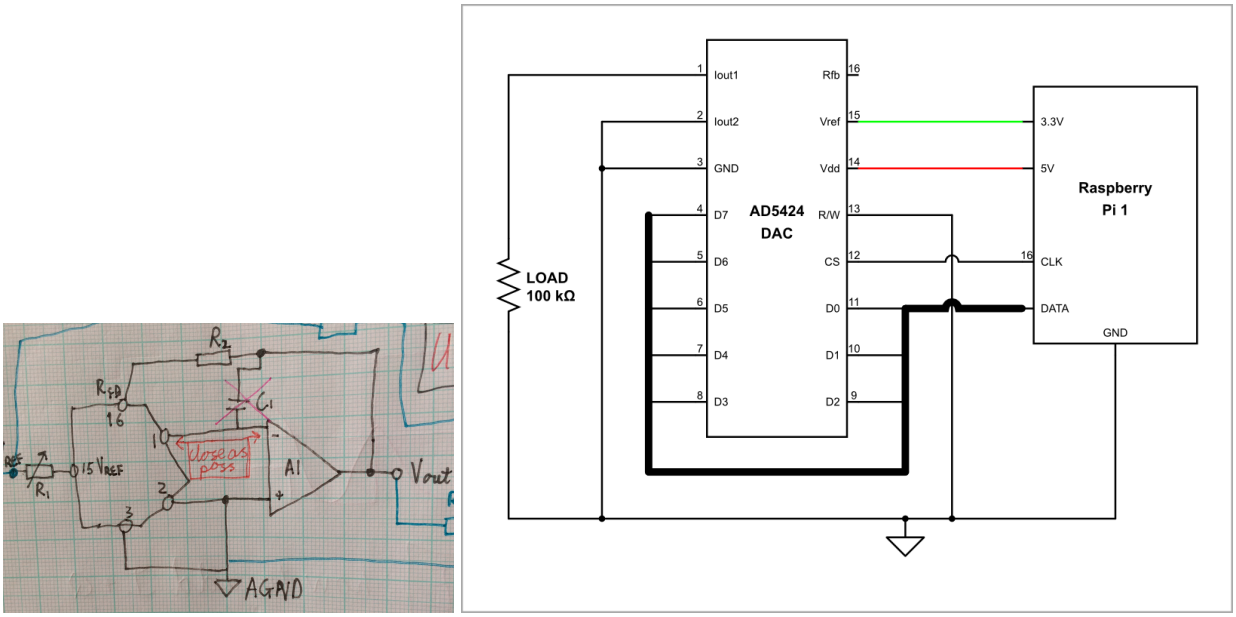


Figure 2.3: Layout of the Digital Analogue Converter

2.2.2 Analogue Digital Converter

The Analogue Digital Converter used is the ZN448,

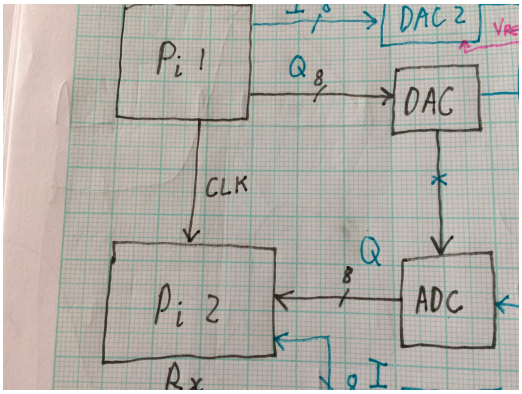


Figure 2.4: Layout of Test Bed for Pulse Amplitude Modulation

2.2.3 Multiplier

This section still needs the technical specifications and layout in the test bed of the multiplier.

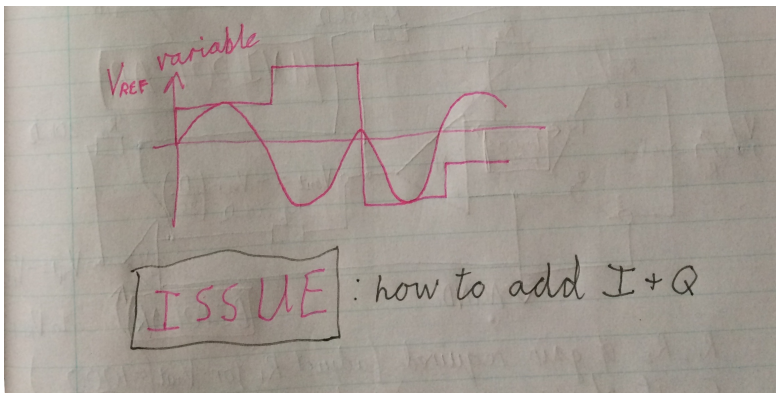


Figure 2.5: Four Quadrant Multiplication of input sinusoid using DAC

The Pi is not able to output negative voltages. As a result, Pulse Amplitude Modulation uses the range 0 to V_{max} rather than $-V_{max}$ to V_{max}. Similarly, Quadrature Amplitude Modulation uses a grid of I and Q values 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 $V_{max/2}$ using a voltage divider. This means that the sine and cos signals would be inverted when multiplied by the values 0 and 1 in the same way they would be for -3 and -1, and the transmitted signal will essentially be a normal PAM or QAM signal plus a DC component of $V_{max/2}$.

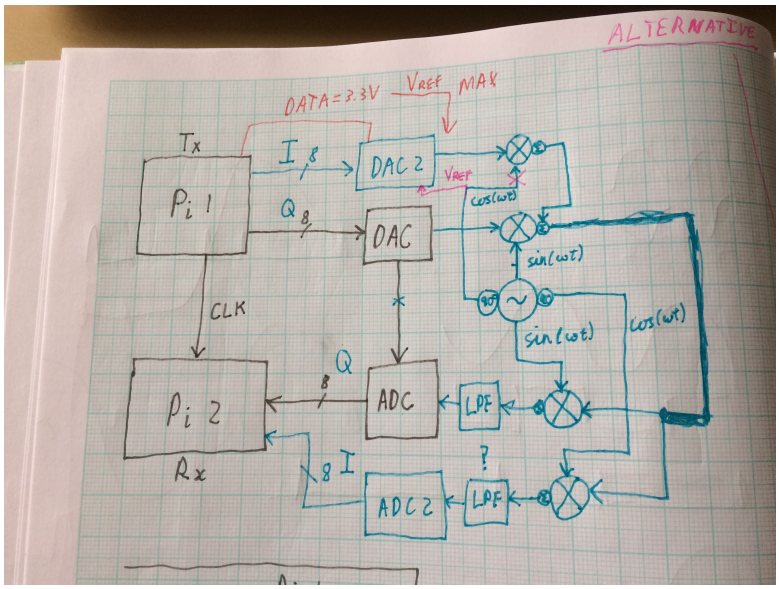


Figure 2.6: Layout of the Test Bed for QAM and Potentially OFDM

2.2.4 Parts Used

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

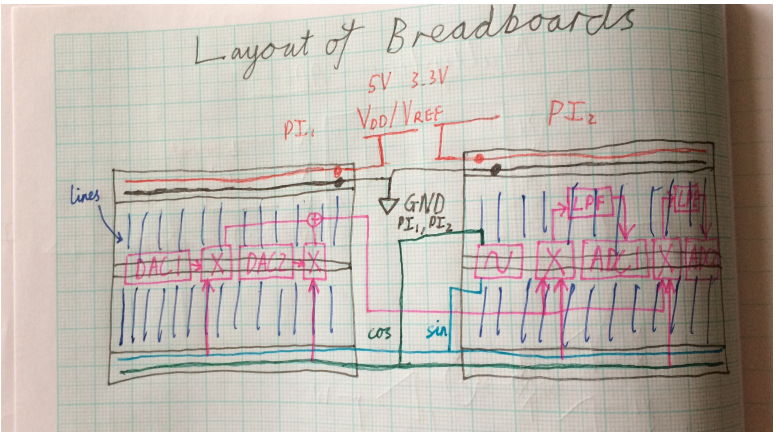


Figure 2.7: The Final Configuration of the Test Bed

sourced. However, there may be more suitable chips available given more time or experience to find them, 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 physical test bed layout and code.

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

- 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 or a drain resistor at the output. It also had some issues with correct value output (See Section 3.3).
- Analogue Digital Converter
- The Quadrature Sinusoid Generator uses an oscillator chip which outputs 90^o out-of-phase square waves, and the chip used was the only simple one which did this or anything close. 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 from the design, making the carrier frequency purely software-dependent. No obvious chip could be found for this and ones that did anything close were expensive.
- 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 (which is at lower voltages) and requires re-amplification before transmission - an oversight when choosing it. A similar chip designed for lower voltages

would be ideal.

- Low Pass Filters to remove high frequency signals in the demodulation part of QAM were again made using fixed-value components, if frequency response of a filter could be altered in software, all frequencies used (carrier and symbol) could be hardware independent.

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 either Python or C. The standard GPIO library which comes installed with Raspbian is RPi.GPIO for Python [11]. This is used for the On-Off Keying part of the communications test bed. Python is relatively slow however, and so a C library is used for the pin-level manipulation for all modulation schemes requiring multi-pin outputs to Digital Analogue Converters to generate multi-level signals. 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 the libraries used and the reasons for choosing C over Python for the advanced modulation schemes. 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 code. This comprises of the Python transmitter *PiComTx_5_DAC.py*, and receiver *PiComRx_5_DAC.py*, as well as the executables compiled from C code for the transmitter *PiTransmit_3*, and receiver *PiReceive*. The main function in the Python files for each the transmitter and receiver is split into On-Off Keying and Advanced Modulation Schemes. On-Off Keying (Section 2.3.1) is the simplest form of clocked communication, and acts as a proof of concept for the Raspberry Pis as a test bed. It is implemented using Python lists to store '1's and '0's to represent the binary stream. These are output using the native RPi.GPIO library.

The OOK part was added into the final code for the Advanced Modulation Schemes (Section 2.3.2) from previous versions retrospectively, as all of the Advanced Schemes are implemented in the same code. This was done to make it easier to conduct all communications tests through a single program interface. It also allows all of the modulation schemes to be used on essentially the same layout with minor changes to the hardware, and this adheres better to the idea of Software Defined Radio being as software-defined and hardware-change-independent as possible (Section 1.2).

The advanced modulation 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. Code for this section implements the data stream as bytes in a *NumPy* array

rather than as bits in a list. This has certain computational advantages but also allows for the integration of image handling with *imageio*. This means that images can be transmitted instead of random data, allowing for visualisation of the bit error rate etc. of the transmission. These schemes also use a separate compiled C module for the actual transmitting and receiving of data.

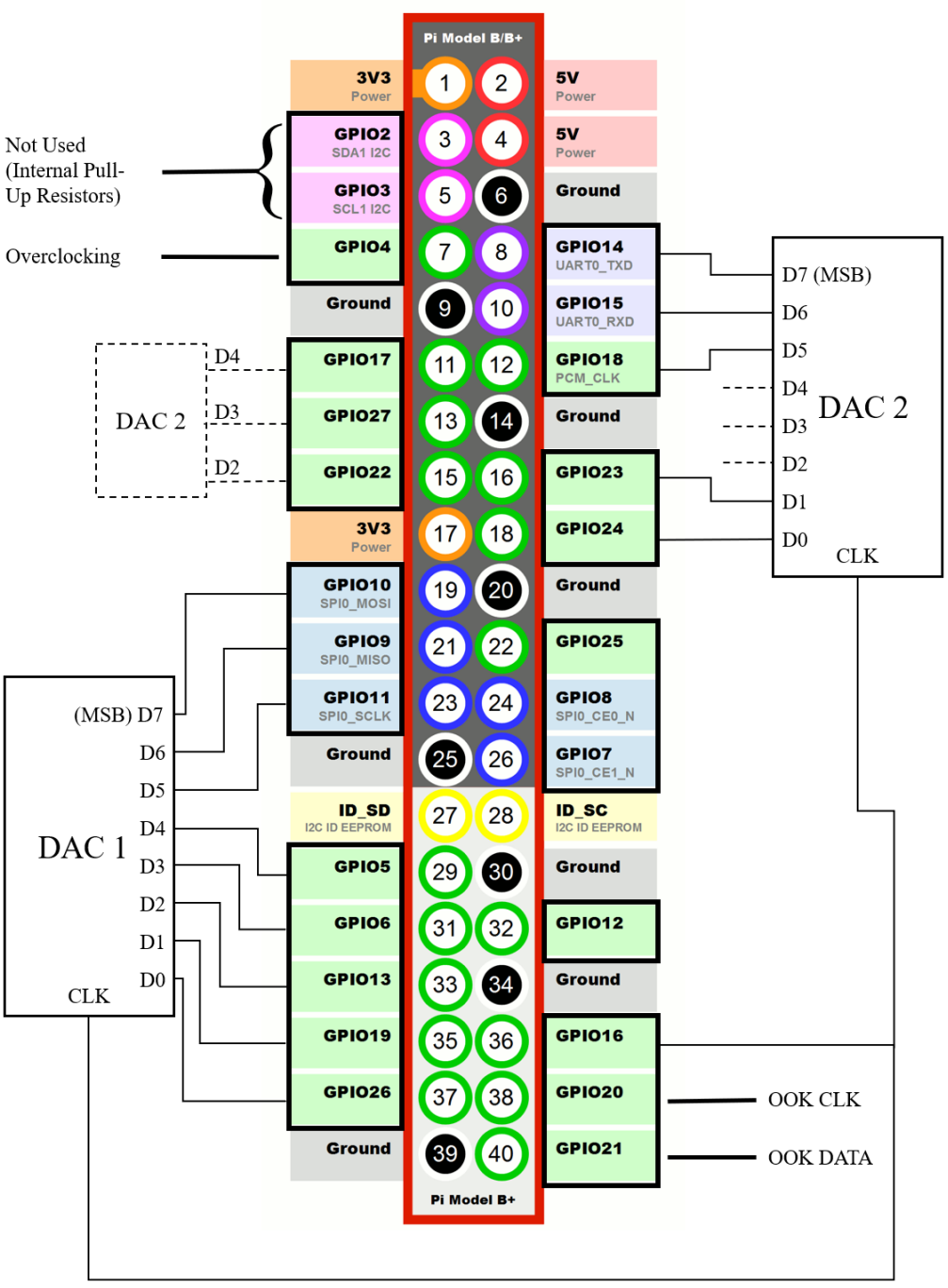


Figure 2.8: Pin Diagram of Connections to Raspberry Pi (reference to image source) - These are alterable fairly easily in the code so changes can be made with little overhead

Pins used currently for the OOK transmission as well as for the DACs are shown in Figure 2.8. The pins with black boxes around them are the accessible GPIO pins, and they are referred to by their

BCM number (GPIO# in the boxes) rather than their BOARD number (consecutive numbers in the circles). Of particular interest due to their differences are pins 2, 3 and 4. Pins 2 and 3 are used for the clock and data lines of *I2̂C* bus communication and so have internal pull-up resistors. Due to the fact that the test bed uses pins set with pull-down resistors, these pins aren't used. Pin 4 is the only pin on the Raspberry Pi B+ with access to a hardware clock which can be programmed for external use. It is thus used for overclocking external components which can be fed a clock signal, which is discussed in Section 3.3.1. The code defines the pins used for transmission as global variables at the start so that the rest of the code can be pin-independent. Excepting the the DAC clock pin (pin 16) which is defined in the C code, Listing 2.2 shows these pin definitions.

```
1    # OOK Pins
2    CLK_PIN = 20
3    DATA_PIN = 21
4    # DAC Pins
5    DAC_PINS_1, DAC_PINS_2 = [10, 9, 11, 5, 6, 13, 19, 26], \
6                             [14, 15, 18, 17, 27, 22, 23, 24]
```

Listing 2.2: Pins used for OOK and the DACs

2.3.1 On-Off Keying

On-Off Keying is a modulation scheme based on using V_{max} as '1' and 0 V as '0'. The transmit section of the test bed loops through the data list, outputting each value to the data pin 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. Using a function which is polling the clock pin as opposed to interrupt-driven solution is not ideal, but it was more easily written and so used for this early-stage modulation scheme.

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 may 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 [6] which is used to make an SSH connection, and then to execute a command on the device which it connects to. 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 can use 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 2.3: 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 over the connection is ignored. All logging 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 does not 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

This section will start by describing the advanced modulation schemes used, and will then it will go on to how the transmitter and receiver implement the schemes in code.

The modulation schemes used are 4-level Pulse Amplitude Modulation (4PAM), 256-level Pulse Amplitude Modulation (256PAM) and 16-Quadrature Amplitude Modulation (16QAM). The communications test bed could be implemented with Orthogonal Frequency Division Multiplexing (OFDM), however there were certain issues and time constraints which meant that this was not realised, but the details of how this would be extended are included. As mentioned in Section 2.2.3, the Raspberry Pi is unable to output negative voltages, and so all symbol values are in a positive voltage range between $V_{min} = 0V$ and $V_{max} = 3.3V$. Therefore, 4PAM and 16QAM use values from the range $\{0, 1, 2, 3\} \times \frac{V_{max}}{3}$, output from one and two DACs respectively. Similarly 256PAM uses 256 values from $\{0, 1, ..., 255\} \times \frac{V_{max}}{255}$

256 PAM is similar to 4PAM and is not included in the flow diagram for space and because it is not viable due to granularity issues, especially with the problem mentioned in Section 3.3 (need to be more specific). OFDM is also conceptualised here but due to the above problem, the continuous value issue mentioned above would need to be fixed before it could be implemented. OFDM could be used with NumPy fftn (N-dim FFT), ifftn (N-dim Inverse FFT).

TRANSMITTER BRUVS – Figure 2.9. Mention the QAM constellation used.

Here we will talk about the receiver generally and how it works - probably enough to separate the flow charts with a page of writing.

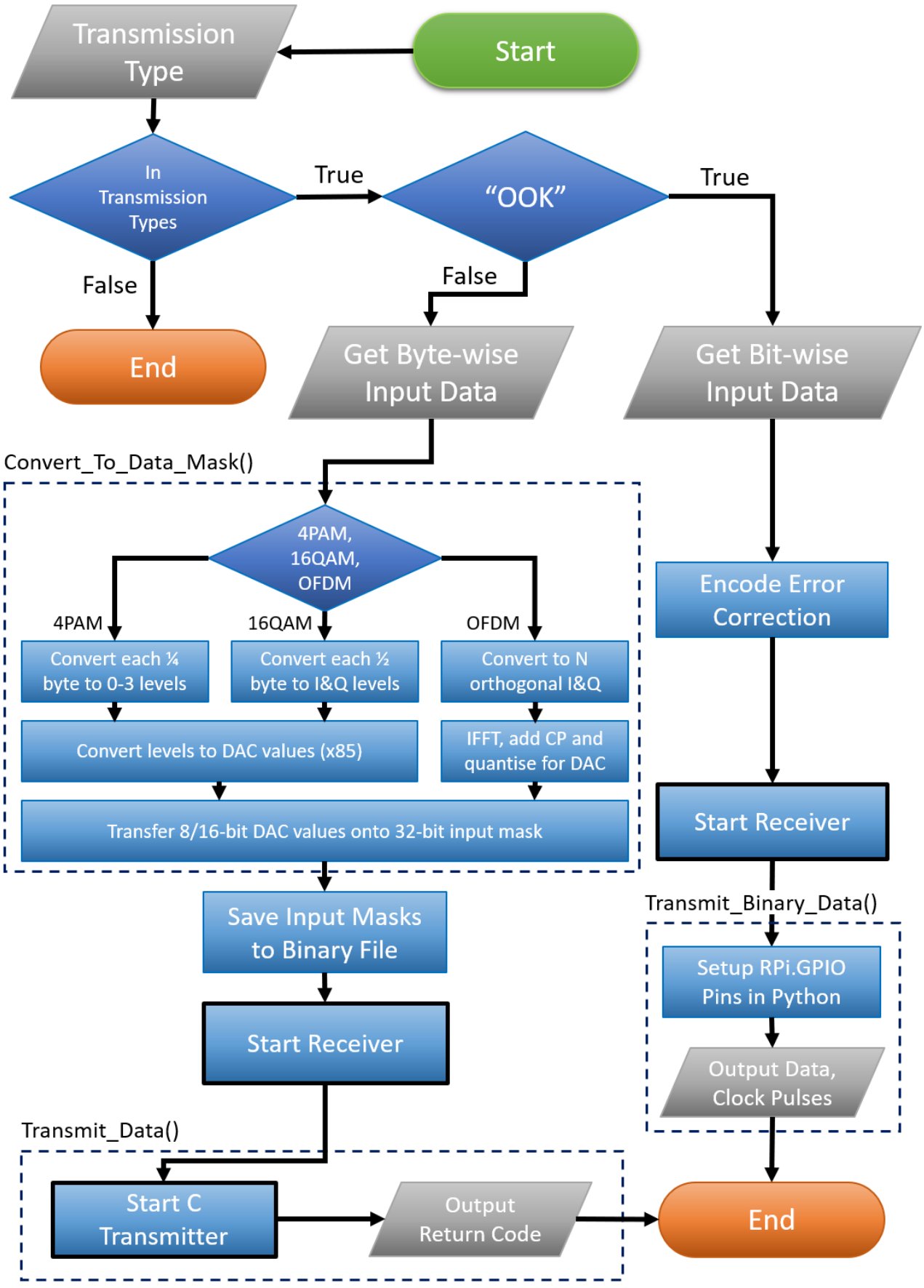


Figure 2.9: Flow Chart for Transmitter Code

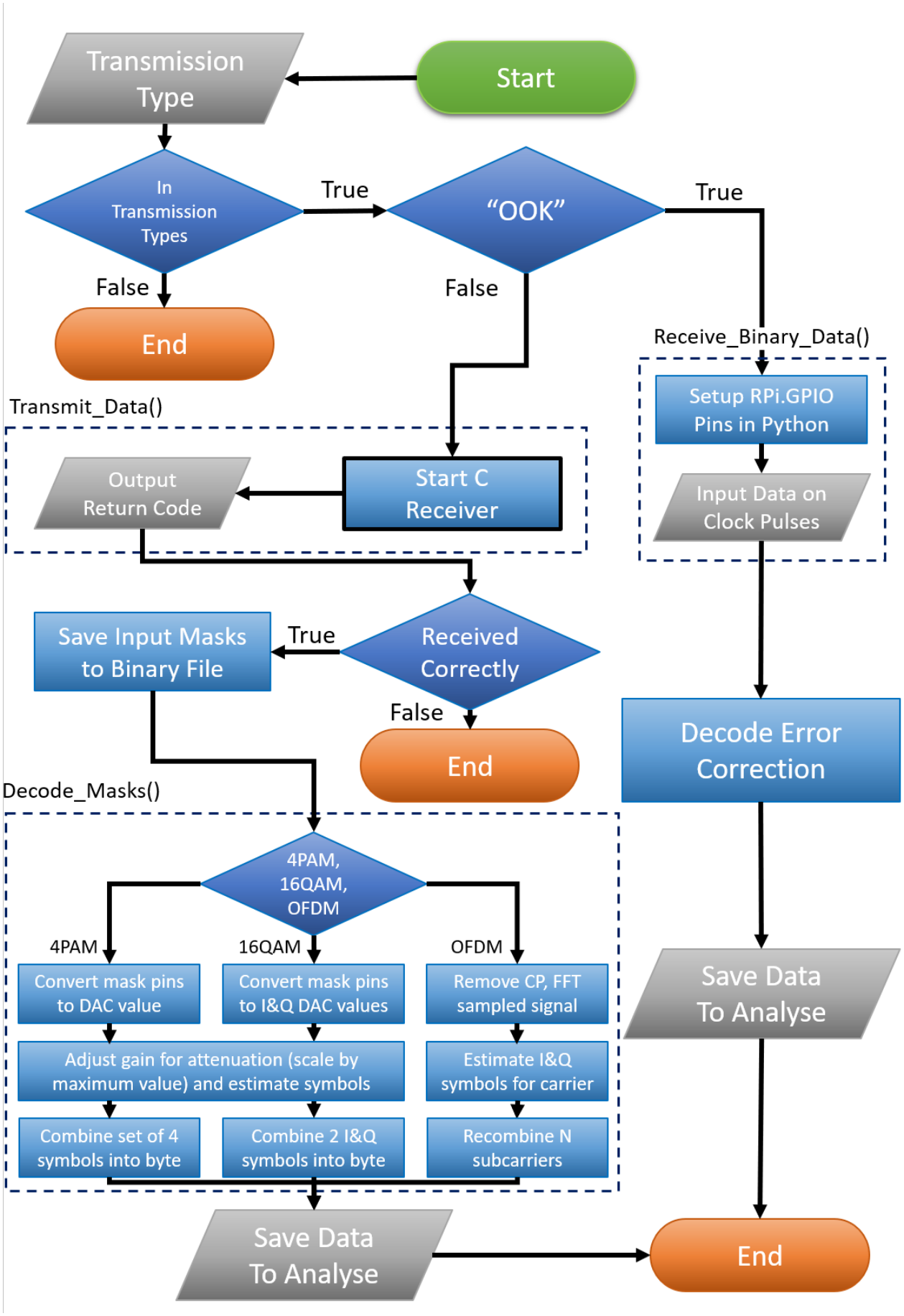


Figure 2.10: Flow Chart for Receiver Code

2.3.2.1 Data and Image Handling

NumPy and imageio, how they work and are used in the code, examples.

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.

PiTransmit 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 is not done during transmission. This also means the Transmit program does not need to know how many DACs 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. There will also be an explanation about what these bit-masks actually mean and refers to, it is exemplified in Figure 2.11.

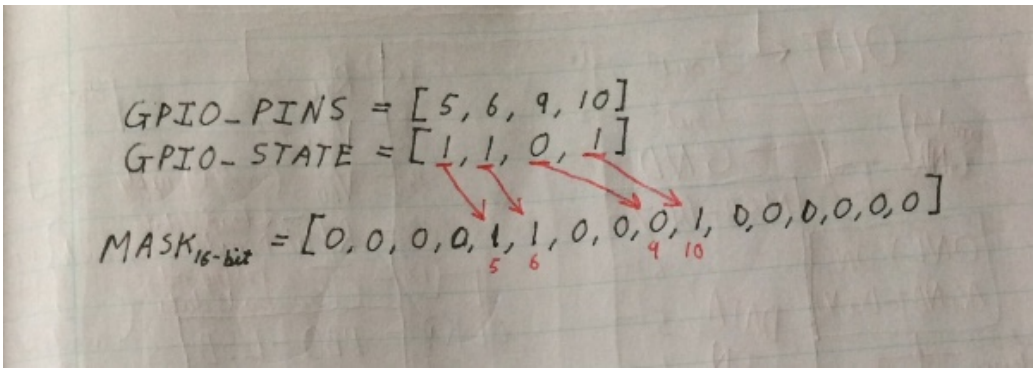


Figure 2.11: Explanation of the concept of mask expansion

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 can output low ($0\,\mathrm{V}$) and high ($3.3\,\mathrm{V}$) levels, and they are powered by the same $3.3\,\mathrm{V}$ rail as the power pins of the same voltage. As a result, there is a maximum current that can be drawn from all of these pins together as well as from each GPIO pin individually.

The Embedded Linux Wiki [12] 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 $50\,\mathrm{mA}$ (this would apply to the power pins and the GPIO pins combined). However that specification was actually a design value for the original Pi, designed to supply $3\,\mathrm{mA}$ for each of its17 pins for $\approx 51\,\mathrm{mA}$ total, according to Gert Van Loo, the hardware engineer of the first Pi's boards [13]. 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 $16\,\mathrm{mA}$ as this is the current to which the electronics on each pad are rated. Mosaic Documentation Web also has a page attempting to define the electrical specifications of the Raspberry Pi [14], and it also suggests that one shouldn't attempt to source or sink more than $16\,\mathrm{mA}$ on an output pin. The pins actually have a set drive strength from $2\,\mathrm{mA} - 16\,\mathrm{mA}$ in $2\,\mathrm{mA}$ increments which is set for a bank of pins and usually set to $8\,\mathrm{mA}$ but even a pin set to $2\,\mathrm{mA}$ drive strength and then loaded so as to draw $2\,\mathrm{mA}$ will not be damaged [15].

The maximum current which can be drawn from an output pin 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 multimeter 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 ($\mathrm{k}\Omega$)
Input	48.93
Input with Pull Up Resistor	NEED TO DOUBLE CHECK
Input with Pull Down Resistor	NEED TO DOUBLE CHECK
Output	NEED TO DOUBLE CHECK
Raspberry Pi ON (No Mode)	NEED TO DOUBLE 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 $48.93\,\mathrm{k}\Omega$ will only draw $67.44\,\mu\mathrm{A}$ of current from a $3.3\,\mathrm{V}$ GPIO pin, three orders of magnitude lower than the maximum current enough that these pins from one Raspberry Pi can be connected directly to those of another. The ground pins of Raspberry Pis must be connected together so that they share a common reference for the data levels.

3.2 Computational Characteristics of the Raspberry Pi

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 FUNCTION 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...

Table from interim report and two screen shots in particular.

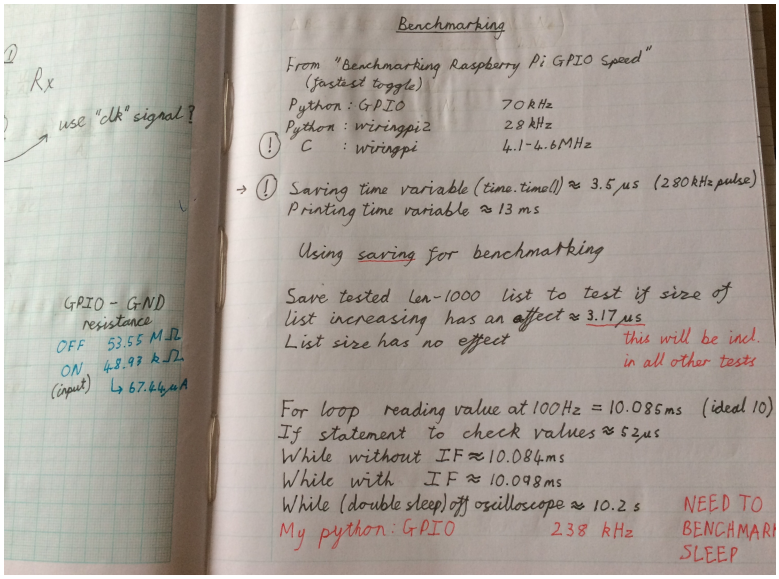


Figure 3.1: Benchmarking initial data collection

3.2.1 Maximum Frequency

3.2.2 Comparing Python and C

DIRECT MEMORY ACCESS,

3.3 Characterising Components of the Test Bed

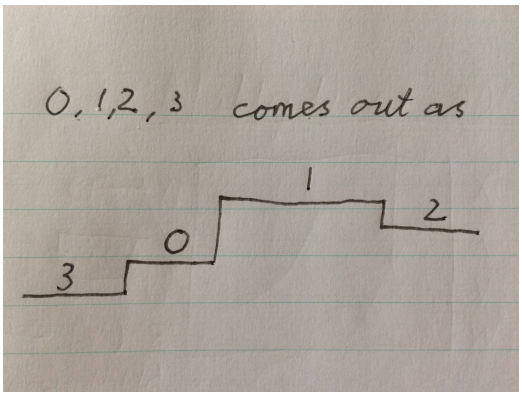


Figure 3.2: Non-continuous Output for Continuous Input Values of the DAC

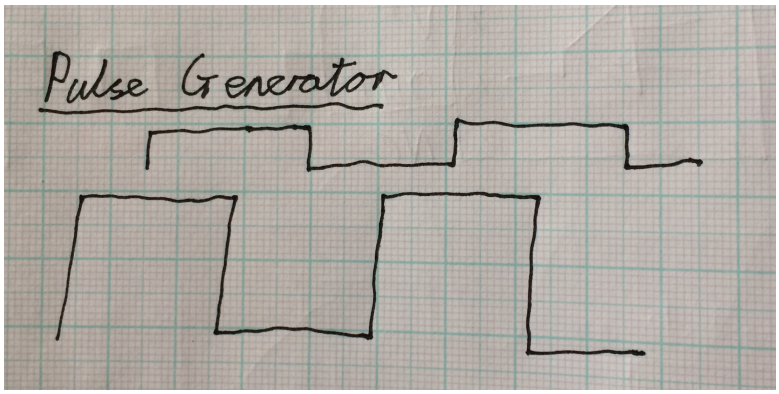


Figure 3.3: Output of pulse generator

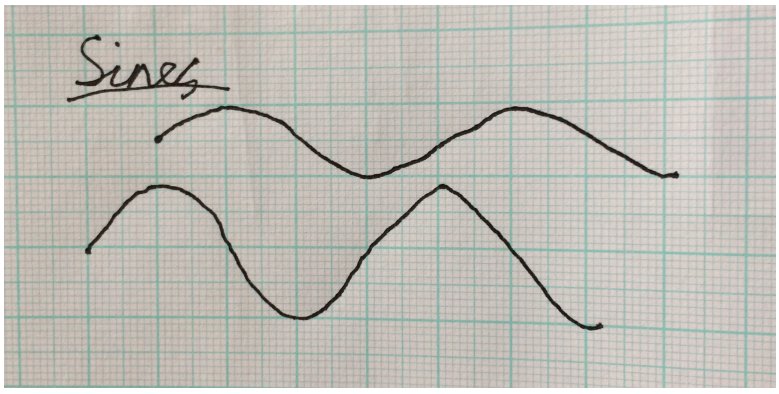


Figure 3.4: Output of pulse generator once filtered to be sine waves

Electrical Components:

- Analogue Digital Converter
- 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 [16].

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.

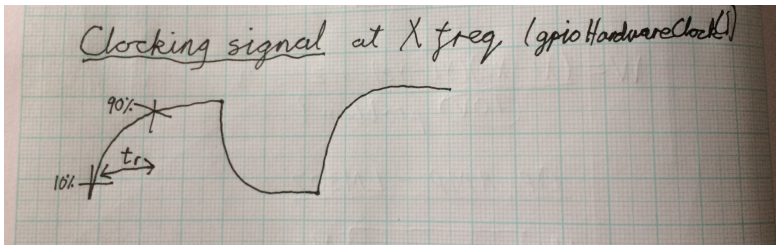


Figure 3.5: This is a high frequency view of the hardware clock for interest and rise-time (max frequency achievable actually) - will analyse screen grab from Oscilloscope and include in report

Chapter 4

Communications Testing

Testing Communications, remember project levels:

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

4.2 Error Rate

Bit error rate of transmission.

4.3 Channel Coding

Syndrome decoding - not a large amount of it covered but rather interesting stuff!

Chapter 5

Conclusion

Here we shall have our conclusion.

Summary, what has been achieved, recommendations for future work.

Bibliography

[1] Raspberry Pi Foundation. *Raspbian*. [Online: Accessed ...] 2018. URL: https://www.raspberrypi.org/downloads/raspbian/.

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

[3] Simon Tatham. *PuTTY: a free SSH and Telnet client*. [Online: Accessed ...] June 8, 2017. URL: https://www.chiark.greenend.org.uk/~sgtatham/putty/.

[4] RealVNC. *Download VNC Viewer*. [Online: Accessed ...] 2018. URL: https://www.realvnc.com/en/connect/download/viewer/.

[5] Eduard Bloch and Christian Lackas. *System Administration Utilities. VPNC*. [Online: Accessed ...] Nov. 23, 2016. URL: https://manpages.debian.org/testing/vpnc/vpnc.8.en.html.

[6] Jeff Forcier. *Welcome to Paramiko!* [Online: Accessed ...] 2018. URL: http://www.paramiko.org/.

[7] NumPy Developers. *NumPy*. [Online: Accessed ...] 2017. URL: http://www.numpy.org/.

[8] Almar Klein. *imageio. Python library for reading and writing image data*. [Online: Accessed ...] 2014. URL: https://imageio.github.io/.

[9] joan2937. *The pigpio library*. [Online: Accessed ...] Apr. 5, 2018. URL: http://abyz.me.uk/rpi/pigpio/index.html.

[10] Scott Chacon. *Git*. [Online: Accessed ...] URL: https://git-scm.com/.

[11] Python Software Foundation. *RPi.GPIO 0.6.3*. [Online: Accessed ...] Oct. 30, 2016. URL: https://pypi.org/project/RPi.GPIO/.

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

[13] 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.

[14] Mosaic Industries. *GPIO Electrical Specifications*. [Online: Accessed ...] URL: http://www.mosaic-industries.com/embedded-systems/microcontroller-projects/raspberry-pi/gpio-pin-electrical-specifications.

[15] Gert Van Loo. *GPIO Pads Control. Errata*. [Online: Accessed ...] Aug. 2, 2012. URL: https://www.scribd.com/doc/101830961/GPIO-Pads-Control2.

[16] joan2937. *pigpio C Interface. gpioHardwareClock*. [Online: Accessed ...] Mar. 20, 2018. URL: http://abyz.me.uk/rpi/pigpio/cif.html#gpioHardwareClock.

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:

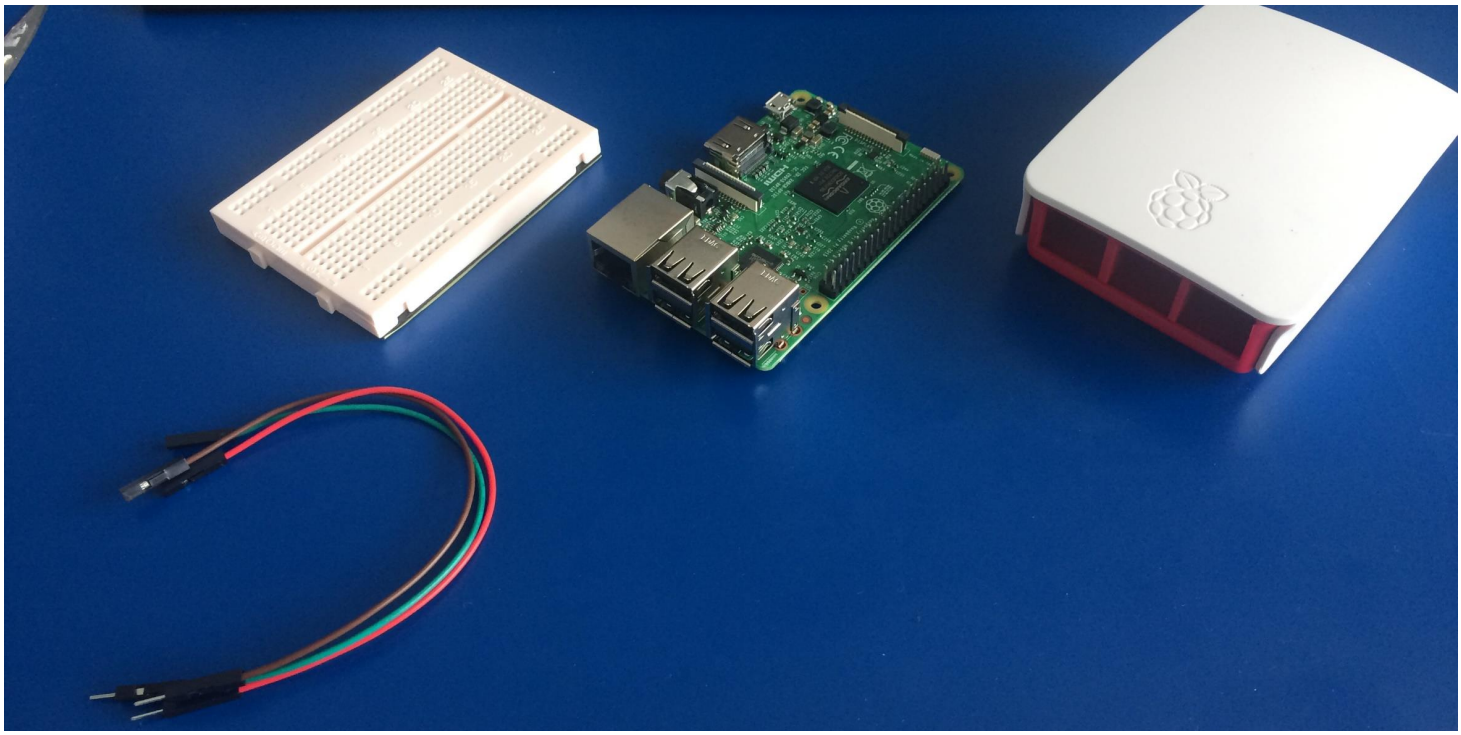
- Consequences: Decide how severe the outcome for each hazard would be if something went wrong (i.e. what are the Consequences?) Death would be "Severe", a minor cut to a finger could be regarded as "Insignificant".
- Likelihood: How likely are these Consequences to actually happen? Highly likely? Remotely 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		