The University of Oxford

Engineering Science

Fourth Year Project

PiCom: A Digital Communication Test Bed Based on Raspberry Pi

---

Candidate Number:

CANDIDATE NUMBER OR NAME - ??PAGE COUNT??

Supervisor

Justin Coon

Layout of title page

Trinity Term, 2018

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

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].

Chat chat chat.

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

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

1.3 (My) Contributions

```python
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 **?? ??**.

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 **??**.

MISSING REFERENCE - How to setup Raspbian https://hackernoon.com/raspberry-pi-headless-install-462ccabd75d0

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. 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 is much easier. In the test bed the Transmitter Pi will be connected to a screen and peripherals in command line, 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. 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, these can 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

```
sudo apt-get update

sudo apt-get install vpnc
sudo apt-get install libffi6 libffi-dev python-dev
sudo pip3 install cryptography paramiko
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
# ... Only for OOK transmission which uses Python not C
# Used in Check_Input_Masks(): from RPiSim.GPIO import GPIO
# ... Only works in Windows to check masks work
```

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:

- VPNC - VPN software used to access the Oxford VPN to use the OWL Wi-Fi network
- Paramiko - Python library used for SSH to start the receiver program, and its dependencies
- NumPy - Python library used for easier array manipulation as well as interface with images
- imageio - Python library used to read and save an image file as a NumPy array
- 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

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 2.1). 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.

Missing figure

Test Bed Set-up For On-Off Keying

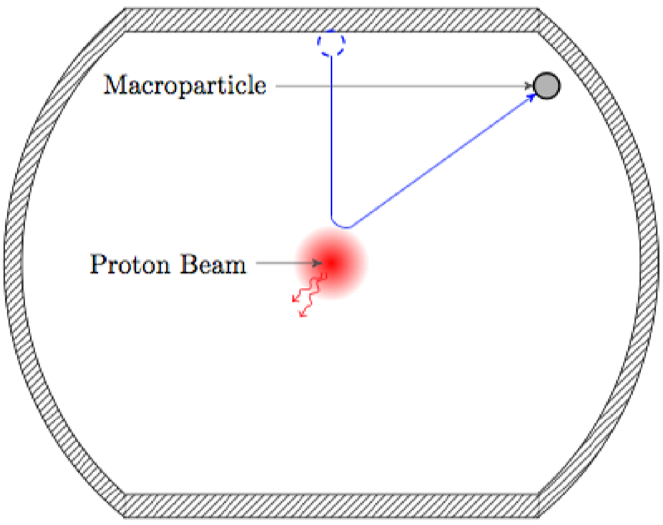

Figure 2.1: Test Bed Set-up For On-Off Keying

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 2.2.1.

Missing figure

Layout of Analogue Digital Converter

2.2.2 Analogue Digital Converter

The Analogue Digital Converter used is the ZN448,

2.2.3 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 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.

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

- 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 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.

2.3.1 On-Off Keying

and *GPIO.wait_for_edge()* on the clock pin to trigger reading of the

DOC LINE FROM TRANSMITTER

```
 '''
OOK has been integrated from the previous versions, uses different pins and runs
using RPi.GPIO and python arrays instead of a C executable (pigpio) and NumPy arrays

256PAM is too finely spaced to use successfully but is nice for setup

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
 '''

/* 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.
 */
```

Listing 2.2: Transmitter Comment

2.3.1.1 Starting the Receiver

Paramiko

2.3.2 Advanced Modulation Schemes

Definitely have a full page flow chart for the transmitter and receiver code

2.3.2.1 Data and Image Handling

2.3.2.2 C Transmitter and Receiver

Chapter 3

Electronic Testing

BLABLABLA.

Write Electro Testing

3.1 Electrical Characteristics of the Raspberry Pi

3.1.1 Maximum Frequency

3.1.2 Impedance

3.2 Comparing Python and C

3.3 Characterising Components of the Test Bed

Electrical Components:

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

3.3.1 Overclocking Components

Chapter 4

Communications Testing

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

4.2 Error Rate

4.3 Channel Coding

Chapter 5

Conclusion

Here we shall have our conclusion.

Write Conclusion

Bibliography

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