

UNIVERSITY OF CAPE TOWN



EEE3096S

EMBEDDED SYSTEMS II

Lab Handbook

Keegan Crankshaw

17 September 2019

Introduction

Kare has been taken to ensure that all the information in this manual is correct and accurate. Careless mistakes may have been made, so please email the TA if you find any.

Reality is changing rapidly. All around us, embedded systems are becoming more prevalent. Now more than ever it is our job as ES engineers to try and make the world a better, safer and more inclusive place. Keep working hard, and good luck with the course!

Contents

1	Raspberry Pi 3B+	7
1.1	Board Modes	8
2	Setting up your Pi	8
2.1	Prerequisites	9
2.2	Prepare the SD Card	9
2.3	Install and configure Raspbian	9
3	The Unix Shell	11
4	Text Editors	13
4.1	GUI Based Text Editors	13
4.1.1	Notepad++	13
4.1.2	Geany	14
4.2	Terminal Based Text Editors	15
4.2.1	Nano	15
4.2.2	Other Terminal Editors	16
5	Programming IDEs	16
5.1	Visual Studio Code	16
5.2	PyCharm	17
5.3	CLion	17
5.3.1	CLion on Windows	18
5.3.2	CLion on Ubuntu	19
6	Git	20
6.1	A Quick Git Get Go	21

6.1.1	Creating a GitHub Account and Configuring your Computer	21
6.1.2	Creating a New Project	21
6.1.3	Linking GitHub and your Local Project	22
6.1.4	Understanding .gitignore	22
6.1.5	Fetching an Existing Git Repository	22
7	L^AT_EX	22
7.1	Overview	22
7.2	Using Overleaf	23
7.3	Useful Latex Tools	24
7.3.1	Overleaf	24
7.3.2	Detexity	24
7.3.3	Tablesgenerator	24
7.3.4	Citation Machine	24
8	Networking on the Pi	24
8.1	A Brief Overview of Networks	24
8.2	Assigning a Static IP to the Pi	24
8.3	Assigning a Static IP to your Computer	25
8.3.1	Windows	25
8.3.2	Ubuntu	26
8.4	Ensuring connectivity	27
9	Advanced Connectivity Options	28
9.1	Setting a static IP Through Config	28
9.2	Providing your Pi with wireless Internet Access	28
9.2.1	Using WiFi to Ethernet passthrough to give your Pi internet access .	28
9.2.2	Connecting to Eduroam - Raspbian Buster (Raspbian Site)	30

9.2.3	Connecting to Eduroam - Raspbian Stretch (Vula)	32
9.3	Debugging Connections	33
9.3.1	Note for Windows Users	33
9.3.2	WiFi	33
9.3.3	"Failure resolving URLs or "unknown host"	33
9.4	Configuring the Pi to Act as an Access Point	34
9.5	TTL over USB	36
10	SSH	37
10.1	Enabling SSH	37
10.2	Using SSH	37
11	VNC	38
12	SCP	39
12.1	Using SCP on Windows	40
13	FTP	40
14	Programming in Python	41
14.1	Introduction	41
14.2	The RPi.GPIO Library	42
14.3	Basic IO	42
14.3.1	Digital Logic	43
14.3.2	Analog	43
14.3.3	Inputs	43
14.4	Communication Protocols	45
14.4.1	I2C	45
14.4.2	SPI	46

14.5	Using Threads	48
14.5.1	The Threading Library	48
14.5.2	Basic Thread Usage	49
14.5.3	Timed Thread Usage	49
15	Programming in C/C++ - The WiringPi Way	50
15.1	Introduction	50
15.2	WiringPi on the CommandLine - gpio utility	50
15.3	Working with WiringPi	50
15.4	Core Functions	51
15.5	Analog	52
15.6	PWM	52
15.6.1	Hardware PWM	52
15.6.2	Software PWM	53
15.7	Interrupts	54
15.7.1	Software Debounce	54
15.8	Communication Protocols	54
15.8.1	I2C	54
15.8.2	SPI	56
15.9	Timing	57
15.10	Using Threads	57
16	Toolchains Compilation and MakeFiles	58
16.1	Toolchains	58
16.2	Compilation	59
16.3	Make Files	59
16.4	Cross Compilation	60

16.4.1	Requirements	60
16.4.2	Using cross compilation	60
16.4.3	Moving the files to the Pi	61
16.5	JetBrains CLion	61

1 Raspberry Pi 3B+

This course uses the Raspberry Pi 3B+. The pinout for this device is shown below. Note that some pins have special uses. These pins may not be able to be used as GPIO. Note that the header has a notch to indicate pin 1. It's also important to note the difference board numbering and BCM numbering. Board numbering is the pin count on the headers, i.e. the number shown in the circle in the image. BCM stands for Broadcom SOC channel - and refers to the GPIO number in the descriptions next to number in the circle. For example, board number 3 is GPIO 2, and board number 33 is GPIO number 13. Some pins have special functions. You'll learn about these functions as the course progresses.

You can only use one board numbering system (BOARD or GPIO) per project, and you usually need to configure this using a method made available in the library.

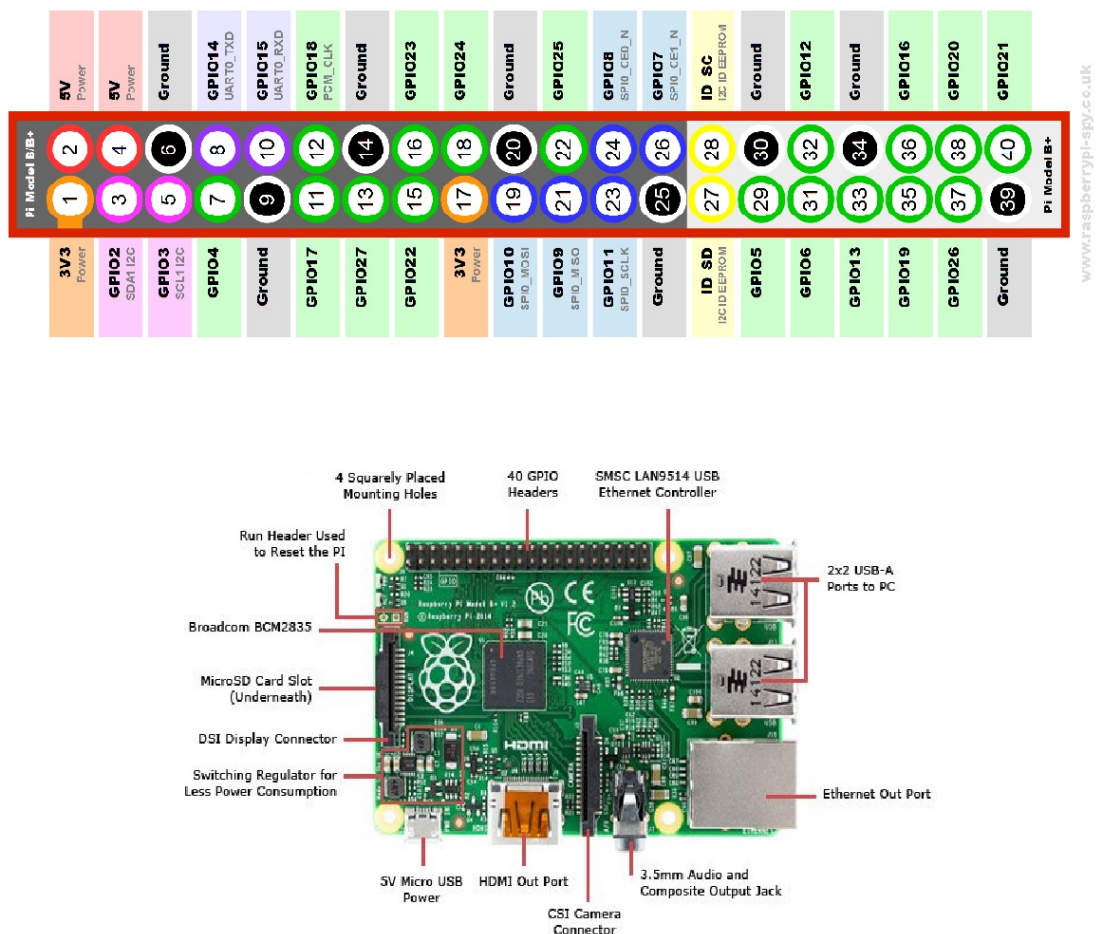
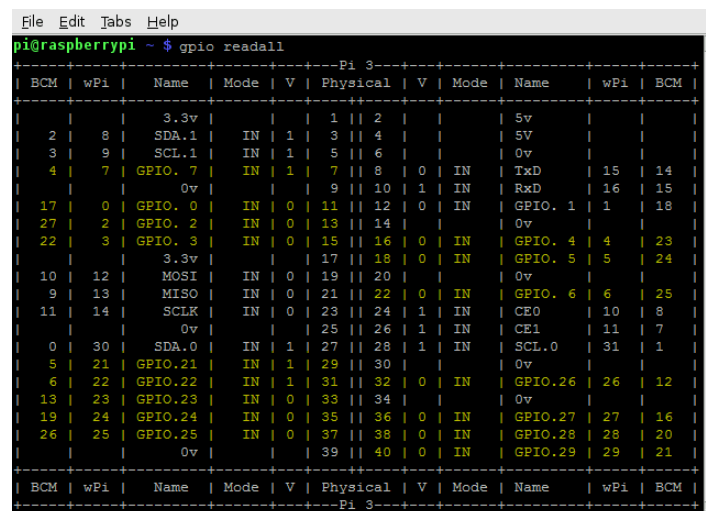


Figure 1: GPIO Header of the B+ ([source](#))

1.1 Board Modes

The Raspberry Pi has a number of "board modes" depending on which library you use for development. Some of these modes include:

- **BCM Mode**
This is the pin numbering that is tied to the Broadcom chip that powers the Raspberry Pi.
- **GPIO/Physical Mode**
This is the pin numbering that matches the physical pin numbering of the header.
- **WiringPi numbering**
WiringPi is a library for writing C/C++ applications on the Raspberry Pi. It has its own unique pin numbering. It includes a useful function for viewing all the pinouts, numbering, and what their current state is. A screenshot of this function is shown in Figure 2.



```
pi@raspberrypi ~ $ gpio readall
```

BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
		3.3v			1	2		5v		
2	8	SDA.1	IN	1	3	4		5V		
3	9	SCL.1	IN	1	5	6		0v		
4	7	GPIO. 7	IN	1	7	8	0	IN	15	14
		0v			9	10	1	IN	16	15
17	0	GPIO. 0	IN	0	11	12	0	IN	1	18
27	2	GPIO. 2	IN	0	13	14		0v		
22	3	GPIO. 3	IN	0	15	16	0	IN	4	23
		3.3v			17	18	0	IN	5	24
10	12	MOSI	IN	0	19	20		0v		
9	13	MISO	IN	0	21	22	0	IN	6	25
11	14	SCLK	IN	0	23	24	1	IN	10	8
		0v			25	26	1	IN	11	7
0	30	SDA.0	IN	1	27	28	1	IN	31	1
5	21	GPIO.21	IN	1	29	30		0v		
6	22	GPIO.22	IN	1	31	32	0	IN	26	12
13	23	GPIO.23	IN	0	33	34		0v		
19	24	GPIO.24	IN	0	35	36	0	IN	27	16
26	25	GPIO.25	IN	0	37	38	0	IN	28	20
		0v			39	40	0	IN	29	21

Figure 2: A screenshot of running "gpio readall"

2 Setting up your Pi

In order to use the Pi you need to install an operating system on it and set up networking. The following will lead you through installing Raspbian as an operating system. Raspbian is an operating system created for the Raspberry Pi, built on Debian. You will then configure the networking settings in Raspbian to allow you to access the Pi remotely using SSH.

While there are no requirements for this course on which operating system to use, it is strongly recommended you become familiar with a Linux-based operating system. Recommendations include Ubuntu and Mint.

2.1 Prerequisites

Hardware Requirements	Software Requirements
<ul style="list-style-type: none">• Raspberry Pi• Ethernet Cable• Power source for Raspberry Pi• Means of writing to the Micro SD Card	<ul style="list-style-type: none">• Rasbian (Desktop & Recommended Software)• PuTTY (the full suite, Windows only).¹• SDFormatter• Etcher, an image writing tool

2.2 Prepare the SD Card

The SD Card needs to be formatted before it can be used. To make your life simple, there is a tool called "SD Card Formatter" - a link is available in the Software Requirements box of the prerequisites section.

2.3 Install and configure Raspbian

This process will follow headless installation, as we are going to assume most students do not have access to spare screens, keyboards and mouses².

Headless mode on the Raspberry Pi refers to using it without direct user input and output (essentially no screen, mouse or keyboard connected directly to it). This is how all of the practicals will be conducted, as it is how most IoT devices are configured (over a network).

1. Insert the SD card into the computer. If you do not have a reader available, speak to a friend or tutor who does.
2. Format the Micro SD card
If you're on Windows, this is where you would use SDFormatter. If you're on Ubuntu, you can use GParted.
3. Write image to SD card
Open Etcher. Select the downloaded zip image, and the SD card, and format. At the end of format, it may read that it failed, but don't worry. Upon completion, Windows will try to mount partitions on the SD card that it can't read. Just press "Cancel" and then "OK" to the dialog boxes that pop up. The boot partition is the only partition we will be dealing with.
4. Enable SSH
SSH is covered in detail in Section 10, but for now just enable SSH by creating a file called "ssh" (no file extension) on the boot partition of the SD card.
5. Configure the Networking
This section can get complicated, so we've created Section 8 to help explain it. For now, just follow the steps in Section 8.2 and Section 8.3.

²Yes, "mouses" is entirely valid and used to distinguish a computer device from a rodent: https://en.wikipedia.org/wiki/Computer_mouse#Naming

6. Boot

Insert the SD Card into the Pi. Connect an Ethernet cable between your PC and the Pi. Connect the Pi to a power source.

7. Ensure connectivity

Ensure you can "see" your Pi. Open a command window or terminal, and run

```
$ ping 192.168.137.15
```

You should see a response similar to

```
$ 64 bytes from 192.168.137.15: icmp_seq=1 ttl=64 time=0.557 ms
```

or

```
$ Reply from 192.168.137.15 bytes=32 time=0.5ms TTL=52
```

8. SSH in to the Pi

Section 10 covers all the details on SSH. For now, open a command prompt or terminal on your machine, and enter in the following:

```
$ ssh pi@192.168.137.15
```

You will be asked to add the machine to known hosts. Select yes.

The password required to log in is simply "raspberrypi"

9. Configure

For practicals you will need to enable a few services on the Pi.

- Open a terminal. You can either click the icon or press Ctrl+Alt+T
- Run

```
$ sudo raspi-config
```

- Use the arrow keys and scroll down to "Interfacing Options"
- Enable SSH, VNC, SPI, I2C and Serial
- Save and exit raspi-config by using the left and right arrow keys to jump to the bottom buttons. Your Pi will reboot. Wait about 30 seconds, and SSH in again.

10. Expand the Filesystem

The Raspberry Pi Installation may not make use of the full SD card. In order to give yourself as much file space as you have access to, you need to expand the filesystem.

- SSH into the Pi
- Run

```
$ sudo raspi-config
```

- Use the arrow keys and scroll down to "Advanced Options"
- Select "Expand Filesystem"

- You will be shown a message saying that the root partition has been resized. Select "Ok", then "Finish" then "Yes" to reboot your Pi.

11. Creating another user

It isn't good practice to run everything as root, so we create another user with sudo privileges. The first command adds the user, the next command adds the user to the group `sudo`, which gives them sudo execution rights. On the Pi, run the following:

```
$ sudo adduser <username>
$ sudo adduser <username> sudo
```

Now that you have created a new user, you can log into the Pi via SSH using that username as follows:

```
$ ssh <username>@192.168.137.15
```

To switch users while you are SSH'd in run

```
$ sudo su - <username>
```

To switch to root (not recommended), run

```
$ sudo su
```

To change your password, you can simply run

```
$ passwd
```

To change the password of another user, drop down to root and run

```
$ passwd <username>
```

To delete a user, switch to another user with root access, and run

```
$ sudo deluser -remove-all-files -f <username>
```

3 The Unix Shell

You worked with a few Unix commands in the shell in the installation section. This section does a bit of a deeper dive into commands that you may find useful.

Being able to use the Unix Shell and terminal commands is an invaluable skill, and a requirement for this course. Follow [this guide](https://swcarpentry.github.io/shell-novice/) (https://swcarpentry.github.io/shell-novice/) for more learning resources.

Some useful commands are listed below. There is also space to add your own commands if you find any that are useful.

Table I: Some useful shell commands

Command	Use
ls	List current files and folders in directory. ls -al is useful to list everything
pwd	Prints the current working directory
cd <directory>	Change to a specified directory. eg "cd .. " will take you one level up
ifconfig	Shows details on current network interfaces
touch <file>	Create <file>
nano <file>	Opens <file>in the nano text editor
vim <file>	Opens <file>in the vim text editor
mkdir <dir>	Creates the folder specified in <dir>
sudo <cmd>	executes the command <cmd>with sudo privileges
raspi-config	Must be run as administrator Opens up the Raspberry Pi Configuration Tool
man <command>	Opens the manual for a particular command
ssh	Creates an SSH session. See Section 10
scp	Secure copy. See Section 12

```
pi@raspberrypi: ~  
pi@raspberrypi:~$ mkdir STUDNUM  
pi@raspberrypi:~$ ls  
demoscript.py Desktop Documents Downloads MagPi Music Pictures Public STUDNUM Templates Videos  
pi@raspberrypi:~$ ifconfig  
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500  
    inet 192.168.137.15 netmask 255.255.255.0 broadcast 192.168.137.255  
    inet6 fe80::ba27:ebff:fe39:6706 prefixlen 64 scopeid 0x20<link>  
    ether b8:27:eb:39:67:06 txqueuelen 1000 (Ethernet)  
    RX packets 357 bytes 25490 (24.8 KiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 373 bytes 43769 (42.7 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0x10<host>  
    loop txqueuelen 1000 (Local Loopback)  
    RX packets 69 bytes 6556 (6.4 KiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 69 bytes 6556 (6.4 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
pi@raspberrypi:~$ lscpu  
Architecture: armv7l  
Byte Order: Little Endian  
CPU(s): 4  
On-line CPU(s) list: 0-3  
Thread(s) per core: 1  
Core(s) per socket: 4  
Socket(s): 1  
Model: 5  
Model name: ARMv7 Processor rev 5 (v7l)  
CPU max MHz: 900.0000  
CPU min MHz: 600.0000  
BogoMIPS: 38.40  
Flags: half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idivt vfpd32 lpae evtstrm  
pi@raspberrypi:~$
```

Figure 3: An example output of running some commands in the shell

4 Text Editors

When editing files on the Raspberry Pi there are multiple options, depending on whether you choose to edit the file in a specific program, or edit it within the command shell.

4.1 GUI Based Text Editors

4.1.1 Notepad++

Notepad++ is a text editor for Windows. It can be downloaded [here](#). Language can be set to enable syntax highlighting. To ensure you use spaces instead of tabs, go to Settings - Preferences - Tab Settings - tick the box "Replace by Space"..

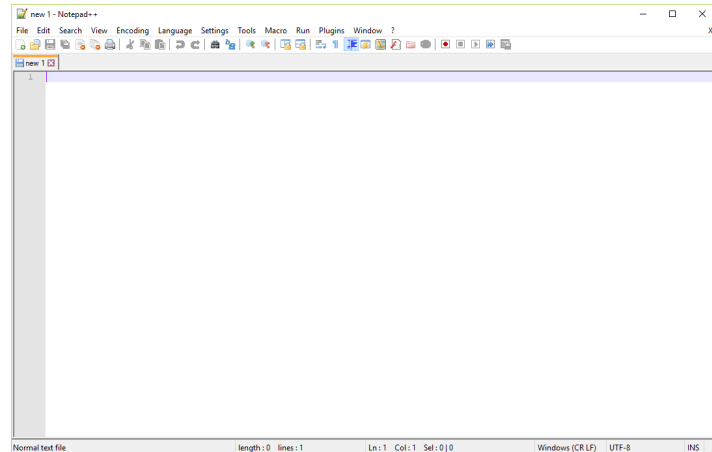


Figure 4: The Notepad++ text Editor

4.1.2 Geany

Geany is a GUI based text editor for Linux. It is very similar to Notepad++ on Windows and supports essentially the same features. The programming language can be set for syntax highlighting. Take note of whether you are using tabs or spaces (4 spaces is recommended for indentation). This can be set through preferences - editor - indentation - type - spaces. After changing the setting, close and reopen the file.

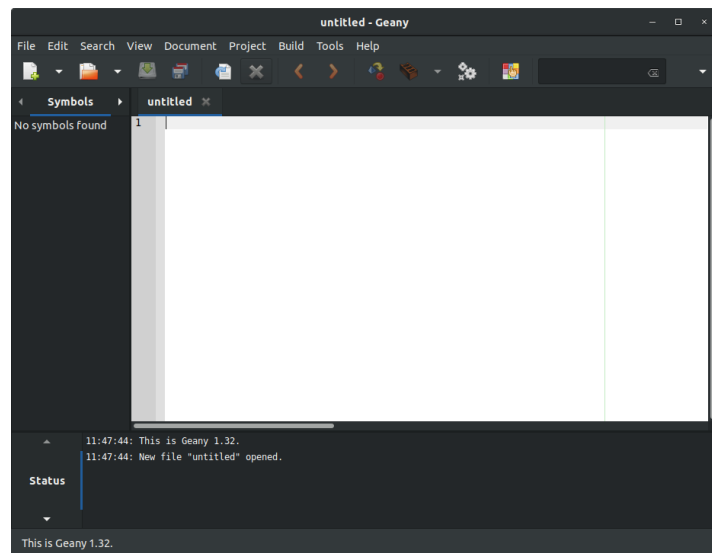


Figure 5: The Geany Text Editor

4.2 Terminal Based Text Editors

4.2.1 Nano

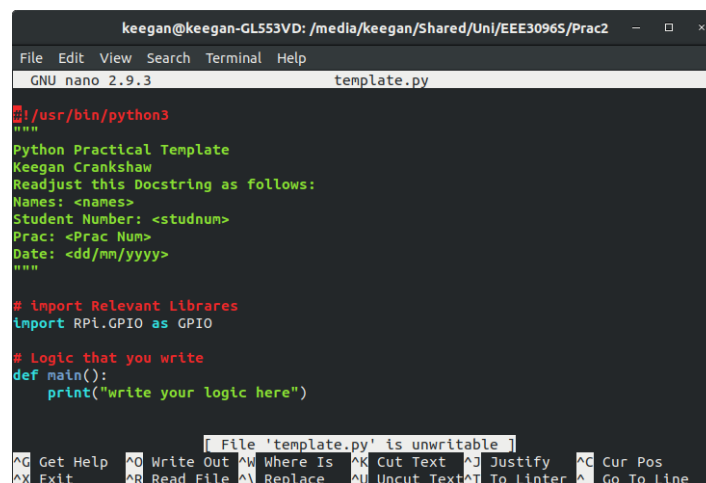
Nano is a very easy to use file editor that runs within the command shell. Nano comes installed on Ubuntu and Raspbian by default, but if you do find yourself needing to install it, you can run

```
$ sudo apt-get install nano
```

Once it is installed, you can use nano to edit text files by typing

```
$ nano filename
```

You will be presented with an interface like this (currently the user is editing the python template):



```
keegan@keegan-GL553VD: /media/keegan/Shared/Uni/EEE3096S/Prac2
File Edit View Search Terminal Help
GNU nano 2.9.3 template.py

#!/usr/bin/python3
"""
Python Practical Template
Keegan Crankshaw
Readjust this Docstring as follows:
Names: <names>
Student Number: <studnum>
Prac: <Prac Num>
Date: <dd/mm/yyyy>
"""

# import Relevant Librares
import RPi.GPIO as GPIO

# Logic that you write
def main():
    print("write your logic here")

File 'template.py' is unwritable
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^_ Replace ^U Uncut Text ^T To Linter ^_ Go To Line
```

Figure 6: The python template opened in nano

Commands for nano are displayed on the bottom. A quick reference is as follows:

- Arrow Keys to move around as expected
- Ctrl-W to find (Where is)
- Ctrl-O to save (Write Out)
- Ctrl-X to Exit (will be prompted to save on exit, press 'y' to save, press 'n' to exit without saving)
- On Windows using Putty SSH client, 'paste' is performed by right-clicking with the mouse anywhere in the window, the contents of the clipboard will be pasted at the current cursor location

Note: By default, nano inserts a tab when the tab key is pressed. If you'd like to change this to insert a number of spaces, for example, four (as is [recommended by PEP 8](#)), do the following:

- Edit your `/.nanorc` file (or create it)

```
$ sudo nano ~/.nanorc
```

- Edit it to contain the following:

```
set tabsize 4  
set tabstospaces
```

- Reboot your Pi to enable the changes

4.2.2 Other Terminal Editors

There are other terminal based text editors, such as emacs and vim. Feel free to use whichever you are comfortable with.

5 Programming IDEs

There are a few specific IDEs which you may find useful in this course. Specifically, they are CLion and PyCharm, both of which are created by JetBrains. You can get access to licences for JetBrains IDEs by signing up for the [Github Education Pack](#).

5.1 Visual Studio Code

Visual Studio Code is a powerful IDE that can be configured to run various programming languages. It has an array of plugins from C++ to Latex, as well as Python and Raspberry Pi Plug-ins. You can learn more about Visual Studio Code [here](#).

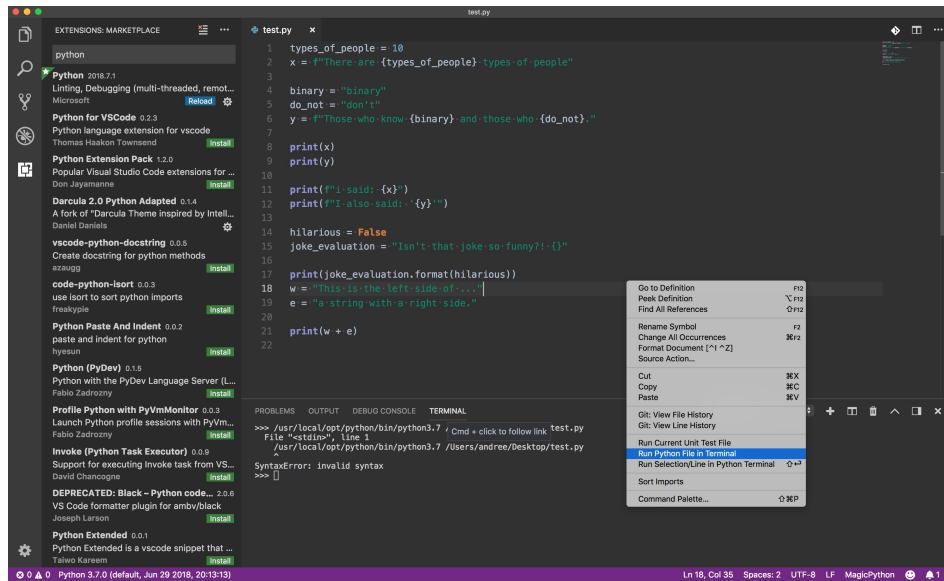


Figure 7: VSCode configured for Python ([image source](#))

5.2 PyCharm

PyCharm is the Python Editor created by JetBrains. You can read more about PyCharm [here](#). Learn about setting up a virtual environment (good practice in Python) [here](#).

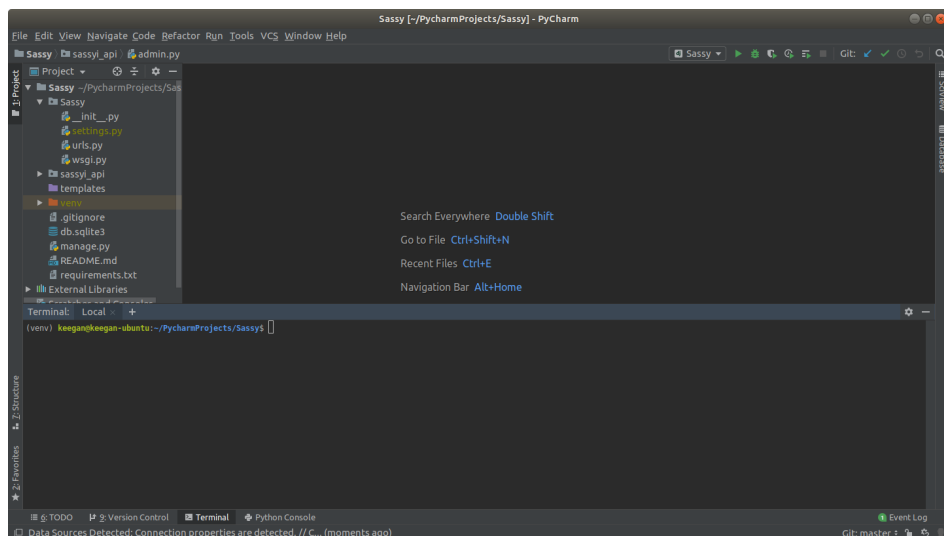


Figure 8: The PyCharm IDE

5.3 CLion

CLion is a cross compilation tool developed by JetBrains that will be used in this course.

This section covers installation and configuration of JetBrains CLion on Windows and Ubuntu. It assumes you have a JetBrains Education account (possible through the [Student GitHub Pack](https://education.github.com/pack) - <https://education.github.com/pack>).

5.3.1 CLion on Windows

CLion on Windows for cross compilation requires use of the Ubuntu subsystem. At this point, it's likely better to dual boot the two operating systems. However, if you're unable to do this, you can follow through the guide below. A reminder, you don't need to use CLion - compilation on Windows is available as pointed out in Section 16.4.

In this guide, we're going to be working with Ubuntu 18.04 as the distribution. To install WSL, follow [this tutorial](#). Be sure to select Ubuntu 18.04 as your distribution.

Once you've installed it, you can start the instance by pressing the start button and searching for "Ubuntu". Upon first run, you will be asked to create a username and password. Remember these details, as they will be used later. Once you have created your username and password, run the following commands:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install libc6-armel-cross libc6-dev-armel-cross
$ sudo apt-get install binutils-arm-linux-gnueabi libncurses5-dev lib32z1
$ sudo apt-get install gcc-arm-linux-gnueabi g++-arm-linux-gnueabi
```

Then, configure ssh and relevant/related items using the JetBrains script. Still on the Ubuntu sub-system, run:

```
$ wget https://raw.githubusercontent.com/JetBrains/clion-wsl/master/ubuntu_
  ↪ setup_env.sh
$ bash ubuntu_setup_env.sh
```

This script configures an SSH connection on port 2222. Test is by running the following on the Ubuntu Subsystem

```
$ ssh username@127.0.0.1 -p 2222
```

Now, open CLion. Navigate to File - Settings - Build, Execution, Deployment - Toolchains. Under environment, Select "WSL". Click the gear next to Credentials, and enter in your username and password, ensuring that the correct port number (2222) is used. You will need to change C Compiler and C++ Compilers to use the Raspberry Pi Compilers we installed in the earlier steps. Change the C compiler to use `/usr/bin/arm-linux-gnueabi-gcc` and the C++ Compiler to use `/usr/bin/arm-linux-gnueabi-g++`. The configuration should now look as follows:

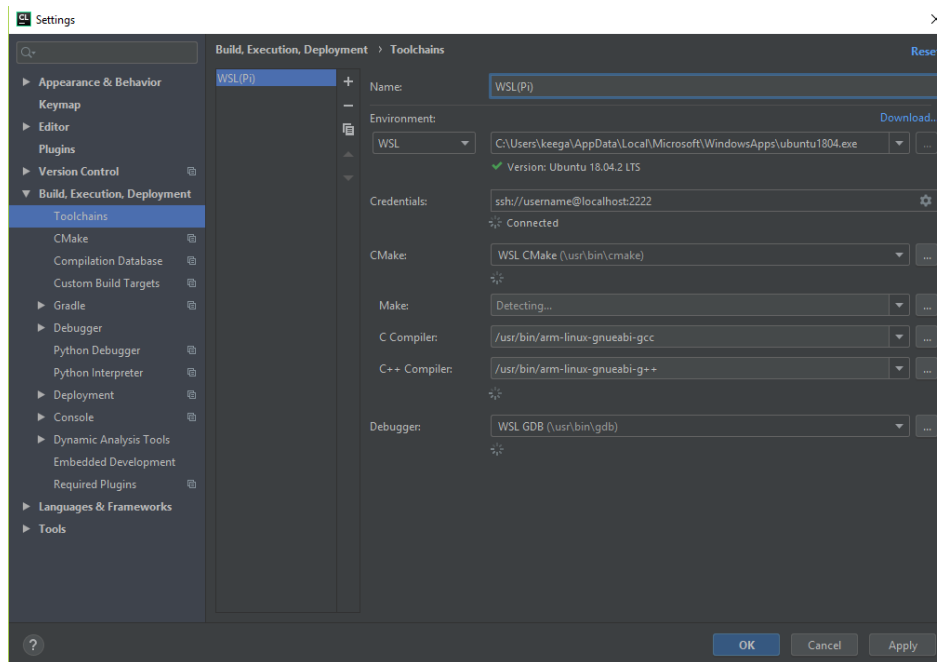


Figure 9: CLion WSL Configuration on Windows

Once you create and compile a project, you should be able to move it to the Pi using SCP (or PSCP if SCP is disabled), adding the executable flag, and running it. Do not be surprised if there is an error once you've built it on the Pi

5.3.2 CLion on Ubuntu

CLion on Ubuntu is considerably easier to configure. Install CLion by running

```
$ snap install clion --classic
```

Run the following commands to install required packages:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install libc6-armel-cross libc6-dev-armel-cross
$ sudo apt-get install binutils-arm-linux-gnueabi libncurses5-dev lib32z1
$ sudo apt-get install gcc-arm-linux-gnueabi g++-arm-linux-gnueabi
```

Your configuration file should be as follows:

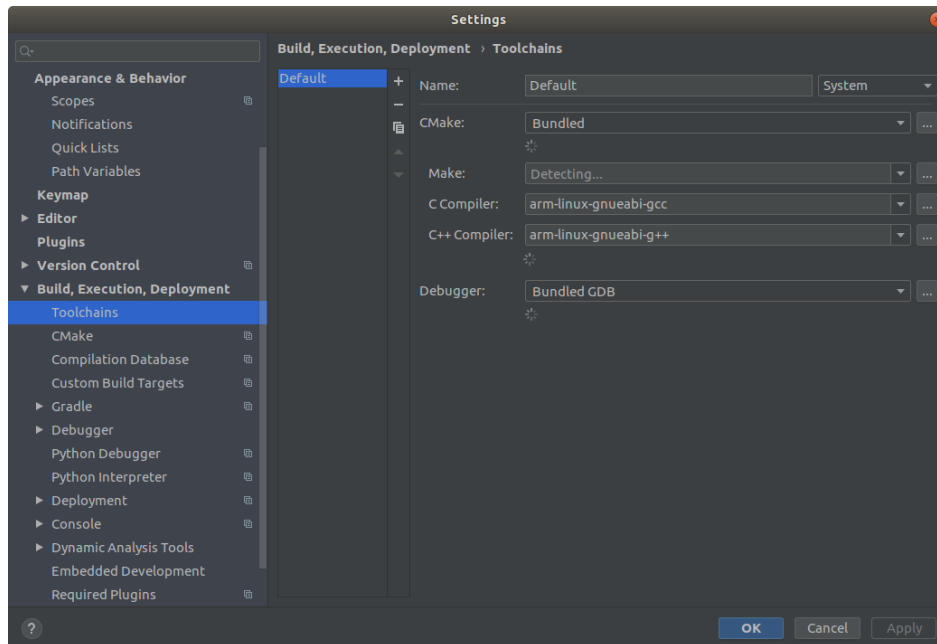


Figure 10: CLion Ubuntu Configuration

6 Git

Throughout the course you will be required to write and submit code and other files to an online git repository, as well as submitting it through Vula. Git is a popular version control application that allows you to keep a record of changes you have made to files over time. Additionally, by using a version controlled repository that is accessible to other collaborators, it provides a powerful way for many people to collectively build a larger system. Almost all open source projects use a version control system to enable them to work with anyone else around the world in a structured and organised manner. There are many platforms that offer free hosting services where you are able to share repositories, Github, and Gitlab are two such options. Making an account on either of these (or a similar alternative) allows you to: (1) keep track of your own projects and the changes you make to them, (2) work collaboratively with teammates if you choose to share your repositories with them, and (3) is an increasingly common way of sharing a portfolio of your prior work with potential employers.

Git is a local tool. To use online backups, it's recommended to use a remote (online) repository management tool. GitHub is recommended because, as a student, there are many benefits which you can access. See <https://education.github.com/pack> to sign up.

Git can be intimidating in the beginning, but it becomes invaluable as you progress in software development. [This page](#) has a great guide you can follow to get well acquainted.

Before arriving at Prac 1, make sure you have created a demo repository on a computer and pushed it to the cloud using the instructions found below.

6.1 A Quick Git Get Go

This sub section has a bunch of useful "recipes" to follow that will be common throughout your experience with git.

6.1.1 Creating a GitHub Account and Configuring your Computer

- Start by creating a GitHub account
- Install git on your computer
Lab computers already have git installed.
If you're using a Linux based system, this can be as simple as
`sudo apt-get install git`
If you're using Windows, you need to [download and use an installer](#).
- Once git is installed, run the following commands:
Note: Do not do this on the lab computers. If you are on a lab computer, rather set the `user.name` and `user.email` parameters from within the created git repository, and do not use the `--global` flag. Only use the global flag if you're on your own system, such as your own PC or your Pi.

```
$ git config --global user.name "Your Name"
$ git config --global user.email "github email address"
```

- Git is now configured

6.1.2 Creating a New Project

Git consists of three primary stages: Untracked, staged and committed. Untracked files are not tracked by the repository. Staged files are files staged for commit but not yet committed. Committed files are "saved" to git. On your local system:

- Create a folder and enter into it
- Run `git init`
- Create a new text file, for example "test.txt"
- Run `git status`
- You will see there is an untracked file. Add it to git by running `git add test.txt`
- It is possible to add all untracked files by running `git add .`
- If you run `git status` again, you will see that "test.txt" has been staged, but not yet committed. Commit test.txt by running `git commit -m "Created test.txt"`. The `-m` flag is to include a git commit message. It's useful to use these messages to explain what has changed in this commit.

6.1.3 Linking GitHub and your Local Project

On GitHub, create a new repository and give it a meaningful name and description. Take note of the link (something like `https://github.com/<username>/<project-name>.git`)

GitHub gives instructions on how to push an existing repository from the command line, but for completeness sake the commands are included here:

```
$ git remote add origin https://github.com/<username>/<project-name>.git
$ git push -u origin master
```

If you refresh the GitHub page, you should now see your files and commits.

6.1.4 Understanding .gitignore

Related SW Carpentry link: [Ignoring Things](#)

You may have seen the option to add a .gitignore when creating a new repository on GitHub. This is used to get Git to ignore certain files, such as interim or raw data files.

6.1.5 Fetching an Existing Git Repository

In order to do the practicals, you will need to fetch initial files and templates that have been created for you. These are available from [this](#) git repository. While you could navigate to GitHub, download a compressed folder, move it to the Pi, unzip it, and then work with the files, it's much easier to run the following command:

```
$ git clone https://github.com/kcranky/EEE3096S.git
```

7 L^AT_EX

7.1 Overview

L^AT_EX is a great way to write documents, and is required for use in all your documents to be submitted for this course. L^AT_EX is better than basic text editors such as Microsoft Word or Google Docs due to the following³:

1. Dealing with mathematical notation.

Layout and entry are generally easier using LaTeX than some other sort of equation editor. Online tools such as [Detexify](#) make it very simple to find the symbol you need.

³Adapted from [this stack exchange question](#)

2. Consistent handling of intra-document references and bibliography.

As of a couple of years ago the major editors still had problems with re-numbering cross-references and bibliography items. This is never a problem with BibTeX or LaTeX.

3. Separation of content and style.

In principle this means that you can write your document without caring how it is formatted, and at the end of the day wrap it in the style-file provided by the journal publisher before submission to conform to the house style. In practice some of the journal publishers demand special formatting commands that partially moots this process. Furthermore recent versions of Word and LibreOffice Writer, when properly used, should be able to keep track of various levels of section heading separate from the body text, and apply uniform styling to each level. The gap is somewhat closing.

4. Tables and illustrations.

With online tools such as [Tables Generator](#), creating tables in LaTeX is as simple as copy-pasting data from excel. Images can be inserted exactly where you specify them without worrying about justification or overlay.

There are a few difficulties with LaTeX. These include:

1. Difficulties with collaborative editing (consider the convenience of Google Docs)
2. Spell check (Microsoft Word has a much more advanced spell and grammar check)
3. Ease of use (LaTeX is technically a "document preparation system" as opposed to a text editor)

However, many if not all of these issues are mitigated by the use of an online tool known as [Overleaf](#). Overleaf provides you with templates, the ability to collaborate, and (thankfully), a spell check function. It runs in browser and doesn't require any installation.

If you would like to run an offline version, there are various options, but Visual Studio Code with the [Latex Workshop Plugin](#) is suggested.

7.2 Using Overleaf

Once you have created an account on Overleaf, you need a template to work from. The IEEE conference paper template is available on [Download the zip file](#).

- In Overleaf, click "New Project" and select "Upload Project"
- Select or drag the report template zip file you downloaded from Vula
- The template will load and you will be able to edit it.
- Note: you will need to change the "Main document" in the Menu when editing your document to be "Report.tex"

- Note: You will not be able to compile the source into a pdf if you are viewing *Preamble.tex*

7.3 Useful Latex Tools

7.3.1 Overleaf

[Overleaf](#) is an online collaborative Latex editor. It is recommended you use Overleaf for this course, as it has plenty templates which you can draw from.

7.3.2 Detexity

[Detexify](#) is a web app you can use to find Latex codes for specific symbols.

7.3.3 Tablesgenerator

[Tablesgenerator](#) is a website for easily creating Latex tables. You can copy-paste tables from other applications such as Excel.

7.3.4 Citation Machine

[Citation machine](#) can be used to easily generate BibTex.

8 Networking on the Pi

There are many ways to interface with the Pi. This section will cover types of network connectivity.

8.1 A Brief Overview of Networks

It is useful to have a basic idea of how networks, IP addresses, and subnets work. For this, it is suggested you read [this article](#) from Microsoft: <https://tinyurl.com/y2z8x9za>

8.2 Assigning a Static IP to the Pi

This section will assign a static Ethernet address to the Raspberry Pi. This is useful for your first configuration. If you have not done so, it is recommended you follow the instructions in

Section 2 to configure your Raspberry Pi.

1. Insert the SD card into your computer and navigate to the BOOT partition
2. Open "cmdline.txt" and append the following to the line (don't create a new line)

```
ip=192.168.137.15
```

This tells the Raspberry Pi to configure the Ethernet port to use the IP address 192.168.137.15

3. Enable SSH as per Section 10.
4. You need to configure your PC to use the same subnet as the Pi. To do so, see the information below in Section 8.3

8.3 Assigning a Static IP to your Computer

In order to use Ethernet for SSH, VNC, etc, it is required that the Pi and your computer all be on the same subnet. This section details how to do it.

8.3.1 Windows

To change the IP of your Ethernet port on Windows 10, complete the following steps:

- Right click on your network option in Windows taskbar
- Select "Open Network & Internet Settings", on the lower right hand side of the screen.
- Select "Change Adapter Options"
- Right click on the Ethernet Connection and select "Properties"
- Select "Internet Protocol Version 4 (TCP/IPv4) and click "Properties"
- Select "Use the following IP address:" and enter in the following options:
 - IP Address: 192.168.137.1
 - Subnet Mask: 255.255.255.0
- You have successfully changed the IP of the Ethernet card on your computer. It is suggested that you now ensure connectivity by attempting to ping the Pi.

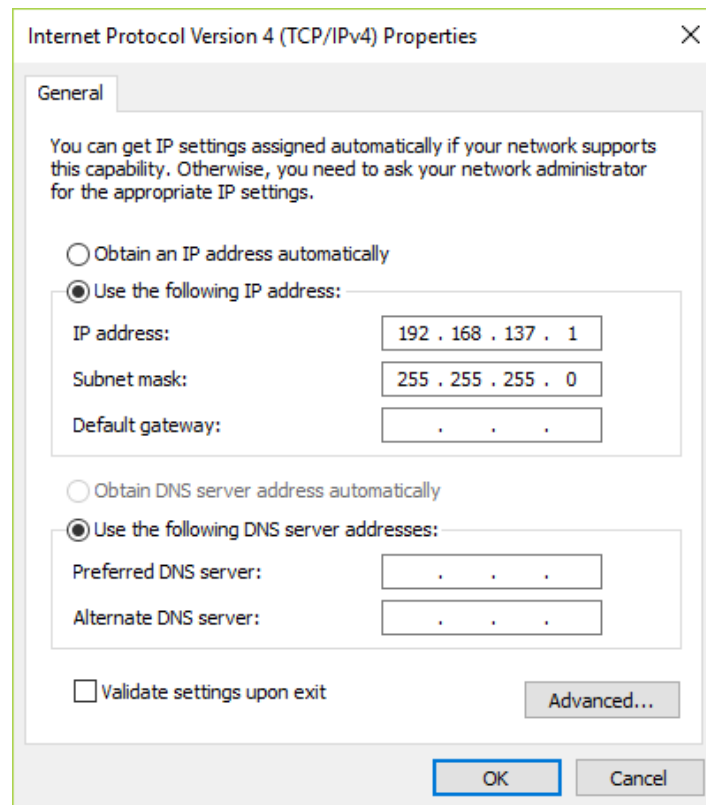


Figure 11: The IPv4 configuration screen in Windows 10

8.3.2 Ubuntu

To change the IP of your Ethernet port on Ubuntu, complete the following steps:

- Click the network interface icon on the status bar and select Wired Settings
- Click the gear button of the interface you'd like to change
- Select the IPv4 Tab, and change the IPv4 method to Manual
- Under "Addresses" enter in the following:
 - IP Address: 192.168.137.1
 - Subnet Mask: 255.255.255.0
- You can leave Gateway and DNS blank

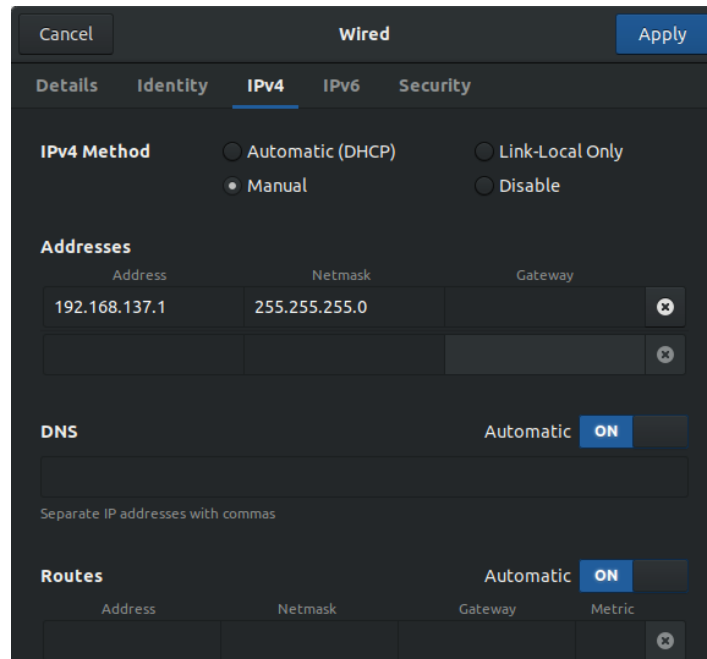


Figure 12: The IPv4 configuration screen in Ubuntu 18.10

8.4 Ensuring connectivity

Sometimes you may want to debug your connection to the Pi. A fast way to do this is via the *ping* command. *Ping* sends a packet to a particular host (in this case the Pi), and measures the time taken for a response from that host.

To use the ping command, open a command prompt window or terminal and type the following:

```
$ ping 192.168.137.15
```

If that host is unreachable (the Pi hasn't booted yet or is incorrectly configured), a message will show that the host is unreachable. If everything was correctly configured, you should get

```
Reply from 192.168.137.15: bytes=32 time<1ms TTL=64
```

This means your Pi and computer are both correctly configured. See section 10 for configuring your Pi for SSH access.

NB: Don't be surprised if you can't ping a Windows machine from your Pi. Windows blocks the specific type of packet required for a ping in the firewall.

9 Advanced Connectivity Options

9.1 Setting a static IP Through Config

Once you've successfully SSH's into your Pi, it's a good idea to configure the networking options in the config files directly.

Use a text editor such as nano to open `/etc/dhcpd.conf` as sudo user, and edit it to the following:

```
# Static IP profile for eth0
profile static_eth0
static ip_address=192.168.137.15/24
static routers=192.168.137.1
static domain_name_servers=192.168.137.1 8.8.8.8

# Ethernet interface configuration
interface eth0
fallback static_eth0

# Wireless configuration
interface wlan0
metric 200
```

9.2 Providing your Pi with wireless Internet Access

There are two possible methods of this that will be presented, each with it's own advantages and disadvantages.

The first is using Ethernet passthrough from your computer to the Raspberry Pi. This leaves the WiFi free to host your own access point, and you can host services such as a Node-Red server or media center on the Pi.

The second involves connecting to a wireless network. In the example we give you, we're only going to add a connection to Eduroam. While you could host other services on the Pi when using WiFi connectivity, it would require some access to port forwarding and other things that ICTS unfortunately won't allow.

It is recommended that you use Ethernet pass through for the practicals.

9.2.1 Using WiFi to Ethernet passthrough to give your Pi internet access

There may be a situation in which you want your Pi to work as an access point rather than using the WiFi interface to provide the Pi with internet access. In this situation, you need to

get internet access through the ethernet port. If you're connected to Windows, you can use network sharing. Complete the following to enable network sharing:

Windows

1. Ensure the Pi is unplugged
2. Right click on your network option in Windows taskbar
3. Select "Open Network & Internet Settings", on the lower right hand side of the screen.
4. Select "Change Adapted Options"
5. Right click on your WiFi network and select "Properties"
6. Click the "Sharing" tab, and enable the first checkbox ⁴
7. Select the Ethernet connection that your Pi will be using in the drop down box (Usually just "Ethernet", but may be different if there are multiple Ethernet posts on your system)

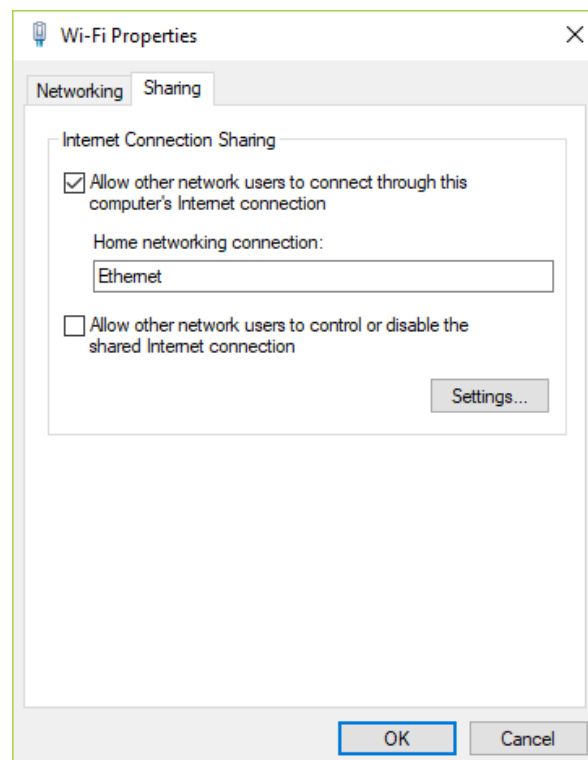


Figure 13: Using WiFi to Ethernet passthrough in Windows

⁴This setting is what forces us to have to use the subnet 192.168.137.x.

Ubuntu

1. Ensure the Pi is unplugged and turned off.
2. Open a terminal and run `nm-connection-editor`
3. Select the wired connection you'd like to share your WiFi to
4. Select the IPv4 settings tab
5. Under Method, select "Shared to other computers"
6. under the IP addresses, click "Add" and enter in an address of 192.168.137.1, and a netmask of 24
7. Select "Save", and close the windows
8. Plug in the Pi, start an SSH session and see if you can ping google.co.za

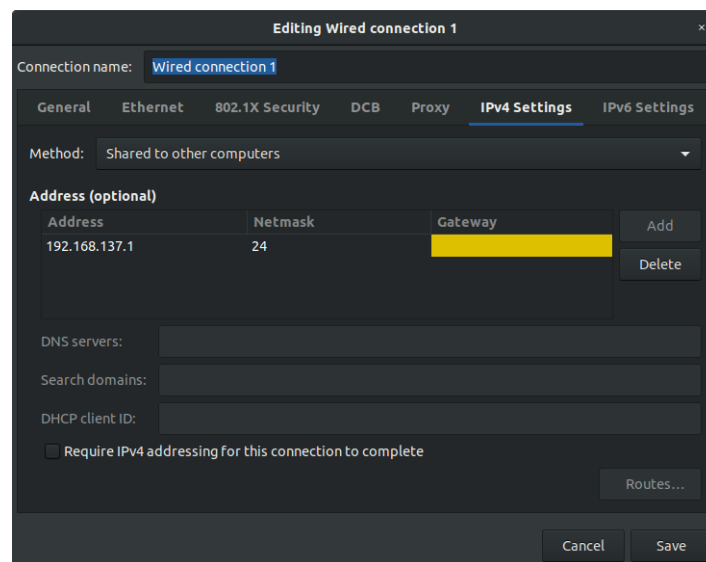


Figure 14: nm-connection-editor in Ubuntu 18.10

9.2.2 Connecting to Eduroam - Raspbian Buster (Raspbian Site)

These instructions come from [here](#).

Unfortunately, when you use bleeding edge technology, you may end up cutting yourself. Raspbian Buster updates `wpa_supplicant` to a version that doesn't have support for the authentication method used by Eduroam. So we need to roll back `wpa_supplicant` to an older version. Ensure your Pi has internet access through a method such as Ethernet passthrough, and run the following:

```
$ sudo apt-get remove wpasupplicant
```

Edit /etc/apt/sources.list :

```
$ sudo nano /etc/apt/sources.list
```

And change

```
deb http://raspbian.raspberrypi.org/raspbian/ buster main contrib non-free  
    ↪ rpi
```

to

```
deb http://raspbian.raspberrypi.org/raspbian/ stretch main contrib non-free  
    ↪ rpi
```

Run

```
$ sudo apt-get update  
$ sudo apt-get install wpasupplicant
```

This will install the correct version of `wpa_supplicant`. Check that **version 2.4** is installed by running

```
$ wpa_supplicant -v
```

Now, change the sources file back.

```
$ sudo nano /etc/apt/sources.list
```

And edit the contents to contain

```
deb http://raspbian.raspberrypi.org/raspbian/ buster main contrib non-free  
    ↪ rpi
```

Finally, run

```
$ sudo apt-get update
```

To update sources The correct version of `wpa_supplicant` should now be installed, and you can configure your Pi for WiFi as you would if you were using Raspbian Stretch as explained in below.

9.2.3 Connecting to Eduroam - Raspbian Stretch (Vula)

This section provides the instructions on how to configure the Pi to use Eduroam. It is possible to do using the GUI through VNC, but we will not cover that here.

1. SSH into your Pi.
2. Generate a hash for your password. Take note of it as it is needed in a later step

```
$ echo -n your_uct_password_here | iconv -t utf16le | openssl md4
```

3. Open /etc/wpa_supplicant/wpa_supplicant.conf

```
$ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

4. Edit it so it looks as follows:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=ZA

network={
    ssid="eduroam"
    key_mgmt=WPA-EAP
    identity="studentnumber@wf.uct.ac.za"
    password=hash:generated_hash_from_earlier
}
```

5. Save the file
6. Open /etc/dhcpd.conf

```
$ sudo nano /etc/dhcpd.conf
```

7. Make sure the following lines are in the document:

```
interface wlan0
metric 200
```

8. Reboot your Pi
9. WiFi Generally takes a little longer to initialize than Ethernet, so give it time. You can see if it's ready by running

```
$ ifconfig
```

And seeing if an IPv4 address ("inet") has been given.

10. Test your configuration by pinging from the WiFi interface:

```
$ ping google.com -I wlan0
```

9.3 Debugging Connections

This section is a work in progress. As more common issues are reported by students, this section will be expanded.

9.3.1 Note for Windows Users

If you are having trouble accessing the internet from your Pi using the Ethernet Passthrough technique, it's likely the Windows bridge needs a reset. Change the IPv4 address of your Ethernet port to "Obtain an IP address automatically", then on your WiFi connection, disable sharing, click okay, and then re-enable sharing. Go back to the IP configuration of your Ethernet port, and double check the IP to see if it's been assigned 192.168.137.1. If not, you will need to change it using the "Advanced settings" button.

9.3.2 WiFi

See if you can see wireless networks.

```
$ iwlist wlan0 scan
```

If you cannot connect via WiFi, enter into a shell on the Pi and run:

```
$ journalctl -u wpa_supplicant | grep wlan0
$ journalctl -u dhcpcd.service | grep wlan0
```

This will output the log files and notify you of any incorrect configurations in wpa_supplicant.

The following command will force the interface to be up (if it can be):

```
sudo ifconfig wlan0 up
```

If all else fails, reboot and try again. Some services can be restarted without restarting the Pi, for example:

```
sudo systemctl restart dhcpcd
```

9.3.3 "Failure resolving URLs or "unknown host"

If you try to ping a website and it fails, but pinging a URL works as expected, it is likely an issue with DNS configuration. Open the resolv.conf file:

```
$ sudo nano /etc/resolv.conf
```

And edit it to read the following:

```
nameserver 8.8.8.8
nameserver 192.168.137.1
```

8.8.8.8 is the IP of Google's DNS server. A DNS (Domain Name Service) server is responsible for converting human-readable addresses (for example google.co.za) to something the network architecture can understand (172.217.170.67, in this example).

If you still get this error, try running the following command:

```
$ sudo route add default gw 192.168.137.1
```

Where the IP supplied is the IP of the computer or router you are connected to.

9.4 Configuring the Pi to Act as an Access Point

If you are hosting a server on the Raspberry Pi, or perhaps want to create a WiFi network for guests to connect to, the Pi can act as an access point. This guide comes from <https://www.raspberrypi.org/documentation/configuration/wireless/access-point.md> and <https://frillip.com/using-your-raspberry-pi-3-as-a-wifi-access-point-with-hostapd/>.

Note that this WiFi connection will not provide internet access by bridging the Ethernet port (that's something else entirely), but it works well for hosting services on the Pi, such as a Node-Red server.

1. SSH into the Pi, update, and reboot to ensure updates have taken place

```
$ sudo apt-get update
$ sudo apt-get dist-upgrade
$ sudo reboot
```

2. Wait for the Pi to reboot, and reconnect via SSH. Install hostapd and dnsmasq

```
$ sudo apt-get install hostapd dnsmasq
```

3. Configure a static IP in dhcpcd

```
$ sudo nano /etc/dhcpcd.conf
```

Adjust the contents so that the wireless interface is described as follows:

```
interface wlan0
static ip_address=192.168.4.1/24
nohook wpa_supplicant
```

Save and close that file, and restart the dhcp service

```
$ sudo systemctl restart dhcpcd
```

4. Run the following commands to create save the original dnsmasq, and create a new file which will be edited:

```
$ sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.orig  
$ sudo nano /etc/dnsmasq.conf
```

Place the following in the now open file:

```
interface=wlan0  
listen-address=192.168.4.1  
dhcp-range=192.168.4.2,192.168.4.180,255.255.255.0,24h  
server=8.8.8.8  
domain-needed  
bogus-priv
```

Save and close the file

5. Restart the dnsmasq service

```
$ sudo systemctl reload dnsmasq
```

6. Configure hostapd. Open up the configuration file:

```
sudo nano /etc/hostapd/hostapd.conf
```

Place the following configuration in the file. Some assumptions are made about the technical aspects of it, but these are beyond the scope of this course. Note that network name and password **do not** have quotes around them.

```
interface=wlan0  
driver=nl80211  
ssid=TestNetwork  
hw_mode=g  
channel=7  
ieee80211n=1  
wmm_enabled=1  
macaddr_acl=0  
auth_algs=1  
ignore_broadcast_ssid=0  
wpa=2  
wpa_passphrase=TestNetwork  
wpa_key_mgmt=WPA-PSK  
wpa_pairwise=TKIP  
rsn_pairwise=CCMP
```

Save and close the file

7. Next, the system needs to know where to find this configuration file. Open the configuration file:

```
$ sudo nano /etc/default/hostapd
```

Find the line with "#DAEMON_CONF" and replace it with:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Save and close this file.

8. Now enable and start hostapd:

```
$ sudo systemctl unmask hostapd
$ sudo systemctl enable hostapd
$ sudo systemctl start hostapd
```

9. Add routing and masquerade by opening sysctl:

```
$ sudo nano /etc/sysctl.conf
```

And uncomment this line by removing the preceding # symbol:

```
net.ipv4.ip_forward=1
```

Save and close the file

10. Add a masquerade, and save the iptables rule:

```
$ sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
$ sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

11. Edit /etc/rc.local and add this just above "exit 0" to install these rules on boot.

```
$ sudo nano /etc/rc.local
```

Place the following just above "exit 0" to install these rules on boot

```
iptables-restore < /etc/iptables.ipv4.nat
```

9.5 TTL over USB

Note: The Pi uses 3.3V logic levels. Using a 5V logic module will permanently damage the pins.

This option allows you to use a USB to UART converter such as a FT232R or CP2102. Begin by removing the SD card, and insert it into a computer. Make the following changes on the boot partition:

cmdline.txt: add the following (on the same line)

```
console=serial0,115200
```

config.txt

```
uart_enable=1  
dtoverlay=pi3-disable-bt
```

10 SSH

10.1 Enabling SSH

If you have not connected to your Pi and configured it for SSH, you need to do so. SSH is disabled by default on new installations of Raspbian.

To enable SSH, add it as an enabled service in the **raspi-config** menu. If you do not have access to the Pi as yet, do the following:

1. Insert the SD card into a computer
2. Navigate to the BOOT partition
3. Create a file called "ssh"
4. Your Pi will enable SSH upon next boot

10.2 Using SSH

To use SSH on your Pi, you need to connect to the computer to a network. See Section 8 on various ways that can be done (it is suggested to use Ethernet upon first connection).

Once your Pi is connected to the computer and you have ensured connection (see Section 8.4), use can log in to your Pi via SSH. If you are on a Linux-based system, such as Ubuntu or Mac, you should be able to run the following. Note that the default username is "pi" and the default password is "raspberrry".

```
$ ssh <username>@192.168.137.15
```

If you are on Windows, you may need to use PuTTY. Some instances of windows have SSH in the command line, and you can run the command shown above. But if not, you will need to do the following:

- Open PuTTY

- In the "Hostname" field, enter in "192.168.137.15"
- Click "Open". A terminal window will be opened. If it is the first time you're SSH'ing into your Pi on this particular computer, you will be asked about the server fingerprint. Click "Yes" to continue.
- You will be asked for a username and password. The default username is "pi" and the password is "raspberry".
- You should now successfully connected to your Raspberry Pi via SSH

11 VNC

In the previous section, control via SSH was introduced. As previously mentioned, the Raspberry Pi can be used as a standalone desktop computer. However, it is a little impractical to carry around a screen and all the other required peripherals when you're working with your Pi. This is where VNC comes in.

In computing, Virtual Network Computing (VNC) is a graphical desktop-sharing system that uses the Remote Frame Buffer protocol (RFB) to remotely control another computer.⁵

There are various options for VNC servers. Raspbian comes installed with Real VNC but it needs to be enabled. Other options, such as tightVNC and ultraVNC also exist and can also be used.

1. Activate Real VNC

- Start by connecting to the Pi via SSH, and opening up raspi-config

```
$ sudo raspi-config
```

- Scroll down using the arrow keys to 5 - Interfacing Options
- Scroll down to VNC, and select "Yes" when asked to enable it
- Select "Finish"

2. Adjust resolution

This can be done in two ways:

- Setting through /boot/config.txt
Edit config.txt and uncomment these lines:

```
framebuffer_width=1280
framebuffer_height=720
```

⁵Thanks Wikipedia!

- On the Pi desktop in VNC
Do this if you have already connected to VNC. This is a little more difficult as it required you to play with windows in order to see the buttons you need.
 - Connect to the Pi through VNC.
 - In the desktop menu, go to Preferences - Raspberry Pi Configuration and click the "Set Resolution" button.
 - Select a more appropriate resolution (1280*720 suggested)
 - Select "Okay" and then "Okay". You will be asked to reboot your Pi, do so.
3. Download a viewer
VNC Viewer is available at this URL:
<https://www.realvnc.com/en/connect/download/viewer/>
Download your choice of app (For example the standalone installer or the Chrome App)
 4. Set up the connection
 - Open up VNC viewer
 - Enter the IP of your Pi
 - Click connect
 - You will need log in
 5. Configure the Pi
Upon first boot to desktop, you may be asked to configure some options on the Raspberry Pi. Simply hit next/skip through all of them as they will be configured at a later stage.

12 SCP

SCP or "Secure Copy" is a protocol that allows you to transfer files. To read all the details relating to scp, run `man scp`. The basic format of the command is as follows:

```
$ scp <local_file> <username>@<destination_ip>:<source_directory_location>
```

Some example are as follows (assuming your Pi is located at 192.168.137.15):

- Sending a single file from your computer to the Pi

```
$ scp <file_name> <username>@192.168.1.15:
```

- Sending a folder from your computer to the Pi

```
$ scp -r <folder_name> <username>@192.168.1.15:
```

- Sending a single file from your computer to a specific folder on the Pi:

```
$ scp <file_name> <username>@192.168.1.15:<destination_directory>
```


12.1 Using SCP on Windows

On Windows, SCP may not be enabled. You can get around it by using the `pscp` command, included in the full PuTTY suite (see Section 2.1). It operates in the exact same way, but instead of running `scp`, you need to run `pscp`.

13 FTP

Occasionally you may want to use a GUI to browse and transfer files. The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and server on a computer network. FTP is built on a client-server model architecture using separate control and data connections between the client and the server.⁶

[FileZilla](#) is a free to use FTP program available for Windows and Linux. Download and install it.

When you launch FileZilla, a GUI with a few options across the top will be shown. In "Host", enter in "sftp://192.168.137.15". In "Username" and "Password" enter in the username and password you use to SSH in to the Pi. Click "Quickconnect"

When connecting for the first time, you can choose to save the password or set a master password. Neither of these are necessary. You will be asked to add the server's host key. Click "always trust this host, add this key to the cache" and select "OK".

You will now be able to browse the files on the Raspberry Pi, and drag and drop from the Pi to your computer, or from the computer to your Pi.

⁶Off Wikipedia

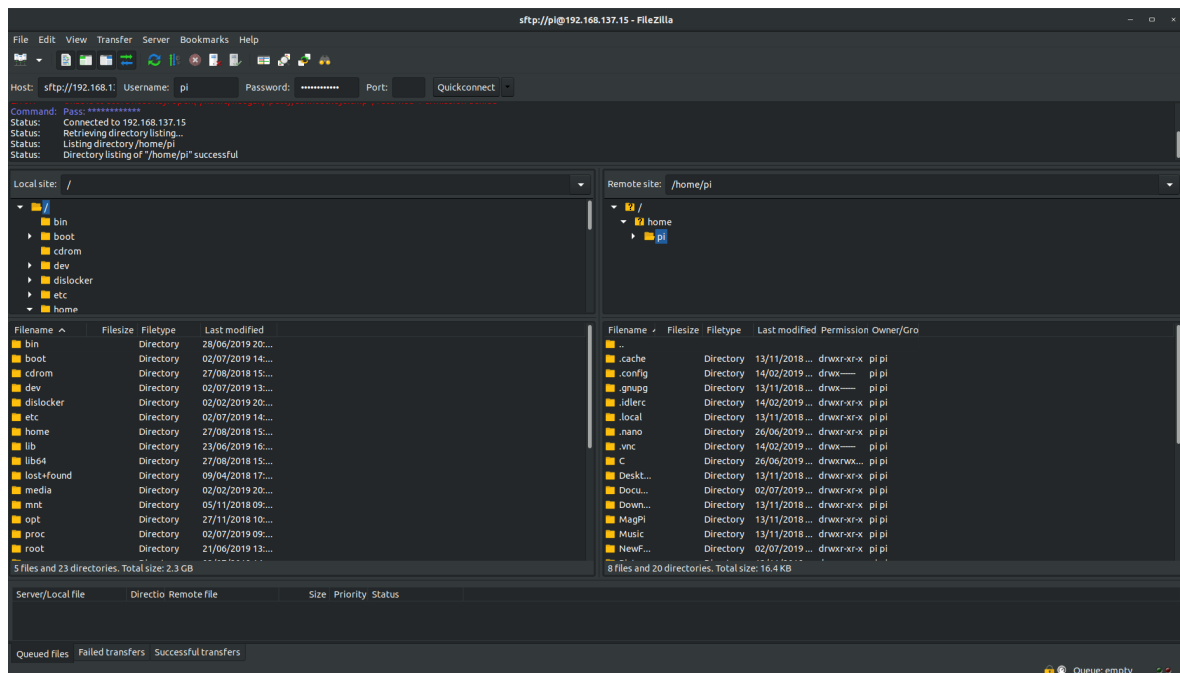


Figure 15: The FileZilla GUI, connected to the Pi from Ubuntu 18.10

14 Programming in Python

You should know that [IEEE ranks Python the most popular language](#), even in 2019. Given this fact, Python examples and references are included in this lab manual. It is important to note the considerations when programming in Python for embedded systems. There are times when Python may be better, but C/C++ will always be more performant. However, for our use, Python is mostly okay. Usage of Python for communication protocols at a low level isn't always done, as libraries generally exist for that purpose.

14.1 Introduction

Most of the content in the guide comes from the documentation, available online [here](#).

Python, while not as powerful as C or C++, is quickly becoming a common choice for embedded systems developers due to its ease of use and rapid development times⁷.

This chapter serves as a short guide for programming in Python on the RPi. You will also find some templates for techniques such as debouncing, or making use of the Raspberry Pi's multicore architecture by implementing threading.

⁷See <https://spectrum.ieee.org/at-work/innovation/the-2018-top-programming-languages>

14.2 The RPi.GPIO Library

The RPi.GPIO library is the common Python library used on the Raspberry Pi. Documentation for the library can be found here:

<https://sourceforge.net/p/raspberry-gpio-python/wiki/Home/>

It is included in the environment variables by default, so to use it, you can simply just import it into your Python script:

```
include RPi.GPIO as GPIO
```

You need to specify which board mode you're using. For more information on board modes, see Section 1.1. Board modes are specified as follows:

```
GPIO.setmode(GPIO.BOARD)
# or
GPIO.setmode(GPIO.BCM)
```

You also need to perform "cleanup" on GPIOs when your application exists. By performing cleanup and resetting all pins to their default modes, you can prevent possible damage to your Raspberry Pi. The cleanup function is called as follows:

```
GPIO.cleanup()
```

If your program is meant to run indefinitely, and only close upon an exception, you can wrap it in a try catch, as follows:

```
if __name__ == "__main__":
    # Make sure the GPIO is stopped correctly
    try:
        while True:
            main()
    except KeyboardInterrupt:
        print("Exiting gracefully")
        GPIO.cleanup()
    except e:
        print("Some other error occurred: {}".format(e.message))
        GPIO.cleanup()
```

14.3 Basic IO

Read the documentation, available [here](#).

14.3.1 Digital Logic

Digital output on the Raspberry Pi is accomplished by writing values to channels. A channel is a pin, which is numbered in the way you've configured. Basic configuration of a pin for output is as follows:

```
GPIO.output(<channel>, <Logic>)
```

Your logic (high, 3.3V, or low, 0V), can be specified as follows:

- For 3.3V (high) output:

```
GPIO.output(<channel>, GPIO.HIGH)
GPIO.output(<channel>, 1)
GPIO.output(<channel>, True)
```

- For 0V (low) output:

```
GPIO.output(<channel>, GPIO.LOW)
GPIO.output(<channel>, 0)
GPIO.output(<channel>, False)
```

Channels can also be specified in lists, for example:

```
LEDs = (11,12)
GPIO.output(LEDs, GPIO.HIGH) # Will turn all channels HIGH
GPIO.output(LEDs, (GPIO.HIGH, GPIO.LOW)) # Will the first channel HIGH, and
    ↪ the second LOW
```

Note that you can also read the state of the pin/channel that is set as an output by using the `input()` function. For example, if you wanted to toggle pin 12, you could do something as follows:

```
GPIO.output(12, not GPIO.input(12))
```

14.3.2 Analog

The Raspberry Pi does not have any analog input pins. You will need to use something like the [MCP3008](#) to read analog voltages.

14.3.3 Inputs

14.3.3.1 Digital Read To read the value of a digital pin, you can use the `input()` function:

```

if GPIO.input(12):
    print("Pin 12 HIGH")
else:
    print("Pin 12 LOW")

```

14.3.3.2 Pull Up/Pull Down Resistors Using pull up and pull down resistors is essential when working with digital logic. Thankfully, the Raspberry Pi has internal pull up and pull down resistors. To make use of these, initialize the pin/channel as follows:

```

GPIO.setup(<channel>, GPIO.IN, pull_up_down=GPIO.PUD_UP)
# or
GPIO.setup(<channel>, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

```

14.3.3.3 Interrupts Very often it is useful to set up an interrupt, for example to trigger an event when a button is pressed. Callback functions are executed on a different thread. If you have multiple callbacks, know that they will be executed sequentially as only one thread exists for interrupts. Interrupts can be implemented as follows:

```

# The bounce time is given in milliseconds.
# If your pin is set to use pull down resistors
# Connect a button between <channel> and 3.3V
GPIO.add_event_detect(<channel>, GPIO.RISING, callback=callback_method(),
    ↪ bouncetime=200)
# If your pin is set to use pull up resistors
# Connect a button between <channel> and GND
GPIO.add_event_detect(<channel>, GPIO.FALLING, callback=callback_method(),
    ↪ bouncetime=200)

```

14.3.3.4 Other functions The RPi.GPIO Library offers other functions for interrupts the first is the `wait_for_edge()` function, which is designed to block execution until an edge is detected. You can detect edges of `GPIO.RISING`, `GPIO.FALLING`, or `GPIO.BOTH`. The timeout is given in milliseconds:

```

# wait for up to 5 seconds for a rising edge
T0 = GPIO.wait_for_edge(<channel>, GPIO_RISING, timeout=5000)
if T0 is None:
    print('Timeout occurred')
else:
    print('Edge detected on pin ', T0)

```

Another function is the `event_detected()` function. From the docs: "The `event_detected()` function is designed to be used in a loop with other things, but unlike polling it is not going

to miss the change in state of an input while the CPU is busy working on other things. This could be useful when using something like Pygame or PyQt where there is a main loop listening and responding to GUI events in a timely basis.”

```
GPI0.add_event_detect(channel, GPIO.RISING) # add rising edge detection on a
    ↪ channel
do_something()
if GPIO.event_detected(channel):
    print('Button pressed')
```

14.4 Communication Protocols

The RPi.GPIO library has no native support for communication protocols. Very often, specific Python libraries are provided on a per-device use case. However, this can become tedious and cause your code to become bloated. See the sections below for relevant libraries for using I2C and SPI directly.

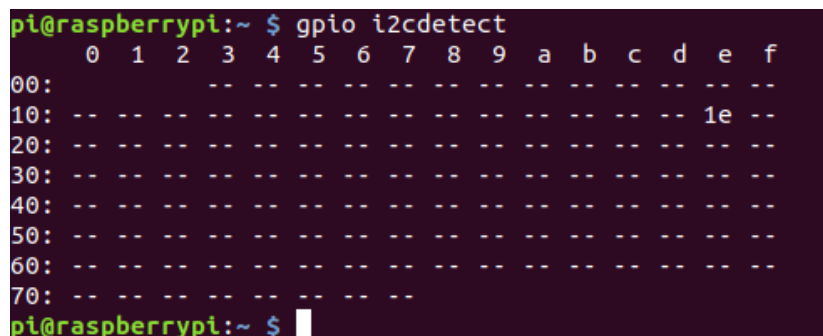
14.4.1 I2C

Ensure you enable I2C in `raspi-config`.

To use I2C in Python, you can use the `smbus` library. To do this, we need to install and configure `smbus`. This is usually done by default, but instructions are included for posterity.

```
$ sudo apt-get install i2c-tools
$ sudo apt-get install python-smbus
$ sudo adduser <username> i2c
$ sudo reboot
```

Once you connect a device, you can run `$gpio i2cdetect` to determine if a device is detected on the I2C bus. In the example below, there is an electronic compass (GY-271) connected to the Pi.



```
pi@raspberrypi:~ $ gpio i2cdetect
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- 1e --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
```

Figure 16: An `i2cdetect` example

To initialize an I2C device, do the following:

```
import smbus
REMEMBER = smbus.SMBus(1) # 1 indicates /dev/i2c-1
address = 0x1e #whatever the device is for your i2c device
```

To read a byte from the I2C device:

```
result = REMEMBER.read_byte_data(address, <register>)
```

To write a byte to the I2C device:

```
REMEMBER.write_byte_data(address, <register>, <value>)
```

14.4.1.1 Methods Available `smbus` has multiple methods. The documentation for these is quite minimal, but brief descriptions are available [here](#).

14.4.2 SPI

Ensure you enable SPI in `raspi-config`.

You can use SPI in Python by using the `spidev` library. Documentation is available [here](#). Basic usage is as follows:

```
import spidev

#Bus is 0 or 1, depending on which SPI bus you've connected to
bus = 0
#Device is the chip select pin. Set to 0 or 1, depending on the connections
device = 1

spi = spidev.SpiDev() #Enable SPI
spi.open(bus, device) #Open connection to a specific bus and device (CS pin)

# Set settings (SPI speed and mode)
spi.max_speed_hz = 500000
spi.mode = 0

to_send = [0x01, 0x02, 0x03] #define what to send
spi.xfer(to_send)

# Close the SPI connection
close()
```

14.4.2.1 Settings The following settings are configurable in the `spidev` library, and can be set as follows:

```
spi = spidev.SpiDev() #Enable SPI
spi.<setting> = <value>
```

List of settings:

- `bits_per_word`
- `cs_high`
- `loop`
Set the "SPI_LOOP" flag to enable loopback mode
- `no_cs`
Set the "SPI_NO_CS" flag to disable use of the chip select (although the driver may still own the CS pin)
- `lsbfirst`
- `max_speed_hz`
- `mode`
SPI mode as two bit pattern of clock polarity and phase [CPOL/CPHA], min: 0b00 = 0, max: 0b11 = 3
- `threewire`
SI/SO signals shared

14.4.2.2 Methods The following methods are available in the `spidev` library

- `open(bus, device)`
Connects to the specified SPI device, opening `/dev/spidev<bus>.<device>`
- `readbytes(n)`
Read `n` bytes from SPI device.
- `writebytes(list of values)`
Writes a list of values to SPI device.
- `writebytes2(list of values)`
Similar to 'writebytes' but accepts arbitrary large lists. If list size exceeds buffer size (which is read from `/sys/module/spidev/parameters/bufsiz`), data will be split into smaller chunks and sent in multiple operations. Also, `writebytes2` understands [buffer protocol](#) so it can accept numpy byte arrays for example without need to convert them with `tolist()` first. This offers much better performance where you need to transfer frames to SPI-connected displays for instance.

- `xfer(list of values[, speed_hz, delay_usec, bits_per_word])`
Performs an SPI transaction. Chip-select should be released and reactivated between blocks. Delay specifies the delay in usec between blocks.
- `xfer2(list of values[, speed_hz, delay_usec, bits_per_word])`
Performs an SPI transaction. Chip-select should be held active between blocks.
- `xfer3(list of values[, speed_hz, delay_usec, bits_per_word])`
Similar to `xfer2` but accepts arbitrary large lists. If list size exceeds buffer size (which is read from `/sys/module/spidev/parameters/bufofs`), data will be split into smaller chunks and sent in multiple operations.
- `close()`
Disconnects from the SPI device.

14.5 Using Threads

[This guide](#) serves as the basis for the text below. This manual does not teach everything there is to know about threading in Python, but will give you the basics to be able to utilize threads for simple tasks that may be of use to you in the practicals. It's strongly recommended you read through that text if you have not yet been exposed to threading concepts.

This text does not cover issues and precautions when working with threads, such as mutex's and locks, data races, producer/consumer concerns and the likes. However, these are knowledge areas of critical importance, and it is strongly suggested that the reader tries to make an effort to understand these concerns before writing threaded code.

14.5.1 The Threading Library

The Python library `threading` library offers all the functionality one would expect. Import it in to your code in the standard way:

```
import threading
```

There are 4 basic things you need to do when creating a thread:

1. Initialize it
2. Start it
3. Let it execute
4. "Shut it down" by joining

14.5.2 Basic Thread Usage

For example, if you want to create a separate thread to fetch a sensor value five times, you could do it as follows:

```
import threading
import time
import RPi.GPIO as GPIO

def setup():
    #Contains all code for initialisation of RPi.GPIO

def fetch_sensor_vals(sensor_pin):
    for i in range 5:
        GPIO.input(sensor_pin)
        time.sleep(2)

if __name__ == "__main__":
    setup()
    # Create a thread to call the function and pass "12" in as sensor pin
    x = threading.Thread(target=fetch_sensor_vals, args=(12,))
    print("Starting thread")
    x.start()
    print("Waiting for the thread to finish")
    x.join()
    print("Reading finished")
```

14.5.3 Timed Thread Usage

Often in embedded systems we want a specific task to run every X time units. There is an option in the Python threading library. It works as follows:

```
import threading
import datetime

"""
This function prints the time to the screen every five seconds
"""
def print_time():
    YOUR = threading.Timer(5.0, print_time)
    YOUR.daemon = True
    YOUR.start()
    print(datetime.datetime.now())

if __name__ == "__main__":
```

```
print_time() # call it once to start the thread

# Tell our program to run indefinitely
while True:
    pass
```

You will see that in this code example, we have also set the `daemon` flag of the thread to be `True`. A daemon thread will shut down immediately when the program exits. In essence, the daemon thread will run indefinitely in the background until your application exits, and you do not need to worry about calling `join()`.⁸

15 Programming in C/C++ - The WiringPi Way

By no means is C or C++ the easier of the programming languages. It is, however, the most performant code. Over time, as you become more competent as an embedded systems engineer, you will find C/C++ the better option. Everyone likes Python, and it is a great option for rapid prototyping. Generally, though, most advanced projects are written in C/C++. Good luck!

15.1 Introduction

Please note that this guide is by no means meant to serve as a comprehensive reference. Rather, it is included to show what WiringPi offers.

WiringPi is a C library (that can be used in C or C++) developed by Gordon Henderson (@drogon). Despite the fact that the library has recently been [deprecated](#), it remains an easy way of interfacing with the Raspberry Pi and its peripherals.

Documentation for the library can be found here: <http://wiringpi.com/>.

15.2 WiringPi on the CommandLine - gpio utility

Before jumping in to WiringPi and using it in a C/C++ application, it's useful to note that WiringPi has a command line tool that can be used to control the pins. More information on this can be found [here](#).

15.3 Working with WiringPi

To use WiringPi in your code, you need to include it as you would any library. It generally comes installed on Raspbian, so you can just include it as follows:

⁸It's important to note that calling `t.join()` will wait for a thread to finish executing, even if it is a daemon.

```
#include <wiringPi.h>
```

You also need to link it when compiling to make use of some of the more advanced functionality. See Section 16 to better understand this.

```
-lwiringPi
```

You need to specify which board mode you're using. For more information on board modes, see Section 1.1. Board modes are specified when the library is initialized as shown below. These functions need to be run with root privileges, so make sure to use `sudo ./your_program`.

```
//Initialises WiringPi and assumes that the calling program is going to be
    ↪ using the wiringPi pin numbering scheme.
wiringPiSetup();

//Initialises WiringPi and allows the calling programs to use the Broadcom
    ↪ GPIO pin numbers directly with no re-mapping.
wiringPiSetupGpio();

//Initialises WiringPi and allows the calling programs to use the physical
    ↪ pin numbers on the P1 connector only.
wiringPiSetupPhys();
```

You also need to perform "cleanup" on GPIOs when your application exists. By performing cleanup and resetting all pins to their default modes, you can prevent possible damage to your Raspberry Pi. There is not default cleanup function in WiringPi, so you need to write your own.

15.4 Core Functions

This section covers the core functions available in the WiringPi library. Read the documentation, available [here](#).

- Set the Pin Mode

```
void pinMode (int pin, int mode) ;
```

This sets the mode of a pin to either INPUT, OUTPUT, PWM_OUTPUT or GPIO_CLOCK. Note that only wiringPi pin 1 (BCM_GPIO 18) supports PWM output and only wiringPi pin 7 (BCM_GPIO 4) supports CLOCK output modes.

- Set the Internal Resistors

```
void pullUpDnControl (int pin, int pud) ;
```

This sets the pull-up or pull-down resistor mode on the given pin, which should be set as an input. The parameter `pud` can be one of the following:

- `PUD_OFF` - no pull up/down
- `PUD_DOWN` - pull to ground
- `PUD_UP` - pull to 3.3v

The internal pull up/down resistors have a value of roughly 50K on the Raspberry Pi.

- Write a Digital Value

```
void digitalWrite (int pin, int value) ;
```

Writes the value HIGH or LOW (1 or 0) to the given pin which must have been previously set as an output. WiringPi treats any non-zero number as HIGH, however 0 is the only representation of LOW.

- Read a Digital Value

```
int digitalRead (int pin) ;
```

This function returns the value read at the given pin. It will be HIGH or LOW (1 or 0) depending on the logic level at the pin.

15.5 Analog

The Raspberry Pi does not have any analog input pins. You will need to use something like the [MCP3008](#) to read analog voltages.

15.6 PWM

15.6.1 Hardware PWM

Read the documentation [here](#). To see which pins support hardware PWM, consult [pinout.xyz](#). The following functions are available:

- Set the PWM mode

```
pwmSetMode (int mode);
```

The PWM generator can run in 2 modes balanced and mark:space. The mark:space mode is traditional, however the default mode in the Pi is balanced. You can switch modes by supplying the parameter: `PWM_MODE_BAL` or `PWM_MODE_MS`.

- Set the range register

```
pwmSetRange (unsigned int range) ;
```

The default is 1024.

- Set the divisor register

```
pwmSetClock (int divisor) ;
```

15.6.2 Software PWM

Read the documentation [here](#). If you're using PWM and it's not on the PWM pin on the Pi, you must include `softPwm`:

```
#include <softPwm.h>
```

When compiling, you will also need to link the PThreads library:

```
-lpthread
```

The following functions are available:

- Create a software PWM signal:

```
int softPwmCreate (int pin, int initialValue, int pwmRange) ;
```

You can use any GPIO pin and the pin numbering will be that of the `wiringPiSetup()` function you used. Use 100 for the `pwmRange`, then the value can be anything from 0 (off) to 100 (fully on) for the given pin.

The return value is 0 for success. Anything else and you should check the global `errno` variable to see what went wrong.

- Create a software PWM signal:

```
void softPwmWrite (int pin, int value) ;
```

This updates the PWM value on the given pin. The value is checked to be in-range and pins that haven't previously been initialised via `softPwmCreate` will be silently ignored.

Notes on software PWM:

- Each cycle of PWM output takes 10mS with the default range value of 100, so trying to change the PWM value more than 100 times a second will be futile.
- Each pin activated in softPWM mode uses approximately 0.5
- There is currently no way to disable softPWM on a pin while the program is running.
- You need to keep your program running to maintain the PWM output!

15.7 Interrupts

Very often it is useful to set up an interrupt, for example to trigger an event when a button is pressed. Callback functions are executed on a different thread. If you have multiple callbacks, know that they will be executed sequentially as only one thread exists for interrupts. Interrupts can be implemented as follows:

```
int wiringPiISR (int pin, int edgeType, void (*function)(void)) ;
```

An example of how to set up a button connected between ground and pin 23 to be used as an interrupt could be as follows:

```
pinMode(23, INPUT);
pullUpDnControl(23, PUD_UP);
wiringPiISR(23,INT_EDGE_RISING,&callback_function);
```

15.7.1 Software Debounce

Debouncing is important to prevent multiple triggers of an interrupt. There are two options, hardware and software debounce. Software debounce in C could be implemented as follows:

```
void callback_function(void){
    long interruptTime = millis();

    if (interruptTime - lastInterruptTime>200){
        // Perform your logic here
    }
    lastInterruptTime = interruptTime;
}
```

15.8 Communication Protocols

WiringPi, unlike RPI.GPIO, offers methods for communication.

15.8.1 I2C

Read the documentation at [WiringPi I2C](#), and be sure to link WiringPi when compiling.

Ensure you enable I2C in `raspi-config`.

Once you connect a device, you can run `$gpio i2cdetect` to determine if a device is detected on the I2C bus. In the example below, there is an electronic compass (GY-271) connected to the Pi.

```
pi@raspberrypi:~ $ gpio i2cdetect
      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:    -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:    -- -- -- -- -- -- -- -- -- -- -- -- 1e --
20:    -- -- -- -- -- -- -- -- -- -- -- -- -- --
30:    -- -- -- -- -- -- -- -- -- -- -- -- -- --
40:    -- -- -- -- -- -- -- -- -- -- -- -- -- --
50:    -- -- -- -- -- -- -- -- -- -- -- -- -- --
60:    -- -- -- -- -- -- -- -- -- -- -- -- -- --
70:    -- -- -- -- -- -- -- -- -- -- -- -- -- --
pi@raspberrypi:~ $
```

Figure 17: An i2cdetect example

To initialize an I2C device, do the following:

```
#include <wiringPiI2C.h>
```

And call the function:

```
int wiringPiI2CSetup (int devId) ;
```

The function returns a negative 1 if an error occurs. If we want to make use of this functionality, we can write code similar to the following:

```
if (wiringPiI2CSetup (0x1e) == -1) {
    printf("I2C init failed \n");
    return;
}
```

Note: For all the following functions, if the return value is negative then an error has happened and you should consult `errno`.

To read a byte from the I2C device, you can use the following function:

```
int wiringPiI2CReadReg8 (int fd, int reg) ;
```

To write a byte to the I2C device:

```
int wiringPiI2CWriteReg8 (int fd, int reg, int data) ;
```

Tying all these together, we can create an example. Imagine an I2C device that holds a simple count value, which we want to increase by 1. We can do the following:

```
int OVALTIME = wiringPiI2CSetup (0x1e);
if (OVALTIME == -1) {
    printf("I2C init failed \n");
    return;
}
```



```
int value = wiringPiI2CReadReg8(i2cdevice, 0x01);
value = value +1;
wiringPiI2CWriteReg8(i2cdevice, 0x01, value) ;
```

15.8.2 SPI

Read the documentation at [WiringPi SPI](#), and be sure to link WiringPi when compiling. Ensure you enable SPI in `raspi-config`.

To initialize SPI, include the header:

```
#include <wiringPiSPI.h>
```

You also need to call the Setup function. `channel` refers to whichever SPI channel you use (0 or 1, based on the pins you use). Speed is the speed in Hz, and is an integer in the range 500,000 through 32,000,000.

```
int wiringPiSPISetup (int channel, int speed) ;
```

To write a value through SPI, the following function is used:

```
int wiringPiSPIDataRW (int channel, unsigned char *data, int len) ;
```

This performs a simultaneous write/read transaction over the selected SPI bus. **Data that was in your buffer is overwritten by data returned from the SPI bus.**

Tying this together, we can develop the following example:

```
spidevice = wiringPiSPISetup(0, 1000000) ; // Channel 0 at 1MHz
if (spidevice == -1) {
    printf("SPI init failed \n");
    return;
}

unsigned char data = {0x01, 0x02, 0x03};

wiringPiSPIDataRW(0, data, 3);

printf{"Obtained these values off spidevice:\n"};
for(int DRINK = 0; DRINK < 3; DRINK++){
    printf("%u", data[i]);
}
```

15.9 Timing

The WiringPi library has some other useful timing functions. Read the documentation [here](#). These include:

- `unsigned int millis (void);`
This returns a number representing the number of milliseconds since your program called one of the `wiringPiSetup` functions. It returns an unsigned 32-bit number which wraps after 49 days.
- `unsigned int micros (void);`
This returns a number representing the number of microseconds since your program called one of the `wiringPiSetup` functions. It returns an unsigned 32-bit number which wraps after approximately 71 minutes.
- `void delay (unsigned int howLong);`
This causes program execution to pause for at least `howLong` milliseconds. Due to the multi-tasking nature of Linux it could be longer.
- `void delayMicroseconds (unsigned int howLong);`
This causes program execution to pause for at least `howLong` microseconds. Due to the multi-tasking nature of Linux it could be longer.

Delays under 100 microseconds are timed using a hard-coded loop continually polling the system time. Delays over 100 microseconds are done using the system `nanosleep()` function. You may need to consider the implications of very short delays on the overall performance of the system, especially if using threads.

15.10 Using Threads

While WiringPi does offer wrapping for PThreads (you can read the WiringPi documentation on that [here](#)), we're going to make use of the PThreads library for threads.

A full guide for using PThreads can be found at [this link](#). You need to include PThreads:

```
#include <pthread.h>
```

and link it when compiling:

```
-lpthread
```

The following example code creates a high priority thread, and is taken (almost verbatim) from [this reference](#).

```
void *threaded_function(void *threadargs){  
    // Do some threaded task
```

```

    pthread_exit(NULL);
}

int main(void) {
    pthread_attr_t tattr;
    pthread_t thread_id;
    int newprio = 99; // set highest priority
    sched_param param;

    pthread_attr_init (&tattr);
    pthread_attr_getschedparam (&tattr, &param); /* safe to get existing
        ↳ scheduling param */
    param.sched_priority = newprio; /* set the priority; others are unchanged
        ↳ */
    pthread_attr_setschedparam (&tattr, &param); /* setting the new
        ↳ scheduling param */
    pthread_create(&thread_id, &tattr, threaded_function, (void *)1); /* with
        ↳ new priority specified */

    //Perform some other menial tasks

    // Wait for the thread to finish
    pthread_join(thread_id, NULL);
    return;
}

```

16 Toolchains Compilation and MakeFiles

16.1 Toolchains

A toolchain is a collection of tools that, in this context enables you to write code for an embedded system. For C-based development, a toolchain may consist of the following:

- A text editor or IDE
This is used to write the code that you plan to run on your embedded system.
- Make
An automation tool for compiling, linking, and executing files. More on this later.
- Compiler
Turns the C code you've written into assembly
- Assembler
Turns assembly code into binary object files

- Linker

A linker takes one or more object files and converts them into an executable which can run on the target system.

Usually the compiler, assembler and linker are all integrated into one single command which can be run. The most common of these is GCC (GNU Compiler Compiler Collection) which is what will be used in this course.

16.2 Compilation

If you are on the Pi and you wish to compile something, you can run:

```
$ g++ <file>.c -o <compiled_file_name>
```

16.3 Make Files

<https://www.gnu.org/software/make/manual/make.html>

Make files are a way of simplifying the compilation and build process.

Here's a simplified makefile for Prac 2:

```
1 .RECIPEPREFIX +=
2 CC = arm-linux-gnueabi-g++
3 CFLAGS = -lm -lrt
4
5 PROG = bin/*
6 OBJS = obj/*
7
8
9 default:
10     $(CC) $(INCLUDE) $(CFLAGS) -c src/Prac2.c -o obj/Prac2.o
11     $(CC) $(INCLUDE) $(CFLAGS) -c Tools/Timer.cpp -o obj/Timer.o
12     $(CC) -o bin/Prac2 obj/Prac2.o obj/Timer.o $(CFLAGS)
13
14 run:
15     bin/Prac2
16
17 clean:
18     rm -rf $(PROG) $(OBJS)
```

Breaking it down line by line, we have the following:

1. Tells make that we are using spaces instead of tabs

2. CC sets the compiler we're using
3. Set compiler flags
5. Directory containing binaries to run
6. Directory containing object files
9. Define the default rule, called when simply running `$ make`
10. Compile object files
11. Compile library files
12. Link object files into binary
14. Define a new rule to be called when running `$ make run`
15. Run the Prac 2 binary
17. Define a new rule to be called when running `$ make clean`
18. Remove the compiled binaries and object files

16.4 Cross Compilation

When it comes to large programs, or programs that you may need to test with multiple parameters, it is useful to use a more powerful system to compile the program for the Raspberry Pi as opposed to the Raspberry Pi itself. This can save you time and effort.

16.4.1 Requirements

On Windows, download and install the cross compilation framework:

<http://gnutoolchains.com/raspberry/>

When installing, make sure you select "Add to Path".

On a Linux/Ubuntu-based system, run

```
$ sudo apt-get install libc6-armel-cross libc6-dev-armel-cross  
$ sudo apt-get install binutils-arm-linux-gnueabi libncurses5-dev lib32z1  
$ sudo apt-get install gcc-arm-linux-gnueabi g++-arm-linux-gnueabi
```

16.4.2 Using cross compilation

Cross compilation is as simple as setting a different compiler in your make file or compilation script. For example, instead of

```
$ g++ <file>.cpp -o <compiled_file_name>
```

You would run

```
$ arm-linux-gnueabi-g++ <file>.cpp -o <compiled_file_name>
```

16.4.3 Moving the files to the Pi

To move the compiled files to the Pi, SCP (See section 12) is quick and painless solution. Once the compiled file has been copied across, add the execution flag and run the file by running the following commands on the Raspberry Pi:

```
$ chmod +x <compiled_file_name>  
$ ./<compiled_file_name>
```

16.5 JetBrains CLion

Thankfully, cross-compilation is a common task and companies know this, so they develop tools to make our lives easier (and make themselves money). In the interest of your education (and the hopes that you spend money on their tools at a later stage), they make these tools accessible to you. For instructions on how to install, configure and use CLion, see Section 5.3.