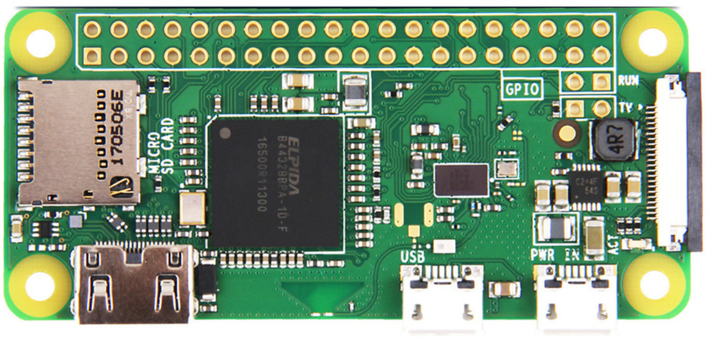
University of Cape Town



EEE3096S/EEE3095S

Embedded Systems II

# Work Packages 2021

30 August 2021

## Abstract

Welcome to the practicals for EEE3096S. These instruction are applicable to all practicals so please take note! It is critical to do the pre-practical/tutorial work. Tutors will not help you with questions with answers that would have been known had you done the pre-practical work. The UCT EE Wiki (wiki.ee.uct.ac.za) is a very useful point to find any additional learning resources you might need. Use the search functionality on the top right of the page.

## Practical Instructions

- **Naming**
  All files submitted to Vula need to be in the format. Marks will be deducted if not.

```
pracnum_studnum1_studnum2.fileformat
```

- **Submission**
  - Work packages consist of a tutorial and a practical. Be sure you submit to the correct submission on Vula!
  - If working in a prac group, do only one submission to Vula for the team (i.e. by one of the team members). Ensure all files submitted to Vula are in the format: pracnum_studnum1_studnum2.fileformat (although if you are submitting a zip file, you do not need to have all the sub-files named according to the student numbers, but do please have the report filename containing student numbers to ensure the marker is aware that it is a team submission).
  - All text assignments must be submitted as pdf, and pdf only.
  - Code within PDFs should not be as a screenshot, but as text.
  - Where appropriate, each pdf should contain a link to your GitHub repository.
- **Groups**
  All practicals are to be completed in groups of 2. If a group of 2 can't be formed, permission will be needed to form a group of 3. You are to collaborate online with your partners. See more here.
- **Mark Deductions**
  Occasionally, marks will be deducted from practicals. These will be conveyed to you in the practical, but many will be consistent across practicals, such as incorrectly names submissions or including code as a screenshot instead of formatted text.
- **Late Penalties**
  Late penalties will be applied for all assignments submitted after the due date without a valid reason. They will work as 10% deduction per day, up to a maximum of 60%. After this, you will receive 0% and have the submission marked as incomplete.

## Contents

# Work Package 3

## 1 Practical - I2C and PWM

You have been put in charge of implementing ES Games's latest proof of concept for a new gambling machine where users will try and guess a number. Of course, being a sensible individual, you do not condone reckless gambling and know when to draw the line. To this end, you tender your two-week resignation and start looking for a job at a gaming company that better aligns with your values, perhaps *Even Mo' Jang*, *Nin-eleven-do* or *Beth-Is-Ma*. However as you still have two weeks, your bosses assign your last task to be implementing the logic and user feedback systems.

### 1.1 Overview

You're going to design a simple guessing game with feedback to the user.

A circuit diagram is shown in Figure 1.1 below.

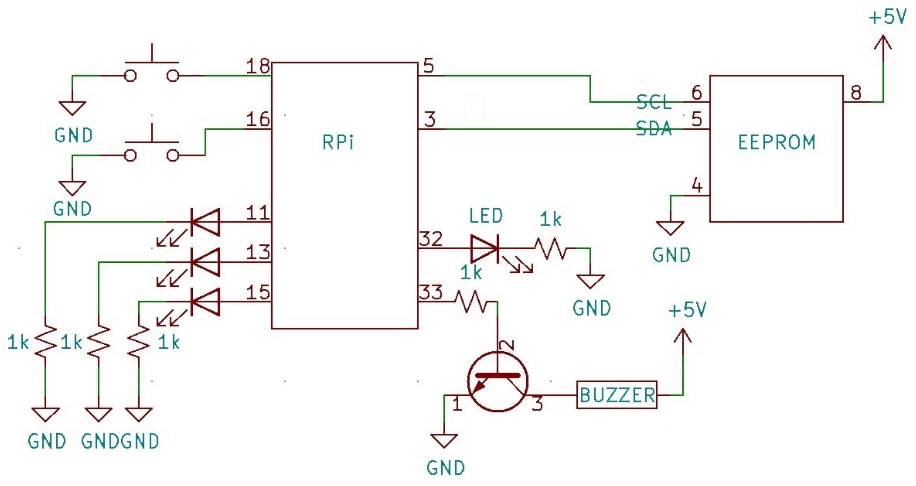


Figure 1.1: The circuit diagram for the system

### 1.2 Requirements

You must do the following:

- Use Python 3!
- Install libraries as required

  - You will need to use pip3, as we're using Python 3
    `sudo apt install python3-pip`
  - You'll need RPi.GPIO for Python 3
    `sudo apt-get install python3-rpi.gpio`
  - You'll need SMBus 2 for Python 3
    `sudo pip3 install smbus2`
- Read up on the data sheet for the EEPROM.
- Fetch the prac code off the GitHub repository

### 1.3 EEPROM

Read through the data sheet. In order to simplify things for you, an EEPROMUtils library has been provided. The library instantiates an EEPROM class on a specified bus with a defined address.

Some things you should know about the EEPROM from the datasheet and how the library has been written:

- The EEPROM is organised into 128 pages consisting of 32 bytes each.
- There are 32768 bits at a word size of 8 bits each, meaning there are 4096 addressable words.

The library provided creates a new abstraction called "blocks":

- Each block can be addressed from 0...n, where n is the maximum amount of blocks
- Each block consists of 4 words (32 bits)
- The write block method will write to as many registers as required by the data passed to it. I.e you can write more than just 4 words using the `write_block` method

The following methods are provided:

- write block
  Writes an array of data starting at the beginning of a particular block
- write byte
  Writes a singular byte to a given address.
- read block
  Reads a given amount of registers from the start of a specified block
- read byte
  Reads a single byte from a specified address.

- clear
  Writes `0x00` to all memory.
- Populate mock scores
  Adds 4 mock scores to the EEPROM.

## 1.4 ASCII

We can't store text in EEPROM, only 1s and 0s. Thankfully Python will convert numbers between any base, but characters are still unknown. To convert between the two, we can use `ord()` and `chr()` in Python.

```
orc('c') = 99 = 0x63 = 0b01100011
chr(99) = chr(0x63) = chr(0b01100011) = 'c'
```

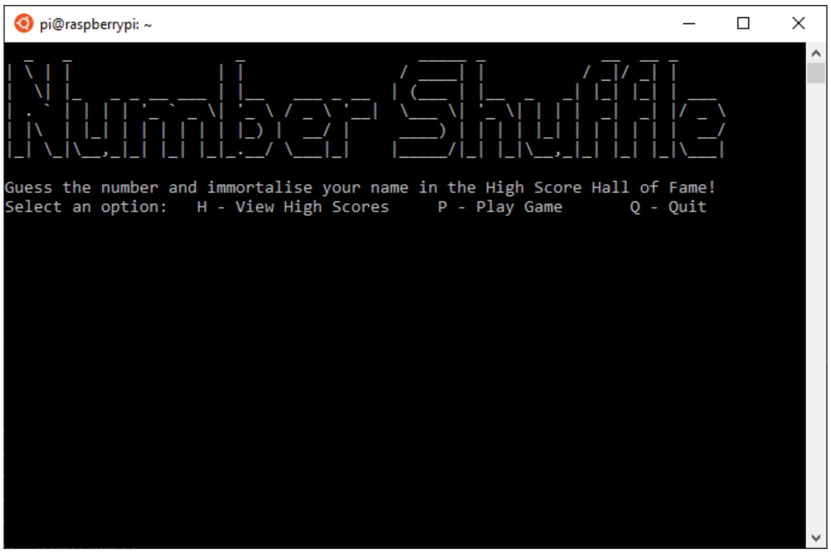
## 1.5 Design overview

The requirements of each component and how they respond and are required to act are detailed by comments in the code source.

1. Two buttons should be connected, using interrupts and debouncing, which do the following:
   (a) Button 1 - Increases the user's guess value and update the value shown on the LEDs
   (b) Button 2 - Submit the user's guess
2. 4 LEDs
   - 3 to show the user's current guess value
   - 1 should show the accuracy of the guess
3. One Buzzer

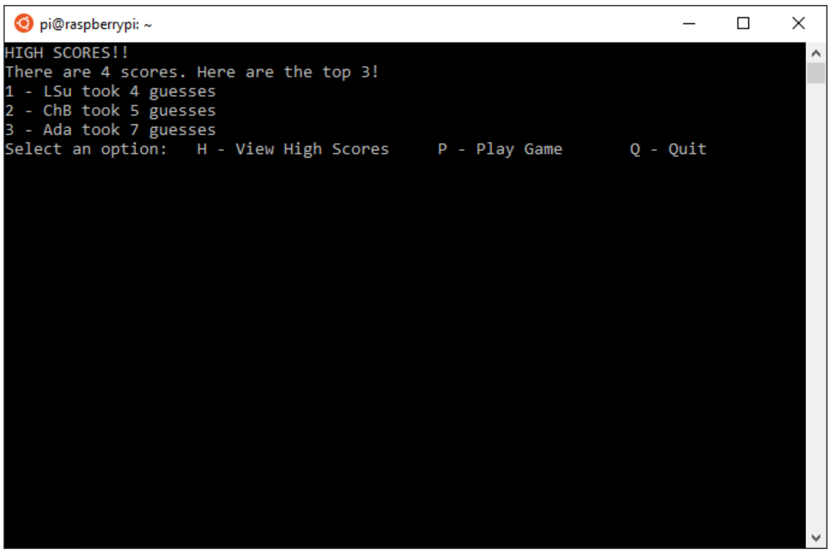
## 1.6 Playing the game

When launched, the user is presented with this screen:



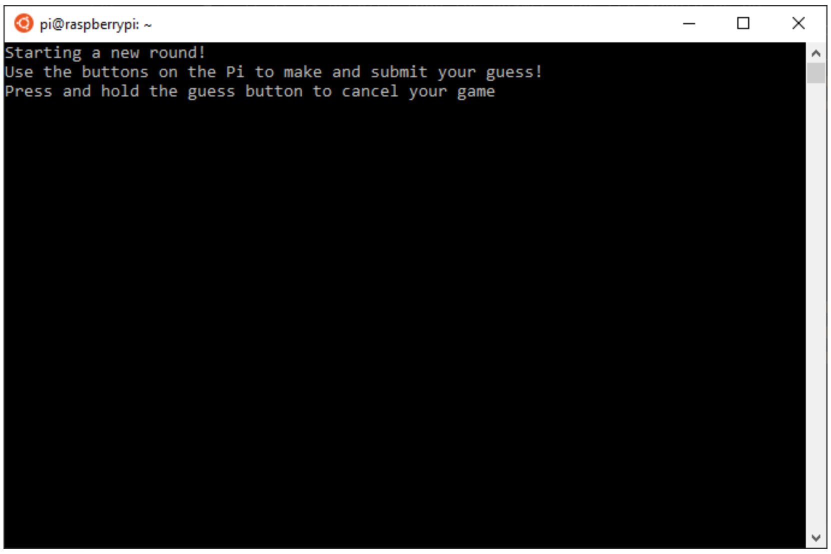
```
 _   _                 _                 ____  _            __  __ _      
| \ | |_   _ _ __ ___ | |__   ___ _ __  / ___|| |__  _   _ / _|/ _| | ___ 
|  \| | | | | '_ ` _ \| '_ \ / _ \ '__| \___ \| '_ \| | | | |_| |_| |/ _ \
| |\  | |_| | | | | | | |_) |  __/ |     ___) | | | | |_| |  _|  _| |  __/
|_| \_|\__,_|_| |_| |_|_.__/ \___|_|    |____/|_| |_|\__,_|_| |_| |_|\___|

Guess the number and immortalise your name in the High Score Hall of Fame!
Select an option:   H - View High Scores     P - Play Game       Q - Quit
```

When the player selects high scores, they should see a screen like this, updated for the latest high scores. Notice how the system knows how many scores are stored but only presents the top three scores.



```
HIGH SCORES!!
There are 4 scores. Here are the top 3!
1 - LSu took 4 guesses
2 - ChB took 5 guesses
3 - Ada took 7 guesses
Select an option:   H - View High Scores     P - Play Game       Q - Quit
```

When a new game is played, this screen is displayed:



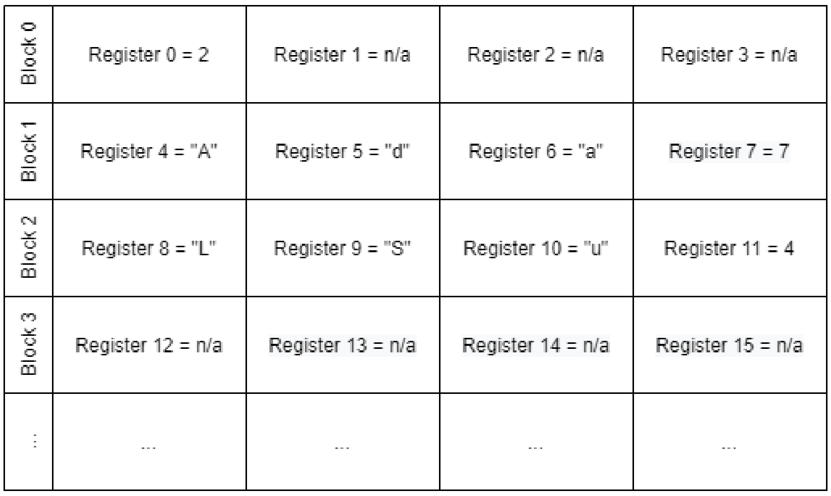
```
Starting a new round!
Use the buttons on the Pi to make and submit your guess!
Press and hold the guess button to cancel your game
```

Notes on gameplay mechanics are as follows:

- At the start of each game, a new random number between 0 and $2^3$ is generated. This is kept secret from the player.
- The user presses an "increase guess" button, and each press updates the value shown on the LEDs.
- When the user submits their guess, the PWM LED and the buzzer update appropriately, as described in the P4.py source file.
- If the user presses and holds the submit guess button, the buzzer and LEDs are turned off and they are taken back to the main menu
- If the user wins by guessing the number correctly, they must be prompted to enter in their name. Your code should handle the scenario that they enter in more than three letters for a name. The system should then store their score to EEPROM and take them back to the main menu.

## 1.7 Storing Scores

High scores will be stored on EEPROM with the first byte of the first block storing the amount of scores, and from the first block onwards. An example of this is shown in the figure below.



| | | | | |
|---|---|---|---|---|
| Block 0 | Register 0 = 2 | Register 1 = n/a | Register 2 = n/a | Register 3 = n/a |
| Block 1 | Register 4 = "A" | Register 5 = "d" | Register 6 = "a" | Register 7 = 7 |
| Block 2 | Register 8 = "L" | Register 9 = "S" | Register 10 = "u" | Register 11 = 4 |
| Block 3 | Register 12 = n/a | Register 13 = n/a | Register 14 = n/a | Register 15 = n/a |
| ... | ... | ... | ... | ... |

It can be seen that the first word in the first block stores the number of scores. From the first block onwards, we store the scores, with the first three words of a block storing the player's name, and the final word storing how many guesses they took.

## 1.8 Walkthrough

1. Start by enabling I2C in raspi-config, under "Interfacing Options"
2. Do a git pull in the prac source folder to fetch the Prac 3 content
3. Build the circuit given to you, taking care not to create any short circuits and configuring things with the correct polarity.
4. Run `$ sudo i2cdetect -y 1` to see if you can see the EEPROM on 0x50
5. Open `p4.py` and write the code required. Some function templates are made available to give you a guide.
    - Be sure to take care with regards to which pin numbering is used! The circuit diagram details board mode, not BCM mode! If you chose to make any adjustments, be sure to cater for the new pin numbering.

## 1.9 Some Hints

1. You will need to debounce your button presses. The Rpi.GPIO library provides appropriate methods.
2. The source code provided to you has a lot of implementation in it already. Ensure you read and understand it before embarking out on the practical.
3. It's important to cleanup the GPIO pins on exit. This has been done for you.

4. You can use visual studio code or Pycharm Professional (which you can get with a GitHub Education account) to use Python over SSH. This is different from cross-compilation. Guides exist online on how to set this up. Here is a video on how to mimic remote development for the Raspberry Pi Zero.

5. You don't need to edit any code in the EEPROM Utils library, but it does provide some insight into how you might apply some logic!

## 1.10 Deliverables

This practical requires a demonstration of the working implementation to a tutor during your lab session.

The submission required for this practical is a singular PDF, correctly named, with the following each having it's own heading. You do not need to write the report in LaTeX, but hopefully after Prac 2 you're encouraged by how nice it is to focus on content rather than formatting using a decent text editor! To switch to a single page in LaTeXfrom the example source given to you in Prac 3, change the documentclass to "onecolumn" instead of "twocolumn". Monospaced text boxes can be created using the `lstlisting` environment.

1. A UML use-case diagram of the system. You should know these details from EEE3097S or any design course. If you need a refresher, watch this YouTube Video.

2. The Prac3.py code contents

    - Each method needs to be in it's own separate text box.
    - All the code needs to be included, even if it was a part of the provided code. Do not include the EEPROM library.
    - Use a monospaced font such as Consolas. Using the correct environment in LaTeXwill do this for you. You can also look into syntax highlighting

**!!!Marks will be deducted for not following instructions.!!!**