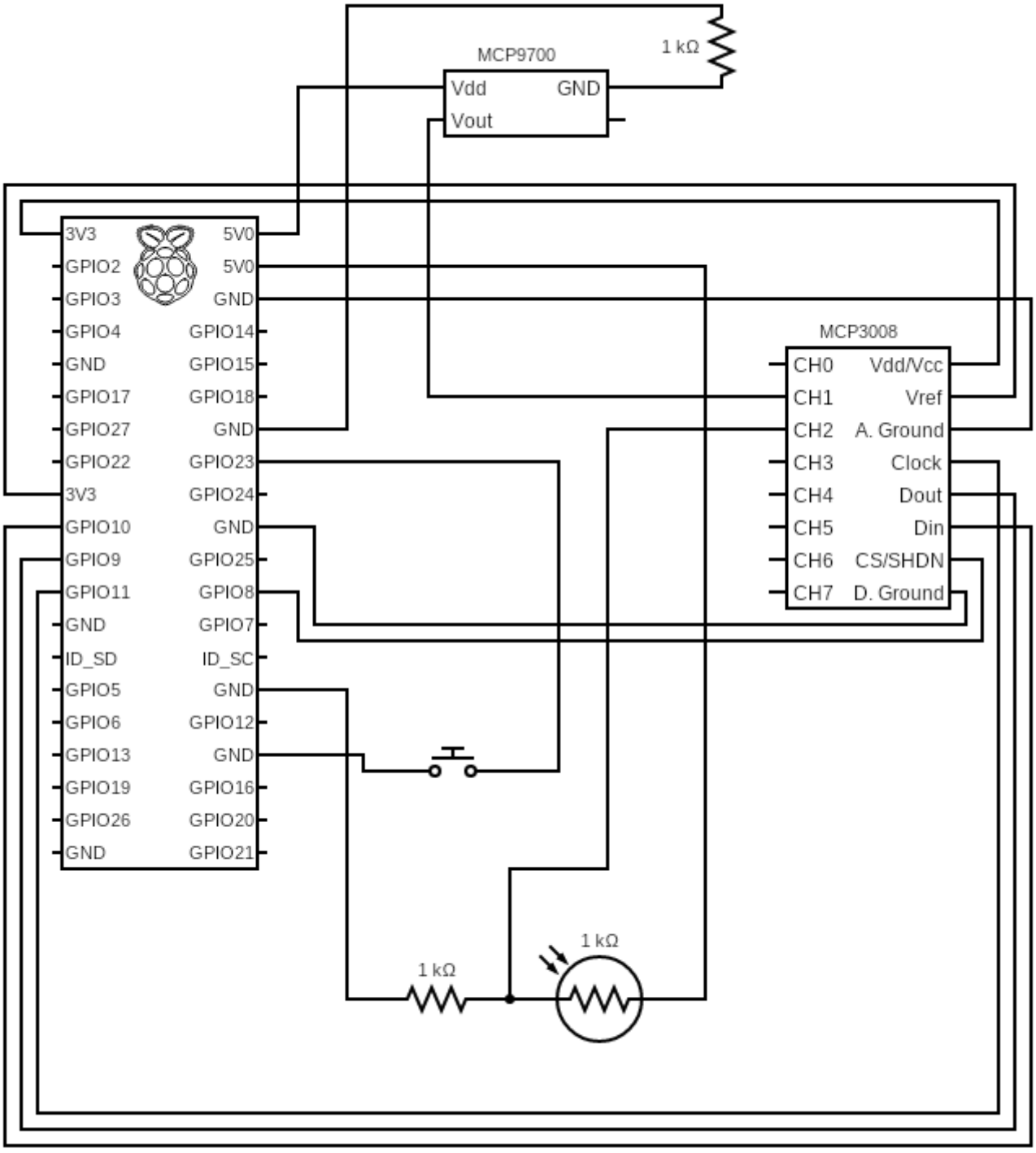
Work Package 4 - Practical

CLXBHE005, KMPKWE002

EEE3096S 2021

Demonstration Video

1. The circuit diagram:



2. Validation and Testing

- Light senser
  - When light is provided on LDR, the LDR decreases the resistance with respect to receiving luminosity on the component's sensitive surface. As LDR varies with light the voltage will be varied. When the light level decreases, the resistance of the LDR increases, as this resistance increases it will cause voltage across LDR to also increase. The voltage we get is sent to ADC to be converted to digital. We can see from our code we get ADC values. We use 10-bit resolution to analogue to digital converter to have optimal accuracy, by this resolution we can validate that the ADC values we got mostly accurate
- Temperature senser
  - The temperature sensor will sample the current room temperature, the sensors' output will then be sent to analogue to digital converter. The output resistance from the sensor must be moderate (not high or low) for it to be compatible with ADC inputs. A high-resolution ADC is required in many cases as the magnitude of the signal from the sensor is quite small hence, we used 10-bit resolution. A high-resolution ADC is required so that the low-level signal from the sensor can be converted into digital information. When the temperature sensor is powered on, the voltage output will be proportional to the temperature. Generally, A temperature of 25 degrees C will result in an output of 0.750V, we can see from our values that the relationship still holds.

3. The Prac4.py code contents

The initialisations and imports are as follows:

```
import RPi.GPIO as GPIO
import busio
import digitalio
import board
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn
import threading
import datetime
import time

# some global variables that need to change as we run the program
frequencyOfReadingData = 10.0
startTime = None

# DEFINE THE PINS USED HERE
btn_submit = 23
```

The setup method is as follows:

```
# Fuction to setup the GPIO Pins
def setup():
    #Button Setup
    GPIO.setup(btn_submit, GPIO.IN, pull_up_down=GPIO.PUD_UP)

    # Setup debouncing and callbacks
    GPIO.add_event_detect(btn_submit, GPIO.FALLING, callback=incrementFrequency, bouncetime
=300)
```

The fetch_sensor_vals method is as follows:

```
# Function to get sensor_values from the ADC
def fetch_sensor_vals(channel):
    # create the spi bus
    spi = busio.SPI(clock=board.SCK, MISO=board.MISO, MOSI=board.MOSI)
    # create the cs (chip select)
    cs = digitalio.DigitalInOut(board.D5)
    # create the mcp object
    mcp = MCP.MCP3008(spi, cs)

    if(channel == 1):
         # create an analog input channel on pin 1
        chan1 = AnalogIn(mcp, MCP.P1)
        return chan1.value, chan1.voltage

    elif(channel == 2):
        # create an analog input channel on pin 2
        chan2 = AnalogIn(mcp, MCP.P2)
        return chan2.value, chan2.voltage
```

The convertToTemp method is as follows:

```
# Function to calculate temperature from
# ADC value, rounded to specified
# number of decimal places.
def convertToTemp(tempArg, places=2):
    temp = ((tempArg)/float(1023))
    temp = round(temp * (1/0.01),places)
    return temp
```

The main method is as follows:

```
# Fuction, it uses threads to read data from the sensors
def main():
    global startTime
    runTime = (datetime.datetime.now() - startTime).total_seconds() # Calculates run time
    global frequencyOfReadingData
    tempADC_Value, tempVoltage = fetch_sensor_vals(1)
    lightADC_Value, lightVolatge = fetch_sensor_vals(2)
    print("{:<11d}{:<16d}{:05.2f} C{:>10d}".format(int(runTime), tempADC_Value, convertToTe
mp(tempADC_Value,2), lightADC_Value))

    thread = threading.Timer(frequencyOfReadingData, main)
    thread.daemon = True  # Daemon threads exit when the program does
    thread.start()
```

The printHeading method is as follows:

```
# Fuction, to print the heading of the table
def printHeading():
    print('{:<11s}{:<16s}{:<13s}{:<17s}'.format("Runtime", "Temp Reading", "Temp", "Light R
eading"))
```

Finally, the main program when executed does the following:

```
if __name__ == "__main__":
    try:
        setup()
        printHeading()
        startTime = datetime.datetime.now()
        main()
        # Tell our program to run indefinitely
        while True:
            pass
    except Exception as e:
        print(e)
    finally:
        GPIO.cleanup()
```