Data Acquisition and Handling (DAH)

Franz Muheim (Course organiser) <f.muheim@ed.ac.uk>
Ben Wynne <b.m.wynne@ed.ac.uk>

Senior Honours
Semester 1, 2019/20

Edinburgh, September 2nd, 2018

Chapter 1

DAH: Course Overview

1.1 Introduction

Data Acquisition and Handling (DAH) is a Senior Honours course, which was introduced in 2014 during a review of whole degree programme. DAH will introduce you to methods and tools of modern Data Acquisition and Handling, including Analog and Digital electronics, reading out sensors (detectors), handling and interpreting data. This course replaces JH Electronics Methods. Note that some parts of Electronics Methods (digital and analog electronics) are now taught in 2nd year Practical Physics. The DAH course will focus on data acquisition and data analysis.

1.2 Schedule

In week 1 (Tuesday 17 and Thursday 19 September 2019) there will be lectures introducing the DAH course. Laboratory work commences in week 1 (Tuesday 17 September 2019) and finishes at the end of week 11 (Thursday 28 November 2019). The laboratory sessions will take place on Tuesday and Thursday afternoons from 14:00 to 17:00 and you need to attend one of these sessions. Timetabling will automatically allocate one of the afternoons for you. The laboratory sessions will be held in JCMB 3301.

- Tuesday 17 September, 2 to 3 pm, Lecture Theatre 100, Joseph Black Building Introduction to course and Lecture 1.
- Thursday 19 September, 2 to 3 pm, Lecture Theatre 2, Hudson Beare Building : Lecture 2.
- week 1, Tuesday 17 or Thursday 19 September, 3 to 5 pm, JCMB room 3301: laboratory work.
- weeks 2 to 11: Tuesday or Thursday 2 to 5 pm, JCMB room 3301: laboratory work.

1.3 Syllabus

The outline syllabus is as follows:

- Analogue signal processing. Treatment of noise. Filtering. Buffering using sample and hold;
- Analogue to digital conversion. Sampling rates. Characteristics & errors;
- Digital to analogue signal conversion;
- Communication protocols (Bus standards). Input/Output (I/O);
- Digital signal processing. Triggering. Fourier transforms;
- Data acquisition using a Raspberry Pi and Arduino and python;
- Advanced data analysis. Multi-parameter likelihood fits;
- Practical examples, e.g. temperature sensors, ultrasound sensors, FFT spectrum analyser, synthesizer, digital signal generators, motion sensors, remote sensing, image processing, with CCDs.

1.4 Learning Outcome

On completion of this course, the student will be able to:

1. Understand core concepts of data acquisition, data handling and data analysis in physical sciences;
2. Apply standard practical laboratory techniques (e.g. routine handling of data acquisition equipment and writing short, procedural computer programs) as directed in a script to achieve a stated goal;
3. Apply advanced practical laboratory techniques (e.g. handling of complex data acquisition equipment, and writing data acquisition computer programs) with limited direction to achieve a stated or open-ended goal;
4. Apply advanced data handling and data analysis techniques (e.g., data selection and representation, multi-parameter likelihood fits and writing data analysis computer programs) with limited direction to achieve a stated or open-ended goal;
5. Present a record of an experiment or computation in an appropriate, clear and logical written form (e.g. laboratory notebook, laboratory report, fully documented computer code), augmented with figures graphs, audio or movies where appropriate.

1.5 Laboratory work

The laboratory sessions of the DAH course will take place in JCMB 3301 on Tuesday and Thursday afternoons from 14:00 to 17:00 (15:00 - 17:00 in week 1). You will need to attend one of these sessions. Timetabling will automatically allocate either Tuesday or Thursday for you. In the laboratory you will work in pairs, so you will need to choose a partner.

The laboratory work consists of checkpoints and projects, which are described in detail in Chapters 2 and 3. You will work with a Raspberry-Pi, which is a credit-card sized computer that plugs into a computer/TV screen and a keyboard. A manual will be provided. To control the Raspberry-Pi you will need to write python code. Example python scripts and code snippets will be provided on github, see `https://github.com/fmuheim/DAH`.

During weeks 1 to 6 of the DAH course you will need to complete six checkpoints. In each checkpoint you will learn a specific aspect of data acquisition or data handling and complete a prescribed number of tasks. You will work in pairs during the checkpoints. Each pair will have their own set of kit, however the Raspberry-Pi's will be shared between the Tuesday and Thursday afternoon sessions. Each pair will be given a yellow box with the required kit in which you can preserve your work for use in the following week. The boxes should be labelled with your names.

You should maintain a clear record of your work in a laboratory notebook as you work through the checkpoints. This must include diagrams of the circuits used. Python code written on the Raspberry-Pi must include explanatory comments and at each check point you need to demonstrate that your code compiles and runs correctly. Each partner will need to maintain their own notebook to demonstrate that a checkpoint has been completed.

In weeks 7 to 11 of the DAH course, you will carry out a project during the laboratory sessions. You will continue to attend during the same afternoon as for the checkpoints. The projects will build upon what you have learned during the checkpoints, but you will also encounter new material. While the checkpoints concentrated on specific data acquisition techniques and data handling methods the projects will allow you to progress towards building a small DAQ and/or data analysis system. The projects will have an open-ended aspect and are an opportunity where you can show your own initiative and demonstrate your experimental and computational skills.

You will be able to choose from a list of projects, but due to the availability of equipment, the number of spaces for each project will be limited. A signup sheet will be provided.

The equipment specific to each DAH project will be available in red boxes. Some of the parts, including Arduinos and loudspeakers, as well as the Raspberry-Pi's, will be shared between Tuesday and Thursday afternoon sessions. In addition, each pair will continue to use the yellow box in which you can preserve your work for use in the following week.

For the DAH project you will continue to work in pairs. Throughout the project, each of you should maintain a clear record of your work in your laboratory notebook. As an example, diagrams of built circuits must be included. Python code written on the Raspberry-Pi must include explanatory comments. Each partner will be required to submit an individual report for the project.

1.6 Assessment

Data Acquisition and Handling is a continuously assessed course. The overall DAH assessment will be made from three parts. The sum of the marks achieved while carrying out the checkpoints will count for 30% of the total course mark. The marks obtained for the DAH project will count for 60% of the total course mark. In addition, there will be a quiz/hand-in which will count for 10% of the total course mark.

1.6.1 Assessment of checkpoints

There are six checkpoints and checkpoints 2 to 6 will be assessed by one of the demonstrators during the laboratory hours. The assessment will be performed when you decide that you have completed the tasks for a checkpoint or parts thereof as allowed by the marking scheme. For each checkpoint a total score of between 8 and 10 marks will be awarded. In total up to 45 marks will be available for the checkpoints. Please note the following.

- While working in pairs, each student will be assessed separately. The marks awarded need not to be the same for both students.
- A marking scheme will be provided. Some of the tasks of each checkpoint may be marked together.
- You will only be awarded marks if you can demonstrate that the relevant circuit functions correctly and that your python code achieves the requested results.
- You will be assessed on your (individual) laboratory notebook and the structure and readability of your (joint) python code.
- You will be required to answer the questions given in the checkpoints.
- You are required to have separate python scripts for each task of a checkpoint. By doing this you will be able to demonstrate all parts of a checkpoint and not just the last task. For some tasks you will need to write results or figures to a file. Make sure that these are not overwritten by the next task.
- You should be able to complete each checkpoint in one afternoon. The deadlines for assessing checkpoints are as follows:

  The deadline for checkpoint 2 is week 3, for checkpoint 3 it is week 4, for checkpoint 4 it is week 5, for checkpoint 5 it is week 6 and for checkpoint 6 it is week 7.

The overall laboratory assessment will be made from the sum of marks of the check points, which constitutes 30% of the total course mark.

1.6.2 Project assessment

Your DAH project will be assessed through the submitted material. This includes your project report and your DAH software (e.g. Python scripts) and, if deemed useful, supple-

mentary material. Guidance on how to prepare these items is given below. The project will be marked according to the University Common Marking Scheme.

Report Preparation

For how to write a proper report we refer you to the workshop slides on report writing in the Senior Honours (SH) Projects course, which are available at `https://www.wiki.ed.ac.uk/display/SP/SH+Projects`. The basic layout of a DAH report will be similar to an SH project report with the main difference being that a good DAH report will be shorter and should typically be approximately 7 pages long. It is expected that the report is typed. Most students use LaTeX or Word, either is fine.

When planning and writing a report, you need to be selective about what to include in your report, it should be a concise technical document. However, it also needs to contain all the information required for the reader to understand what was achieved, i.e. with your report you need to be able to demonstrate to what extend the project was carried out successfully. It is often useful to include circuit diagrams, pictures of the setup, or plots of measurements. A good report would allow a fellow student to be able to reproduce your work. Students are advised to start writing their report as the project progresses. Experience shows that report writing will take longer than anticipated.

Each partner is required to submit an individual report for the project. If you are working with a partner, this report must make it clear which parts of the project were carried out together, which parts are only your work, and which parts were only carried out by your partner. The report must contain a signed declaration, which will be available on Learn.

Programming Code: e.g. Python

Programming code for the DAH project, e.g. using Python, written on the Raspberry-Pi or on another computer, must include explanatory comments. When reading a (python) script, a reader should easily be able to understand what the script will do. All code written (in python or another programming language) for the DAH project will need to be submitted using the "Assessment" tool on Learn. The files should be bundled up in a .zip or .tar file. A README file should be included. Submission details will be provided.

Supplementary Material

You are encouraged to submit supplementary material if you consider the material as a part of the project that does not fit into the report format. This could include your laboratory notebook, output files produced by running a python script, short videos that demonstrate how your project works, ... These can be submitted using the "Assessment" tool on Learn, or separately on a USB drive or a link to a webpage. If you have questions about the suitability of material, please consult with the Course Organiser. All such supplementary material must be clearly listed in an appendix to the report and referred to in the main text.

Submission Deadline:

The assessed material for the DAH projects will need to be submitted by **12.00 NOON on Friday, 29th November 2019**. By the deadline you must have submitted

- an electronic version of your project report to Turnitin via Learn. At the beginning of your submission you will need to declare an "Own Work Declaration";
- supplementary material via Learn. If required (usually not the case), you can submit supplementary material to the Teaching Office in JCMB (Room 4309);
- and any supplementary material.

The marks obtained for the DAH project will count for 60% of the total course mark. Reports submitted after the deadline will receive a penalty of 5% (equivalent to 3 marks out of 60) for each calendar day by which the deadline is exceeded. Students who, for good reason, find they are unable to meet the deadline, should contact the DAH Project Organiser and Course Secretary before the deadline.

1.6.3 Quiz

Midway during the course, you will be need to submit a quiz/hand-in on questions about data acquisition and handling material. You will be given two weeks to solve these questions on your own time, i.e. you should not use laboratory hours to solve the quiz questions. The exact deadline for handing in the quiz will be announced on Learn, it will be around the end of week 7 of the semester. The quiz will count for 10% of the total course mark.

1.7 Plagiarism:

The University regulations on plagiarism apply, see Section 27 of the Taught Assessment Regulations, available online at `http://www.ed.ac.uk/schools-departments/academic-services/policies-regulations/regulations/assessment`.

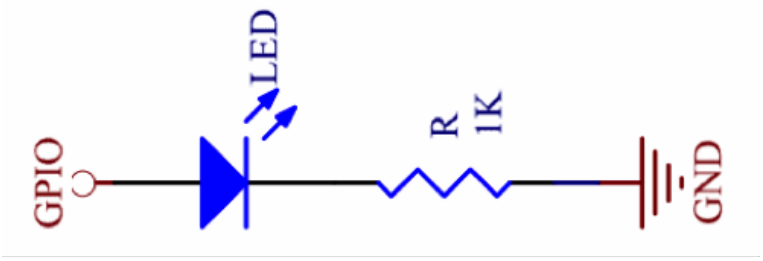
Chapter 2

DAH Checkpoints

Important note: Please consult the DAH manual to familiarise yourself with the equipment, including the Raspberry-Pi, LEDs, temperature sensors, ADCs, DACs, I/O, switches, breadboard and connectors. The manual contains detailed instructions on how to operate the Raspberry-Pi. It is suggested to use the chromium web browser. To copy python code snippets (see below) into your python scripts, download these files from github, see `https://github.com/fmuheim/DAH`. Data sheets for all electronic elements are available from Dropbox, see: `https://www.dropbox.com/sh/gfnisnh4ntnum1d/AAAWtNL_AhcpR8PZ_QmqZpsja?n=112609310`. The DAH manual also provides information on how to start and stop the webIOPi web server.

2.1 Checkpoint 1: LEDs and Switches

1.1. Control an LED with the Raspberry-Pi by completing the following steps. Connect the Raspberry-Pi to a breadboard and start the webIOPi web server. Start the chromium web browser and go to the webIOPi header webpage. Set GPIO 24 to OUT, connect this output to an LED with a 1 kOhm resistor in series to ground, and switch the LED on and off using the web interface. Repeat the exercise with negative logic (active low) by connecting the LED with a 1 kOhm resistor in series to 3.3 V. Draw a schematic diagram for this circuit, see example below.



1.2. Using positive logic (active high) connect a push-button Switch between 3.3V and GPIO23 and, with a 1kOhm resistor in series, to ground and explain what happens on the webIOPi header webpage. Draw a schematic diagram for this circuit. Afterwards stop the webIOPi web server.

1.3. Connect an LED with a 1 kOhm resistor as in checkpoint 1.1 above. Start python interactively to control GPIO 24. Using this code template, import the GPIO library and turn the LED on and off.

```
studentnn@dahpimm  ~ $ python3
```

```
# Import GPIO library
import RPi.GPIO as GPIO

# Configure standard GPIO mode
# "BCM" refers to the Broadcom processor
GPIO.setmode(GPIO.BCM)

# Define the pin number for the LED
LED0 = 24

# Control the LED
GPIO.setup(LED0, GPIO.OUT) # Set Pin as output
GPIO.output(LED0, GPIO.HIGH) # Turn on the LED
GPIO.output(LED0, GPIO.LOW) # Turn off the LED
```

../scripts/checkpoint_1a.py

Alter the first python script to blink an LED, connected with either positive or negative logic, by adding a while loop.

```
import time

# Loop for ever
while True:

# Toggle LED after time in seconds as defined inside sleep method
    value = not GPIO.input(LED0)
    GPIO.output(LED0, value)
    time.sleep(2)
```

../scripts/checkpoint_1b.py

1.4. Connect the push button switch using using negative logic (active low) and the LED circuit with positive logic (active high). Write a python script to toggle the LED state every time the push button switch is pushed.

```
# Imports
import RPi.GPIO as GPIO

# Set which GPIO pin is connected to the switch
#    Insert your code here

# Read the switch and if it is pressed toggle the state of the LED
if (GPIO.input(SWITCH0) == GPIO.LOW):

# Read the value of the LED, invert it and save it to variable value
# Write variable value to the LED
#    Insert your code here
```

../scripts/checkpoint_1c.py

2.2 Checkpoint 2: ADC, DAC and SPI BUS

Most experimental observables are continuous: their values can vary by arbitrarily small amounts. However, we record measurements as discrete values: a number with some range of uncertainty. Creating a numerical (digital) measurement from a continuous (analogue) signal is called digitisation, and is performed by an Analogue to Digital Converter (ADC). The digital information can then be analysed with a computer.

In this checkpoint we will use an ADC to read information from a light sensor into the Raspberry Pi. We will also perform the opposite task, varying the brightness of an LED by converting a numerical output from the Raspberry Pi into the corresponding voltage level using a Digital to Analogue Converter (DAC).

2.1. Connect an ADC MCP3208 chip to the Raspberry Pi using the SPI Interface on pin GPIO 8 (SPI_CE0). Make sure that all required connections between the MCP3208 and the Raspberry-Pi are made (see pin-out sheet). Use a multimeter to check that power (VDD) and ground (AGND and DGND) are correctly connected. Explain what all connections are for.

Connect a Light Dependent Resistor (LDR) and a 4.7 kΩ resistor as a voltage divider between 3.3 and 0 V, using an ADC input channel to measure the voltage in the middle. Use python interactively to read the voltages of all eight ADC channels. Try reading a specific ADC channel, and experiment with all the other methods given below. You are encouraged to consult the following webpage `http://webiopi.trouch.com/`(→ Tutorials → Using Devices → Analogue).

Explain how the ADC works and what the meaning of the return values of each method is. What is the primary ADC output and how is the voltage output calculated from this? Cover the LDR with your hand, and explain how the ADC readings change.

```
# Import ADC chip libraries
from webiopi.devices.analog.mcp3x0x import MCP3208

# Define ADC on Chip Enable 0 (CE0/GPIO8)
ADC0 = MCP3208(chip=0)

# Read all ADC channels in Volts.
print ( ADC0.analogReadAllVolt() )

# Play with the following methods
ADC0.analogCount()
ADC0.analogResolution()
ADC0.analogMaximum()
ADC0.analogReference()
ADC0.analogRead(channel)
ADC0.analogReadFloat(channel)
ADC0.analogReadVolt(channel)
ADC0.analogReadAll()
ADC0.analogReadAllFloat()
ADC0.analogReadAllVolt()
```

../scripts/checkpoint_2a.py

[3 marks]

2.2. Leave the ADC in place, but also connect the DAC MCP4922 chip to the Raspberry-Pi using the other SPI Interface on GPIO 7 (SPI_CE1). Make sure that all required connections between the MCP4922 and the Raspberry-Pi are made (see pin-out sheet). Use a multimeter to check that power (VDD) and ground (AVSS) are correctly connected. Explain how the DAC works and what all connections are for.

Use python interactively to set a value — e.g. 1.3 V — to output VOUTA of the DAC. Measure this voltage with a multimeter.

```
# Import DAC chip libraries
from webiopi.devices.analog.mcp492X import MCP492X

# Define DAC on Chip Enable 1 (CE1/GPIO7)
DAC1 = MCP492X(chip=1, channelCount=2, vref=3.3)

# Output 1.3V on channel 0 of DAC1
print ('output 1.3V on channel 0 of DAC1')
DAC1.analogWriteVolt(0, 1.3)

# Print value of channel 0 of DAC1
print ('value of register for channel 0 of DAC1')
print (DAC1.analogReadVolt(0))
```

../scripts/checkpoint_2b.py

[2 marks]

2.3. Connect one output of the DAC to an LED. Write a python script that varies the brightness of the LED by setting a series of different values for the output voltage of the DAC. Now arrange the circuit so that the LED is next to the LDR, and the change in brightness can be measured. The laboratory is quite bright relative to an LED, so you might need to cover the breadboard to show a convincing change. Modify your script to read the ADC input each time you set the DAC output. Write the DAC setting and measured ADC values at each step to an output file with comments such that the content of the file will explain your work.

[4 marks]

2.3 Checkpoint 3: Generating and Sampling Analogue Signals

3.1. Connect an ADC chip (MCP3208) to the Raspberry Pi, as in Checkpoint 2. Verify that the ADC works with a DC voltage produced with a potentiometer or an LDR in series with a 4.7 kΩ resistor

Use the signal generator to produce a repetitive signal, e.g. a sinusoidal waveform. Display the output on the oscilloscope. Set the amplitude of the signal such that the waveform can be read by the ADC chip, which can sample between 0 V and VREF = 3.3 V. Set the frequency to 10 Hz.

Connect the output of the signal generator to an ADC input channel. Using python in interactive mode read a few samples of the ADC output. Comment on what you measure. [Caveat: Don't connect a signal with a voltage outside the range of the ADC chip, which could destroy it and the Raspberry Pi.]

[1 marks]

3.2. Measure the waveform produced by the signal generator by writing a python script that records 100 ADC samples and displays these on a graph. Always label plots correctly with title and axes and save these to a file (pdf format). [Hint: Use the pylab interface for plotting graphs. Example files are available on github.]

[2 marks]

3.3. Calibrate the voltage scale of the ADC output with respect to the voltage scale displayed on the oscilloscope by using a square waveform that closely matches the ADC input range. First connect the signal from the signal generator to the oscilloscope. Read the input voltage for the high and low sections of the square waveform off the oscilloscope screen. For this use the trigger threshold dial to determine these voltage levels as precisely as possible. Then connect the signal to an ADC input channel. Write a python script that takes 100 ADC samples and writes these into a file, then expand the script to determine the average ADC voltages for the high and low sections of the square waveform and record the results. Reduce the amplitude of the input waveform by a factor of two and repeat above procedure. Plot the four calibration measurements, i.e. the measured ADC voltages (from the two sets of measurements of the high and low sections) versus the four input voltages (measured with the oscilloscope) on a graph and comment.

[2 marks]

3.4. What is the maximum signal frequency with which you can properly record a given repetitive signal? First, consider which would be the best waveform for this investigation. What is the sampling frequency? Explain what happens when a signal is undersampled. Make a plot with a waveform that is undersampled.

[2 marks]

3.5. Connect a DAC chip (MCP4922) to the Raspberry Pi, as in Checkpoint 2. Using a python script, generate a sinusoidal waveform on the DAC output and plot the waveform to a graph.

Use the ADC chip (MCP3208) to digitise the waveform generated by the DAC. Write a python script that takes 100 DAC samples and 100 ADC measurements and plot these on the same graph. Take into account the limitations encountered in 3.4.

[3 marks]

2.4 Checkpoint 4: Input/Output I/O and I2C BUS

4.1. Input/Output (I/O) Expander chips enable the user to connect many devices having the same or similar functions. With the Raspberry Pi this can be achieved using the I2C bus. Connect the PCF8574AN chip (I2C BUS Expander) to the Raspberry-Pi. Make sure that all required connections between the PCF8574AN chip and the Raspberry-Pi are made, see pin-out sheet. Explain how the I/O Expander works and what the SDA, SCL and A0, A1, A2 address lines are.

Using negative logic connect an LED to output P0 of the PCF8574AN Expander chip. Write a python script to blink the LED using negative logic, see checkpoint 1 for setting up a while loop. Why is negative logic necessary?

You may consult the webIOPi webpage `http://webiopi.trouch.com/` (→ Tutorials → Using Devices → Digital) to find information on the GPIO expander chip.

NB: The GPIO library used here is different to that used in Checkpoint 1. The RPi.GPIO definitions are not compatible with the webIOPi I/O expander commands.

```
# Imports
import webiopi
import time
from webiopi.devices.digital.pcf8574 import PCF8574A

# Retrieve GPIO lib
GPIO = webiopi.GPIO

# Setup chip
mcp = PCF8574A(slave=0x38)

# Set which PCF8574 GPIO pin is connected to the LED (negative logic)
LED0 = 0

# Setup GPIOs
mcp.setFunction(LED0, GPIO.OUT) #Set Pin as output

# Turn on the LED for the first time
mcp.digitalWrite(LED0, GPIO.LOW)

# Loop for ever
#    Insert your code here
# Include a delay
time.sleep(0.10)
```

../scripts/checkpoint_4a.py

[3 marks]

4.2. Connect an additional 3 LEDs to outputs P1, P2 and P3 of the PCF8574AN Expander chip. Write a python script which turns the LEDs on and off in a predefined pattern, such as a running light. Consider P0 to P3 as default, but you are encouraged to play with other patterns.

[2 marks]

4.3. Consult the webpage `http://webiopi.trouch.com/` (→ Tutorials → Using Devices → Digital) for the GPIO Expander. Use the portWrite(value) method to manipulate all four LEDs at the same time. Connect a push button switch to pin P4 of the expander chip. Write a python script such that an LED pattern toggles every time the button is pushed.

```
# Loop for ever
while True:
    # Read the switch and if it is pressed toggle the state of the LED
    if (mcp.digitalRead(SWITCH0) == GPIO.LOW):
        # Insert your code here

        # dummy write to reset switch register
        mcp.digitalWrite(SWITCH0, GPIO.HIGH)
```

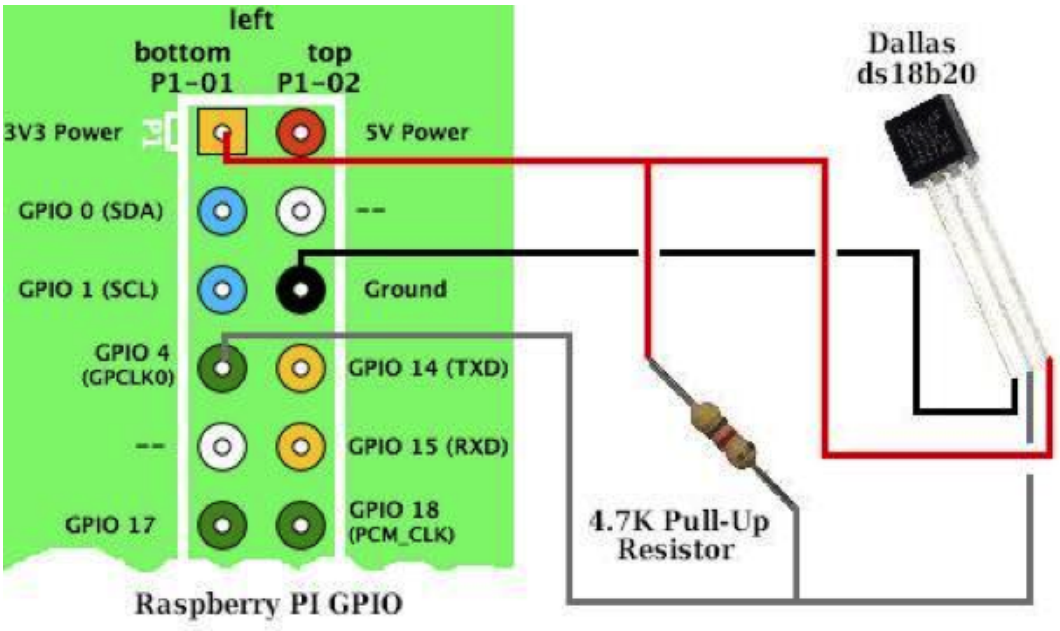
../scripts/checkpoint_4b.py

[3 marks]

2.5 Checkpoint 5: Temperature Sensors

5.1. We will be using DS18B20 temperature sensors for this checkpoint. Take a look at the datasheet here: http://www.adafruit.com/datasheets/DS18B20.pdf or download it from the DAH Dropbox.

Connect a DS182B0 temperature sensor to your Raspberry pi (look at the flat front of the sensor to get it the right way around):



What is the interface between the DS18B20 temperature sensor and the Raspberry Pi? Explain how it works.

Locate the sensor output by finding the file that has the serial number of your sensor:

```
studentnn@dahpimm ~ $  cd /sys/bus/w1/devices
studentnn@dahpimm /sys/bus/w1/devices $ ls
10-00080265b6d6 w1_bus_master1
```

where n = 1 to 50 and mm = 01 to 22. Note that your temperature sensor won't be called 10-00080265b6d6, this is just an example.

Now read the sensor output, i.e. the raw temperature measurement:

```
studentnn@dahpimm /sys/bus/w1/devices ~ $ cd 10-00080265b6d6
studentnn@dahpimm /sys/bus/w1/devices/10-00080265b6d6 $ cat w1_slave
30 00 4b 46 ff ff 0d 10 29 : crc=29 YES
30 00 4b 46 ff ff 0d 10 29 t=23937
```

This should be interpreted as 23.937 centigrade (degree Celsius).

WebIOPi provides a simple way to access the temperature sensor data in python. It is best to test this by running python in interactive mode first.

```
studentnn@dahpimm ~ $ python3
```

```
# Import
from webiopi.devices.sensor.onewiretemp import DS18S20

# Readout temperature sensor
tmp0 = DS18S20(slave="10-00080265b6d6")
tmp0.getCelsius()
```

../scripts/checkpoint_5a.py

[2 marks]

5.2. Measure temperature with the DS18B20 sensor versus time. Choose a sensible time interval. Write a python script to make a graph of 50 temperature measurements versus time. Always label plots correctly with title and axes and save these to a file.

As a second step the graph should update itself as each temperature measurement is made. Write a python script for this purpose. Once this is working, play with it by touching the temperature sensor with you fingers. Describe what is happening and sketch it in your lab book.

The following code example shows how to display a plot that updates regularly:

```
import pylab
import matplotlib.animation as animation
import datetime

# Empty arrays of time and measurement values to plot
timeValues = [ ]
measurements = [ ]

# Set up the plot object
plotFigure = pylab.figure()

# The function to call each time the plot is updated
def updatePlot( i ):

    timeValues.append( datetime.datetime.now() ) # Store the current time
    measurements.append( MEASUREMENT )           # Store the measurement
    plotFigure.clear()                           # Clear the old plot
    pylab.plot( timeValues, measurements )       # Make the new plot


# Make the animated plot
ani = animation.FuncAnimation( plotFigure, updatePlot, interval=1000 )
pylab.show()
```

../scripts/checkpoint_5b.py

[3 marks]

5.3. Add another temperature sensor to your circuit by connecting it in parallel with the existing one. You don't need to make separate connections to the Raspberry Pi: your new sensor can share these with the existing one (Just make sure that you connect it the right way around).

Find its serial number in the w1/devices folder like before. Now ensure that you can read out your two temperature sensors simultaneously in python. You can test this by running python in interactive mode first. Note that your temperature sensors will have different serial numbers.

```
studentnn@dahpimm  ~ $ python3
```

```
# Import
from webiopi.devices.sensor.onewiretemp import DS18S20

# Readout temperature sensor
tmp0 = DS18S20(slave="10-00080265b6d6")
tmp1 = DS18S20(slave="10-000802cb3b5c")
print ( str( tmp0.getCelsius() ) + ", " + str( tmp1.getCelsius() ) )
```

../scripts/checkpoint_5c.py

What is the smallest change in temperature that a sensor can report? Explain this with reference to the datasheet, and how the temperature information is encoded.

Investigate the accuracy of the sensor readings by looking at the stability of the measurement with time, and by comparing the outputs of your two sensors. You can probably assume that the ambient temperature in the lab is constant, but shielding your sensors from breezes may help.

Modify your graphing code from part 5.2 to make histograms of the temperature measurements of the two sensors.

You can use the pylab histogram command:

```
# Pylab makes graph plotting very easy:
import pylab

# Make a histogram with NumberOfBins bins
# in the range binMinimum to binMaximum
pylab.hist( someData, bins=NumberOfBins, range=[binMinimum, binMaximum] )
```

../scripts/checkpoint_5d.py

Make also a graph of the difference between the temperature of the two sensors. Determine the RMS value of a set of temperature difference measurements between the two sensors. Explain your results. Are these consistent with the datasheet?

[3 marks]

2.6 Checkpoint 6: Data Handling and Analysis

6.1. This is a data handling exercise and the Raspberry Pi is not required. Thus this checkpoint is best carried out using the Physics CPlab computers. There is a CPlab computer available on every desk in the DAH laboratory. You can use the following python libraries.

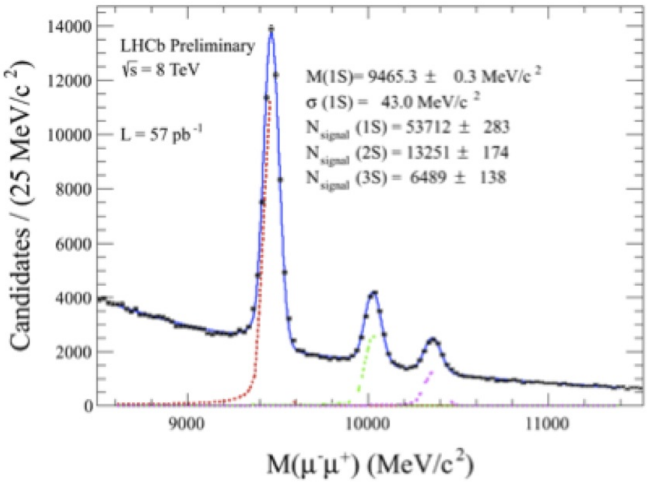
```
# Import
# pylab has a LOT of useful things in it.
import pylab

# numpy is the fundamental package for scientific computing with Python.
import numpy as np

# Make a histogram with arrays of nr. of entries and of bin edges
entries, binedges, patches = pylab.hist(xmass, bins = Nbins, range = [
    binMin,binMax])
```

../scripts/checkpoint_6a.py

The LHCb experiment at the Large Hadron Collider at CERN has recorded a sample of muon pairs with invariant masses in the range of 8.5 to 11 GeV/c^2. Three clear peaks are observed in this mass spectrum. These correspond to the production of Upsilon mesons, which are bound states of a b and a anti-b quark. These states are known as the Υ(1S), Υ(2S) and Υ(3S) mesons where the Υ(1S) meson is the ground state and the Υ(2S) and Υ(3S) states are radial excitations (for LHCb paper, see DOI: 10.1007/JHEP06(2013)064).



Download the file upsilons-mass-xaa.txt from the DAH Dropbox, which contains the invariant masses of a large number of muon pairs in units of GeV/c^2 in text format. Write a python script that reads the data from this file and plots a histogram of all the masses, choosing a reasonable bin width. Always label plots correctly with title and axes and save these to a file. [Hint: The bin width should be chosen such that each of the three peaks is clearly resolved and represented by a sufficient number of bins for analysis.]

[2 marks]

6.2. Determine the masses of the three particles by determining the bins with the highest number of entries in the peak regions. Divide the histogram into three peak regions and write a local peak finding method for this part. What are the mass differences between the Υ(2S) and Υ(3S) states with respect to the Υ(1S) meson?

[2 marks]

6.3. Determine the mass of the Υ(1S) meson and its statistical uncertainty. This can be achieved by several methods. First by looking at the mass spectrum, choose a suitable region around the Υ(1S) mass peak. Calculate the mean, the unbiased variance and standard deviation for the events in this region. Use these values to determine the standard deviation of the mean. Comment on possible biases for this method.

The mass peaks corresponding to the three Υ mesons can be described reasonably well by a Gaussian function, $f(x) = \frac{N}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-/\mu)^2}{2\sigma^2}\right)$ where x is the invariant mass of the muon pairs, μ is the mass of the Υ(1S) meson, σ is the Gaussian width (mass resolution) and N is the total number of signal events. By inspecting visually the muon-pair mass spectrum, determine the Full Width Half Maximum (FWHM) of the Υ(1S) mass peak. Assuming a Gaussian signal shape estimate the mass resolution σ from the FWHM. Compare this result for the mass resolution of the Υ(1S) peak with the standard deviation in the signal region determined above and comment.

[3 marks]

In the muon-pair mass spectrum define a signal region of width ± 150 MeV/c^2 around the Υ(1S) peak position and determine the number of events N in this region. Define an upper and lower sideband region where there are only background events. These sidebands should each be half as wide as the signal region and located at masses equidistant from the Υ(1S) peak position. Assuming that the background is falling linearly with the muon-pair mass, determine the number of background events B in the signal region (below the Υ(1S) mass peak). Perform either a linear least squares fit in the sideband regions or use the sideband subtraction method for this. Determine the number of signal events S in the signal region.

Alternatively, if you know how to perform a fit you may choose fitting the Υ(1S) peak in the mass spectrum for this part.

Compare your mass measurement with the Particle Data Group (`pdg.lbl.gov`) and comment. [Hint: The PDG lists the properties of particles. Select "pdgLive - Interactive Listings" followed by "Mesons b anti-b" to find the Υ(1S) meson.]

[3 marks]

Chapter 3

DAH Projects

3.1 Project B: Building an FFT Spectrum Analyser

3.1.1 Goals of project

You will develop and build a real-time spectrum analyser. This device will sample an analog signal and perform a Fast Fourier Transform (FFT) analysis and the results can be displayed and/or written to a file. The ADC MC3208 chip, which was used during the checkpoints, could be a part of this project.

The projects have an open-ended aspect and are an opportunity where you can show your own initiative and demonstrate your experimental and computational skills.

3.1.2 Equipment for project B

For this project you will need the following items:

- Raspberry-Pi
- Signal generator
- Analog to Digital Converter (MCP3208)
- Arduino Uno
- USB cable to connect Arduino to Raspberry-Pi.
- Microphone with preamplifier if you want to analyse acoustic signals.

Some of this equipment will be located in the red box B. You will have to share some items between Tuesday and Thursday sessions.

3.1.3 Building the FFT spectrum analyser

We suggest that you start by reading up on Fourier transformations, which you learned in a 3rd year course, and the Fast Fourier Transform as a particularly effective numerical implementation. You may also want to familiarise yourself with the Arduino micro controller, e.g. at `http://www.arduino.cc/`. It is suggested to develop and build the spectrum analyser in a modular way, e.g. to develop the FFT analysis separately from the real-time data acquisition, before combining these tasks.

There are many ways of completing this project. To give you a better idea of what may be required, take a look at the three examples below, listed in the order of increasing complexity.

1. Use the webIOPi framework. This is the simplest approach because it requires almost no new knowledge. You would use the signal generator to produce a waveform and connect the ADC (MC3208) chip to the Raspberry-PI using the SPI interface. Use the ADC chip (MCP3208) to sample the signal at regular intervals, fill an array of a fixed length (say, 1024 samples) and perform an FFT analysis of the data using a Numpy library (routine `numpy.fft.rfft()`). You would then save the absolute values of the FFT data to file and display these on a screen. This task would allow you to develop the FFT analysis and the displays. The amount of work for this is similar to a checkpoint.

   While such an FFT analyser is simple to build and control, it has severe limitations. webIOPi allows only for a very low sampling rate (as studied in Checkpoint 3). There can be missing parts in the recorded samples ("data holes") as the Raspberry-Pi is not a real-time computer. The Raspberry-Pi has a multitasking operating system, which can interrupt the data taking, especially if sampled at too large a rate.

2. Use the MCP3208 ADC chip, but control and read its output directly through the serial interface SPI using a python library called `spidev`, which is already installed on the Raspberry-Pi. The spidev library has a routine called `xfer()` that sends an array of bytes to the ADC through the SPI interface and receives ADC data in response to this request. Use `xfer()` to instruct the ADC to measure the voltage. To read the voltage from CH0 of the ADC one needs to send three bytes [6,0,0] which instructs the ADC to select channel CH0, measure the voltage and then send back another three bytes (ignore the first byte, the second and the third are the high and the low byte of the 12-bit integer output of the ADC) to the Raspberry-Pi.

```
# instruction for ADC to measure a voltage sample

import spidev
spi=spidev.SpiDev() # initialization
spi.open(0,0)       # opens the port
spi.max_speed_hz=1000000  # sets the clock speed to 1MHz = max

# recommended for VDD=3.3V
# to read ADC output
adc=spi.xfer2([6,0,0])  # read voltage from CH0 of MCP3208
# after executing this commabd, "adc" will be an array of three bytes,
# where the 2nd and 3rd will hold the output of the ADC
```

../scripts/project_B_a.py

This method is much faster than webIOPi and it is possible to record up to 5k samples per second, however real-time sampling is not possible and there can be holes in the data.

3. Use the Arduino, a real-time micro-controller, which is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals, The Arduino is based on an 8-bit 16MHz chip ATmega328 and does not have an operating system by default, so it supports real-time computing, i.e. the program does not get interrupted. The Arduino also has a built-in 10-bit ADC. For this project, the Arduino is used as an ADC and a data buffer connected to the Raspberry-Pi and it will be made available pre-programmed. From your perspective the Arduino will behave as "black box", i.e. an ADC with a buffer memory that can be accessed through an USB interface. (Note: If you are interested, you may familiarise yourself with Arduino-IDE package and upload the code, `ADCforRPI4.ino`, available on `https://github.com/fmuheim/DAH`. You will need to write python code that communicates with the Arduino using the `serial` library.

```
import serial
import numpy

# open the serial port with baud rate 115200bps
# this is the rate used by the Arduino microcontroller
# Arduino will be on port ttyACM0 by default
ser=serial.Serial('/dev/ttyACM0',115200, timeout=5)

# a small delay may be necessary to give the microcontroller time to
    respond
# you may want to experimentally find the shortest delay that makes the
    code work
time.sleep(1)

# clear the serial port buffer
# this is sometimes required to properly initialize communication between
    R-Pi and Arduino
ser.flushInput()

# a small delay may be necessary to give the microcontroller time to
    respond
# you may want to experimentally find the shortest delay that makes the
    code work
time.sleep(1)

# send desired sampling frequency to Arduino and order it to begin
    sampling
# the first value can be anything between 4 and 255. The larger the value
    the slower the sampling will be
# the second value should always be 2
ser.write(bytes([4,2]))

# a small delay may be necessary to give the microcontroller time to
    respond
# you may want to experimentally find the shortest delay that makes the
    code work
time.sleep(1)
```

```
# read data sample from Arduino ADC (1024 bytes, 8-bit resolution)
data=ser.read(1024)

# convert data to an array of integers (value = 0..255, 0 = zero voltage,
    255 = +5V)
values=numpy.frombuffer(data,dtype=numpy.uint8)
```

../scripts/project_B_b.py

The 8-bit resolution simplifies data transfer and processing, since each sample equals exactly one byte of data. In fact, 8 bits correspond to the effective resolution of the ADC at the largest supported sampling frequencies where the two least significant bits (of the 10-bit ADC) are essentially noise, This method allows for fast sampling rates of up to 100k samples per second in real-time and there are no data holes.

3.1.4 Performing an FFT spectrum analysis

Performing an FFT analysis a data sample is straightforward, for example you can use a Numpy library routine, `numpy.fft.rfft()`. In all cases the data should be saved to a file and displayed on the screen. Once the FFT spectrum analyser is working and the main functionalities for saving and displaying the data spectrum is achieved, there are several possibilities to take this project forward in an open-ended way.

1. You will use FFT to analyse the spectrum of a several repetitive signal shapes, including a sinusoidal, a square and a saw waveform generated by the function generator, and compare the amplitudes of different harmonics to mathematically predicted ones for that signal.
2. Investigate which of the many python libraries that are available for drawing on the screen best suits your needs. For example, mathplotlib can be employed to produce high-quality figures, but may be too slow to present data in real time. The `pygame` library supports drawing simple figures (points, lines, etc.) and is very fast, simple to use, and sufficient for making simple plots of the input signals and its FFT spectrum.
3. There are additional methods for generating signals. For example you can use a microphone to record acoustic signals and perform FFT analyses on these samples. Can you use it e.g. to determine the pitch of a whistled note?
4. The Arduino is used as a "black box" for this project. Learning how to program was deemed to be incompatible with the timescale of the project. However, if you have used the Arduino before or are sufficiently curious, you are welcome to investigate. There might be also other options for reading the data, and you are encouraged to investigate this.

3.1.5 Project planning

The project descriptions are generally significantly less detailed than what was made available for the checkpoints. Any material covered during checkpoints including python code

examples are assumed to be known. Only essential and new information is provided and you are expected to take care of the details. Python code snippets are provided where necessary, but you will have to understand yourself what they do. It is recommend that you google for information about your project on the web, including data sheets of components and python libraries, if applicable. Python scripts should be well structured, either using functions or classes.

The timeline will vary between different projects, but in general, it is recommended that you plan your work as follows:

- weeks 7, 8 & 9: Building your gadget and/or writing code for project;
- week 9, 10: Analysis of data or equivalent, prepare supplementary material;
- week 10, 11: Finish writing of project report and prepare submission.

Note that you are advised to start writing your report as the project progresses.

For guidance on report writing, how the projects will be assessed, plagiarism and the submission deadline, please consult the DAH course booklet and the DAH grade descriptors, available on Learn.

3.2 Project C: Building a Synthesizer

3.2.1 Goals of project

You will develop and build a small synthesizer with which you will be able to play small tunes. This project builds upon the work carried out in checkpoint 4 where you worked with an I/O expander chip. With the synthesizer you will be able to play simple tunes, but there is scope for developing the synthesizer much further.

The projects have an open-ended aspect and are an opportunity where you can show your own initiative and demonstrate your experimental and computational skills.

3.2.2 Equipment for project C

For this project you will need the following items:

- Raspberry-Pi
- Amplified Speakers
- MCP23S17 IO Expender
- 1K Resistors
- Pushbuttons

Some of this equipment will be located in the red box C. You will have to share some items between Tuesday and Thursday sessions.

3.2.3 Building the synthesizer

The SPI I/O expander chip (MCP23S17) is supported by the webIOPi framework. In checkpoint 4 you used an 8 bit I2C expander chip in a very similar way, so you can build upon this work. For this project we recommend that you use the MCP23S17 chip, which is a 16 bit SPI expander. While the two expander chips are different devices the same python methods can be applied when using webIOPi.

```
# Set up GPIO library
import webiopi
GPIO = webiopi.GPIO

# Syntax to import the MCP23S17 I/O expander chip.
# documentation on webiopi.trouch.com is not up to date.
from webiopi.devices.digital.mcp23XXX import MCP23S17

# Define the CS pin and hardware address (A0, A1, A2 all grounded)
mcp0 = MCP23S17( chip=0, slave=0x20 )
```

../scripts/project_C_c.py

NB: The documentation for the MCP23S17 chip is a little confusing because it allows I2C-style addresses as well as SPI-style chip select (CS) signals. You can use multuple MCP23S17 devices with the same CS setting provided they have different hardware addresses, but if you want to include other SPI devices in the project they will need a separate CS as usual.

1. Start your project using switch buttons to play sounds at first and then you can think about including additional forms of input (or output). Remember that you can use components from the checkpoints, as well as what is in the project box.
2. You can base your project on the `PyGame` package, which is already installed on the Raspberry-Pi. You can find a more information on the following webpage: `https://www.pygame.org`. Here is an example of how to use this package as a synthesiser, playing a simple sine-wave note.

```
import numpy as np
import pygame
import time
import math
import pylab as pl

# Some constants
outputRate = 44100
maxAmplitude = np.iinfo( np.int16 ).max

# Create an array containing a sine wave
def SineWave( pitch, volume, duration ):

    global outputRate, maxAmplitude

    # Create the output buffer
    totalSamples = int( outputRate * duration )
    outputBuffer = np.zeros( ( totalSamples, 2 ), dtype=np.int16 )

    # Calculate amplitude
    amplitude = int( maxAmplitude * volume )

    # Calculate change in the wave for each output sample
    waveStep = float( pitch / outputRate ) * 2.0 * math.pi

    # Fill buffer
    for i in range( totalSamples ):

        # Left channel
        outputBuffer[i][0] = amplitude * np.sin( i * waveStep )

        # Right channel
        outputBuffer[i][1] = amplitude * np.sin( i * waveStep )

    return outputBuffer

# Set up the audio output - only once!
# 2-channel (stereo), 16-bit signed integer value output at 44khz
pygame.mixer.init( frequency=outputRate, channels=2, size=-16)
```

```
# Create a note (C5)
sin523 = SineWave( 523, 1.0, 1.0 )

# Make a plot of the wave if you like
pl.plot( sin523[0:100] ) # Just the first 100 values for clarity
pl.show()

# Play sound
noteC5 = pygame.mixer.Sound( buffer=sin523 )
noteC5.play()

# Keep the program active until the note is complete
time.sleep(2)
```

../scripts/project_C_a.py

Once you have set up the hardware, play a few notes and make sure you understand how the synthesizer works. In Appendix 1 of project C, shown below, you can find the frequencies for musical notes.

3.2.4 Applications using the synthesizer

1. You can play a tune and record it.
2. Try to implement 'polyphony:' make the synthesizer able to play more than one note at the same time. In reality the Raspberry Pi is producing multiple interleaving sounds, but we perceive them as concurrently played sounds.

   Polyphony requires that the synthesizer is able to detect two or more buttons being pressed at the same time. Capturing two or more buttons being pressed at once can be achieved by reading the I/O expander as a parallel bus instead of reading the state of each button in sequence. When using parallel or port reading, one should save the state of the port as a single variable. Each binary bit of this variable corresponds to the state of a single input button.
3. You could experiment with different wave forms.
4. As well as synthesising your own sounds, you can use pre-recorded audio samples from a piano in `.wav` format.

   You can download a compressed file with the audio samples from the DAH DropBox page. The `PyGame` package can play most `.wav` format files in just the same way as your synthesised notes. Below an example is given of how to play two sounds at the same time.

```
import pygame, time

# Set up the audio output - only once!
# 2-channel (stereo), 16-bit signed integer value output at 44khz
pygame.mixer.init(frequency=44100, channels=2, size=-16)

# Load piano samples
noteC5 = pygame.mixer.Sound( "piano_samples/C5.wav" )
noteD5 = pygame.mixer.Sound( "piano_samples/D5.wav" )
```

```
# Play samples
noteC5.play()
noteD5.play()

# Keep the program active until the notes are complete
time.sleep(2)
```

../scripts/project_C_b.py

3.2.5 Project planning

The project descriptions are generally significantly less detailed than what was made available for the checkpoints. Any material covered during checkpoints including python code examples are assumed to be known. Only essential and new information is provided and you are expected to take care of the details. Python code snippets are provided where necessary, but you will have to understand yourself what they do. It is recommend that you google for information about your project on the web, including data sheets of components and python libraries, if applicable. Python scripts should be well structured, either using functions or classes.

The timeline will vary between different projects, but in general, it is recommended that you plan your work as follows:

- weeks 7, 8 & 9: Building your gadget and/or writing code for project;
- week 9, 10: Analysis of data or equivalent, prepare supplementary material;
- week 10, 11: Finish writing of project report and prepare submission.

Note that you are advised to start writing your report as the project progresses.

For guidance on report writing, how the projects will be assessed, plagiarism and the submission deadline, please consult the DAH course booklet and the DAH grade descriptors, available on Learn.

Appendix 1 of Project C: Frequency Table for Notes

Frequency [Hz]	Rounded Frequency	Note	MIDI #
27.5	28	A0	21
29.1352	29	A#0	22
30.8677	31	B0	23
32.7032	33	C1	24
34.6478	35	C#1	25
36.7081	37	D1	26
38.8909	39	D#1	27
41.2034	41	E1	28
43.6535	44	F1	29
46.2493	46	F#1	30
48.9994	49	G1	31
51.9131	52	G#1	32
55	55	A1	33
58.2705	58	A#1	34
61.7354	62	B1	35
65.4064	65	C2	36
69.2957	69	C#2	37
73.4162	73	D2	38
77.7817	78	D#2	39
82.4069	82	E2	40
87.3071	87	F2	41
92.4986	92	F#2	42
97.9989	98	G2	43
103.8262	104	G#2	44
110	110	A2	45
116.5409	117	A#2	46
123.4708	123	B2	47
130.8128	131	C3	48
138.5913	139	C#3	49
146.8324	147	D3	50
155.5635	156	D#3	51
164.8138	165	E3	52
174.6141	175	F3	53
184.9972	185	F#3	54
195.9977	196	G3	55
207.6523	208	G#3	56
220	220	A3	57
233.0819	233	A#3	58
246.9417	247	B3	59
261.6256	262	C4	60
277.1826	277	C#4	61
293.6648	294	D4	62
311.127	311	D#4	63
329.6276	330	E4	64
349.2282	349	F4	65

Frequency [Hz]	Rounded Frequency	Note	MIDI #
329.6276	330	E4	64
349.2282	349	F4	65
369.9944	370	F#4	66
391.9954	392	G4	67
415.3047	415	G#4	68
440	440	A4	69
466.1638	466	A#4	70
493.8833	494	B4	71
523.2511	523	C5	72
554.3653	554	C#5	73
587.3295	587	D5	74
622.254	622	D#5	75
659.2551	659	E5	76
698.4565	698	F5	77
739.9888	740	F#5	78
783.9909	784	G5	79
830.6094	831	G#5	80
880	880	A5	81
932.3275	932	A#5	82
987.7666	988	B5	83
1046.5023	1047	C6	84
1108.7305	1109	C#6	85
1174.6591	1175	D6	86
1244.5079	1245	D#6	87
1318.5102	1319	E6	88
1396.9129	1397	F6	89
1479.9777	1480	F#6	90
1567.9817	1568	G6	91
1661.2188	1661	G#6	92
1760	1760	A6	93
1864.655	1865	A#6	94
1975.5332	1976	B6	95
2093.0045	2093	C7	96
2217.461	2217	C#7	97
2349.3181	2349	D7	98
2489.0159	2489	D#7	99
2637.0205	2637	E7	100
2793.8259	2794	F7	101
2959.9554	2960	F#7	102
3135.9635	3136	G7	103
3322.4376	3322	G#7	104
3520	3520	A7	105
3729.3101	3729	A#7	106
3951.0664	3951	B7	107
4186.009	4186	C8	108

3.3 Project D: Building an Ultrasonic Range Finder

3.3.1 Goals of project

You will develop and build an ultrasonic range finder. This device will use an ultrasound transducer as a distance sensor.

The projects have an open-ended aspect and are an opportunity where you can show your own initiative and demonstrate your experimental and computational skills.

3.3.2 Equipment for project D

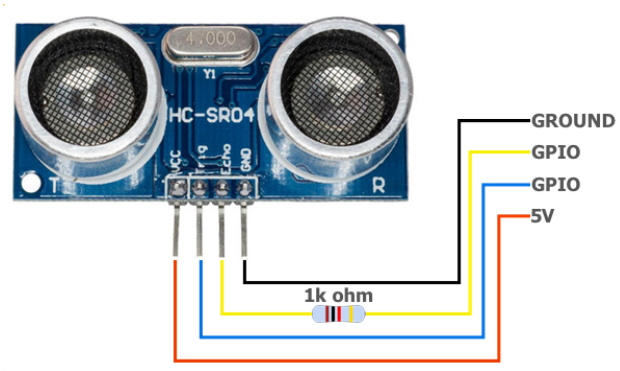
For this project you will need the following items:

- Raspberry-Pi
- Ultrasonic Transducer module HC-SR04
- 1 kΩ Resistor
- Loudspeakers

Some of this equipment will be located in the red box D. You will have to share some items between Tuesday and Thursday sessions.

3.3.3 Building the ultrasonic range finder

You will develop an ultrasonic range finder using a HC-SR04 Distance Sensor. Start by connecting the HC-SR04 to the Raspberry-Pi, as shown in the Figure below. Test the circuit using the code examples, which are provided in Appendix 1 of project D below. You will need to make a few changes in your script to get it to work.



Hints: The output of the HC-SR04 module is a logic level one of 5 V. The Raspberry-Pi works with 3.3V logic signals. A 1 kΩ resistor in series will limit the current in the circuit to prevent damaging the Raspberry Pi.

1. Work out how the HC-SR04 sensor operates. What are the "Trig" and "Echo" pins of the HC-SR04 module used for? Use the oscilloscope to capture these signals when running the code example.
2. Use these data to explain in detail how the module works and how it is possible to measure distance using ultrasonic waves.
3. Familiarise yourself with the `RPi` framework, see e.g. `https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage`. Using the `RPi` framework complete the function `def reading(sensor):` in your script and test your circuit.
4. What are the minimum and maximum distance that your Range Finder can measure?

3.3.4 Applications using the ultrasonic range finder

Now that you have a functioning ultrasonic range finder there are several possible applications.

1. Try to understand better the properties and limitations of your device. What are the minimum and maximum size of objects the range finder is able to detect? Play with reflecting surfaces of different areas and define the angular range (span of angles) within the range finder works? Is this approach suitable for large or small objects? Is there a relation between the distance and the size of the reflection area?
2. Develop an application of your choice. An example could be an electronic "Car Parking Assistant" where sounds are created and LEDs flash (or similar) if the sensor gets too close to an object.
3. Perform a calibration of the distance scale of your application.

Note that the loudspeakers are power hungry, so they should be connected via USB to the monitor.

To play an audio file (sampled note) you can use an application such as `aplay`. Below an example is given of how to play a note using a python script.

```
import os
while True:
    if ( switch1 == True  ):
        os.system('aplay -q C5.wav &')
```

../scripts/project_D_a.py

3.3.5 Project planning

The project descriptions are generally significantly less detailed than what was made available for the checkpoints. Any material covered during checkpoints including python code

examples are assumed to be known. Only essential and new information is provided and you are expected to take care of the details. Python code snippets are provided where necessary, but you will have to understand yourself what they do. It is recommend that you google for information about your project on the web, including data sheets of components and python libraries, if applicable. Python scripts should be well structured, either using functions or classes.

The timeline will vary between different projects, but in general, it is recommended that you plan your work as follows:

- weeks 7, 8 & 9: Building your gadget and/or writing code for project;
- week 9, 10: Analysis of data or equivalent, prepare supplementary material;
- week 10, 11: Finish writing of project report and prepare submission.

Note that you are advised to start writing your report as the project progresses.

For guidance on report writing, how the projects will be assessed, plagiarism and the submission deadline, please consult the DAH course booklet and the DAH grade descriptors, available on Learn.

Appendix 1 of project D: Example Code

```
#!/usr/bin/python

def reading(sensor):

# remember to change the GPIO values below to match your sensors
# GPIO output = the pin that's connected to "Trig" on the sensor
# GPIO input = the pin that's connected to "Echo" on the sensor

    TRIG = 17
    ECHO = 27

    import time
    import RPi.GPIO as GPIO

    # Disable any warning message such as GPIO pins in use
    GPIO.setwarnings(False)
    GPIO.setmode(GPIO.BCM)

    if sensor == 0:

        # Setup the GPIO pins for TRIG and ECHO, including defining
        # if these are input or output pins

        # Insert your code here


        time.sleep(0.3)

        # sensor manual says a pulse length of 10Us will trigger the
```

```
        # sensor to transmit 8 cycles of ultrasonic burst at 40kHz and
        # wait for the reflected ultrasonic burst to be received

        # to get a pulse length of 10Us we need to start the pulse, then
        # wait for 10 microseconds, then stop the pulse. This will
        # result in the pulse length being 10Us.
        GPIO.output(TRIG, True)
        time.sleep(0.00001)
        GPIO.output(TRIG, False)

        # listen to the input pin. 0 means nothing is happening. Once a
        # signal is received the value will be 1 so the while loop
        # stops and has the last recorded time the signal was 0
        while GPIO.input(ECHO) == 0:
          signaloff = time.time()

        # listen to the input pin. Once a signal is received, record the
        # time the signal came through
        while GPIO.input(ECHO) == 1:
          signalon = time.time()

        # work out the difference in the two recorded times above to
        # calculate the distance of an object in front of the sensor
        timepassed = signalon - signaloff

        # we now have our distance but it's not in a useful unit of
        # measurement. So now we convert this distance into centimetres
        # Define relation between "distance" and "timepassed"

        # Insert your code here

        # return the distance of an object in front of the sensor in cm
        return distance

        # we're no longer using the GPIO, so tell software we're done
        GPIO.cleanup()

    else:
        print ("Incorrect usonic() function variable.")

print (reading(0))
```

../scripts/project_D_b.py

3.4 Project E: Building a Precision Refrigerator

3.4.1 Goals of project

The course organiser would like to celebrate the DAH course with a glass of fine wine, chilled to exactly the right temperature. You will design and build a thermostat to precisely control the temperature of a fluid. Regrettably, fine wine could not be made available for teaching purposes and it will be replaced by a beaker of tap water for this project. You will use a Peltier thermoelectric device to cool the liquid. The temperature of the liquid will be monitored using 1-wire temperature sensors, which were used during the checkpoints.

The projects have an open-ended aspect and are an opportunity where you can show your own initiative and demonstrate your experimental and computational skills.

3.4.2 Equipment for project E

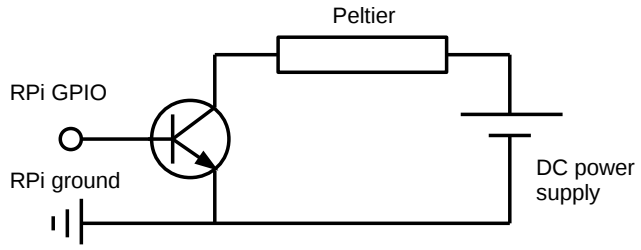
For this project you will need the following items:

- Raspberry-Pi
- DC power supply
- Peltier thermoelectric device
- 2x temperature sensor S18B20
- Power transistor Darlington TO-220
- Heat sinks
- Beaker

Some of this equipment will be located in the red box E. You will have to share some items between Tuesday and Thursday sessions.

3.4.3 Building the refrigerator

You will need to build a circuit to supply power to the Peltier element as the Raspberry-Pi cannot supply enough current itself. You can use the power supplies available in the lab, to produce a DC voltage and control it using a power transistor connected to a GPIO pin on the Raspberry-Pi. Use a 1 A, 1 V supply for the Peltier.



Example circuit diagram for Peltier control.

Research Peltier devices: note that one side gets hot as the other side gets cold, so you will need to place the hot side in contact with a heat sink. Do not power the device for more than a few seconds without this heat sink, or you may damage it! You may also need a heat sink for the transistor.

The temperature of the liquid must be monitored, so you will need to find a way to get the 1-wire sensors near to it. You will be provided with a waterproof temperature sensor, with part number DS18B20, so you will need to adjust your code appropriately if reading it with webIOPi.

3.4.4 Monitoring and controlling the temperature

Aim first to be able to precisely control the liquid at a few centigrade below room temperature before trying to find out how cold you can make the liquid. The Peltier device will take a long time to cool the liquid much below room temperature, so don't waste time waiting for this.

During Checkpoint 5 you already learned how to read information from the 1-wire temperature sensors, but you should now consider this in more detail. How precisely can you measure the temperature of the liquid? Are there systematic effects that you can account for by calibrating your sensors? What will you do about temperature gradients across your refrigerator when the Peltier element is switched on? Remember that you have additional (but not waterproof!) sensors that you used in Checkpoint 5 — these might be useful, and you have already studied their performance.

Your project should include a way of displaying the current temperature of the liquid, and previous measurements. You should also include a visual indication of the state of the Peltier, rather than just poking it to see if it feels cold.

The circuit to power and control your Peltier device is simple, but you should think carefully about when your python code should switch the cooling on and off. Rather than having a simple threshold temperature for turning the cooling on and off, you may want to use a hysteresis loop. Rapidly toggling power to the Peltier in an uncontrolled fashion is unlikely to give good performance.

You could enhance the scope of your project in the following ways.

1. Try varying the cooling power of your Peltier using pulse-width modulation. Here you would use your Raspberry Pi GPIO pin to create a square wave to toggle the power

transistor on and off. You then adjust what percentage of time is spent in the on or off state: the 'duty cycle.'

2. Write an interface that allows the user to start and stop the temperature control programme, show the status and temperature of the Peltier, and change the temperature value of the thermostat.
3. Consider running the Peltier as a heating element. What changes are required to the sensor, the circuit and the control programs?

3.4.5 Project planning

The project descriptions are generally significantly less detailed than what was made available for the checkpoints. Any material covered during checkpoints including python code examples are assumed to be known. Only essential and new information is provided and you are expected to take care of the details. Python code snippets are provided where necessary, but you will have to understand yourself what they do. It is recommend that you google for information about your project on the web, including data sheets of components and python libraries, if applicable. Python scripts should be well structured, either using functions or classes.

The timeline will vary between different projects, but in general, it is recommended that you plan your work as follows:

- weeks 7, 8 & 9: Building your gadget and/or writing code for project;
- week 9, 10: Analysis of data or equivalent, prepare supplementary material;
- week 10, 11: Finish writing of project report and prepare submission.

Note that you are advised to start writing your report as the project progresses.

For guidance on report writing, how the projects will be assessed, plagiarism and the submission deadline, please consult the DAH course booklet and the DAH grade descriptors, available on Learn.

3.5 Project F1: Make Accurate Measurements of Particle Masses

3.5.1 Goals of project

You will use LHCb data on the invariant mass of particle candidates that you were introduced to during a checkpoint. You will analyse this in a much more sophisticated way and closer to the actual analysis performed leading to its publication. You will use the maximum likelihood process to fit different mass model shapes to the data. From this you will determine the parameters of the mass model for the three signal peaks, and their errors. You will start with a very simple Gaussian mass model. You will then improve this and use a more sophisticated model.

The projects have an open-ended aspect and are an opportunity where you can show your own initiative and demonstrate your experimental and computational skills.

3.5.2 Equipment for project F1

Note: this is a data analysis project, which does not require use of the Raspberry Pi. This project is best carried out using the Physics CPlab computers. No other equipment is needed. There is a CPlab computer available on every desk in the DAH laboratory. You may also use your own laptop computer, but you will need to be able to install python and python packages on your own.

3.5.3 Detailed project description

You were previously introduced to the LHCb Upsilon data. The LHCb experiment at the Large Hadron Collider at CERN has recorded a sample of muon pairs with invariant masses in the range of 8.5 to 11 GeV/c^2. Three clear peaks are observed in this mass spectrum. These correspond to the production of Upsilon mesons, which are bound states of a b and a anti-b quark. These states are known as the Υ(1S), Υ(2S) and Υ(3S) mesons where the Υ(1S) meson is the ground state and the Υ(2S) and Υ(3S) states are radial excitations (for LHCb paper, see DOI: 10.1007/JHEP06(2013)064).

During checkpoint 6, you performed some very simple "peak finding". In this project you are going to do the analysis much like it would actually be carried out in a particle physics experiment.

Download the files ups-15.bin, ups-15-small.bin and mc.bin from the DAH Dropbox, These files contain the data recorded by LHCb in 2015 and a subset with a factor 5 less data. In addition, there is a file with simulated data (Monte Carlo). The files are written in binary format and contain five observables for a large number of muon pairs

- invariant mass of muon pair in GeV/c^2;

- transverse momentum $p_\perp$ of muon pair in GeV/c;
- rapidity η of muon pair;
- momentum p of muon pair in GeV/c;
- transverse momentum $p_{\perp,1}$ of first muon in GeV/c;
- transverse momentum $p_{\perp,2}$ of second muon in GeV/c.

Write a python script that reads the data from this file, see below. Plot histograms of all six variables, choosing a reasonable bin width. Always label plots correctly with title and axes and save these to a file.

```
import numpy as np

# import data
# xmass = np.loadtxt(sys.argv[1])
f = open("datafiles/ups-15-small.bin","r")
datalist = np.fromfile(f,dtype=np.float32)

# number of events
nevent = len(datalist)/6
xdata = np.split(datalist,nevent)
print(xdata[0])

# make list of invariant mass of events
xmass = []
for i in range(0,nevent):
    xmass.append(xdata[i][0]/1000.)
    if i < 10:
        print(xmass[i])
```

../scripts/project_F_a.py

1. Consider first the Upsilon(1S) (Υ(1S)) particle, which is the particle with the lowest mass, i.e. the left most peak in the plot. Construct a composite probability density function (PDF) for the invariant mass of the muon pairs, which contains two components:

   - A Gaussian shape to fit the Υ(1S) mass peak;
   - A shallow falling exponential to fit the background shape of the mass spectrum underneath and around the peak.

2. Use this PDF in a Maximum Likelihood fit to determine the parameters of the PDF. Note hat it is essential that the composite PDF remains normalised to 1 over the range of the fit.

   Determine the Υ(1S) meson mass and yield, and all other parameters, and their and errors.

   You should be able to obtain the parameter errors directly from the minimization engine of your choice (scipy.optimize.minimise, scipy.optimise.curve_fit, lmfit, see `https://lmfit.github.io/lmfit-py/` or Minuit). Depending on your choice you will be

able to chose different minimising methods. It would be good to show that you understand these by obtaining them yourself from the parameters of the Gaussian signal fit - this is described in the data handling lectures.

Plot the fitted signal shape on top of the data.

3. Now consider the entire mass range, and perform a simultaneous fit for all three Upsilon peaks, and the underlying background. Again you should always report the parameter values, and their errors. Plot the fitted signal shape on top of the data.

4. The results so far probably look "quite good" by eye. i.e. the signal shape plotted on top of the data probably looks like it fits quite well. However this can be misleading when performing a precision measurement. You should make a plot of what are called the "residuals". A residual is the difference between the data in the binned histogram and the best-fit mass model value for the centre of that bin. Describe what you see.

5. There are several ways to enhance the scope of the project. For example, if the single Gaussian mass model does not fit the data perfectly, one can try other mass models, i.e. by using a signal PDF that goes beyond a single Gaussian function. Examples are:

   - A PDF comprising a function which is the sum of two Gaussian functions (i.e. one narrow and one wide Gaussian function to fit a single Upsilon mass peak)
   - A Crystal Ball function, which incorporates a non-Gaussian tail at the lower end of the mass peak. The functional shape is described elsewhere, e.g. see: `http://en.wikipedia.org/wiki/Crystal_Ball_function`.

   Implement one or both of these functions in your PDF. First fit this PDF using the simulated Monte Carlo data. This allows you to see better how the additional parameters should improve the description of the tails of the signal peak. Then fit the PDF to the LHCb data and see how much better they are at describing the data. Consider fixing some of the shape parameters from the fit to the Monte Carlo sample.

6. Make 2-dimensional plots of the additional observables vs the invariant mass of the muon pair. Find out if you can improve the purity of your signal sample. The purity is defined as the ratio of signal events over all events in a region.

7. As on open ended activity, you will see in the paper that the analysis is also done by dividing the data up into bins of transverse momentum (pT) and rapidity (eta). You can explore doing this analysis yourself. It is somewhat more complex as you have to think about which are common parameters (e.g. masses) and which are not (e.g. background fractions).

8. Read the publication and see what is said about systematic errors. Make a reasonable attempt at determining some systematic errors on the masses.

3.5.4 Project planning

The project descriptions are generally significantly less detailed than what was made available for the checkpoints. Any material covered during checkpoints including python code examples are assumed to be known. Only essential and new information is provided and you are expected to take care of the details. Python code snippets are provided where necessary,

but you will have to understand yourself what they do. It is recommend that you google for information about your project on the web, including data sheets of components and python libraries, if applicable. Python scripts should be well structured, either using functions or classes.

The timeline will vary between different projects, but in general, it is recommended that you plan your work as follows:

- weeks 7, 8 & 9: Building your gadget and/or writing code for project;
- week 9, 10: Analysis of data or equivalent, prepare supplementary material;
- week 10, 11: Finish writing of project report and prepare submission.

Note that you are advised to start writing your report as the project progresses.

For guidance on report writing, how the projects will be assessed, plagiarism and the submission deadline, please consult the DAH course booklet and the DAH grade descriptors, available on Learn.

3.6 Project G: Building a Remote Sensing System

3.6.1 Goals of project

You will develop a remote sensing system, similar to a weather station. A WiFi micro-controller will be used to acquire temperature and humidity data and publish these on a embedded web server. You will use the Raspberry-Pi to download these data and build a monitoring system.

The projects have an open-ended aspect and are an opportunity where you can show your own initiative and demonstrate your experimental and computational skills.

3.6.2 Equipment for project G

For this project you will need the following items:

- Raspberry-Pi
- Adafruit HUZZAH ESP8266 WiFi micro-controller
- DHT22 Temperature Humidity Sensor
- DC power supply
- USB to TTL UART 6PIN CP2102 Module Serial Converter
- Adafruit RGB 16x2 LCD and Keypad Kit
- Windows PC in laboratory
- WiFi hotspot

Some of this equipment will be located in the red box G. You will have to share some items between Tuesday and Thursday sessions.

3.6.3 Building the Remote Sensing System

You will develop a remote sensing system, similar to a weather station. For this you need to put together different elements, including connecting a temperature and humidity sensor to a micro-controller, programming the micro-controller, connecting an LCD display to the Raspberry-Pi, and remotely connecting the sensing system to the Raspberry-Pi via a WiFi hotspot. It is suggested that you familiarise yourself with all the components by consulting the corresponding web pages and reading their technical specifications. A micro-controller is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals, so it is more limited than a Raspberry-Pi. Here we will use the Adafruit HUZZAH ESP8266 WiFi micro-controller, see `https://learn.adafruit.com/adafruit-huzzah-esp8266-breakout/overview`.

1. Mount the Huzzah ESP8266 micro-controller on a breadboard. Use a DC power supply to provide the +5V. Please note that it is always recommended to switch off the power supply when connecting devices or changing the wiring. Connect the 3.3V output and GND to the rails on the breakout board. Then use the USB to Serial cable - Receive, Transmit and Ground Lines (Rx,TX, GND) - to connect the ESP8266 with to the Windows PC, located on the desk in the right-hand corner of the DAH laboratory. When connected, reset theESP8266 by pushing the reset button while holding the GIPO0 button. This will make the micro-controller ready to boot and a dimmed red LED will show.

2. To program the ESP8266 you need to start the Arduino-IDE package, which is installed on the Windows PC. Using the instructions on `https://learn.adafruit.com/adafruit-huzzah-esp8266-breakout/using-arduino-ide`, go to Arduino → File → Preferences and copy the following link into the Additional Board Manager URLs field:
   `http://arduino.esp8266.com/stable/package_esp8266com_index.json`
   Then you invoke the Board Manager (Tools → Board → Boards Manager), search for say "ESP" and install "esp8266 by ESP8266 Community". If successful, the ESP8266 should appear under Tools → Board. Check that the CPU Frequency is set correctly (80 MHz) on the micro-controller and set upload baud rate (115200) and the matching port of your USB to serial cable (e.g. COM4).

   In addition you need to install two libraries. Use the library manager (Sketch → Include Libraries → Library Manager), first search for say "Adafruit Unified" and install "Adafruit Unified Sensor library by Adafruit" then search for say "DHT" and install "DHT sensor library by Adafruit".

3. As a first programming exercise, perform the "Blink Test". Consult the instructions on `.../using-arduino-ide`. Copy the code into an Arduino IDE sketch, compile the code (Sketch → Verify / Compile) and upload (⇒) to the micro-controller. The sketch will start immediately - you'll see the LED blinking. Hooray!

4. The next step is to connect the temperature and humidity sensor DHT22, see `http://www.adafruit.com/products/385`, to the micro-controller. The DHT22 requires three lines (3.3V, GND and data). Use GPIO #2 as serial input on the ESP8266. Remember to switch off the power supply during this process.

5. You are now ready to program the temperature and humidity web server. Download the example code into the Arduino-IDE application, using the instructions from `https://learn.adafruit.com/esp8266-temperature-slash-humidity-webserver/code`. You will need to change this code by adding the name and password of the WiFi hotspot.

```
const char* ssid     = "DAHLADHOC";
const char* password = "dahlab15";
```

../scripts/wifi.ino

   Compile the code (Sketch → Verify / Compile) and upload it to the micro-controller. Recall to put first the micro-controller ready into boot state.

6. In order to connect the ESP8266 micro-controller to the WiFi hotspot, which is located on the desk in the left corner of the laboratory, switch on the WiFi hotspot, if necessary,

push the reset button on the micro-controller and open a Serial Monitor on the Arduino-IDE (→ Tools → Serial monitor). The ESP8266 & DHT22 sensor system should start to work and you should obtain the IP address of the HTTP server, see:

```
Working to connect ....................................................

DHT Weather Reading Server
IP Address: 192.168.1.2
HTTP server  started
```

7. The final step is to read the DHT22 & ESP8266 remotely with the on-board WiFi adapter of the Raspberry-Pi. To enable WiFi on the Raspberry-Pi, execute the following command:

   ```
   studentnn@dahpimm  ~ $ wpa_cli enable wlan0
   ```

   Check the network connection and select the WiFi hotspot. You might have to type in the password. Using the Chromium web browser, you can now read the sensor by connecting to the correct IP address, e.g. `http://192.168.1.2`. Find out how to read temperature and humidity from the DHT22 Adafruit webpage, `https://learn.adafruit.com/esp8266-temperature-slash-humidity-webserver/using-the-webserver`.

3.6.4 Using the Remote Sensing System

Reading temperature and humidity using a web browser is straightforward, but you want to go beyond single measurements. Write a python script which samples a series of measurements and displays these. For this you need to parse the web server data. Use the instructions at https://docs.python.org/3/howto/urllib2.html to write a script that will regularly read the temperature and humidity from the remote sensor and print the data onto the screen and/or file. What would be a reasonable update frequency? Further possible applications could include the following:

1. Write a well structured python script in which reading the temperature and humidity sensor are functions or classes. Convert temperature values from Fahrenheit into Celsius.
2. Write a script which displays an animated series of measurements in real time. Use a hot air-blower/hair-dryer to vary the temperature. Avoid applying the heat-source for too long to prevent the DHT22 from starting to melt. Plot the temperature versus humidity and explain what you observe.
3. Display the temperature and humidity on an LCD display. Connect the LCD display to top of the Raspberry-Pi. Make sure that you use the provided GPIO port Extender plug to keep the LCD board from touching the USB and Ethernet ports of the Raspberry-Pi. Download the python script `char_lcd_plate.py` from `https://github.com/fmuheim/DAH`. Run this script to understand how to program the LCD display. Consulting the instructions at

`https://learn.adafruit.com/adafruit-16x2-character-lcd-plus-keypad-for-raspberry-usage`, write a python script which regularly reads and updates temperature (in Celsius) and humidity on the LCD display.

3.6.5 Project planning

The project descriptions are generally significantly less detailed than what was made available for the checkpoints. Any material covered during checkpoints including python code examples are assumed to be known. Only essential and new information is provided and you are expected to take care of the details. Python code snippets are provided where necessary, but you will have to understand yourself what they do. It is recommend that you google for information about your project on the web, including data sheets of components and python libraries, if applicable. Python scripts should be well structured, either using functions or classes.

The timeline will vary between different projects, but in general, it is recommended that you plan your work as follows:

- weeks 7, 8 & 9: Building your gadget and/or writing code for project;
- week 9, 10: Analysis of data or equivalent, prepare supplementary material;
- week 10, 11: Finish writing of project report and prepare submission.

Note that you are advised to start writing your report as the project progresses.

For guidance on report writing, how the projects will be assessed, plagiarism and the submission deadline, please consult the DAH course booklet and the DAH grade descriptors, available on Learn.

3.7 Project X: Develop a Project or Suggest Your Own Project

3.7.1 Goals of project

We have a few possible projects where the project descriptions and goals are not advanced enough to be included in the list of projects A to G, see the list below. If you are interested in one of these, please let us know. In addition, if you have an idea of what you want to do with the Raspberry-Pi, please tell us and we will discuss it. Please note that we will have to make a decision if your own project or one from the list below is feasible within the time scale of the DAH course.

3.7.2 List of possible projects

We give here a list of projects using the Raspberry-Pi and/or the Arduino, where more development work is required and little or no project description exists.

- Building an oscilloscope using the Arduino micro-controller as an ADC and the Raspberry-Pi as the DAQ to display waveforms.
- Building a motion sensor using an accelerometer connected to an ADC and read out by the Raspberry Pi.
- Build a CCTV-like imaging capturing triggered by a motions sensor, for info see `http://www.raspberrypi.org/learning/python-picamera-setup/`.
- Control switches, LEDs and relays using the PiFace Digital expansion boards, see `http://www.piface.org.uk/products/piface_digital/`.
- Controlling a toy train set-up in the laboratory, for info see the following web link: `http://www.mathworks.co.uk/company/newsletters/articles/adding-fun-to-first-year-c` `html?nocookie=true`. You will need to bring your own toy train set-up.
- Suggest your own project.

3.7.3 Project planning

The project descriptions are generally significantly less detailed than what was made available for the checkpoints. Any material covered during checkpoints including python code examples are assumed to be known. Only essential and new information is provided and you are expected to take care of the details. Python code snippets are provided where necessary, but you will have to understand yourself what they do. It is recommend that you google for information about your project on the web, including data sheets of components and python libraries, if applicable. Python scripts should be well structured, either using functions or classes.

The timeline will vary between different projects, but in general, it is recommended that you plan your work as follows:

- weeks 7, 8 & 9: Building your gadget and/or writing code for project;
- week 9, 10: Analysis of data or equivalent, prepare supplementary material;
- week 10, 11: Finish writing of project report and prepare submission.

Note that you are advised to start writing your report as the project progresses.

For guidance on report writing, how the projects will be assessed, plagiarism and the submission deadline, please consult the DAH course booklet and the DAH grade descriptors, available on Learn.