UNIVERSITY of York

Degree Examinations 2017/18

DEPARTMENT OF COMPUTER SCIENCE

Programming of micro-controllers (PROM)

Open Assessment

Issued: Monday 16th April, 2018

Submission due: 9:00am, Monday 14th May, 2018

Demonstration: 14th May – 16th May 2018 (week 5 of Summer term)

Feedback and marks due: Friday 8th June, 2018

All students should submit their lab logbook individually through the electronic submission system: http://www.cs.york.ac.uk/student/assessment/submit/ by **9:00 am, Monday 14th May, 2018**. Your name and group number should be on the front cover of your lab book. Do not include exam numbers as these are private to individuals.

Each group should submit their software e.g. code, schematics,or SPICE simulations, that will form part of the assessed demonstration through the electronic submission system: https://www.cs.york.ac.uk/student/assessment/submit/ by **9:00 am, Monday 14th May, 2018**. Any group member can submit their group's software.

Each group should submit a printed copy of the minutes from their weekly project meetings at the time of their assessed demonstration in **Summer term week 5**. The names of all your group members and your group number should be on the front cover of your assessment. Do not include exam numbers as these are private to individuals.

Demonstrations will take place in **week 5 of the Summer term**.

An assessment (or part of an assessment) submitted after the deadline will be marked initially as if it had been handed in on time, but the Board of Examiners will normally apply a lateness penalty to the whole assessment.

The feedback and marks date is guided by departmental policy but, in exceptional cases, there may be a delay. In these cases, all students expecting feedback will be emailed by the module owner with a revised feedback date. The date that students can expect to see their feedback is published on the module descriptor: http://www.cs.york.ac.uk/modules/

Your attention is drawn to the section about Academic Misconduct in the Departmental Statement on Assessment:

http://www.cs.york.ac.uk/student/assessment/policies/

Any queries on this assessment should be addressed to: Dr Mike Freeman (mjf@cs.york.ac.uk)

PROM Open Assessment 2018

Introduction

The ICAR cockroach traps have failed, the new breed of super cockroaches have overwhelmed the University :(. Your group has been commissioned to develop an electronic monitoring system to assess the level of infestation and identify where and how these cockroaches are entering the building.

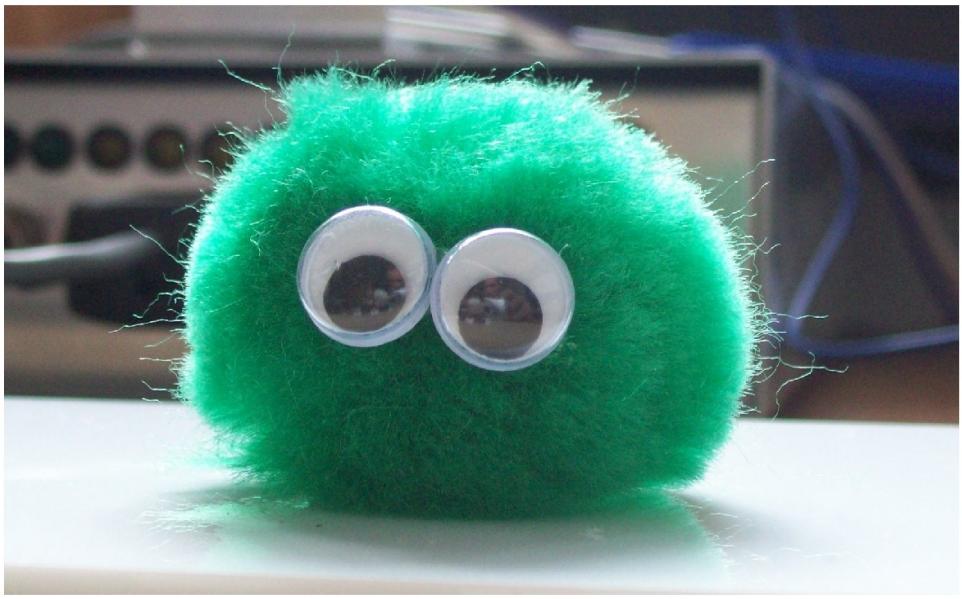


Figure 1: super cockroach

This open assessment will be performed in groups of typically three people. Each person within a group will take on one project management role:

- Hardware : cabling, circuitry external to micro-controller.
- Interfacing : boundary between hardware and software, signal conditioning, noise reduction.
- Software : controlling software architecture, data processing.

The person performing each of these roles is responsible for the allocation of related work packages within the group e.g. the person performing the Hardware role is **not** necessarily responsible for implementing the hardware, but will co-ordinate its implementation. It is expected that hardware and software development will be shared equally amongst all group members. The exception to this statement is that each role is allocated a specific task, described in the next section, which forms part of that person's individual mark. A group member's awarded mark is made up from their individual mark (10%), plus the group mark (90%).

There will be overlap between roles, these boundaries need to be decided upon within each group.

Possible solutions to assessed tasks should not be shared with other groups. The University's guidelines on academic integrity and in particular how to avoid collusion can be found at:

http://www.york.ac.uk/integrity/collusion.html

This open assessment is intended to be performed during laboratory sessions. If

required, the hardware laboratory is also accessible Monday – Friday, 9:00 – 17:00. Each group member must keep a record of the work performed during each laboratory session i.e. a lab book, briefly describing their work and work assigned as part of their role.

Each group member's lab book must be submitted electronically at the end of the project through this web page:

https://www.cs.york.ac.uk/student/assessment/submit/

There will be no differential awarding of marks unless group members indicate that the group has not functioned, and this is substantiated by log books and attendance records. The log book is not otherwise consulted. Where it is clear from all indicators that a student has made no contribution to the group work then a mark of zero will be awarded to that student.

Any software e.g. code, schematics, or SPICE simulations, that will form part of the assessed demonstration must be submitted electronically. Submitted software may be uploaded onto a test system for you. Any group member can submit their group's software through this web page:

https://www.cs.york.ac.uk/student/assessment/submit/

Marking will be via a demonstration. Each group will be assigned a 15 minute time slot, Summer term week 5, in which to demonstrate their final solution. Marks are awarded for the correct operation of system functions. A full breakdown of marks is given at the end of this document. During the demonstration you will be asked to justify the selection of component values used e.g. to show design documentation, calculations used to determine resistor, capacitor values etc.

During week 5 the hardware labs will be closed, groups will not be allowed access to their hardware except for a 15 minute setup time before they start their group presentation.

If you have any questions relating to this open assessment, or if a member of your group is not adequately engaging with this material then please contact me either after a lecture, practical, or by email mjf@cs.york.ac.uk.

System Core Requirements

The main sensor that will be used to monitor cockroach movements is a USB camera, as shown in figure 2. The system shall indicate the state of the camera using a YELLOW LED:

- on : active, an image is being recorded
- off : idle, camera is not in use.

Using images captured by this device the system must identify if a cockroach is present within an image. All cockroaches will be of a similar size, shape and colour to the one shown in figure 1. The system shall capture a minimum of one image every two seconds, once processed these images should be deleted.

To allow the camera to operate at night a tungsten filament bulb light source will be used. The camera is designed to operate in normal environmental ambient lighting conditions. Therefore, rapid changes in light intensity, the very bright (direct) light and the very dark will cause the camera to saturate or underexpose an image i.e. produce a white or black image. To ensure correct camera illumination levels a closed loop lighting control system must be constructed. However, depending on installation requirements, the user may choose to select a manual mode and set a

fixed illumination level. Switching between automatic and manual modes is controlled by a single push button, toggling the system's state. The current system state is indicated by two LEDs, RED=Manual, GREEN=automatic.

Integrated into the USB camera is a microphone, accessed using 3.5mm plug as shown in figures 2 and 3. This sensor is interfaced into the system to detect the noises produced by the cockroaches as they burrow through walls and other material that would obscure them from the camera's view i.e. a seismometer.

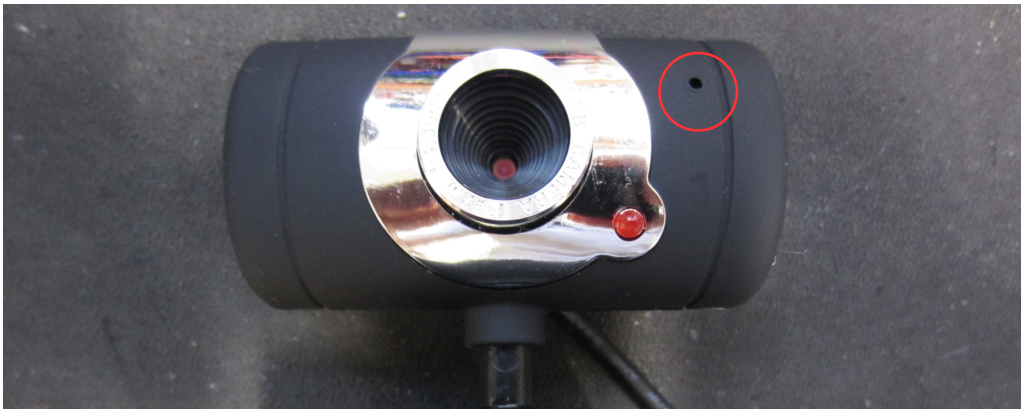


Figure 2: camera (microphone hole circled)

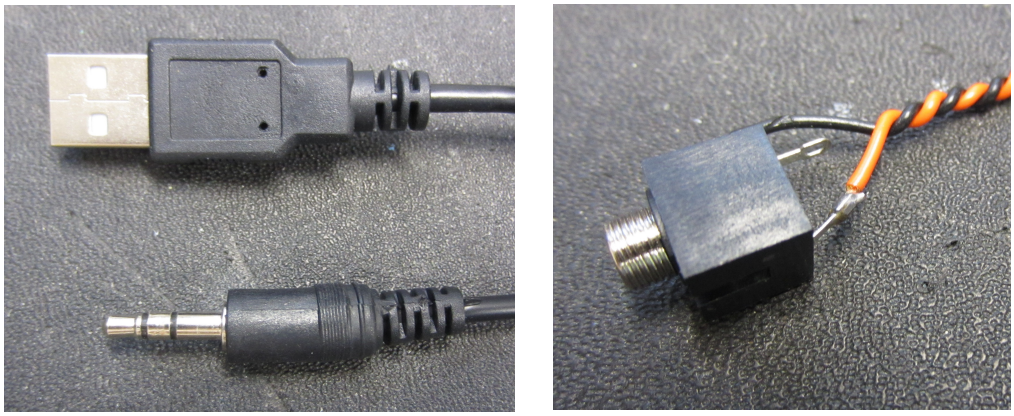


Figure 3: microphone 3.5mm plug and socket

Images shall be captured on the Raspberry Pi using the `fswebcam` command line program. Initially images will be stored as `.jpg` files. To find out more information about this program, on the Raspberry Pi type:

```
man fswebcam
```

Command line programs will be called from the system's python program using the `os` package i.e. the miscellaneous operating system interfaces package.

Within this program images must be stored and processed using ASCII encoded, Portable PixMap (`ppm`) or Portable PixGray (`pgm`) file formats. To convert `jpg` files into `ppm` files and perform other image processing tasks the ImageMagick command line programs can be called from within the system's python program. An example shell script program to perform this conversion can be found in Appendix G. For more information refer to:

Image formats : https://en.wikipedia.org/wiki/Netpbm_format
ImageMagick software tools : https://www.imagemagick.org/script/index.php

Note, to simplify reading and writing image files it is recommended that you use the ASCII encoded versions of these formats i.e. P2 and P3.

The system is required to detect if a cockroach is present within an image. When a cockroach is detected the system will print "`Image: Cockroach Detected`" on the screen and the current date and time. For the purposes of debugging, the system should also save a `ppm` image file to disk, placing a RED rectangle around the area it believes contains a cockroach. The filename given to this file should contain the current date and time.

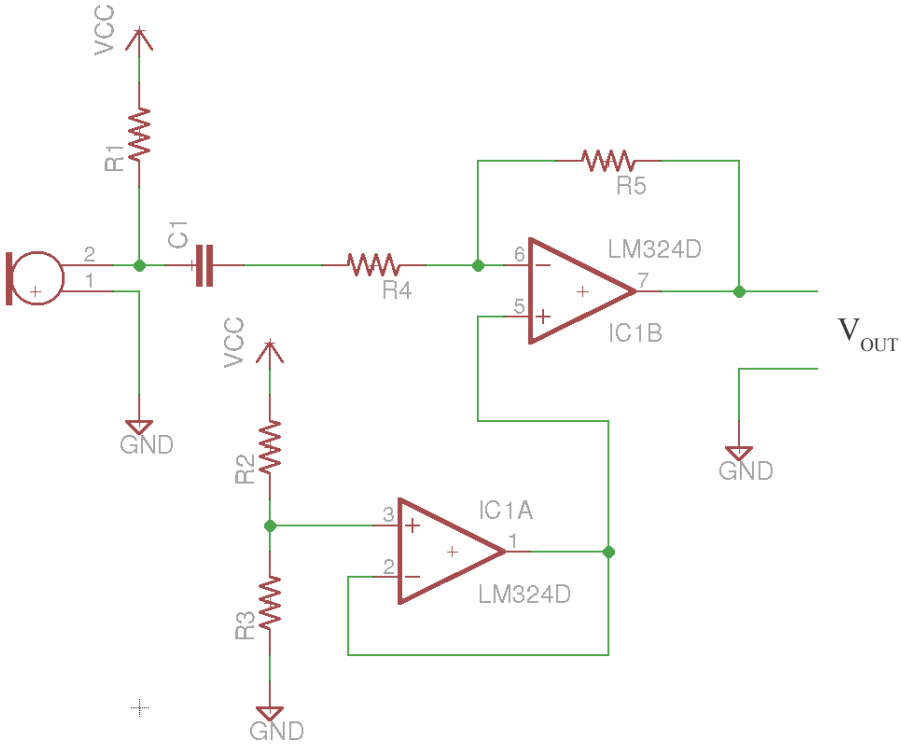


Figure 4: microphone pre-amplifier

The signal produced by the microphone is too small to be processed directly and should be amplified using the circuit shown in figure 4. Component values should be selected to produce an output signal centred about 1.5V, producing a maximum output of approximately 3.0V and a minimum output of not less than 0V.

Note: depending on the level of sensitivity required, further amplification stages may be required. When the camera is operating background noise levels will increase, therefore, you may wish to modify this circuit by placing a capacitor in parallel with the feedback resistor R5 to reduce high frequency gain.

The operational amplifier used to implement this circuit must be a LM324, powered using +5V and 0V power supply. You are **not** allowed to use the 741 operational amplifier, or +12 / -12V power supplies to build this circuit. Circuits using either of these will be awarded a zero mark. Guidance on operating operational amplifiers from a single rail supply i.e. +5V and 0V is given in the VLE.

To interface the Raspberry Pi to the amplified microphone signal an AD799, analogue to digital converter (ADC) has already been attached to the I2C bus. The data sheet for this device can be found on the VLE (AD799.pdf). To access data from this device an example program `adcTest.py` is shown in Appendix F. This code gives you initial guidance on how to test and use this device. The AD799 is a four channel ADC, access to these pins is shown in Appendix E.

Within the python file `constants.py` define a threshold constant for the microphone's ADC signal. When this user defined level is exceeded the system should print "`Microphone: Cockroach Detected`" on the screen and the current date and time.

All inputs and outputs used for switches and LEDs must be connected and controlled using an I2C PCF8574A GPIO expander. These signals must **not** be connected directly to the Raspberry Pi GPIO lines. When driving LEDs appropriate buffers / driver circuits should be used. When the state button is pressed the corresponding input signal will be driven a logic '0', when released a pull-up resistor should provide the default logic '1' state.

Owing to the mechanical properties of the switches used false signals may be produced when they are pressed i.e. when they change state from a logic 0 → 1 or from a logic 1 → 0. To prevent these false transitions changing the system's state, additional filtering my be required. For more information about this noise source and pull-up resistors refer to:

Digital inputs : https://en.wikipedia.org/wiki/Pull-up_resistor
Noise sources : https://en.wikipedia.org/wiki/Switch#Contact_bounce

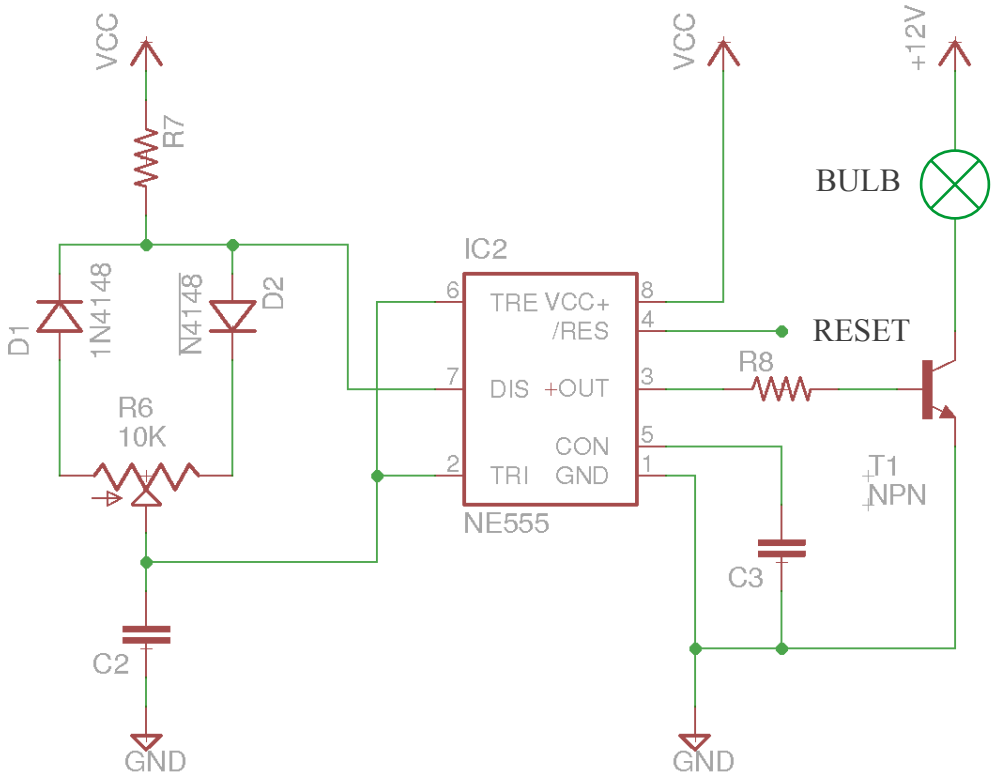


Figure 5: PWM light controller

The brightness of the tungsten bulb must be controlled using the PWM circuit shown in figure 5. This circuit uses a 555 timer, the mark-space ratio is determined by the position of the 10K variable resistor R6 and the capacitor C2. The value of capacitor should be selected to give a wide range of control i.e. illumination levels. An initial value of 1uF should be used. The open collector transistor driver stage must be implemented using the appropriate transistor driver on the FESC laboratory PCB or appropriate NPN transistor. For more information on 555 timer refer to:

555 timer : https://tinyurl.com/q98jymc

In automatic mode the system can control the position of variable resistor R6 using a

servo motor, as shown in figure 6. This motor is coupled to the variable resistor's drive shaft and can be controlled using a Raspberry Pi GPIO line. For more information on how to implement this refer to:

PWM GPIO : https://www.youtube.com/watch?v=ddlDgUymbxc

The position of the variable resistor in automatic mode is determined by the resistance of an LDR sensor. For more information on this sensor refer to the appropriate FESC laboratory script. Using this sensor, as the ambient light level decreases the mark-space ratio of the PWM signal produced by the 555 timer should be increased to maintain a constant light level and vice-versa. This control functionality must be implemented within the python program as a PID controller. The sensitivity of this controller i.e. how quickly it responds to changes in light level, must be defined using constants defined in the python file `constants.py`. For more information on this topic refer to:

PID control : https://en.wikipedia.org/wiki/PID_controller
Python example : https://tinyurl.com/33vu6yw

In manual mode the user must be able to set the position of the variable resistor i.e. rotate the drive shaft. To achieve this the system must switch the servo motor's supply voltage off using a Raspberry Pi GPIO line, or the I2C GPIO expander. As the servo supply will need approximately 250mA this output signal must be buffered using the appropriate transistor driver stage on the FESC laboratory PCB or appropriate NPN transistor.

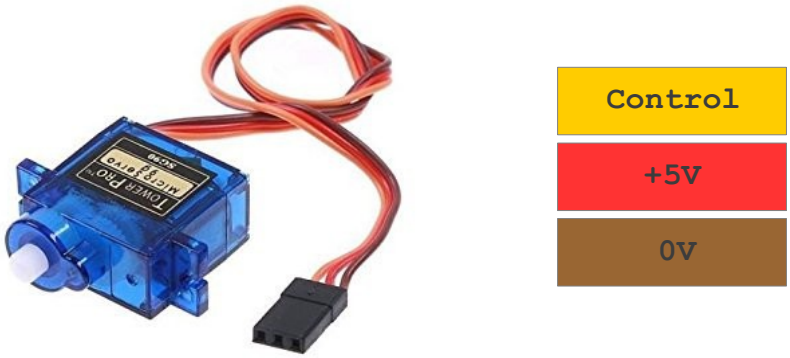


Figure 6: Servo (left), connections (right)

Each role within the group is allocated a specific task, this will form 10% of each group members overall mark.

- Hardware : using Vero-board construct a push button and LED (R,Y,G) board with appropriate resistors. You are also required to solder the 3.5mm audio socket. Connecting wires to this circuit should be selected using an appropriate colour scheme and cut to fit into breadboard i.e. minimising exposed copper and excessive wire lengths.

- Interfacing : construct a hardware de-bounce circuit on bread board for the push button. Also implement a software routine to de-bounce this switch, you may not use software events or callbacks. A nice article on switch de-bounce can be found here:

  Switch de-bounce : http://tinyurl.com/z6uermg

- Software : using the eight onboard LEDs shown in Appendix B, construct a linear display to show the microphone signal strength, as shown in figure 7. This should be updated in 'real-time'. For more information on the onboard

LEDs i.e. which Raspberry Pi GPIO pins are connected to which LED refer to Raspberry Pi circuit diagrams on the VLE. The maximum and minimum microphone signal threshold associated with each LED transition must be defined using constants in the python file `constants.py`.

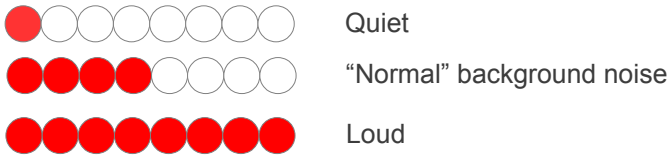


Figure 7: LED display

All PCBs used with the Raspberry Pi were designed using the Eagle PCB software. A free version is available from:

http://www.cadsoftusa.com/

The original Eagle circuit diagrams and PCB layouts are available on the VLE. They are also available as `.pdf` versions.

All software must be implemented in python. Software should be split into appropriate functions or structured using object orientated design techniques.

During the demonstration groups are only allowed to edit constants within the python file `constants.py`, therefore, software should be designed accordingly to allow each system function to be demonstrated.

Optional Software Components

To score maximum marks during your demonstration you will need to explain how your software operates and justify the software architectures chosen.

1) GUI v1

In addition to the plain text messages printed to the terminal implement a GUI using the Tkinter python package. You should also improve the functionality of this display by adding:

- camera and microphone status
- a counter of the number of cockroaches identified using the camera
- a counter of the number of cockroaches identified using the microphone
- system state: manual or automatic
- an ON/OFF button for the PWM light controller. Additional hardware may be required to implement this function. A Raspberry Pi GPIO line, or the I2C GPIO expander may be used.
- an ON/OFF button for the servo motor's supply.

2) GUI v2

Modify the GUI in part (1) to display the image captured by the camera. Also add a “SNAP” button to trigger the camera functions i.e. override the 2 second update rate.

3) Button Press Piezoelectric Buzzer

When the push button is pressed the system should indicate that this action has been detected by producing a short beep using a piezoelectric buzzer. This signal may be driven from a Raspberry Pi GPIO line, or the I2C GPIO expander. The length of the beep should be 1 second i.e. independent of the duration that the button is held in the closed position.

4) Interrupt Based Button Pressed Detection

Using the `INT` output on the PCF8574A GPIO expander and an additional Raspberry Pi GPIO line modify the system's python program to implement an interrupt driven, button pressed detection routine.

5) Image Processing

To improve bug detection accuracy image processing techniques can be used to help reduce background noise. You may wish to consider using morphological image processing techniques such as erosion and dilation, as described in the links below:

https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm
https://homepages.inf.ed.ac.uk/rbf/HIPR2/filtops.htm

During the demonstration you must be able to turn these features on/off by editing the python file `constants.py`. The effectiveness of these techniques will be judged via the debugging `ppm` image file. To score full marks there must be a visible reduction in noise. You are allowed to select a suitable background to help demonstrate the operation of these algorithms.

6) Multiple Cockroach Detection

Expanding on the system's core functionality, the system should be capable of identifying multiple individual cockroaches within a single image. In addition to printing the required message on the screen, the system should also print the number of cockroaches in the image. Again, for the purposes of debugging it should save an image file to disk, placing a RED rectangle around the area it believes contains each cockroach.

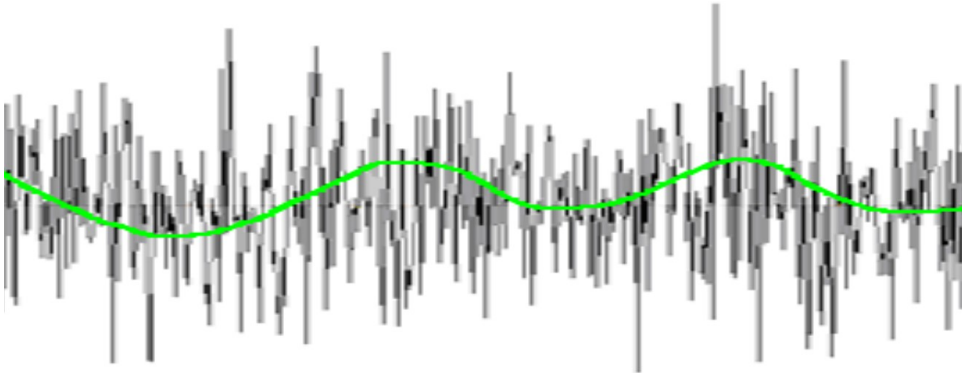


Figure 8: signal noise

7) Signal Filtering

A common problem in many systems is noise e.g. transient spikes, or white noise on top of the required signal, as shown in figure 8. A simple method for removing noise “spikes” is to use a median filter, based on a fixed size, sliding window i.e. a set of input samples. Consider the seven element window below:

```
05 50 10 07 00 11 09        -    Raw input data
00 05 07 09 10 11 50        -    Sorted data
00 05 07 09 10 11 50        -    Selection
50 10 07 00 11 09 03        -    Update raw data
```

Initially the system waits until it has captured seven samples. The filter successively removes the higher and lower values returning the middle value. If an even number of samples are taken the mean of the two remaining values is returned; in this case the value 9 is selected. The window is then updated, the first sample is removed and a new value added i.e. the window is shifted one place to the right. Implement a

median filter for the microphone signal, using a window size of 10. You may not use existing python libraries to implement this filter i.e. you must implement your own software function using standard python data types and instructions.

8) Detection Log File

When the system's main python program is terminated by pressing `CNTL-C` the detection time, date and type i.e. was a cockroach identified in an image or using the microphone should be recorded in a `.csv` log file.

Optional Hardware Components

To score maximum marks during your demonstration you will need to explain how the circuits in this section operate and justify your selection of resistor and capacitor values. You will be expected to show all calculations used in the design of these circuits.

1) Power Supply Noise

A common problem associated with analogue signals is noise e.g. transient spikes, or white noise. These unwanted signals can be caused by high speed digital circuits or radiated from high voltage cabling. Add smoothing capacitors to the supply rails on the bread-board to minimise mains noise and other sources of 'high' frequency noise. For more information on this topic refer to:

Noise : http://en.wikipedia.org/wiki/Decoupling_capacitor

2) Peak Detector

To ensure that the system does not miss short transient peaks within the microphone signal construct a peak detector as shown in figure 9. This circuit should be designed to maintain the peak output voltage between ADC sampling periods.

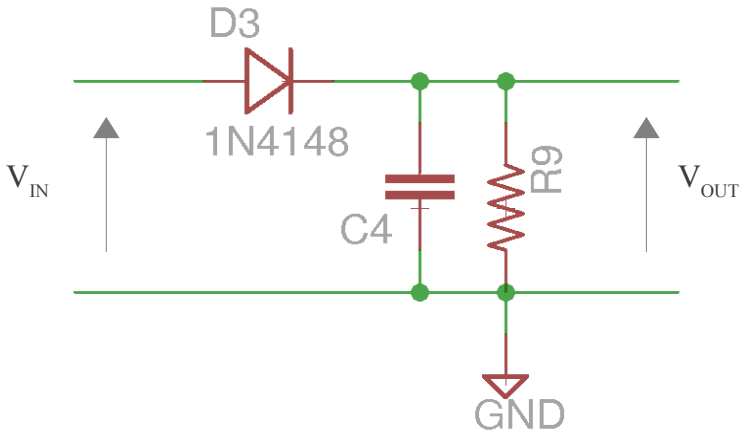


Figure 9: peak detector

3) Signal Filtering

To further improve the quality of the microphone signal a low pass filter should be constructed to remove high frequency noise. Refer to the appropriate FESC lectures and laboratory scripts for further information. For more information on how to design and implement an active low pass filter refer to:

Low pass filters : http://tinyurl.com/6caw4u

4) Piezoelectric Buzzer Oscillator

Replace the software generated square wave used to drive the piezoelectric buzzer with the circuit shown in figure 10. You may not use other circuits. The output of this

circuit `VOUT` **must** be combined with digital logic gates to produce the final buzzer circuit. The activation of this signal must be controlled using the PCF8574A GPIO expander. For more information on how to design and implement this circuit refer to:

Operational amplifier oscillator : https://tinyurl.com/ya3g2fj6

The operational amplifier used to implement this circuit must be a LM324, powered using +5V and 0V power supply. You are **not** allowed to use the 741 operational amplifier, or +12 / -12V power supplies to build this circuit. Circuits using either of these will be awarded a zero mark. Guidance on operating operational amplifiers from a single rail supply i.e. +5V and 0V is given in the VLE.

If you have implemented the software version of this buzzer you will need to demonstrate each implementation individually using separate piezoelectric buzzers. The python file `constants.py` must be used to enable or disable the software or hardware implementations.

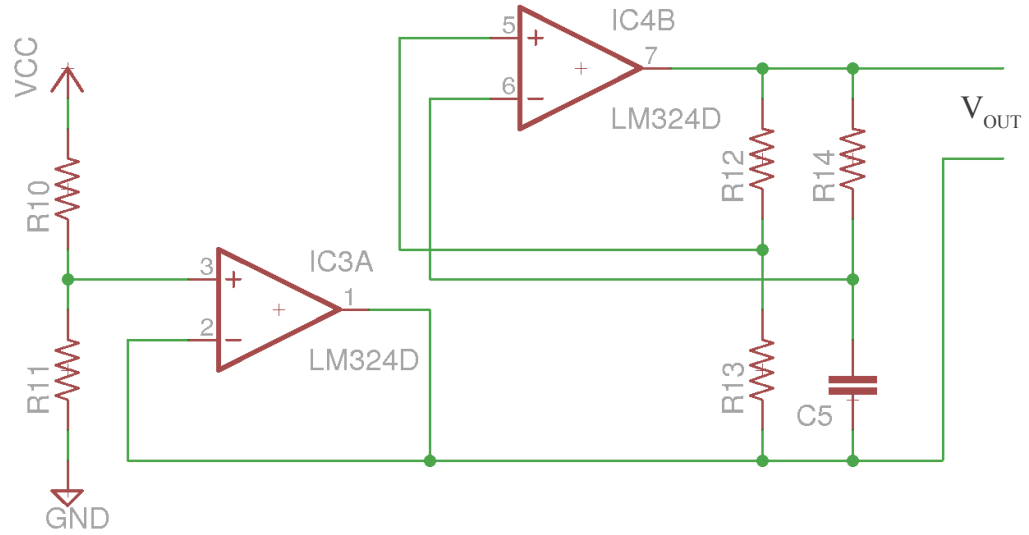


Figure 10: square wave oscillator

5) Piezoelectric Buzzer Driver

To increase the loudness of the piezoelectric buzzer used in this system a push-pull driver stage using five 7404 inverters can be constructed. One possible implementation is shown in figure 11.

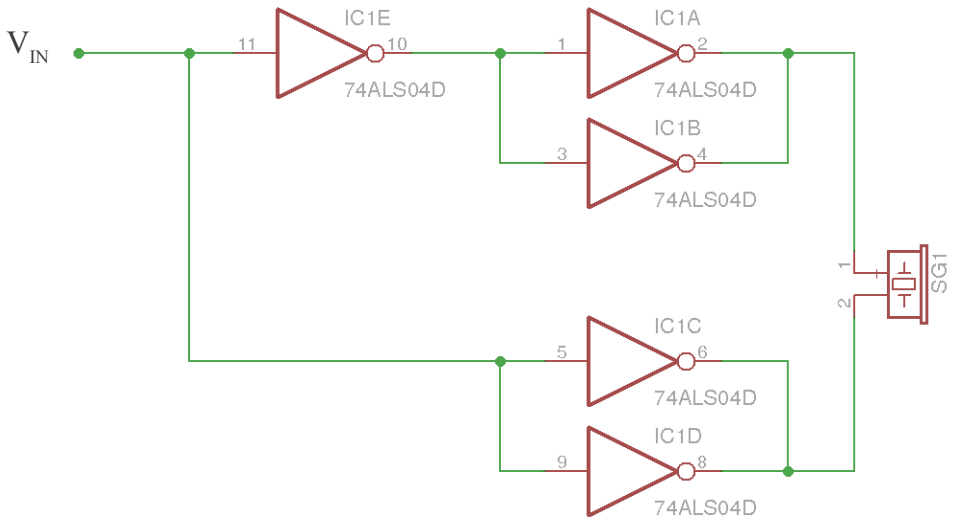


Figure 11: push-pull driver

This circuit may be attached to either the software or hardware buzzer. For more information on how to design and implement an exponential horn refer to:

Acoustic horn : https://en.wikipedia.org/wiki/Horn_loudspeaker
Logarithmic Horns : http://cq.cx/horn.pl

6) Button Press Piezoelectric Buzzer

Construct a monostable timer using either a 555 timer or an operational amplifier to produce a 1 second output pulse. This timer must be triggered by an output from the PCF8574A GPIO expander. Using digital logic gates combine this signal with the oscillator output shown in figure 10, such that when a button press is detected by the system's python program a 1 second beep is generated without the use of software timing functions. For more information on how to design and implement this type of timer refer to:

555 timer : https://tinyurl.com/q98jymc
Operational amplifier timer : https://tinyurl.com/ybuvbefn

7) 7 segment LED encoder

Design a digital decoder circuit to convert a 3bit binary value from the PCF8574A GPIO expander into signals capable of controlling a 7 segment LED display:

Input Value (3bit)	*7 segment character*
0	0
1	1
2	2
3	3
4	BLANK
5	BLANK
6	BLANK
7	BLANK

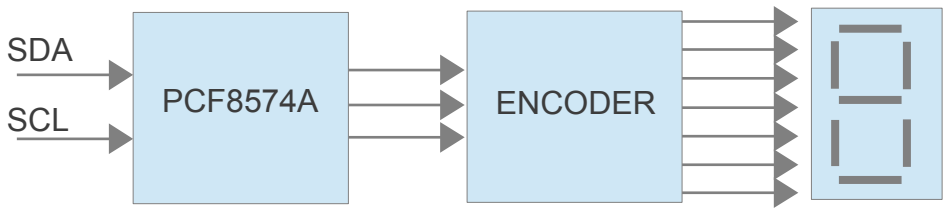


Figure 12: 7 segment display

When the system first starts, this display it should count down from 3 to 0, updating the display every second. When the system is operational the display should be blanked. When an image is taken the system will update this display to indicate the number of cockroaches present within an image. After a one second delay the display should be blanked.

8) Digital PWM controller

Using the I2C PCA9534 GPIO expander shown in Appendix D create an 6-bit data bus to control a digital PWM controller:

- 4 bits : PWM level, 0 = low (mostly off), 15=high (mostly on)
- 2 bits : OFF/PWM/ON, set the output to a logic 0, set the output to the selected PWM ratio, or set the output to a logic 1.

This 6-bit binary bus can **not** be implemented using Raspberry Pi GPIO lines, or a

different I2C device. The digital logic circuit used to generate the PWM signal must be constructed using TTL logic gates. The output of this circuit should be used to control a second tungsten filament bulb light source. The PWM frequency should be selected to give a wide range of control i.e. illumination levels.

A block diagram of one possible implementation is shown in figure 13. It is recommended that the following components are used.

- 74193 : up/down counter
- 7485 : comparator
- 7408, 7432, 7404 : AND, OR, NOT gates
- 555 timer : clock
- 2N4401 : NPN transistor

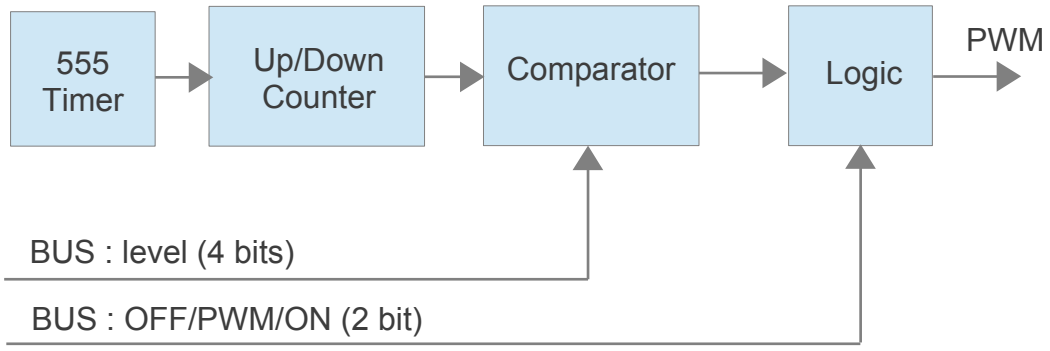


Figure 13: PWM block diagram

The state of this controller i.e. OFF/PWM/ON and PWM level, must be controlled using constants defined in the python file `constants.py`. You may also add buttons or text entry boxes to the GUI (if implemented) to control this circuit.

Marking

A group member's awarded mark is made up from their individual mark (10%), plus the group mark (90%).

Individual marks are assessed upon:

- Hardware :
  - Construction / soldering = 5 marks
  - Functionality = 5 marks
- Interfacing :
  - Construction / coding = 5 marks
  - Functionality = 5 marks
- Software :
  - Coding = 5 marks
  - Functionality = 5 marks

All solutions must implement the system's core functionality. Any combination of optional software or hardware components may be implemented e.g. you can have software and hardware buzzers. The final group mark is capped at 90%, 65% is awarded for the system's core functionality and 25% for optional software or hardware components.

Core components

1) RED, YELLOW, GREEN LED	= 4 marks
2) Push button	= 4 mark
3) Image cockroach detection, display message	= 16 marks
4) Debug image saved (ppm format)	= 5 marks
5) Microphone cockroach detection, display message	= 15 marks
6) PWM generator	= 8 marks
7) Automatic light control	= 8 marks
8) Manual light control	= 5 marks
Total core elements	= 65 marks

Optional software components:

1) GUI v1	= 4 marks
2) GUI v2	= 4 marks
3) Button Press Piezoelectric Buzzer	= 2 marks
4) Interrupt Based Button Pressed Detection	= 2 marks
5) Image processing **Note**, maximum marks will only be awarded if a combination of different image processing techniques are used.	= 8 marks
6) Multiple Cockroach Detection	= 10 marks
7) Signal Filtering	= 2 marks
8) Detection Log File	= 3 marks
Total software elements	= 35 marks

Optional hardware components:

1) Power Supply Noise	= 2 marks
2) Signal Filtering	= 4 marks
3) Peak Detector	= 2 marks
4) Piezoelectric Oscillator	= 4 marks
5) Piezoelectric Buzzer Driver	= 3 marks
6) Button Press Piezoelectric Buzzer	= 4 marks
7) LED encoder	= 5 marks
8) Digital PWM controller	= 10 marks
Total hardware elements	= 34 marks

If system functions do not meet the specified requirements, markers will decide upon an appropriate weighting for that element of the assessment. If markers decide that an implemented system function does not meet a minimum functional level a weighting of 0% will be applied.

As part of the demonstrations the group will be asked to explain the function and purpose of both software and hardware components within their system. If markers decide that a group's explanation does not meet a minimum level, a weighting of 0% will be applied to that element of the assessment.

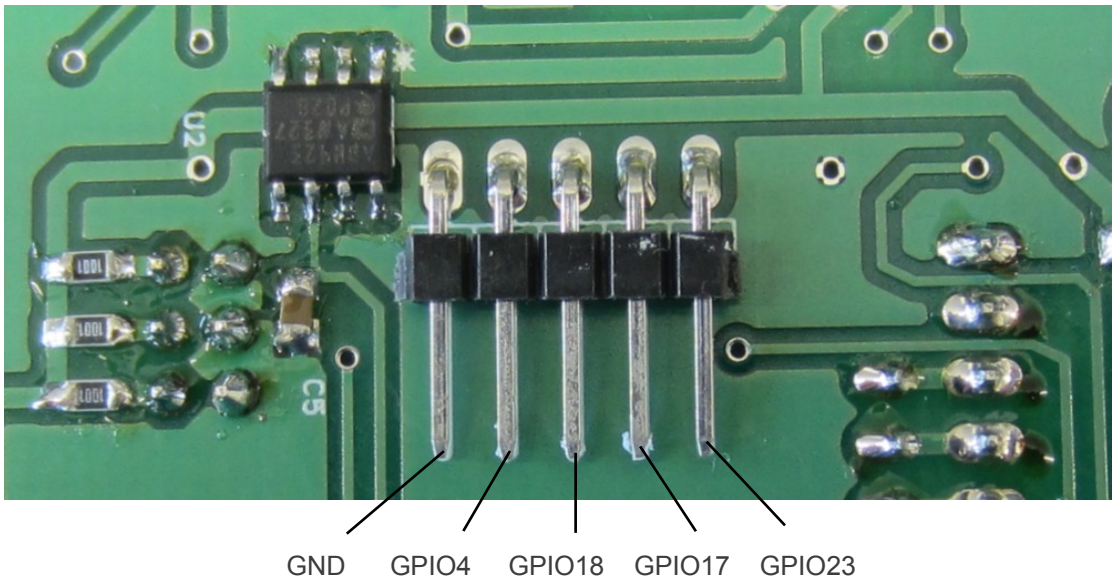
During the demonstration groups are only allowed to edit constants within the python file `constants.py`, therefore, software should be designed accordingly to allow each system function to be demonstrated.

Group Work

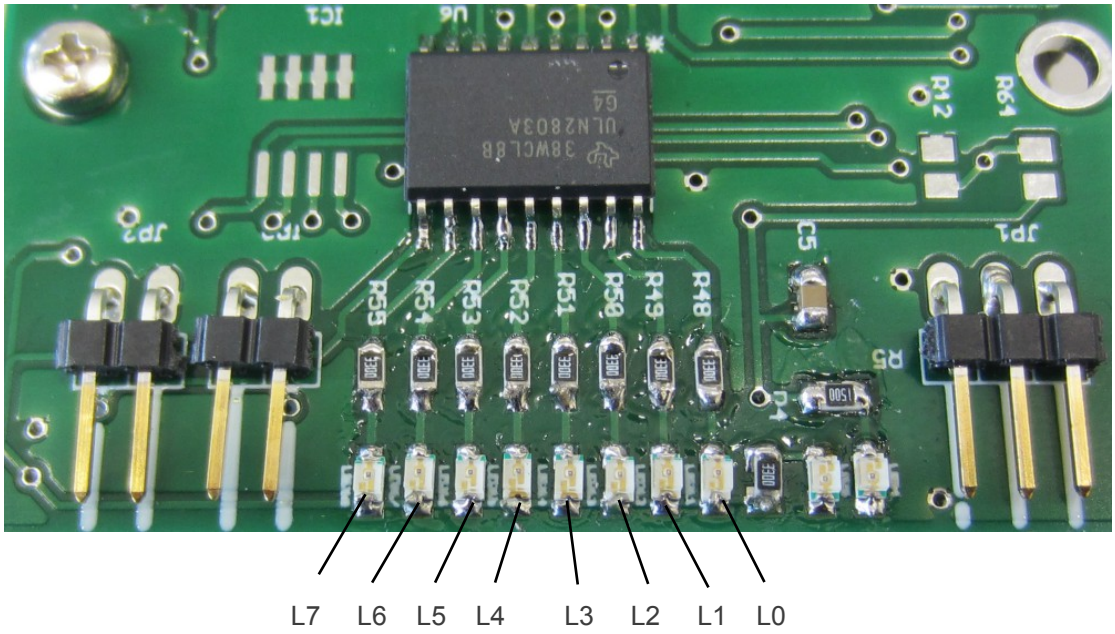
Each group is required to have at least one project meeting a week. These meetings should be minuted, recording the people present, the work completed since the last meeting, any problems encountered and actions to be performed by each group

member before the next meeting. As part of the demonstration each group is required to hand in a printed copy of these minutes.

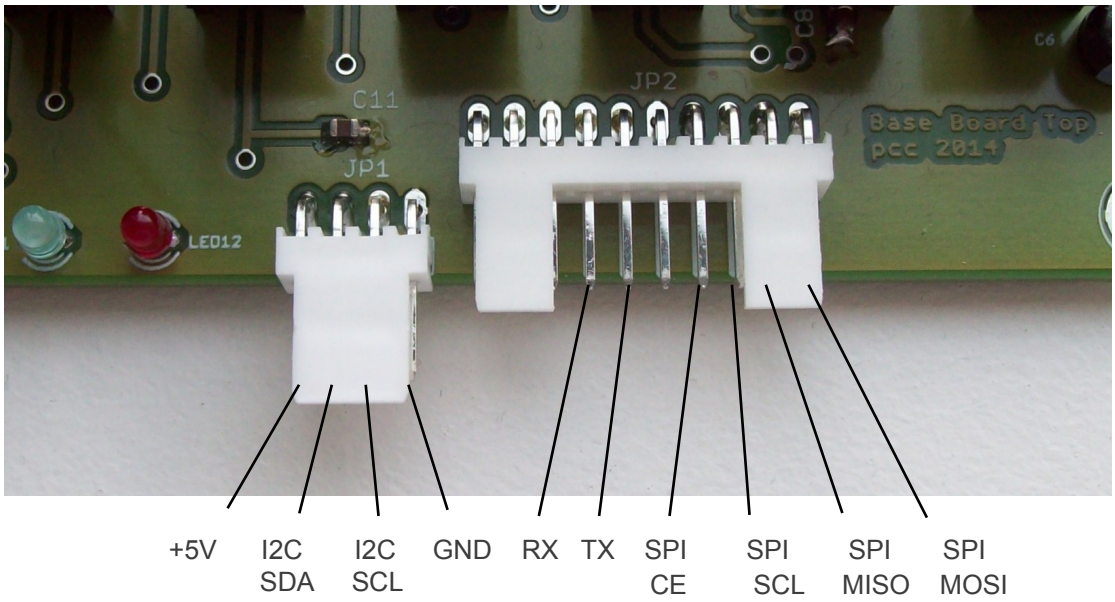
Appendix A: GPIO signals



Appendix B: GPIO LEDs



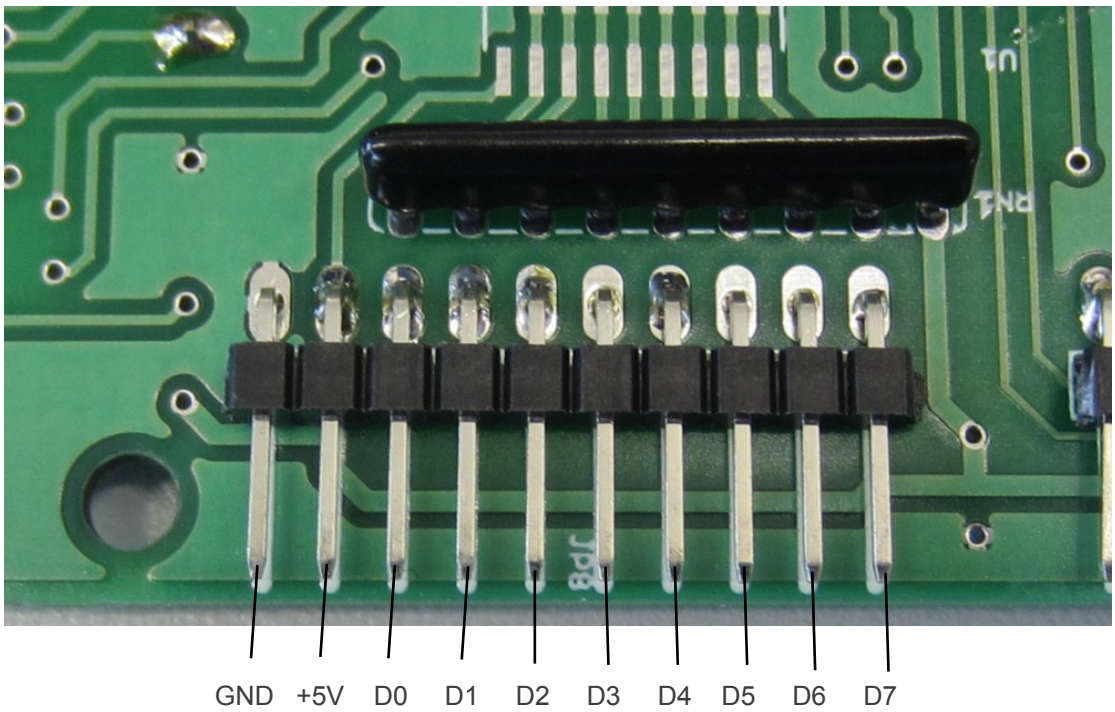
Appendix C: I2C / SPI inputs & Outputs



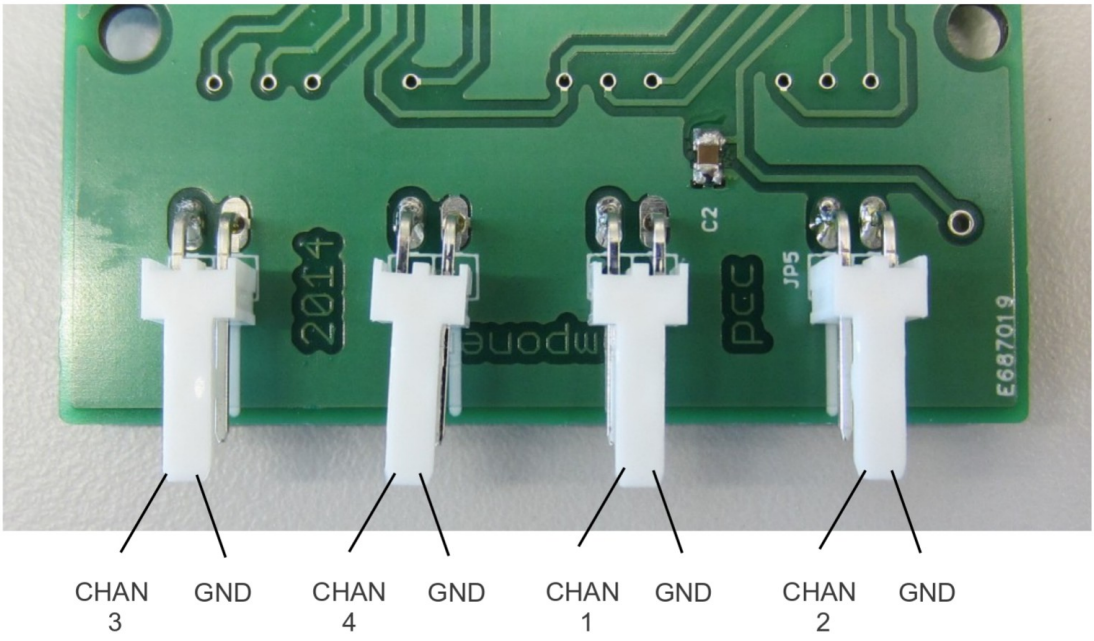
Note, the following pins can also be used as general purpose input outputs (GPIO)

- SPI MOSI : GPIO 10
- SPI MISO : GPIO 9
- SPI SCL : GPIO 11
- SPI CE : GPIO 8
- I2C SCL : GPIO 3
- I2C SDA : GPIO 2
- Serial TX : GPIO 14
- Serial RX : GPIO 15

Appendix D: I2C inputs / Outputs



Appendix E: ADC channels



Appendix F: ADC Code

```
# ADC TEST CODE

# TEST 1: to check if the ADC's I2C interface is working,
# at the command line type : sudo i2cdetect -y 1

# This will scan the I2C bus, returning active addresses.
# If the ADC is work address 0x21 will be listed.

# First words of advice : ** READ THE ADC's DATA SHEET **
# When you have done this consider the Python code below.
# I would suggest using the ADC in mode 2. A description
# is on page 29. To read an input you must first write a
# command word to the ADC, triggering the conversion.
# Then the converted value can be read back. HOWEVER, the
# 16 bit value read back contains additional information,
# as described on page 20.

import smbus
import time
I2CADDR = 0x21

bus = smbus.SMBus(1)
bus.write_byte( I2CADDR, 0x20 )
tmp = bus.read_word_data( I2CADDR, 0x00 )

# The value in the variable tmp needs further processing.
# Problem 1: the ADC returns the converted value back to
# the Pi in the wrong byte order. The Pi is expecting
# HIGH_BYTE:LOW_BYTE, but is receives LOW_BYTE:HIGH_BYTE.
# Therefore the bytes in the 16bit value in tmp must be
# swapped.

# Problem 2: as described on page 20. The top four bits
# of the received data packet do not contain numerical
# data. Therefore, these bits must be zeroed.

# Second words of advice : Bitwise functions don't try
# and convert the data into an ASCII string.

# Final words of advice : ** READ THE ADC's DATA SHEET **
```

Appendix G: Image Capture Script

```
#!/bin/sh

# To identify supported camera resolutions, at the
# command line type:
#
# v4l2-ctl --list-formats-ext
#
# Replace X and Y with the desired resolution

fswebcam -r XxY --no-banner image.jpg
convert image.jpg -compress none image.ppm
```