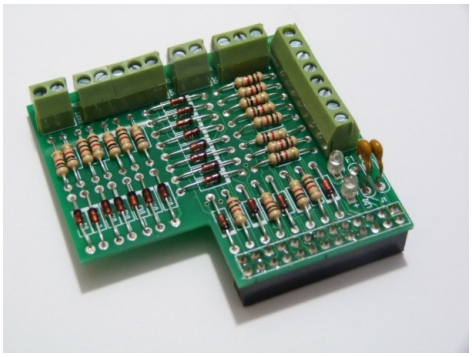
Custard Pi 1 - Breakout Board with protection for the Raspberry Pi GPIO
Full Technical Documentation

CONTENTS

INTRODUCTION

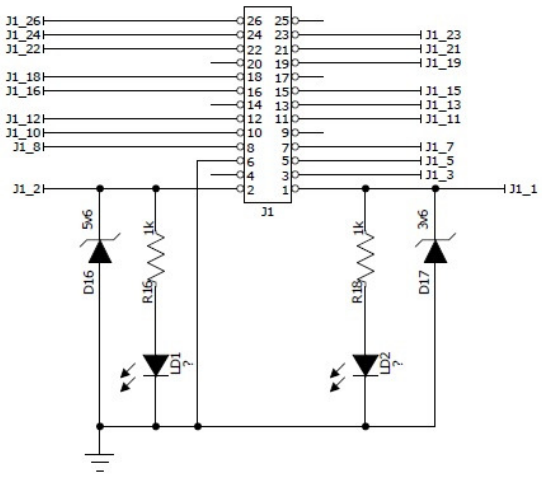
The Raspberry Pi GPIO allows the control of external electronics. There are two rows of 13 pins which are brought out to a 26 way header on the edge of the board. The Custard Pi 1 board simply plugs into the Raspberry Pi GPIO connector and allows users to quickly connect to all the pins. At the same time it protects the Raspberry Pi from possible damage from the wrong voltage being accidentally connected to the GPIO.



Custard Pi 1

CIRCUIT DESCRIPTION

When the Custard Pi is plugged into the GPIO, two LEDs come ON, showing that the 5V and 3.3V rail are working correctly.



GPIO connections showing power rails

The 3.3V is supplied on pin 1 of the GPIO and the 5V is supplied on pin 2. The 2 LEDs are connected to these pins with a 1k current limiting resistor. Note that there are no connections to pins 4, 9, 14, 17, 20 & 25 of the GPIO. On Revision 1 Raspberry Pi boards nothing should be connected to these pins. On Revision 2 boards, these are connected to 5V, 3.3V or Gnd as shown in the chart below.

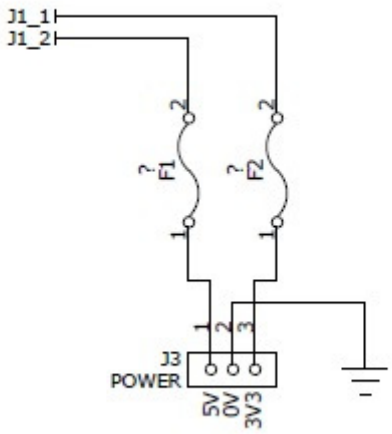
Rev 2	Rev 1	Function	Pin Numbers		Function	Rev 1	Rev 2
		Power 3.3 V	1	2	Power 5V		
		I2C SDA	3	4	***	DNC	5V
		I2C SCL	5	6	Gnd		
		GPIO 4	7	8	UART TXD		
Gnd	DNC	***	9	10	UART RXD		
		GPIO 17	11	12	GPIO 18		
GPIO 27	GPIO 21	***	13	14	***	DNC	Gnd
		GPIO 22	15	16	GPIO 23		
3.3V	DNC	***	17	18	GPIO 24		
		SPI MOSI	19	20	***	DNC	Gnd
		SPI MISO	21	22	GPIO 25		
		SPI CLK	23	24	SPI CE0		
Gnd	DNC	***	25	26	SPI CE1		

Layout of GPIO port pins on Rev 1 and Rev 2 boards

This chart shows the layout of the GPIO port pins. It looks quite complex, but once it is described piece by piece, it will be easier to understand.

Power pins (J3)

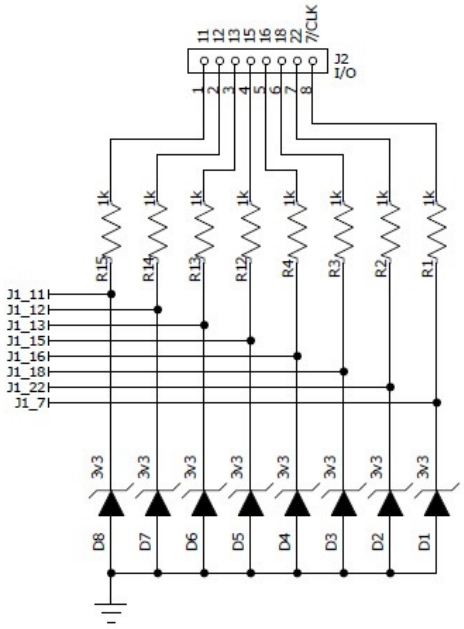
These are brought out to connector J3 on the Custard Pi 1 and have a fuse fitted to each line. This is to prevent the user from drawing too much current from the Raspberry Pi. The fuses are resettable and are both rated at 0.1 Amp (100 m Amp).



5V and 3.3V pins with fuses

General Purpose Input Output (I/O) pins (J2)

The pins marked green are general purpose digital input output pins. These are pins 11, 12, 13, 15, 16, 18, 22 and 7. They can be set high (to 3.3V) or low (0V) by program control from the Raspberry Pi. The Pi can also read whether these pins are high or low, say to a switch being pressed.



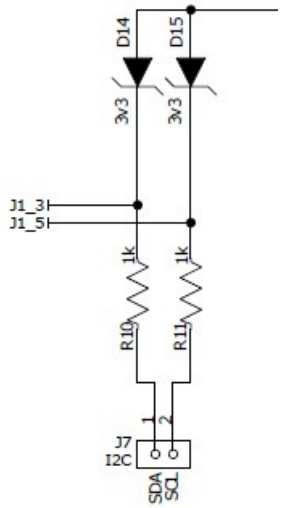
General purpose I/O pins

Each of these pins is protected by a 3.3V zener and a 1k ohm current limiting resistor. The 3.3V zener prevents any voltages in excess of 3.3V from being applied to the pins of the Integrated Circuit on the Raspberry Pi and damaging it. It also protects from negative voltages being applied to the pins. Due to the diode action of the zener voltages on the pins are limited to -0.7V.

The 1k ohm (1000 ohm) current limiting resistors are there to prevent too much current flowing in the zener if a wrong voltage was to be connected. However on later Custard Pi 1 boards this has been reduced to 220 ohms. The 1000 ohm resistor has colour bands brown, black red and the 220 ohm resistor has colour bands red, red and brown.

I2C serial bus (J7)

This bus is used to interconnect various integrated circuits such as A to D and D to A convertors, Port expanders and tone generators. The bus has a bi-directional data line and a clock line and can interface to many devices connected to the same 2 lines as the devices are addressable individually.

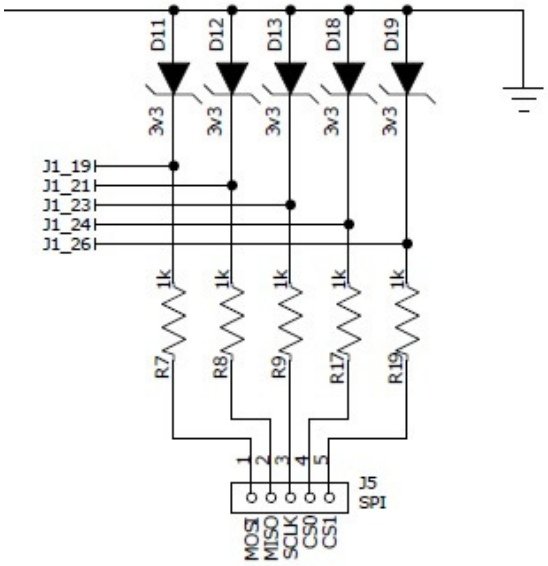


I2C bus pins

Internally, the Raspberry Pi uses two pull up resistors (1.8 k ohms) on these 2 pins. If you are planning to use the I2C bus then make sure that the current limiting resistors are 220 ohm and not 1000 ohms. If you have a Custard Pi 1 board with a 1k ohm resistor fitted, then either replace this with a 220 ohm resistor or solder a 220 ohm resistor on top of the 1k ohm resistor.

SPI serial bus (J5)

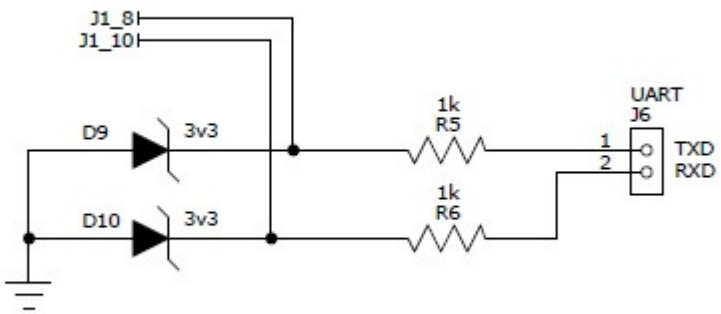
This serial bus can also be used to interface to a number of external integrated circuits. However it is different from the I2C bus in that it has separate data out and data in lines and the devices are not addressable. There are separate chip enable lines for each integrated circuit. The Raspberry Pi SPI bus is provided with 2 chip enable outputs.



SPI bus pins

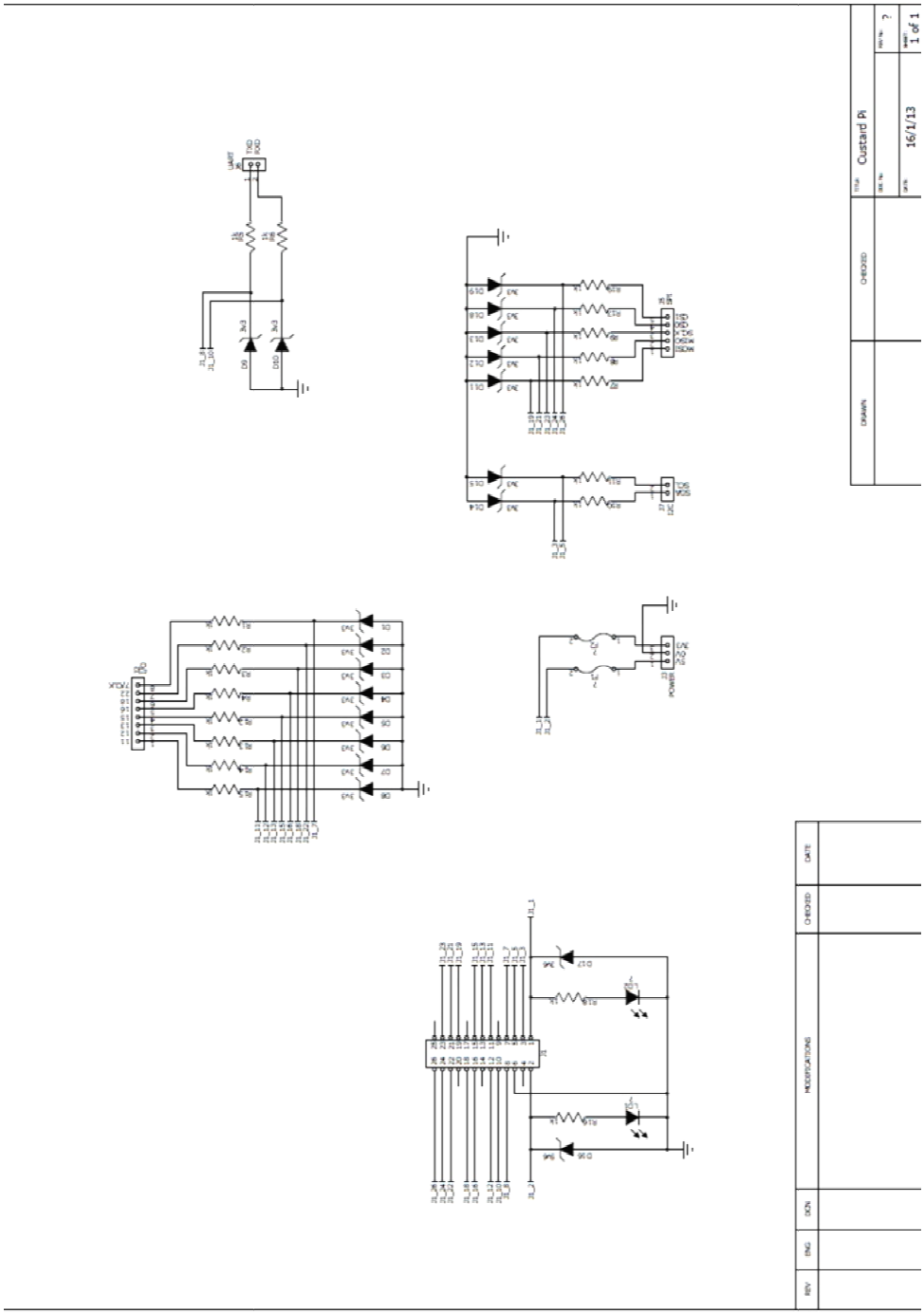
UART serial port (J6)

This is a serial port and can be used to communicate with external devices equipped with an RS232 serial interface.



UART pins

SCHEMATIC



Custard Pi 1 schematic

Note: On the latest version boards, 220 ohms are fitted instead of 1 k ohm.

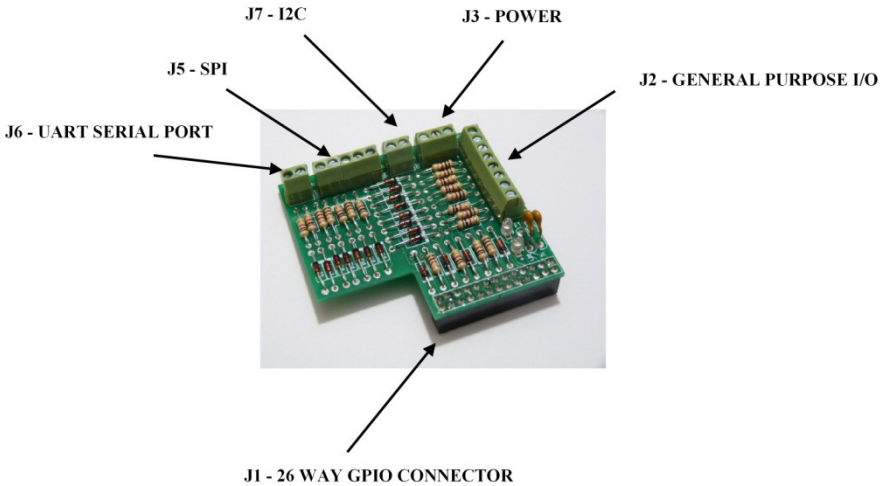
PARTS LIST

Custard Pi -1 Protection layer for the Raspberry Pi GPIO (rev 3 August 23rd 2013)

Description	Circuit reference	Notes
Printed Circuit Board (PCB)		
26 way connector	J1	Solder to underside of PCB
4 x 2 way screw terminal connectors	J2	Interlock the four terminals before soldering with wire access facing the edge of the PCB.
1 x 3 way screw terminal connector	J3	Make sure the wire access is facing the edge of the PCB
1 x 2 way screw terminal connector	J7	Make sure the wire access is facing the edge of the PCB
1 x 2way and 1 x 3 way screw terminal connectors	J5	Interlock the two terminals before soldering with wire access facing the edge of the PCB.
1 x 2 way screw terminal connector	J6	Make sure the wire access is facing the edge of the PCB
2 x Multifuse	F1, F2	Can be inserted either way round
2 x Leds	LD1, LD2	Make sure that the longer leg is inserted into hole marked +
19 x 220 ohm resistors	R1 to R19	Can be inserted either way round
18 x 3.6V zener	D1 to D15, D17 to D19	The black line on the zener has to line up with the line on the PCB
1 x 5.6V zener	D16	The black line on the zener has to line up with the line on the PCB
2 x sticky pads		To isolate the Custard Pi from Raspberry Pi components

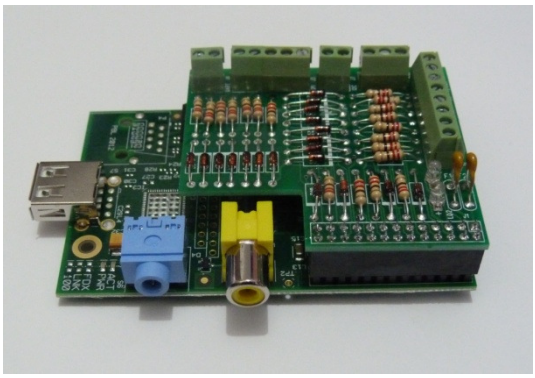
THE CUSTARD PI 1 ASSEMBLY

The positions of the connectors are shown below. These are mini screw terminals into which wires can be quickly connected.



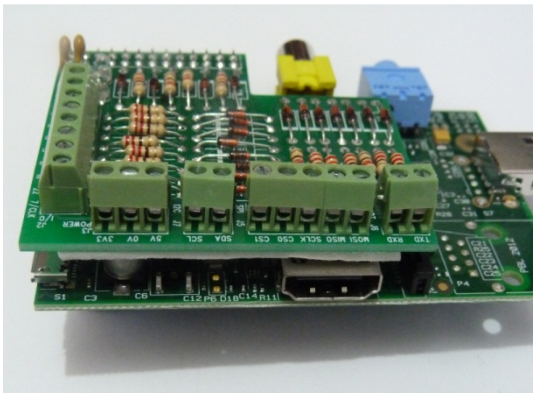
Positions of the connectors

This is a compact assembly that simply plugs into the Raspberry Pi GPIO. This can be done even with the Raspberry Pi is powered. Just make sure that the 2 power LEDs are on as soon as you plug in. If not there could be a fault with the Custard Pi 1 or it has not been plugged in properly.



Custard Pi 1 plugged into the Raspberry Pi GPIO

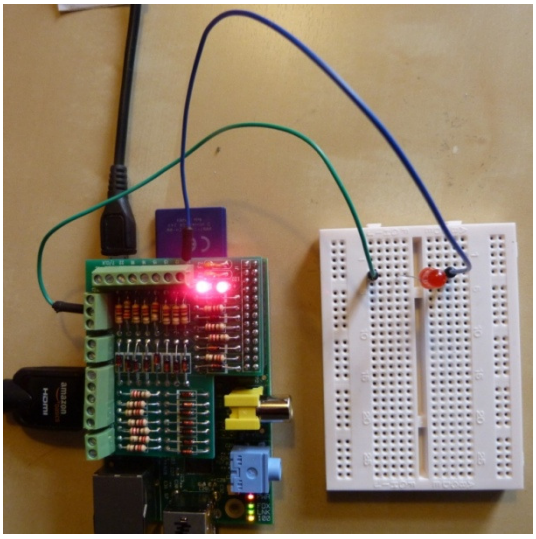
There is a risk of shorting between the pins on the base of the Custard Pi 1 and some of the components of the Raspberry Pi, such as the HDMI connector or capacitor C6. For this reason, the Custard Pi is supplied with a length of double sided sticky pad to act as insulation. If the Custard Pi 1 is bought as a kit of parts for self assembly, then sticky pads are supplied and must be used.



Sticky pads to insulate Custard Pi 1 from Raspberry Pi

PROJECT 1 - FLASHING an LED

Driving LEDs from the Custard Pi 1 is very easy. As there is a current limiting resistor built in (1k on early versions, 220 ohm on later versions). All one has to do is to connect an LED between one of the pins on connector J1 and Gnd. Just make sure that the long leg on the LED is connected to the pin and the short leg is connected to Gnd. In the code below, we assume that the LED is connected to pin 11 of J2, which is one of the general purpose I/O pins.



Custard Pi 1 connected to an LED

```
#sample Python code to flash an led
#www.sf-innovations.co.uk
import RPi.GPIO as GPIO                 # import GPIO library
import time                             #import time library
GPIO.setmode(GPIO.BOARD)                #use board pin numbers

GPIO.setup(11, GPIO.OUT)                #setup pin 11 as output

for x in range (0,10):                  #repeat for x=0 to 9
   GPIO.output(11, True)                #set pin 11 high
   time.sleep(0.2)                      #wait 0.2 seconds
   GPIO.output(11, False)               #set pin 11 low
   time.sleep(0.2)                      #wait 0.2 seconds

GPIO.cleanup()                          #tidy up GPIO port
import sys                              #exit program
sys.exit()
```

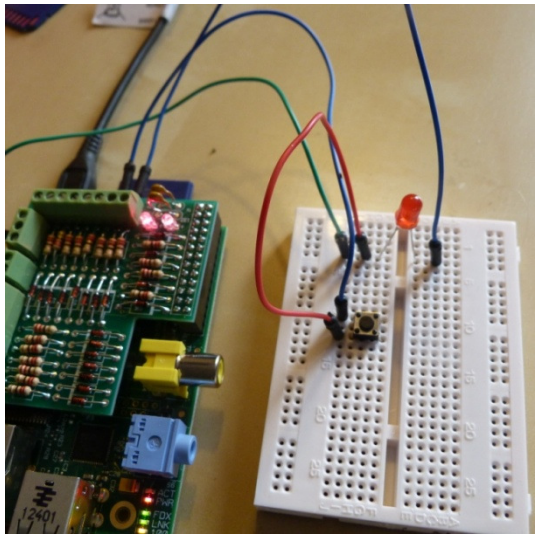
If you would like the LED to flash faster, then change the time.sleep(0.2) to a smaller value. For example time.sleep(0.1) would make the LED flash twice as fast. Both the time.sleep commands will need to be changed to halve the LED ON time and the LED OFF time.

If you would like the LED to carry on flashing 50 times, instead of just 10, then change the command “for x in range (0,10):” to “for x in range (0,50):”.

The GPIO.setmode command uses the board pin numbers as opposed to the port numbers of the IC used to control the GPIO port. In my experience it is much easier to use the pin numbers as these are clearly identified on the Custard Pi board.

PROJECT 2 - READING A SWITCH

In this mini project, we look at reading a switch and flash the LED only when the switch is pressed. The LED is connected to pin 11 as before. Connect the switch between pin 12 and Gnd. When the switch is pressed, pin 12 will be taken low. The Python code for this is presented below.



Custard Pi 1 connected to a switch and an LED

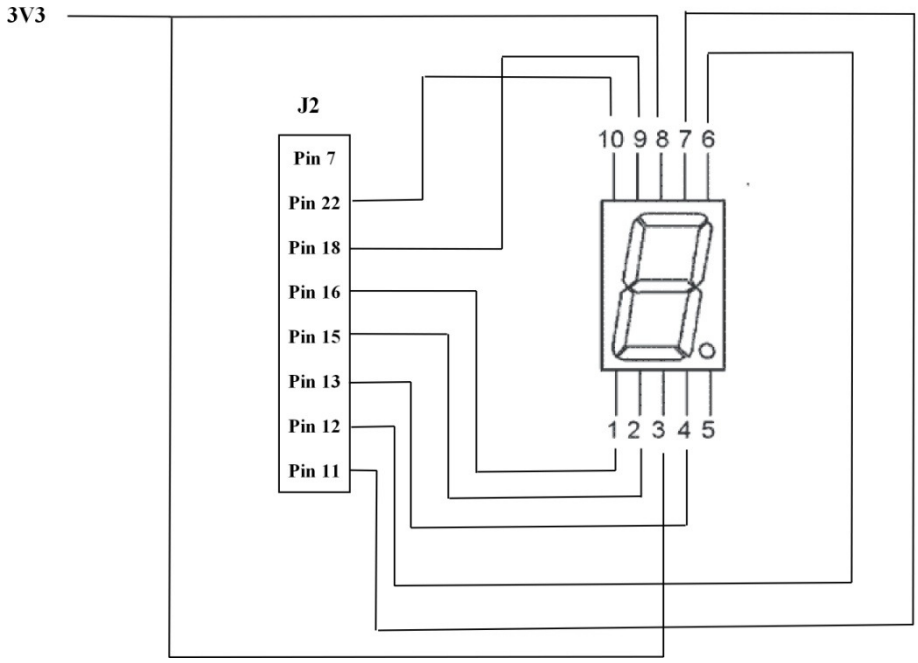
```
#sample Python code to flash an led when a switch is pressed
#www.sf-innovations.co.uk
import RPi.GPIO as GPIO              # import GPIO library
import time                          #import time library
GPIO.setmode(GPIO.BOARD)             #use board pin numbers
GPIO.setwarnings(False)
GPIO.setup(11, GPIO.OUT)             #setup pin 11 as output
GPIO.setup(12, GPIO.IN, pull_up_down=GPIO.PUD_UP)
                                     #setup pin 12 as input with pull up
while True:                          #do forever
    while GPIO.input(12)==False:     #while switch is pressed
       GPIO.output(11, True)         #set pin 11 high
       time.sleep(0.2)               #wait 0.2 seconds
       GPIO.output(11, False)        #set pin 11 low
       time.sleep(0.2)               #wait 0.2 seconds
GPIO.cleanup()                       #tidy up GPIO port
import sys                           #exit program
sys.exit()
```

This code is similar to the previous code but the LED flash is only executed if pin 12 goes low (FALSE) when the switch is pressed. Otherwise the “while True” command keeps the program in an endless loop, waiting for the switch to be pressed.

To exit the program, the user has to press CTRL and C at the same time on the keyboard. Because this exits the program without cleaning up the GPIO interface, we use the command “GPIO.setwarnings(False)” command to stop any warnings from being displayed.

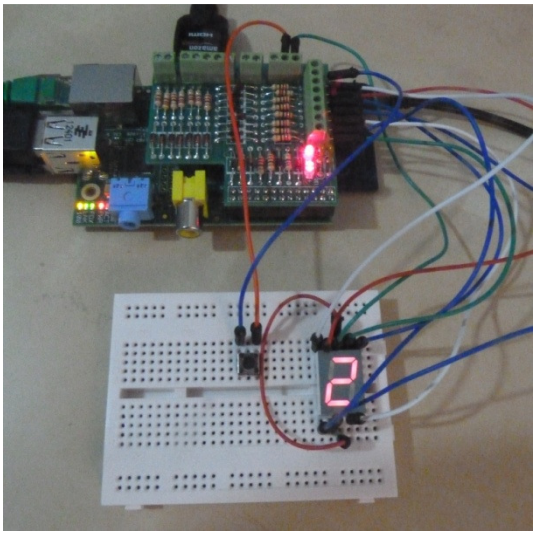
PROJECT 3 - ELECTRONIC DICE

This project uses a 7-segment display and a switch to simulate the roll of a dice. We will use 7 pins from J2 as outputs to drive the 7-segment display and the 8th pin as an input to read the switch. The drawing below shows how to connect up the 7-segment display to the Custard Pi 1.



Connecting the 7-segment display to the Custard Pi 1

Connect a switch between pin 7 of connector J2 and Gnd so that pin7 goes low (False) when the switch is pressed.



Electronic Dice using the Custard Pi 1

The Python code for the electronic Dice is presented below. When the program is started, the 7-segment shows the digit 0. When the switch is pressed, the 7-segment display will randomly display a digit from 1 to 6. This will stay on the display until the switch is pressed again.

```
#!/usr/bin/env python
#sample Python code to display a random digit
#from 1 to 6 when a switch is pressed
#www.sf-innovations.co.uk
import RPi.GPIO as GPIO
import time
import random

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BOARD)

#setup output pins
```

```
GPIO.setup(11, GPIO.OUT)
GPIO.setup(12, GPIO.OUT)
GPIO.setup(13, GPIO.OUT)
GPIO.setup(15, GPIO.OUT)
GPIO.setup(16, GPIO.OUT)
GPIO.setup(18, GPIO.OUT)
GPIO.setup(22, GPIO.OUT)
#setup inpt pin with pull up resistor
GPIO.setup(7, GPIO.IN, pull_up_down=GPIO.PUD_UP)

#define 7 segment digits
digitclr=[1,1,1,1,1,1,1]
digit0=[0,0,0,0,0,0,1]
digit1=[1,0,0,1,1,1,1]
digit2=[0,0,1,0,0,1,0]
digit3=[0,0,0,0,1,1,0]
digit4=[1,0,0,1,1,0,0]
digit5=[0,1,0,0,1,0,0]
digit6=[0,1,0,0,0,0,0]

gpin=[11,12,13,15,16,18,22]

#routine to clear and then write to display
def digdisp(digit):
    for x in range (0,7):
        GPIO.output(gpin[x], digitclr[x])
    time.sleep(0.5)
    for x in range (0,7):
        GPIO.output(gpin[x], digit[x])

#wait for switch to be released
def swwait():
    while GPIO.input(7)==False:
        time.sleep(0.1)

#display random digit
def randigit(digit):
    digdisp(digit)
    swwait()

#initialise by clearing display and writing 0
for x in range (0,7):
        GPIO.output(gpin[x], digitclr[x])
digdisp (digit0)

#main routine to read switch and display random digit from 1 to 6
while True:
    if GPIO.input(7)==False:
        rand = random.randint(1,6)
        if rand == 1:
            randigit(digit1)
        if rand == 2:
            randigit(digit2)
        if rand == 3:
            randigit(digit3)
        if rand == 4:
            randigit(digit4)
        if rand == 5:
            randigit(digit5)
        if rand == 6:
            randigit(digit6)

#tidy up
GPIO.cleanup()
import sys
sys.exit()
```