SF Innovations Ltd

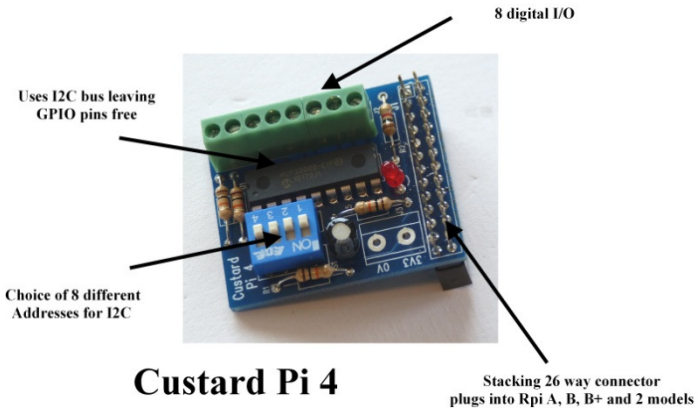
Custard Pi 4 - 8 Digital I/O card using I2C for the Raspberry Pi GPIO

User Instructions (22nd April 2016)

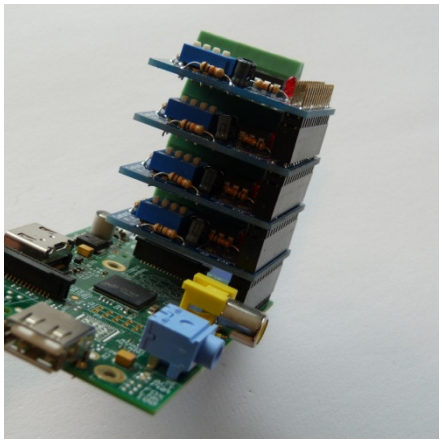
Contents

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 4 board simply plugs into the Raspberry Pi GPIO connector and provides eight digital I/O.



Each Custard Pi 4 allows the user complete flexibility in how each Digital I/O is configured. So for example with a single card the user can have 8 inputs, 8 outputs, 4 inputs & 4 outputs or indeed any other combination just by setting up the I/O expander IC (MCP23008) as required.

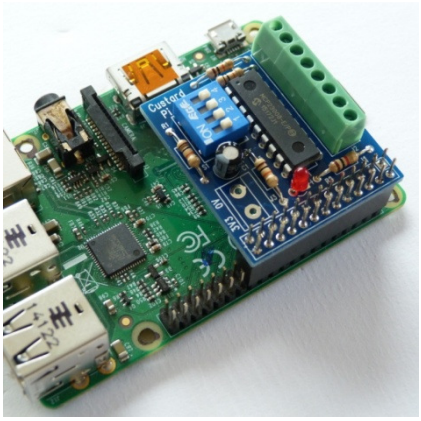


As the I2C address on each card be changed the user can use upto 8 of these with a single Raspberry Pi. This allows the user even more flexibility and some of the possible combinations are shown below.

* 64 digital outputs
* 64 digital inputs
* 32 digital outputs and 32 digital inputs

***** All this without using a single GPIO digital I/O pin *****

The Custard Pi 4 uses a stacking 26 way connector. What this means is that when it is plugged into the Raspberry Pi GPIO, these pins are still available for other accessories. The Custard Pi 4 can also be plugged into the 40 pin GPIO provided on the later raspberry Pi's such as the A+. B+, Raspberry Pi 2,3 and Zero models.



CE Compliance and Safety Information

* This product simply plugs into the Raspberry Pi GPIO to allow users to interface their products and projects to the Raspberry Pi. It uses the 3.3V supply from the Raspberry Pi to power the ADC and the DAC. For this reason it is outside the scope of the LVD Directive.

* The connection of incompatible devices to this product may cause damage and invalidate the warranty.

* All devices connected to this product must comply with all relevant standards to ensure that safety and performance is not compromised.

* This product complies with the Class B limit for Electromagnetic Radiation when used with the Raspberry Pi.

* This product provides reasonable protection against harmful interference in a residential installation. However there could be deterioration in performance in the presence of strong RF fields. For this reason, this product should not be used in any safety critical applications. Any wires connected to this product should be less than 2 metres in length. Avoid handling the PCB while it is powered. Only handle by the edges to avoid the risk of ESD damage.

* If this product is being incorporated in a commercial product which uses either all CE marked products or some CE marked and some non CE marked product, it is the responsibility of the system integrator to assess the end product for compliance with the relevant EU Directives.

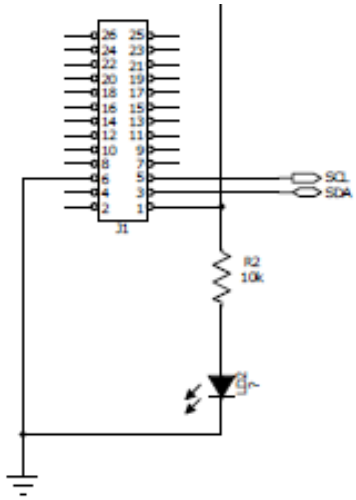
* This product complies with the requirements of the RoHS regulations.

Circuit Description

When the Custard Pi 4 is plugged into the GPIO, an LED comes ON, showing that the 3.3V rail is working correctly.

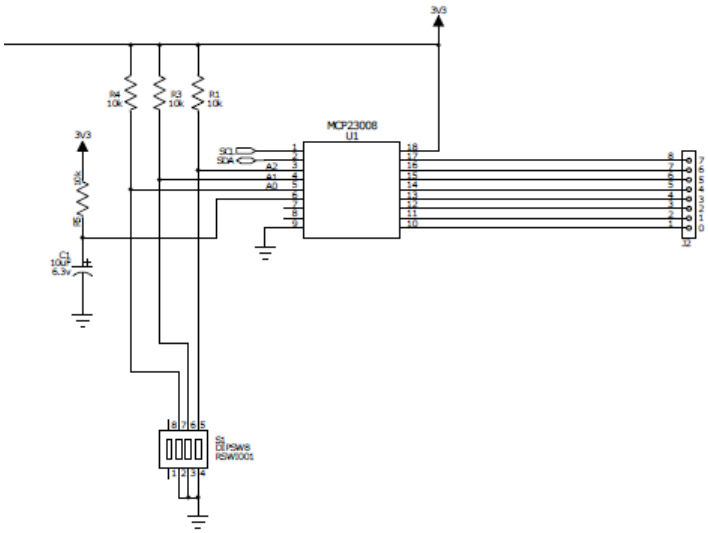
The diagram below shows the connections to the GPIO. Pin 1 is the 3.3V supply and pin 6 is the 0V connection. The other 2 pins shows are the connections to the I2C bus and are defined as follows.

Pin 3: SDA - I2C data
Pin 5: SCL - I2C clock



GPIO connections

The connections to the 8 channel I2C digital buffer is shown below.



The 8 digital I/O are not voltage limited or protected in any way. It is up to the user to make sure that no dangerous voltages are presented to the Custard Pi 4.

How to set the I2C address

The 4 way DIL switch allows the user to select the I2C address. When multiple Custard Pi 4s are used with a single Raspberry Pi then different I2C address mus be set for each of them. This is set as shown below.

S1 pos 2	S1 pos 3	S1 pos 4	Address
ON	ON	ON	add0
OFF	ON	ON	add1
ON	OFF	ON	add2
OFF	OFF	ON	add3
ON	ON	OFF	add4
OFF	ON	OFF	add5
ON	OFF	OFF	add6
OFF	OFF	OFF	add7

Position 1 is not used.

Note: add0 to add7 refers to the addresses set in the functions supplied by SF Innovations. If writing your own I2C routines please note that these map as follows.

add 0	ox20
add 1	ox21
add 2	0x22
add 3	0x23
add 4	0x24
add 5	0x25
add 6	0x26
add 7	0x27

How to set up the I2C buffer IC (MCP23008)

The sample programs below give an example of how to set up the MCP23008 to provide 4 outputs and 4 inputs. The information presented below should allow the user to adapt the sample program for different applications.

IODIR (0x00) - This register determines whether a pin is an input or an output.

INOUT - This variable is set to the desired value. For example '0x0F' sets bit 0 to 3 as inputs and bits 4 to 7 as outputs. If the user requires these all to be outputs then INOUT needs to be set to '0x00'. If they all need to be inouts then INOUT needs to be set to '0xFF'.

The program lines below sets the registers in the MCP23008.

```
def setinandout(address):
    #set inputs & outputs
    bus.write_byte_data(address, iodir, inout)
```

GPIO (0x09) - This holds the value of the inputs. The command to fetch this from the MCP23008 is shown below.

```
def readbit(address, bit):
    #read status of bit 0 to 3
    bitvalue = bus.read_byte_data(address, gpio) & bit
    return bitvalue
```

OLAT (0x0A) - The value in this variable appears at the digital outputs. Each individual bit can be set or cleared by using the definitions and commands below. The command first reads the value already set on the output and then gates it with a High or a Low to set or clear the pin.

```
#set output HIGH
ONbit4= 0x10
ONbit5= 0x20
ONbit6= 0x40
ONbit7= 0x80

#set output LOW
OFFbit4= 0xEF
OFFbit5= 0xDF
OFFbit6= 0xBF
OFFbit7= 0x7F

def setbit(address, byte):
    #sets selected port pin
    outstatus = bus.read_byte_data(address, olat) | byte
    bus.write_byte_data(address, gpio, outstatus)
```

```
def clrbit(address, byte):
    #clears selected port pin
    outstatus = bus.read_byte_data(address, olat) & byte
```

The information presented here along with the sample programs should allow the user to configure the Custard Pi 4 to any applications that requires a different set of output and input configurations.

Sample code for setting up and using the Custard Pi 4

There are two programs listed below. One has some functions defined which is then used by the second program to test out the Custard Pi 4.

The functions - cpi4.py

This program defines the I2C bus addresses that are available. It sets up the I2C 8 port buffer (MCP23008) as 4 inputs and 4 outputs. Bits 0 to 3 are set as inputs and bits 4 to 7 are set as outputs. It then defines the values to set each output as a High or a Low (ONbit and OFFbit). Then it defines functions which can be called from the test routine.

```
#1/usr/bin/env python
import time
import smbus

#****************************************
# Custard Pi 4 resources v1.0 22nd April 2016

#I2C addresses
#Set suitable address on Custard Pi 4

add0= 0x20
add1= 0x21
add2= 0x22
add3= 0x23
add4= 0x24
add5= 0x25
add6= 0x26
add7= 0x27

bus=smbus.SMBus(1)

#set IODIR register
iodir= 0x00
#set bit 0 to 3 as inputs, bits 4 to 7 as outputs
inout= 0x0F
#set GPIO register
gpio= 0x09
#set output latch
olat=0x0A

#set output HIGH
ONbit4= 0x10
ONbit5= 0x20
ONbit6= 0x40
ONbit7= 0x80

#set output LOW
OFFbit4= 0xEF
OFFbit5= 0xDF
OFFbit6= 0xBF
OFFbit7= 0x7F

def setbit(address, byte):
```

```
    #sets selected port pin
    outstatus = bus.read_byte_data(address, olat) | byte
    bus.write_byte_data(address, gpio, outstatus)

def clrbit(address, byte):
    #clears selected port pin
    outstatus = bus.read_byte_data(address, olat) & byte
    bus.write_byte_data (address, gpio, outstatus)

def readbit(address, bit):
    #read status of bit 0 to 3
    bitvalue = bus.read_byte_data(address, gpio) & bit
    return bitvalue

def setinandout(address):
    #set inputs & outputs
    bus.write_byte_data(address, iodir, inout)

def setpullups(address):
    #set inputs & outputs
    bus.write_byte_data(address, 0x06, 0xFF)

def alloff(address):
    #clear all outputs low
    bus.write_byte_data (address, gpio, 0x00)

#**************************************
```

The test routine - cpi4test.py

This test routine defines the I2C address to be used (add0. It then uses the command 'setinandout' to set bit 0 to 3 as inputs and bit 4 to 7 as outputs. All the outputs have pullups enabled and then switched to zero as default.

The test routine then sets each output High in turn and then Low again after a 0.5S pause. It then reads each of the 4 input in turn and prints out the status to the screen

```
#1/usr/bin/env python

import time
import cpi4

#start program

board1=cpi4.add0

cpi4.setinandout(board1)

cpi4.setpullups(board1)

cpi4.alloff(board1)

while True:
    cpi4.setbit(board1, cpi4.ONbit4)
    time.sleep (0.2)
    cpi4.setbit(board1, cpi4.ONbit5)
    time.sleep (0.2)
    cpi4.setbit(board1, cpi4.ONbit6)
    time.sleep (0.2)
    cpi4.setbit(board1, cpi4.ONbit7)

    time.sleep (0.5)
```

```
    cpi4.clrbit(board1, cpi4.OFFbit4)
    cpi4.clrbit(board1, cpi4.OFFbit5)
    cpi4.clrbit(board1, cpi4.OFFbit6)
    cpi4.clrbit(board1, cpi4.OFFbit7)

    time.sleep (0.5)

    bit0 = (cpi4.readbit (board1, 0x01))
    bit1 = (cpi4.readbit (board1, 0x02))
    bit2 = (cpi4.readbit (board1, 0x04))
    bit3 = (cpi4.readbit (board1, 0x08))

    if bit0 == 0:
        print "bit0 low"
    else:
        print "bit0 high"

    if bit1 == 0:
        print "bit1 low"
    else:
        print "bit1 high"

    if bit2 == 0:
        print "bit2 low"
    else:
        print "bit2 high"

    if bit3 == 0:
        print "bit3 low"
    else:
        print "bit3 high"

    print "**************************"

import sys
sys.exit()
```

End of Document