Project 15 - Reading a keypad with the Raspberry Pi

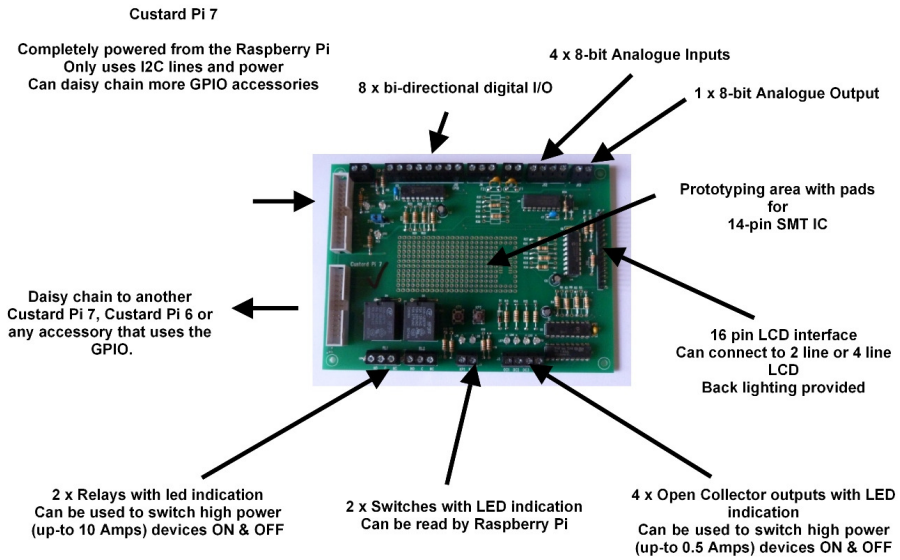
Outline

This application note describes how to read a 3 x 4 data keypad using the Raspberry Pi. Any of the Raspberry Pi models can be used including B, B+ and the Raspberry Pi 2.

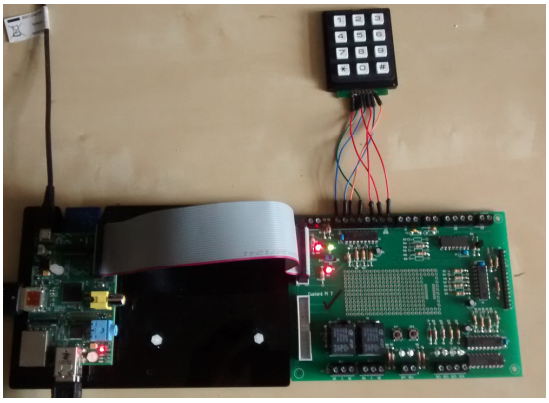
Hardware

3 x 4 data keypad: This is a common low-cost keypad for data entry in an industrial control system. It has 3 columns and 4 rows.

Custard Pi 7: Although the GPIO pins of the Raspberry Pi can be used for this project, we are going to use the Custard Pi 7 for this. This has an 8 bit bi-directional port which can be used to interface to the keypad. THis project will form the basis of further projects based around the Custard Pi 7.

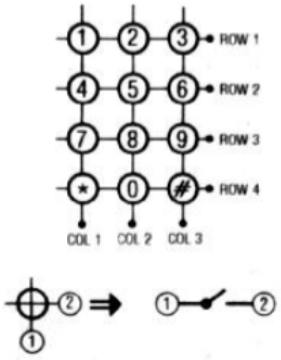


For ease of assembly, this project uses a Custard Pi base to mount the Raspberry Pi. The assembled hardware is shown below.



The layout of the keypad is shown here. When a key is pressed the particular row and column is connected together.

Circuit Diagram



OUTPUT ARRANGEMENT	
Output pin no.	Symbols
1	
2	Col. 2
3	Row 1
4	Col. 1
5	Row 4
6	Col. 3
7	Row 3
8	Row 2
9	
10	

At the bottom of the keypad the connections go from left to right as shown above. Pin 1 has no solder pad on it. Then we have Column 2, Row 1 etc up to Row 2 which is at the right hand side of the keypad. The connections between this and connector J10 of the Custard Pi 7 are as follows.

Output pin no on keypad	Reference	J10 connector number of CPi 7	reference
1	no solder pad		
2	Col 2	pin 1	input bit 1
3	Row 1	pin 4	output bit 4
4	Col 1	pin 0	input bit 0
5	Row 4	pin 7	output bit 7
6	Col 3	pin 2	input bit 2
7	Row 3	pin 6	output bit 6
8	Row 2	pin 5	output bit 5
9	no solder pad		

Software

To read the keypad, we are going to take each row low in turn and then scan column 1, 2 and 3 to see which one of these goes low. If column 1 goes low when row 1 is taken low then this means that key 1 has been pressed. If column 2 goes low when row 3 is taken low then this means that key 8 has been pressed.

However there is more to reading the keypad then just scanning the columns when each row is taken low. We need to 'debounce' the key inputs. When a key is pressed, it does not settle down straight away. There is mechanical 'bounce' where the key makes and breaks the electrical connection very fast before it settles down. If one does not 'debounce' in the software, a single key press can be read as a multiple key entry.

One way to debounce a key is to read it twice and if one gets a low both times then accept this as a genuine key entry. The software below uses a flag for each key to monitor this. The flag is at status 0 by default. On the first key press the status changes to 1. On the second key press the key press is accepted and the status changes to 1.

Once a key press is detected, it is important not to read it again until the key has been released. Flag status 2 and 3 are used to monitor this and also to 'debounce' the key release.

The software prints 'x pressed' and 'x released' to the screen when these events are detected. Keys 1 to 9 are read and displayed. It uses the cpi7y routine (which is listed in the Appendix 2) to read and write to the 8-bit port on the Custard Pi 7 board. The I2C address of the port is 0x26.

The I2C bus needs to be enabled on the Raspberry Pi before we can use it. Appendix 1 shows how to do this as well as to check the I2C bus address of all the devices on the Custard Pi 7

```
#1/usr/bin/env python
#program to read keypad and print results to screen

import time
import cpi7y            #routine to manage MCP23008 8-bit port expander

#start program

board1=cpi7y.add6       #set I2C address

cpi7y.setinandout(board1)   #set LSB 4 bits as inputs and MSB 4 bits as outputs
cpi7y.setpullups(board1)    #set pullup resistors on all pins

# set the 4 outpins to high
cpi7y.setbit(board1, cpi7y.ONbit4)
cpi7y.setbit(board1, cpi7y.ONbit5)
cpi7y.setbit(board1, cpi7y.ONbit6)
cpi7y.setbit(board1, cpi7y.ONbit7)

# initialise flags
flag1 = 0
flag2 = 0
flag3 = 0
flag4 = 0
flag5 = 0
flag6 = 0
flag7 = 0
flag8 = 0
flag9 = 0
flag0 = 0
flagstar = 0
flaghash = 0


while True:
    cpi7y.clrbit(board1, cpi7y.OFFbit4)     #take row 1 low

    bit0 = (cpi7y.readbit (board1, 0x01))   #read col 1
    if bit0 == 0:                           #flag = 0 default
        if flag1 <2:                        #flag = 1 key press 1st time
            if flag1 == 1:                  #flag = 2 key pressed 2nd time
                print "1 pressed"           #flag = 3 key released 1st time
                flag1=2
            else:
```

```
            flag1=1
if bit0 ==1:
    if flag1 >1:
        if flag1 == 3:
            print "1 released"
            flag1=0
        else:
            flag1=3

bit1 = (cpi7y.readbit (board1, 0x02))   #read col 2
if bit1 == 0:                           #flag = 0 default
    if flag2 <2:                        #flag = 1 key press 1st time
        if flag2 == 1:                  #flag = 2 key pressed 2nd time
            print "2 pressed"           #flag = 3 key released 1st time
            flag2=2
        else:
            flag2=1
if bit1 ==2:
    if flag2 >1:
        if flag2 == 3:
            print "2 released"
            flag2=0
        else:
            flag2=3


bit2 = (cpi7y.readbit (board1, 0x04))   #read col 3
if bit2 == 0:                           #flag = 0 default
    if flag3 <2:                        #flag = 1 key press 1st time
        if flag3 == 1:                  #flag = 2 key pressed 2nd time
            print "3 pressed"           #flag = 3 key released 1st time
            flag3=2
        else:
            flag3=1
if bit2 ==4:
    if flag3 >1:
        if flag3 == 3:
            print "3 released"
            flag3=0
        else:
            flag3=3

cpi7y.setbit(board1, cpi7y.ONbit4)      #take row 1 high again

cpi7y.clrbit(board1, cpi7y.OFFbit5)     #take row 2 low

bit0 = (cpi7y.readbit (board1, 0x01))   #read col 1
if bit0 == 0:                           #flag = 0 default
    if flag4 <2:                        #flag = 1 key press 1st time
        if flag4 == 1:                  #flag = 2 key pressed 2nd time
            print "4 pressed"           #flag = 3 key released 1st time
            flag4=2
        else:
            flag4=1
if bit0 ==1:
    if flag4 >1:
        if flag4 == 3:
            print "4 released"
            flag4=0
        else:
            flag4=3

bit1 = (cpi7y.readbit (board1, 0x02))   #read col 2
if bit1 == 0:                           #flag = 0 default
    if flag5 <2:                        #flag = 1 key press 1st time
        if flag5 == 1:                  #flag = 2 key pressed 2nd time
            print "5 pressed"           #flag = 3 key released 1st time
            flag5=2
        else:
            flag5=1
if bit1 ==2:
    if flag5 >1:
        if flag5 == 3:
            print "5 released"
            flag5=0
        else:
            flag5=3
```

```
    bit2 = (cpi7y.readbit (board1, 0x04))   #read col 3
    if bit2 == 0:                           #flag = 0 default
        if flag6 <2:                        #flag = 1 key press 1st time
            if flag6 == 1:                  #flag = 2 key pressed 2nd time
                print "6 pressed"           #flag = 3 key released 1st time
                flag6=2
            else:
                flag6=1
    if bit2 ==4:
        if flag6 >1:
            if flag6 == 3:
                print "6 released"
                flag6=0
            else:
                flag6=3

    cpi7y.setbit(board1, cpi7y.ONbit5)      #take row 2 high again

    cpi7y.clrbit(board1, cpi7y.OFFbit6)     #take row 3 low

    bit0 = (cpi7y.readbit (board1, 0x01))   #read col 1
    if bit0 == 0:                           #flag = 0 default
        if flag7 <2:                        #flag = 1 key press 1st time
            if flag7 == 1:                  #flag = 2 key pressed 2nd time
                print "7 pressed"           #flag = 3 key released 1st time
                flag7=2
            else:
                flag7=1
    if bit0 ==1:
        if flag7 >1:
            if flag7 == 3:
                print "7 released"
                flag7=0
            else:
                flag7=3

    bit1 = (cpi7y.readbit (board1, 0x02))   #read col 2
    if bit1 == 0:                           #flag = 0 default
        if flag8 <2:                        #flag = 1 key press 1st time
            if flag8 == 1:                  #flag = 2 key pressed 2nd time
                print "8 pressed"           #flag = 3 key released 1st time
                flag8=2
            else:
                flag8=1
    if bit1 ==2:
        if flag8 >1:
            if flag8 == 3:
                print "8 released"
                flag8=0
            else:
                flag8=3


    bit2 = (cpi7y.readbit (board1, 0x04))   #read col 3
    if bit2 == 0:                           #flag = 0 default
        if flag9 <2:                        #flag = 1 key press 1st time
            if flag9 == 1:                  #flag = 2 key pressed 2nd time
                print "9 pressed"           #flag = 3 key released 1st time
                flag9=2
            else:
                flag9=1
    if bit2 ==4:
        if flag9 >1:
            if flag9 == 3:
                print "9 released"
                flag9=0
            else:
                flag9=3

    cpi7y.setbit(board1, cpi7y.ONbit6)      #take row 3 high again
```

(Download this code from here: http://www.sf-innovations.co.uk/downloads)

Notes:

The Custard Pi 7 and Custard Base are available from amazon.co.uk and directly from SF Innovations. The code presented here can be downloaded from the SF Innovations website.

Ideas for Advanced projects

Add code to allow the software to read keys *, 0 and # and display this on the screen.

Set a 4 digit PIN number in software and when the user enters this print “WELCOME” on the screen. If they enter the wrong one then print “WRONG” on the screen. Use the ‘#’ key as the enter key.

APPENDIX 1

Setting up the I2C Bus

By default, the I2C bus routines are turned off in the operating system. The following steps need to be followed to enable these.

Step 1

At the command prompt type:

```
sudo nano /etc/modules
```

This uses the nano editor to make some changes to the modules file.Add the following two lines to this file

i2c-bcm2708
i2c-dev

Then save and exit the file using CTRL-x and Y.

Step 2

Make sure that you have the I2C utilities installed by executing the following two commands. The Pi will need to be connected to the Internet for this.

```
sudo apt-get install python-smbus
sudo apt-get install i2c-tools
```

If you get a 404 error do an update first:

```
sudo apt-get update
```

Note : The installation could take a few minutes to do, depend on how busy the server is.

Now add a new user to the i2c group:

```
sudo adduser pi i2c
```

Step 3

On the Raspberry Pi, the I2C and the SPI buses are usually disabled. This is done in the /etc/modprobe.d/raspi-blacklist.conf file.

If this file is not present then there is nothing to be done. Otherwise edit the file by typing the following at the command prompt.

```
sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

If the I2C and the SPI is blacklisted, you will see the following commands.

blacklist spi-bcm2708
blacklist i2c-bcm2708
Insert a # in front of these to comment them out.

Then save and exit the file using CTRL-x and Y.

After editing the file, you will need to reboot for the changes to take effect.

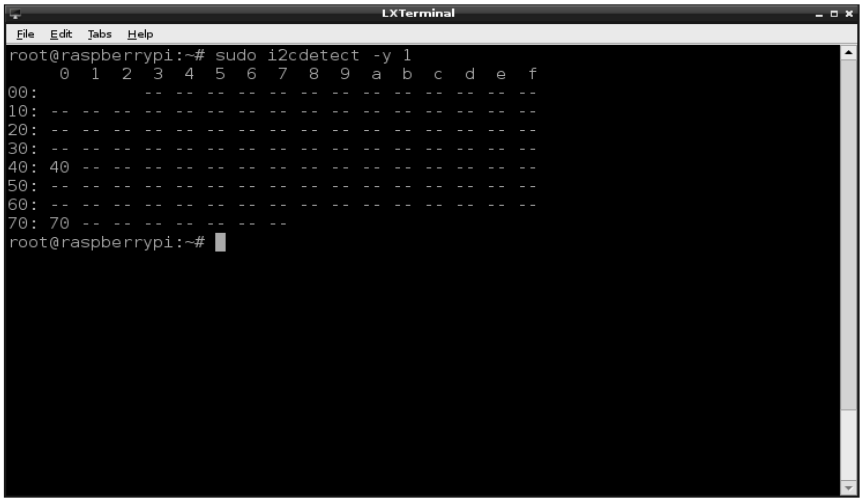
Step 4

Now we need to test if the I2C bus is working correctly.
Connect up the Custard Pi 6 board (or any other I2C bus device) and run the following command.

`sudo i2cdetect -y 1` (for Rev 2 boards which uses port 1)

Or
`sudo i2cdetect -y 0` (for Rev 1 boards which uses port 0)

If everything is OK, then the I2C address of the device will be shown as on the following slide. This shows two devices with address 40 and 70 in hexadecimal code. **(Note: We should get 25, 26, 27 and 4f for our project when the Custard Pi 7 is connected up).**



Now that we have set up the I2C serial bus routines, we are ready to drive the relays on the Custard Pi 6.

APPENDIX 2

This routine assists the user in writing to and reading from the 8-bit port expander using the I2C bus.

(Download this code from here: http://www.sf-innovations.co.uk/downloads)

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

#*****************************************
# Custard Pi 7 resources v1.0 5th Dec 2013

#I2C addresses
#Set suitable address on Custard Pi 7

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)

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