Hochschule Esslingen
University of Applied Sciences

I2C Configuration

Masterquad 2015

in the degree course ASM-SB
of the Faculty Graduate School
ASM2

Oliver Breuning
Martin Brodbeck
Jürgen Schmidt
Phillip Woditsch

Periode: Summersemester 2015
Professor: Prof. Dr. Jörg Friedrich

General Information

Contents

List of Figures

List of Tables

1 I^2C Configuration

This document describes all necessary steps to get the two I^2C Busses of the Raspberry Pi (Model B+) up and running. Because of different operating modes of the devices using the I^2C-Bus the usage of both busses is necessary.

HINT: These steps are not necessary if you install the Rasbian Image of the Projekt. There everything should be configured.

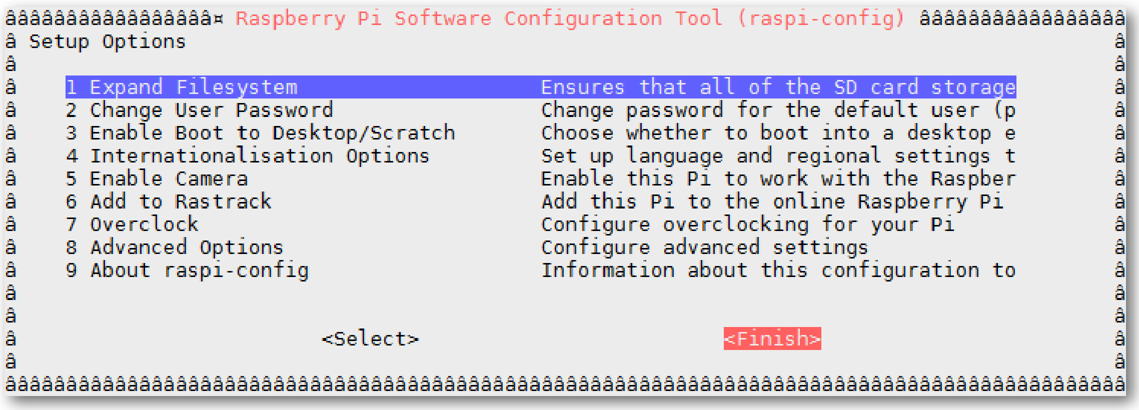
1.1 raspi-config

Enable I^2C using raspi-config utility.

From the command line type:

```
sudo raspi-config
```

This will open the raspi-config utility.



```
ââââââââââââââââââ¤ Raspberry Pi Software Configuration Tool (raspi-config) ââââââââââââââââââ
â Setup Options                                                                              â
â                                                                                            â
â    1 Expand Filesystem                    Ensures that all of the SD card storage          â
â    2 Change User Password                 Change password for the default user (p          â
â    3 Enable Boot to Desktop/Scratch       Choose whether to boot into a desktop e          â
â    4 Internationalisation Options         Set up language and regional settings t          â
â    5 Enable Camera                        Enable this Pi to work with the Raspber          â
â    6 Add to Rastrack                      Add this Pi to the online Raspberry Pi           â
â    7 Overclock                            Configure overclocking for your Pi               â
â    8 Advanced Options                     Configure advanced settings                      â
â    9 About raspi-config                   Information about this configuration to          â
â                                                                                            â
â                                                                                            â
â                        <Select>                        <Finish>                            â
â                                                                                            â
ââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââ
```

Figure 1.1: raspi-config

Now complete the following steps:

Select: "8 - Advanced Options"
Select: "A7 - I^{2}C"
Select: "Yes"
The Screen will ask if you want the interface to be enabled:
Select: "Yes"
Select: "OK"
The Screen will ask if you want the module to be loaded by default:
Select: "Yes"
The Screen will state the module will be loaded by default:
Select: "OK"
Select "Finish" to return to the command line
When you next reboot the I^{2}C module will be loaded.

1.2 Module File

Next we need to edit manually the modules file using:

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

and add the following lines:

```
i2c-bcm2708
i2c-dev
```

Use CTRL-X, then Y, then RETURN to save the file and exit.

1.3 I^{2}CTools

For hardware monitoring, device identification, and troubleshooting we install "i2c-tools".

```
sudo apt-get update
sudo apt-get install i2c-tools
```

Now shutdown your system, disconnect the power to your Pi and you are ready to connect your I^{2}C-hardware.

1.4 Test I^2C-1

Check if I^2C is enabled:

When you power up or reboot your Pi you can check the I^2C module is running by using the following command:
`lsmod | grep i2c_`
That will list all the modules starting with "i2c_". If it lists "i2c_bcm2708" then the module is running correctly.

Testing Hardware:
Once you've connected your hardware double check the wiring. Make sure 5V is going to the correct pins and you've got not short circuits. Power up the Pi and wait for it to boot. Then type the following command:

```
sudo i2cdetect -y 1
```

With e.g. a sensor connected the output looks e.g. like this:

```
    0 1 2 3 4 5 6 7 8 9 a b c d e f
00:       - - - - - - - - - - - - -
10: - - - - - - - - - - - - - - - -
20: - - - - - - - - - - - - - - - -
30: - - - - - - - - - - - - - - - -
40: - - - - - - - - - - - - - - - -
50: - - - - - - - - - - - - - - - -
60: - - 62 - - - - - - - - - - - - -
70: - - - - - - - -
```

This shows that one device is connected and its address is 0x62.

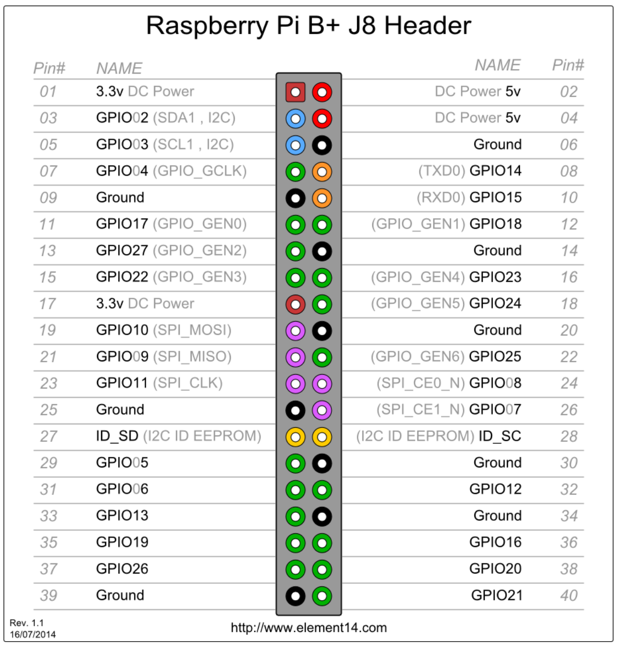
1.5 Set up I^{2}C-0

In normal configuration the second I^{2}C-Bus of the Raspberry Pi is set up as two of the output pins of the DSI Display Connector resp. the CSI Camera Connector.
To make the setup of the quadrocopter as easy as possible and with respect to the weight and soldering/cabling these output pins were redirecteed to two of the 40 pins of the GPIO Header.

This gets done by useage of a Python-script (see below) which gets excecuted while booting the system. To get this configuration running two additional files need to be edited.

In "/boot/cmdline.txt"
`bcm2708.vc_i2c_override=1`
has to be added
in "/etc/modprobe.d/i2c_o_enable.conf"
`blacklist snd_soc_tas5713`
has to be added.

After this the GPIO port 27 is configured as SDA0 and the GPIO port 28 as SCL0.


Raspberry Pi B+ J8 Header

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I2C)	DC Power 5v	04
05	GPIO03 (SCL1 , I2C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)	(I2C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Rev. 1.1
16/07/2014
http://www.element14.com

Figure 1.2: GPIOs [2]

2 http://www.element14.com/community/servlet/JiveServlet/previewBody/68203-102-6-294412/GPIO.png

Listing 1.1: I^{2}C0 Port-Configuration

```
#!/usr/bin/python
#!/usr/bin/env python
#
#

# #######
# For I2C configuration test
import os
import mmap
bplus=0
BCM2708_PERI_BASE=0x20000000
GPIO_BASE=(BCM2708_PERI_BASE + 0x00200000)
BLOCK_SIZE=4096

def _strto32bit_(str):
    return ((ord(str[3])<<24) + (ord(str[2])<<16) + (ord(str[1])<<8) + ord(str[0]))

def _32bittostr_(val):
    return chr(val&0xff) + chr((val>>8)&0xff) + chr((val>>16)&0xff) + chr((val>>24)&0xff)

def get_revision():
    with open('/proc/cpuinfo') as lines:
        for line in lines:
            if line.startswith('Revision'):
                return int(line.strip()[-4:],16)
    raise RuntimeError('No␣revision␣found.')

def i2cConfig():
    if get_revision() >= 10:
        print "B+␣or␣CM␣detected."
        s0 = 0b00000000000000000000100100100100
        s2 = 0b00000000000000000000000000000000
    if get_revision() <=9:
        s0 = 0b00000000000000000000100100000000
        s2 = 0b00100100000000000000000000000000
    if get_revision() <= 3:
        print "Rev␣2␣or␣greater␣Raspberry␣Pi␣required."
        return
    # Use /dev/mem to gain access to peripheral registers
    mf=os.open("/dev/mem", os.O_RDWR|os.O_SYNC)
    m = mmap.mmap(mf,BLOCK_SIZE, mmap.MAP_SHARED,
            mmap.PROT_READ|mmap.PROT_WRITE, offset=GPIO_BASE)
    # can close the file after we have mmap
    os.close(mf)
    # Read function select registers
    # GPFSEL0 -- GPIO 0,1 I2C0   GPIO 2,3 I2C1
    m.seek(0)
    reg0=_strto32bit_(m.read(4))
    # GPFSEL2 -- GPIO 28,29 I2C0
    m.seek(8)
    reg2=_strto32bit_(m.read(4))
    # print bin(reg0)[2:].zfill(32)[2:]
    # print bin(reg2)[2:].zfill(32)[2:]

    # GPFSEL0 bits --> x[26] SCL0[3] SDA0[3]
    #                        GPIO    GPIO
    m0 = 0b00000000000000000000111111111111
    #s0 = 0b00000000000000000000100100100100
    b0 = reg0 & m0
    if b0 <> s0:
        #print "reg0 I2C configuration not correct. Updating."
        reg0 = (reg0 & ~m0) | s0
        m.seek(0)
        m.write(_32bittostr_(reg0))

    # GPFSEL2 bits --> x[2] SCL0[3] SDA0[3] x[24]
    m2 = 0b00111111000000000000000000000000
    b2 = reg2 & m2
    if b2 <> s2:
        #print "reg2 I2C configuration not correct. Updating."
        reg2 = (reg2 & ~m2) | s2
        m.seek(8)
        m.write(_32bittostr_(reg2))

    # No longer need the mmap
    m.close()


if __name__ == '__main__':
    i2cConfig()
```