Multimedia Technieken

2015 – 2016

Controlling an I2C Slave Device from User Space

katholieke hogeschool vives
associatie KU Leuven

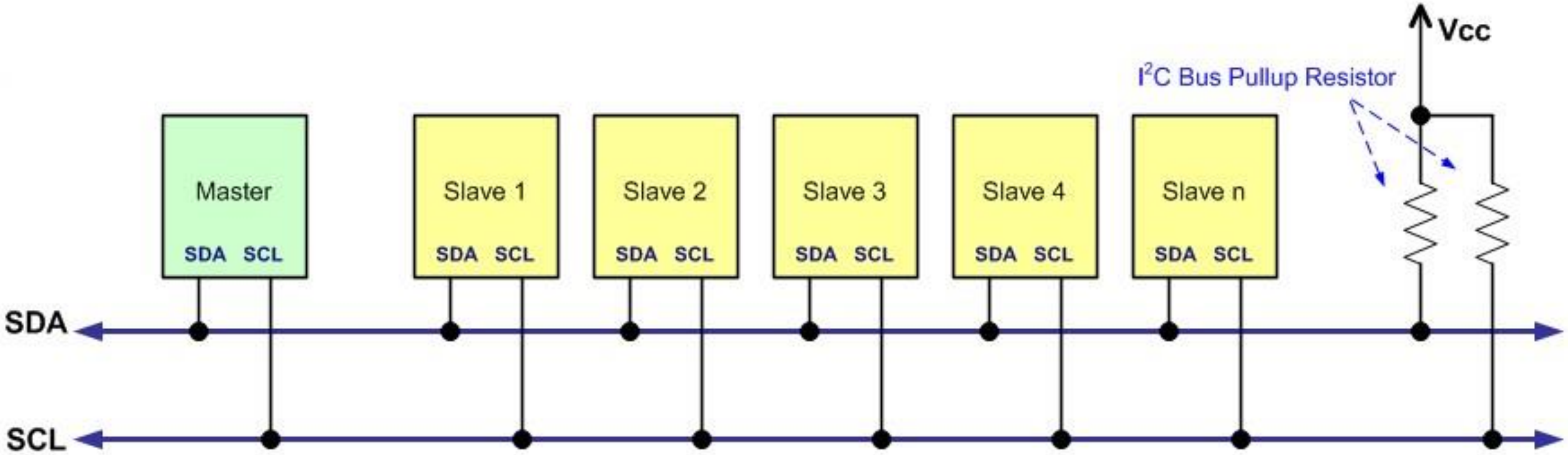
Controlling an I2C Slave Device from User Space

I2C in a Nutshell

katholieke hogeschool
associatie KU Leuven
vives

I2C in a Nutshell

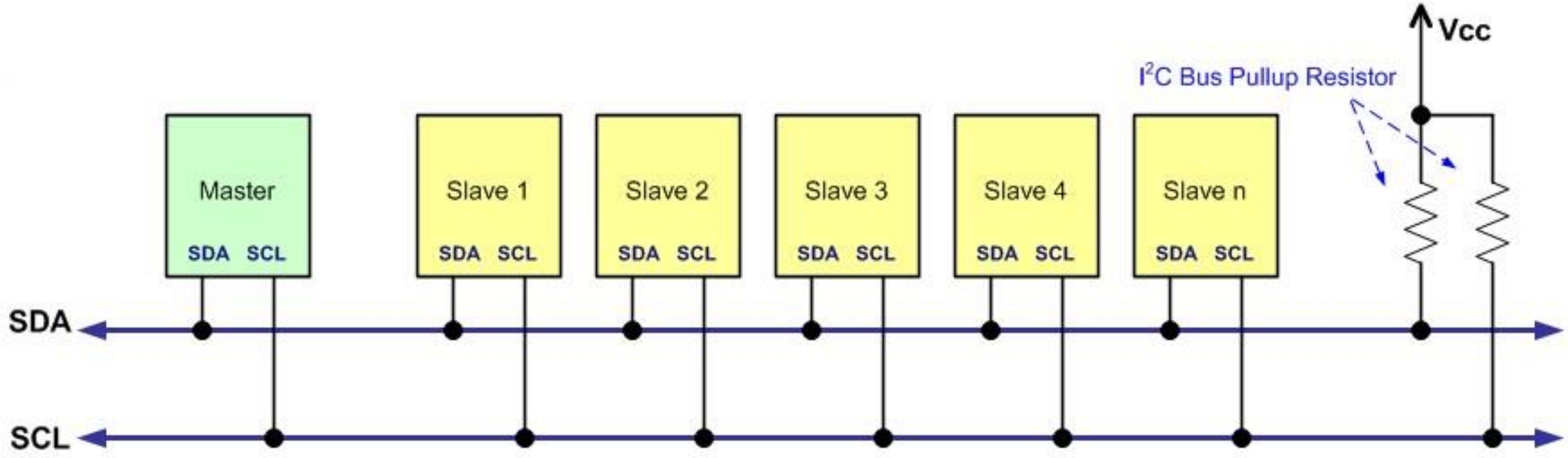
- Why I2C is commonly used to control external devices
  - Its a common standard
  - Its "fast" for low-speed devices
  - Bus architecture (multiple devices can be connected)
  - Easy to use
  - Wide support (most uC’s and processors have build in I2C peripheral)
  - Only 2 communication lines needed (SDA and SCL)



Typical I²C Device Connection

The Physical Bus

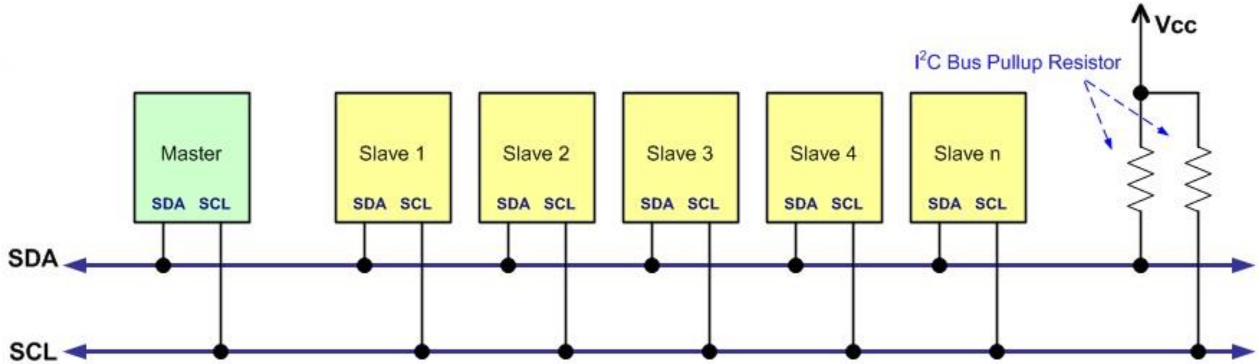
- The bus consists of just two wires, called SCL and SDA, and a ground
  - SCL is the clock line.
    - It is used to synchronize all data transfers over the I2C bus.
  - SDA is the data line.
  - The SCL and SDA lines are connected to all devices on the I2C bus.
- There does need to be a third wire which is the ground



Typical I^{2}C Device Connection

The Physical Bus

- Both SCL and SDA lines are "open drain" drivers.
  - What this means is that the chip can drive its output low, but it cannot drive it high.
  - For the line to be able to go high you must provide pull-up resistors to Vcc
  - There should be a resistor from the SCL line to Vcc and another from the SDA line to Vcc.
  - You only need one set of pull-up resistors for the whole I2C bus, not for each device.
- Vcc depends on the devices used. Typically 5V or 3V3



Typical I²C Device Connection

Masters and Slaves

- The devices on the I2C bus are either masters or slaves.
- The master is always the device that drives the SCL clock line
- The slaves are the devices that respond to the master.
- A slave cannot initiate a transfer over the I2C bus, only a master can do that.
- There can be, and usually are, multiple slaves on the I2C bus, however there is normally only one master.
  - It is possible to have multiple masters, but it is unusual.
- Slaves will never initiate a transfer.
  - Both master and slave can transfer data over the I2C bus, but that transfer is always controlled by the master.

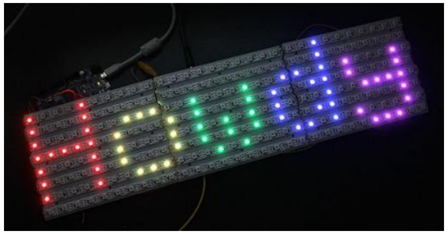
Controlling an I2C Slave Device from User Space

Our Goal

katholieke hogeschool
associatie KU Leuven
vives

Our Goal

- Connect the Raspberry Pi 2 to the mBed i2c NeoPixel driver
- Write a user space application and let it communicate with the slave device
  - We will need to cross-compile the source to the ARM architecture
- In this setup the Raspberry Pi will act as a master while the mbed NeoPixel driver is the slave



Controlling an I2C Slave Device from User Space

Enabling I2C on the Raspberry Pi 2

katholieke hogeschool
associatie KU Leuven

Step 1 - Enable i2c

- Use the raspi-config utility to enable i2c

```
pi@raspberrypi ~ $ sudo raspi-config
```

```
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
ââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââââ
```

- Select ‘Advanced Options’ => ‘I2C’ => ‘Enable’

Step 2 - Edit Module File

- Open the modules file

```
pi@raspberrypi ~ $ sudo nano /etc/modules
```

- And add the following to it

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

- This will make sure the i2c device modules are loaded when the kernel is booted

Step 3 - Install Utilities

- Install i2c-tools which has a bus discovery tool which is really handy

```
pi@raspberrypi ~ $ sudo apt-get update
pi@raspberrypi ~ $ sudo apt-get install i2c-tools
```

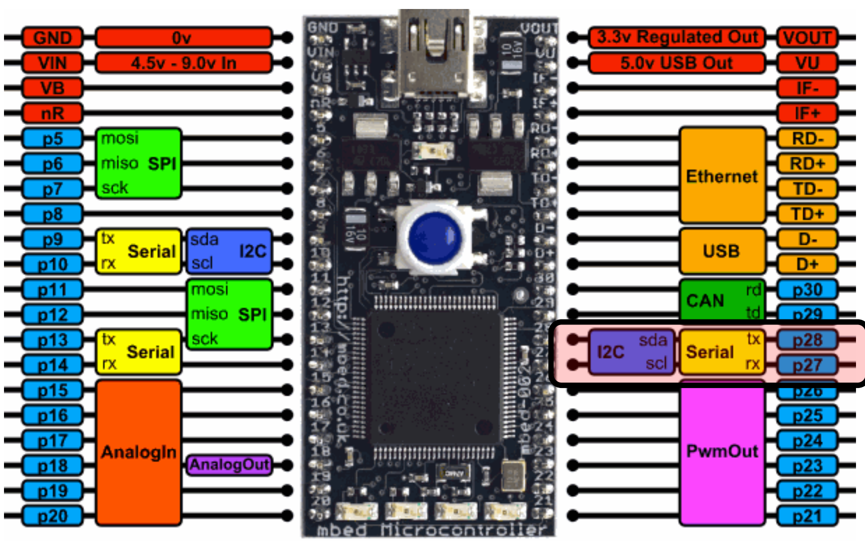
- Shutdown the Raspberry Pi

```
pi@raspberrypi ~ $ sudo halt
```

- Disconnect the power adapter and connect your i2c device

Step 4 - Connecting the i2c of mBed to the Raspberry Pi 2

- Raspberry Pi runs at 3.3V
- mBed runs at 3.3V
- So no level shifting is required
- We do not need to add pull-up resistors because the Raspberry Pi 2 already has pull-ups of 1k8 on each i2c line



Raspberry Pi2 GPIO Header

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I^2C)	DC Power 5v	04
05	GPIO03 (SCL1 , I^2C)	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 (I^2C ID EEPROM)	(I^2C 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
26/01/2014
http://www.element14.com

Step 5 - Scanning the bus using i2cdetect

- Once the two embedded boards are connected using i2c you can use the i2cdetect tool to scan the bus for slave devices
- For we need to check "/dev" for available i2c busses

```
pi@raspberrypi ~ $ cd /dev
pi@raspberrypi ~ $ ls i2c-*
```

- Look for "i2c-x" where x is a number

- Use the i2cdetect tool to scan the bus and replace x with the number of the actual device bus

```
pi@raspberrypi ~ $ i2cdetect -r x
```

Step 5 - Scanning the bus using i2cdetect

- Example

```
pi@raspberrypi ~ $ sudo i2cdetect -r 1
```

- You should get similar output when the mBed is connected

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- 48 -- -- -- 4C -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

- 0x40 is the NeoPixel slave. Can you explain the other two devices ?

7-bit or 8-bit address ?

- Also notice that the mBed uses the 8-bit address (R/W) LSB included while Linux uses the 7-bit address
  - You will need to use the 7-bit address in you C++ program !

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: 40 -- -- -- -- -- -- -- 48 -- -- -- 4C -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

- Part from the mBed slave code:

```
I2cSettings settings;
settings.frequency = 100000;
settings.address = 0x80;
```

Controlling an I2C Slave Device from User Space

The mBed NeoPixel Driver Slave

katholieke hogeschool
associatie KU Leuven

The mBed NeoPixel Driver Slave

- The mBed controls a NeoPixel string of RGB LED's
  - The size of a string and number of strings can be configured in the slave code
    - A string is a continuous connection of NeoPixel LED's

- Slave code is available on mBed as library
  - Import the library 'NeoPixelI2cDriver'
  - Search for 'neopixel'

The mBed NeoPixel Driver Slave

- The NeoPixel data line needs to be connected to the SPI pin (pin 5) on the mBed
  - Everything can be tested without NeoPixel string (debug info is outputted to console)
- To get some console output from the slave device you can use a terminal program such as Putty to connect to the serial interface over USB at a speed of 115200 baud.
- The slave device also has an alive LED which will blink periodically as long as the device is operational and responsive.
- The I2C bus operates at 100kHz and the slave device address is 0x80 (0x40 7-bit).

The mBed NeoPixel Driver Slave - main.cpp

https://developer.mbed.org/users/dwini/code/NeoPixelI2cSlave/

```
#include "mbed.h"
#include "neopixel_string.h"
#include "i2c_device.h"
#include "neopixel_string_factory.h"
#include "neopixel_i2c_daemon.h"

// This must be an SPI MOSI pin.
#define DATA_PIN p5
#define STRING_SIZE 8

#define DEBUG_MODE 1
#include "log.h"

Serial pc(USBTX, USBRX); // tx, rx
```

The mBed NeoPixel Driver Slave - main.cpp

https://developer.mbed.org/users/dwini/code/NeoPixelI2cSlave/

```
int main() {
    pc.baud(115200);
    SimplyLog::Log::i("Neopixel driver loading\r\n");

    I2cSettings settings;
    settings.frequency = 100000;
    settings.address = 0x80;

    SimplyLog::Log::i("Slave is working @ %dHz\r\n", settings.frequency);
    SimplyLog::Log::i("Slave is working @ SLAVE_ADDRESS = 0x%x\r\n", settings.address);

    SimplyLog::Log::i("Creating NeoPixel String\r\n");
    NeoPixelString * first_string = NeoPixelStringFactory::createNeoPixelString(DATA_PIN, STRING_SIZE);

    SimplyLog::Log::i("Creating I2cDevice\r\n");
    I2cDevice i2c(p28, p27, &settings);

    SimplyLog::Log::i("Creating NeoPixel I2c Daemon\r\n");
    NeoPixelI2cDaemon neo(&i2c, LED1);
    neo.attachPixelString(first_string);

    SimplyLog::Log::i("Listening in blocking mode\r\n");
    neo.listen(true);

    while(1) { }
}
```

The mBed NeoPixel Driver Slave - Possible Transactions

- @ the moment there are three commands that can be send to the slave
  - OFF (0x01) which will turn off all the LED's
  - DIAGNOSTIC (0x02) which will run a pre-programmed diagnostic routine
  - SINGLE_COLOR (0x03) which allows all LED's to be programmed with a given color

- OFF and DIAGNOSTIC and single byte transactions
- SINGLE_COLOR is a four-byte transactions
  - First byte is the command (0x03)
  - Following three bytes are the RGB values (0 - 255 or 0x00 to 0xFF)

The mBed NeoPixel Driver Slave - Possible Transactions

- The master can also request some information from the slave by sending a read request
  - @ the moment the slave will answer with a single byte value, namely the number of strings attached to the mBed (which will be 1).

Controlling an I2C Slave Device from User Space

Raspberry Pi 2 User Space Master Program

katholieke hogeschool
associatie KU Leuven
vives

Start from the Base

- Some decent information can be found @ http://elinux.org/Interfacing_with_I2C_Devices

- You can start from the example program that I created
  - Not very OOP, that is YOUR task

- See GitHub:
  - https://github.com/BioBoost/pi_i2c_master_neopixel_driver.git

Controlling an I2C Slave Device from User Space

Assignment

katholieke hogeschool vives
associatie KU Leuven

Assignment

- Create a C++ program to control the mBed slave device from User Space
  - Use OOP !!
    - No single class or 400 lines main function programs
      - This will get you flunked !
      - Prove what you have learned and what you are capable off !
- This needs to be incorporated in the final program of the group

- Requirements
  - Fading (from bright to dark and vice versa) or Stroboscope (with adjustable timings)
  - Set colors (per LED not just one color for the whole string)

- You can extend the slave program as you seem fit