F28HS Hardware-Software Interface: Systems Programming

Hans-Wolfgang Loidl

School of Mathematical and Computer Sciences,
Heriot-Watt University, Edinburgh

Semester 2 — 2018/19

[0] No proprietary software has been used in producing these slides

Outline

Lecture 4.
Programming external devices

Basics of the I^{2}C interface

- So far we always used the GPIO interface to directly connect external devices.
- This is the easiest interface to use.
- It is however limited in the number of connections and devices you can connect with.
- A more general interface is the **I^{2}C interface** or the **I^{2}C bus**.

[0] Based on the article The I^{2}C-bus of the Raspberry Pi (Der I^{2}C-Bus des Raspberry Pi) (in German), Raspberry Pi Geek 01/15

Basics of the I^{2}C interface

- I^{2}C is a serial master-slave bus.
- It is serial, i. e.communication is one bit at a time.
- It allows to connect several masters (data-providers) with several slaves (data-consumers)
- It is designed for short-distance communication, i. e.communication on a board
- Therefore it is also used in the standard Linux kernel to monitor, e. g.temperature and other system health information
- I^{2}C was originally developed by Philips in the 1980s, and has become an industry standard.

Technical detail on I^{2}C

- Communication uses 2 connections:
  - a serial data line (**SDA**)
  - a serial clock line (**SCL**) for synchronising the communication
- Both connections use pull-up resistors to encode one bit (high potential = **1**)
- The two sides of the communication are
  - a **master** that sends the clock information and initiates communication
  - a **slave** that receives the data
- Typical communication rates are between 100 kb/s (standard mode) and 5 Mb/s (ultra fast mode)
- **NB:** I^{2}C was **not** designed for communicating large volumes of data

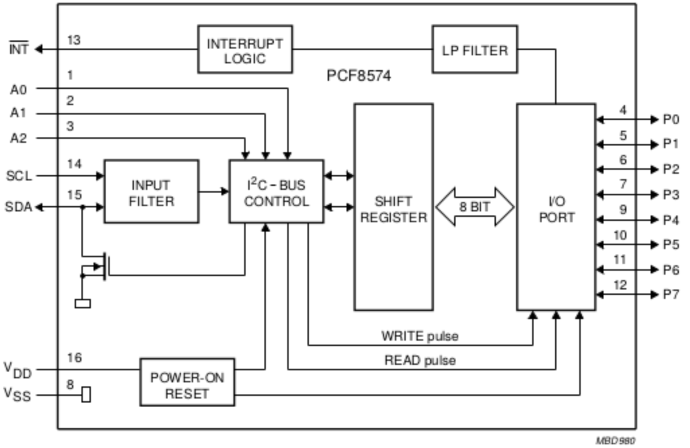
Technical detail on I^{2}C

- I^{2}C uses a 7-bit address space, i. e.128 possible addresses of which 16 are reserved.
- The 8-th bit indicates the direction of the data transfer between master and slave.
- The usable address-space is defined in the technical documentation of the device. E. g.

PCF8574	Port-Expander	0x20 – 0x27
PCF8583	Clock/Calendar	0xA0 – 0xA2

- The device PCF8583 is a chip that provides an external clock, with three registers starting at `0xA0`
- As an example we will now use the PCF8574 port-expander, which is accessed through address `0x20`.
- This can be used to e. g. control an LCD display over just one data channel.

Block Diagram of the PCF8574 Port Expander



NB: 1 input data channel (**SDA**), 8 output data channels (**P0** . . . **P7**)

[0]From PCF8574 Data Sheet

What’s happening on the wires?

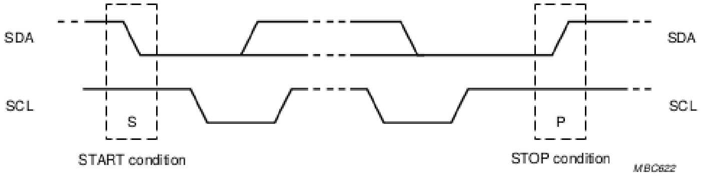


Fig.6 Definition of start and stop conditions.

- signals start with `HIGH`
- a change in the SDA signal, with SCL `HIGH`, indicates start/stop

[0] From PCF8574 Data Sheet

How are the bits transferred?

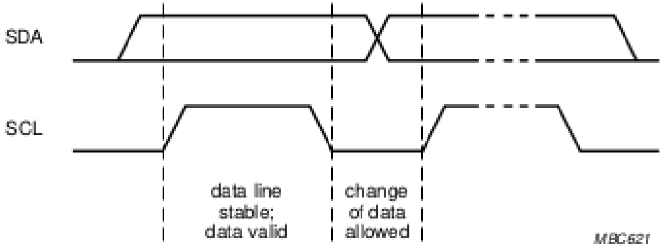


Fig.5 Bit transfer.

- one bit is transferred during each clock pulse
- data is sampled while the SCL line is `HIGH`
- the SDA line needs to be stable during this `HIGH` period

A typical system configuration using I2C

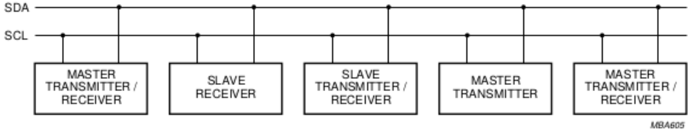

SDA
SCL
MASTER TRANSMITTER / RECEIVER
SLAVE RECEIVER
SLAVE TRANSMITTER / RECEIVER
MASTER TRANSMITTER
MASTER TRANSMITTER / RECEIVER
MBA605

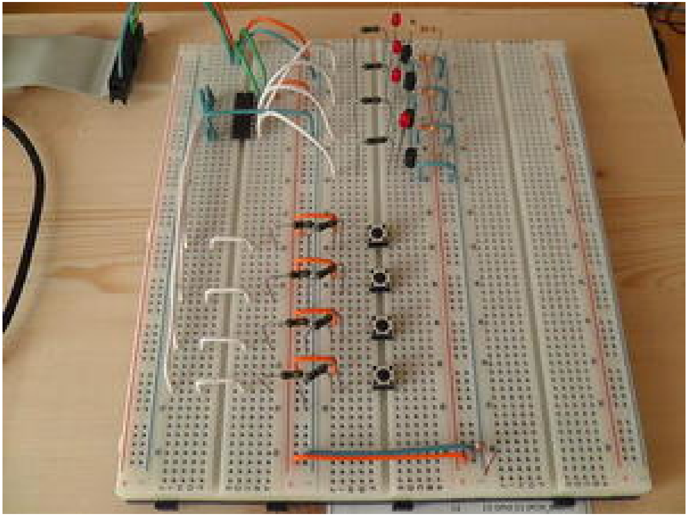
Fig.7 System configuration.

- lines are (quasi-)bidirectional
- a device generating a message is a “transmitter”
- a device receiving is the “receiver”
- the controller of the message is the “master”
- the receivers of the message are the “slaves”

I^{2}C on the Raspberry Pi 2

- On the RPi2 the following pins provide an I^{2}C interface: physical Pin 03 (**SDA**) and Pin 05 (**SCL**) (these are pins 2 and 4 in the BCM numbering)
- In the following example we will use these pins to connect a PCF8574 device.
- In our configuration we connect the device with four buttons and LEDs as shown in the picture below.

Test configuration



[0] From The I^{2}C-bus of the Raspberry Pi (Der I^{2}C-Bus des Raspberry Pi) (in German), Raspberry Pi Geek 01/15

Software configuration

- We use the `wiringPi` library that we have installed and discussed before.

- We also need the `i2c-tools` package for the drivers communicating over the I^{2}C bus
- To install `i2c-tools` do the following:

```
> sudo apt-get install i2c-tools
> sudo adduser pi i2c
> gpio load i2c
```

- We can now use `i2cdetect` to check the connection between our RPi2 and the external device:

```
> i2cdetect -y 1
```

- This shows that we can reach the device through address `0x20`
- The 4 high-bits in that address refer to the LEDs, the 4 low-bits refer to the buttons

Software configuration

- Initially all lines are at high, so all LEDs should light up
- To turn LEDs off, one-by-one we execute:

```
> i2cset -y 1 0x20 0x00
> i2cset -y 1 0x20 0x10
> i2cset -y 1 0x20 0x20
> i2cset -y 1 0x20 0x40
> i2cset -y 1 0x20 0x80
```

- Now we want to configure the button as an input device:

```
> i2cset -y 1 0x20 0x0f
> watch 'i2cget -y 1 0x20'
```

- Using `watch` we continously get output about the current value issued by the button
- Pressing the button will change the observed value

A C API for I^2C

- Now we want to use the I^2C-bus to programmatically control external devices
- We use the following API provided by Gordon Henderson's wiringPi library:

```
int wiringPiI2CSetup (const int devId)
```
  Open the I2C device, and regsiter the target device
```
int wiringPiI2CRead (int fd)
```
  Simple device read
```
int wiringPiI2CWrite (int fd, int data)
```
  Simple device write
```
int wiringPiI2CReadReg8 (int fd, int reg)
```
  Read an 8-bit value from a regsiter on the device
```
int wiringPiI2CWriteReg8 (int fd, int reg, int val
```
  Write a 8-bit value to the given register

and similar read/write interface for 16-bit values.

Sample Source for I^{2}C

Using this interface we can make the LEDs blink one-by-one:

```
#include <wiringPiI2C.h>
int main(void) {
  int handle = wiringPiI2CSetup(0x20) ;
  wiringPiI2CWrite(handle, 0x10);
  delay(5000);
  wiringPiI2CWrite(handle, 0x20);
  delay(5000);
  wiringPiI2CWrite(handle, 0x40);
  delay(5000);
  wiringPiI2CWrite(handle, 0x80);
  delay(5000);
  wiringPiI2CWrite(handle, 0x00);
  return 0;
}
```

NB: We access the LEDs as a bitmask on the high 4-bits, setting the low 4-bits to zero in each case.

Further Reading & Hacking

- The I^2C-bus of the Raspberry Pi (Der I^2C-Bus des Raspberry Pi) (in German), Raspberry Pi Geek 01/15
- Data sheet of the PCF8574 port-expander
- I^2CTutorial
- Configuring I^2C, SMBus on Raspbian Linux
- Using wiringPi on the PCF8574
- Using an PCF8574 to control an LCD display
- Another guide how to use an PCF8574 to control an LCD display