|  |
| --- |
| Embedded Systems Project |
| Visualizing measurements of a smoke sensor on different displays |

|  |
| --- |
| Soufiane Salama & Roy Aarts  [Datum] |

# The project

As assignment for the course Embedded Systems we have chosen to make a system where a Raspberry Pi is connected to two different kinds of displays and to a gas/smoke sensor. The purpose is to poll data of the sensor and visualize it on the displays.

# MQ-2 (SPI)

# GSM Module

# LCD 16\*2 (I2C)

The second display we used is the well-known 16\*2 LCD display (HD44780U). The one we used has the PCF8574AT I2C driver connected to it.

### I2C Protocol

### Wiring Scheme

### HD44780U Registers

On the datasheet of the HD44780u…

But the I2C driver is not connected to all the pins of the LCD, this means we need to program the LCD by 4-bit and thus we always need to send data two times, one time for DB7-DB4 and after for DB3-DB0. Also it’s necessary to have a rising and falling edge on the CS (Enable) pin, so first CS=0, then CS=1 and lasts CS=0.

Practical steps:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Connection on PCF8574AT** | **P7** | **P6** | **P5** | **P4** | **P3** | **P2** | **P1** | **P0** |
| **Connection on HD44780U** | **DB7** | **DB6** | **DB5** | **DB4** | **Backlight** | **CS** | **RW** | **RS** |
|  |  |  |  |  |  |  |  |  |
| Initialize (Function Set) HIGH BITS  **0x33 -> 0x38** | **0** | **0** | **1** | **1** | **1** | **0/1/0** | **0** | **0** |
| Initialize (Function Set) LOW BITS  **0x33 -> 0x38** | **0** | **0** | **1** | **1** | **1** | **0/1/0** | **0** | **0** |
|  |  |  |  |  |  |  |  |  |
| Initialize (Function Set) HIGH BITS  **0x32 -> 0x38** | **0** | **0** | **1** | **1** | **1** | **0/1/0** | **0** | **0** |
| Initialize (Function Set) LOW BITS  **0x32 -> 0x28** | **0** | **0** | **1** | **1** | **1** | **0/1/0** | **0** | **0** |
|  |  |  |  |  |  |  |  |  |
| Display On, Cursor Off, Blink Off HIGH BITS  **0x0C -> 0x08** | **0** | **0** | **1** | **1** | **1** | **0/1/0** | **0** | **0** |
| Display On, Cursor Off, Blink Off HIGH BITS  **0x0C -> 0xC8** | **0** | **0** | **1** | **1** | **1** | **0/1/0** | **0** | **0** |
|  |  |  |  |  |  |  |  |  |
| Function Set (2-lines, 5\*8 Dots) HIGH BITS  **0x28 -> 0x28** |  |  |  |  |  |  |  |  |
| Function Set (2-lines, 5\*8 Dots) LOW BITS  **0x28 -> 0x88** |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Clear Display HIGH BITS  **0x01 -> 0x08** | **0** | **0** | **0** | **0** | **1** | **0/1/0** | **0** | **0** |
| Clear Display LOW BITS  **0x01 -> 0x18** | **0** | **0** | **0** | **1** | **1** | **0/1/0** | **0** | **0** |

### Raspberry Pi Registers

To make use of the I2C protocol we had to make use of one of the three available BSC registers. I2C makes use of six registers (page 28):

* Control register (C), “*is used to enable interrupts, clear the FIFO, define a read or write operation and start a transfer”.* This register has seven fields: I2C Enable, Interrupt on RX, Interrupt on TX, Interrupt on Done, Start Transfer, FIFO Clear and Read Transfer
* Status register (S): “*is used to record activity status, errors and interrupt requests*”. This register has
* Data Length register (DLEN)
* FIFO register (DATA)
* Clock Divider register (CDIV)

Practical steps:

1. Set the I2C slave address of the LCD display, in this case it was “0x3f”.

i2c.base[I2C\_A] = 0x3f;

1. Set the two GPIO pins, pin 2 for SDA and pin 3 for SCL, as Alternative Function 0

gpio\_set\_alt(2, 0);

gpio\_set\_alt(3, 0);

1. Set the Data Length Register on the number of bytes you want to send. In our case it is 1.

i2c.base[I2C\_DLEN] |= 1;

1. Then if you want to send a byte, you need to put it in the FIFO.

i2c.base[I2C\_FIFO] |= send\_value;

1. Before sending the data that we put in the FIFO, we need to clear three fields in the Status register that we will check on errors after transferring the data. These are the CLKT, ERR and DONE fields. We can clear these fields by writing a 1 to them.

We have done this by adding following line to our defines.

#define I2C\_CLEAR\_STATUS I2C\_CLKT|I2C\_ERR|I2C\_DONE

In our program we can start the clear by the following line.

i2c.base[I2C\_S] |= I2C\_CLEAR\_STATUS;

1. To send the data that is in the FIFO, we need to enable I2C and start the transfer by writing a 1 to fields I2CEN and ST of the Control register. We have implement this the same as previous step.

#define I2C\_START\_WRITE I2C\_I2CEN|I2C\_ST

i2c.base[I2C\_C] |= I2C\_START\_WRITE; *// Start Write (see #define)*

1. Last step is to check on errors. We do this by checking the fields that we cleared at step 5.

while(!((i2c.base[I2C\_S]) & i2c.base[I2C\_DONE])) {

systim\_waitus(5000);

}