I2C LED Matrix

Generated by Doxygen 1.5.7.1

Wed May 6 20:07:40 2009

Contents

1 I2C LED Matrix **1**

1.1 Introduction . . . 1

1.2 Purpose . . . 1

1.3 I2C communication . . . 1

1.3.1 Usage . . . 2

1.3.2 Displaying numbers . . . 2

1.4 Building and installing . . . 3

1.5 Drawbacks . . . 3

1.6 Files in the distribution . . . 3

1.7 Thanks! . . . 3

1.8 About the license . . . 4

2 File Index **5**

2.1 File List . . . 5

3 File Documentation **7**

3.1 main.c File Reference . . . 7

3.1.1 Detailed Description . . . 7

3.1.2 Function Documentation . . . 8

3.1.2.1 init_ports . . . 8

3.1.2.2 main . . . 8

3.1.2.3 selectDigit . . . 8

3.1.2.4 showByte . . . 9

3.1.2.5 showDigitByte . . . 9

3.2 twislave.c File Reference . . . 10

3.2.1 Detailed Description . . . 10

3.2.2 Define Documentation . . . 11

3.2.2.1 TWCR_ACK . . . 11

3.2.2.2 TWCR_NACK . . . 11

3.2.2.3 TWCR_RESET 11
3.2.3 Function Documentation 11
3.2.3.1 init_twi_slave 11
3.2.3.2 ISR 11
3.2.4 Variable Documentation 12
3.2.4.1 buffer_adr 12
3.3 twislave.h File Reference 13
3.3.1 Detailed Description 13
3.3.2 Define Documentation 14
3.3.2.1 buffer_size 14
3.3.3 Function Documentation 14
3.3.3.1 init_twi_slave 14
3.3.4 Variable Documentation 15
3.3.4.1 rxbuffer 15
3.3.4.2 txbuffer 15

Chapter 1

I2C LED Matrix

1.1 Introduction

This project turns an AVR ATmega8 microcontroller into a LED controller for a matrix of 8x8 LEDs. The controller is acting as I2C-slave, so you can control the patterns to display via this bus (also known as TWI, Two Wire Interface).

1.2 Purpose

For my next project, I need to display number values on seven-segment-displays. I bought a bunch of 4-digit-displays a while ago, now I'm going to put them to a use. They are built with four digits in one case, and 12 pins on the underside. Eight of them are the cathodes of the segments (seven segments plus dot), four are the anodes. One for each digit.

You can imagine these modules as a matrix of four times eight LEDs, as can be seen in the included circuit. I use two of these, so I have a matrix of eight times eight LEDs.

The rows and columns of this matrix are connected to the microcontroller, so it can power them row by row. This has two advantages: at first a maximum of eight LEDs is powered at a time, so power consumption is lowered. And at second you need only 16 pins of the controller to address a total of 64 LEDs.

Driving the LEDs in this way makes them flicker a bit, but the controller is fast enough to keep the flickering way above the level you would be able to recognize.

I could have connected my display modules directly to the main controller of my next project, but I don't have enough free pins on that. As a further benefit, multiplexing the LEDs on a second controller makes the main program easier to write, since I don't have to mind the timing. So the solution is to use a cheap ATmega8 as LED driver and use the I2C-bus to tell it what to display.

1.3 I2C communication

The ATmega8 has a built-in hardware I2C-interface, so it doesn't take very much code to use it. Nevertheless, I used a little library that **Uwe Grosse-Wortmann (uwegw)** published on roboternetz.de. I only reformatted it a bit to make the code resemble my style. It is well commented, but the comments are in german. Since only one global array, one init-function and an interrupt service routine are used, it shouldn't be too hard for english-speaking people to figure out how it is used.

1.3.1 Usage

On the other end of the communication, I used the excellent **Procyon AVRlib written by Pascal Stang**. You can find it here.

A basic code example would look like this:

```
#define I2C_LEDMATRIX 0x10   // address of the device
timerInit();                 // initialize timers
timerPause(100);             // give everything a little time to settle
i2cInit();                   // initialize i2c function library
timerPause(100);             // wait a bit more
while (1) {                  // endless loop
    uint8_t buffer[9];       // prepare buffer
                             // loop until 255
    for (uint8_t i = 0; i <= 255; i++) {
                             // set all bytes of the buffer to value i
        memset(buffer, i, sizeof(buffer));
                             // send the buffer via I2C-bus
        i2cMasterSend(I2C_LEDMATRIX, sizeof(buffer), buffer);
        timerPause(500);     // wait, so you have the time to watch
    }
}
```

Note: the buffer doesn't contain any numbers that should be displayed on 7segment-displays. At least not in this example. It only holds bit-patterns.

1.3.2 Displaying numbers

If you solder 7segment displays to the unit and intend to display numbers or characters on it, you need to define them on the master-side of the bus. I didn't include the definitions in this library because I want the master to have the full flexibility of displaying whatever it wants to, even if it are no numbers.

However, if you are going to use 7segment displays, definition of the numbers still depends on how you soldered them to the controller. I don't know if the pin-outs are commonly standardized.

To give an example of how you would implement this, here is a fragment of code that defines hexadecimal numbers for usage on my displays:

```
// Names of the segments:
//      aaaaa
//     f     b
//     f     b
//      ggggg
//     e     c
//     e     c
//      ddddd  h
uint8_t characters[16];
//               c          e          g          a          h          f          b          d
characters[ 0] = (1 << 0) | (1 << 1) | (0 << 2) | (1 << 3) | (0 << 4) | (1 << 5) | (1 << 6) | (1 << 7); /
characters[ 1] = (1 << 0) | (0 << 1) | (0 << 2) | (0 << 3) | (0 << 4) | (0 << 5) | (1 << 6) | (0 << 7); /
characters[ 2] = (0 << 0) | (1 << 1) | (1 << 2) | (1 << 3) | (0 << 4) | (0 << 5) | (1 << 6) | (1 << 7); /
characters[ 3] = (1 << 0) | (0 << 1) | (1 << 2) | (1 << 3) | (0 << 4) | (0 << 5) | (1 << 6) | (1 << 7); /
characters[ 4] = (1 << 0) | (0 << 1) | (1 << 2) | (0 << 3) | (0 << 4) | (1 << 5) | (1 << 6) | (0 << 7); /
characters[ 5] = (1 << 0) | (0 << 1) | (1 << 2) | (1 << 3) | (0 << 4) | (1 << 5) | (0 << 6) | (1 << 7); /
characters[ 6] = (1 << 0) | (1 << 1) | (1 << 2) | (1 << 3) | (0 << 4) | (1 << 5) | (0 << 6) | (1 << 7); /
characters[ 7] = (1 << 0) | (0 << 1) | (0 << 2) | (1 << 3) | (0 << 4) | (0 << 5) | (1 << 6) | (0 << 7); /
characters[ 8] = (1 << 0) | (1 << 1) | (1 << 2) | (1 << 3) | (0 << 4) | (1 << 5) | (1 << 6) | (1 << 7); /
characters[ 9] = (1 << 0) | (0 << 1) | (1 << 2) | (1 << 3) | (0 << 4) | (1 << 5) | (1 << 6) | (1 << 7); /
characters[10] = (1 << 0) | (1 << 1) | (1 << 2) | (1 << 3) | (0 << 4) | (1 << 5) | (1 << 6) | (0 << 7); /
characters[11] = (1 << 0) | (1 << 1) | (1 << 2) | (0 << 3) | (0 << 4) | (1 << 5) | (0 << 6) | (1 << 7); /
characters[12] = (0 << 0) | (1 << 1) | (0 << 2) | (1 << 3) | (0 << 4) | (1 << 5) | (0 << 6) | (1 << 7); /
```

```
characters[13] = (1 << 0) | (1 << 1) | (1 << 2) | (0 << 3) | (0 << 4) | (0 << 5) | (1 << 6) | (1 << 7); /
characters[14] = (0 << 0) | (1 << 1) | (1 << 2) | (1 << 3) | (0 << 4) | (1 << 5) | (0 << 6) | (1 << 7); /
characters[15] = (0 << 0) | (1 << 1) | (1 << 2) | (1 << 3) | (0 << 4) | (1 << 5) | (0 << 6) | (0 << 7); /
```

1.4 Building and installing

The firmware is built and installed on the controller with the included makefile. You might need to need to customize it to match your individual environment.

If you take a brand-new controller you shouldn't have to hassle with the fuses of the controller. The internal oscillator at 1MHz is enough to keep the display flicker-free. The settings I used are included in the makefile, so you can use it to reset controllers you already changed in other projects.

Oh, and if you want the slave to use an I2C-address different from 0x10: no problem. Just change it in the code.

1.5 Drawbacks

Till now, the device worked in all situations I tested it in. So far everything is fine.

1.6 Files in the distribution

- *Readme.txt:* Documentation, created from the htmldoc-directory.
- *htmldoc/*: Documentation, created from main.c.
- *refman.pdf:* Documentation, created from main.c.
- *main.c:* Source code of the firmware.
- *main_*.hex: Compiled version of the firmware.
- *twislave.c:* I2C-library.
- *twislave.h:* I2C-library.
- *project.doxygen:* Support for creating the documentation.
- *License.txt:* Public license for all contents of this project.
- *Changelog.txt:* Logfile documenting changes in soft-, firm- and hardware.

1.7 Thanks!

I'd like to thank the authors of the libraries I used: **Uwe Grosse-Wortmann (uwegw)** for the I2C-slave and **Pascal Stang** for the Procyon AVRlib.

1.8 About the license

My work is licensed under the GNU General Public License (GPL). A copy of the GPL is included in License.txt.

(c) 2008 by Ronald Schaten - http://www.schatenseite.de

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

main.c (Firmware for the i2c-ledmatrix) 7
twislave.c (I2C slave library) 10
twislave.h (I2C slave library) 13

Chapter 3

File Documentation

3.1 main.c File Reference

firmware for the i2c-ledmatrix

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>
#include <util/delay.h>
#include <avr/pgmspace.h>
#include "twislave.h"
```

Functions

- void init_ports (void)
  initialize hardware
- void selectDigit (uint8_t digit)
  select which digit should be displayed
- void showByte (uint8_t byte)
  set output of the currently selected digit
- void showDigitByte (uint8_t digit, uint8_t byte)
  show a pattern on a certain digit (or row, if you don't use 7segment displays).
- int main (void)
  main-function.

3.1.1 Detailed Description

firmware for the i2c-ledmatrix

this is a really simple piece of code, since the main work is done by the I2C-library.

Author:

Ronald Schaten <ronald@schatenseite.de>

Version:

Id

main.c,v 1.1 2008/07/16 05:44:45 rschaten Exp

Definition in file main.c.

3.1.2 Function Documentation

3.1.2.1 void init_ports (void)

initialize hardware

Definition at line 200 of file main.c.

Referenced by main().

3.1.2.2 int main (void)

main-function.

initializes everything and contains the main loop which controls the actual output. the rxbuffer[] is filled from the I2C-library, so we just have go through the array and display its values on the corresponding digit.

Returns:

An integer. Whatever... :-)

Definition at line 271 of file main.c.

References init_ports(), init_twi_slave(), rxbuffer, and showDigitByte().

3.1.2.3 void selectDigit (uint8_t *digit*)

select which digit should be displayed

Parameters:

digit number of the digit

Definition at line 216 of file main.c.

Referenced by showDigitByte().

3.1.2.4 void showByte (uint8_t *byte*)

set output of the currently selected digit

Parameters:

byte bit-pattern to show

Definition at line 247 of file main.c.

Referenced by showDigitByte().

3.1.2.5 void showDigitByte (uint8_t *digit*, uint8_t *byte*)

show a pattern on a certain digit (or row, if you don't use 7segment displays).

the output is cleared before selecting the new digit, so there won't be 'shadows' on the display.

Parameters:

digit number of the digit

byte bit-pattern to show

Definition at line 258 of file main.c.

References selectDigit(), and showByte().

Referenced by main().

3.2 twislave.c File Reference

I2C slave library.

```
#include <util/twi.h>
#include <avr/interrupt.h>
#include <stdint.h>
#include "twislave.h"
```

Defines

- #define TWCR_ACK TWCR = (1 << TWEN) | (1 << TWIE) | (1 << TWINT) | (1 << TWEA) | (0 << TWSTA) | (0 << TWSTO) | (0 << TWWC);
- #define TWCR_NACK TWCR = (1 << TWEN) | (1 << TWIE) | (1 << TWINT) | (0 << TWEA) | (0 << TWSTA) | (0 << TWSTO) | (0 << TWWC);
- #define TWCR_RESET TWCR = (1 << TWEN) | (1 << TWIE) | (1 << TWINT) | (1 << TWEA) | (0 << TWSTA) | (0 << TWSTO) | (0 << TWWC);

Functions

- void init_twi_slave (uint8_t adr)

  Initalisierung des TWI-Interface.

- ISR (TWI_vect)

  ISR, die bei einem Ereignis auf dem Bus ausgeloest wird.

Variables

- volatile uint8_t buffer_adr

3.2.1 Detailed Description

I2C slave library.

taken from http://www.roboternetz.de/wissen/index.php/TWI_Slave_mit_avr-gcc

Author:

Uwe Grosse-Wortmann (uwegw), reformatted by Ronald Schaten <ronald@schatenseite.de>

Version:

Id

twislave.c,v 1.1 2008/07/16 05:44:45 rschaten Exp

License: GNU GPL v2 (see License.txt)

Definition in file twislave.c.

3.2.2 Define Documentation

3.2.2.1 #define TWCR_ACK TWCR = (1 << TWEN) | (1 << TWIE) | (1 << TWINT) | (1 << TWEA) | (0 << TWSTA) | (0 << TWSTO) | (0 << TWWC);

Definition at line 49 of file twislave.c.

Referenced by ISR().

3.2.2.2 #define TWCR_NACK TWCR = (1 << TWEN) | (1 << TWIE) | (1 << TWINT) | (0 << TWEA) | (0 << TWSTA) | (0 << TWSTO) | (0 << TWWC);

Definition at line 51 of file twislave.c.

Referenced by ISR().

3.2.2.3 #define TWCR_RESET TWCR = (1 << TWEN) | (1 << TWIE) | (1 << TWINT) | (1 << TWEA) | (0 << TWSTA) | (0 << TWSTO) | (0 << TWWC);

Definition at line 53 of file twislave.c.

Referenced by ISR().

3.2.3 Function Documentation

3.2.3.1 void init_twi_slave (uint8_t *adr*)

Initalisierung des TWI-Interface.

Muss zu Beginn aufgerufen werden, sowie bei einem Wechsel der Slave Adresse.

Parameters:

adr gewuenschte Slave-Adresse

Definition at line 37 of file twislave.c.

References buffer_adr.

Referenced by main().

3.2.3.2 ISR (TWI_vect)

ISR, die bei einem Ereignis auf dem Bus ausgeloest wird.

Im Register TWSR befindet sich dann ein Statuscode, anhand dessen die Situation festgestellt werden kann.

Definition at line 63 of file twislave.c.

References buffer_adr, buffer_size, rxbuffer, TWCR_ACK, TWCR_NACK, TWCR_RESET, and txbuffer.

3.2.4 Variable Documentation

3.2.4.1 volatile uint8_t buffer_adr

Definition at line 30 of file twislave.c.

Referenced by init_twi_slave(), and ISR().

3.3 twislave.h File Reference

I2C slave library.

```
#include <util/twi.h>
#include <avr/interrupt.h>
#include <stdint.h>
```

Defines

- #define buffer_size 9

  Groesse der Buffer in Byte (2..254).

Functions

- void init_twi_slave (uint8_t adr)

  Initalisierung des TWI-Interface.

Variables

- volatile uint8_t rxbuffer [buffer_size]

  Der Buffer, in dem die empfangenen Daten gespeichert werden.

- volatile uint8_t txbuffer [buffer_size]

  Der Sendebuffer, der vom Master ausgelesen werden kann.

3.3.1 Detailed Description

I2C slave library.

taken from http://www.roboternetz.de/wissen/index.php/TWI_Slave_mit_-avr-gcc

Betrieb eines AVRs mit Hardware-TWI-Schnittstelle als Slave. Zu Beginn muss init_twi_slave mit der gewuenschten Slave-Adresse als Parameter aufgerufen werden. Der Datenaustausch mit dem Master erfolgt ueber die Buffer rxbuffer und txbuffer, auf die von Master und Slave zugegriffen werden kann. rxbuffer und txbuffer sind globale Variablen (Array aus uint8_t). Die Ansteuerung des rxbuffers, in den der Master schreiben kann, erfolgt aehnlich wie bei einem normalen I2C-EEPROM. Man sendet zunaechst die Bufferposition, an die man schreiben will, und dann die Daten. Die Bufferposition wird automatisch hochgezaehlt, sodass man mehrere Datenbytes hintereinander schreiben kann, ohne jedesmal die Bufferadresse zu schreiben. Um den txbuffer vom Master aus zu lesen, uebertraegt man zunaechst in einem Schreibzugriff die gewuenschte Bufferposition und liest dann nach einem repeated start die Daten aus. Die Bufferposition wird automatisch hochgezaehlt, sodass man mehrere Datenbytes hintereinander lesen kann, ohne jedesmal die Bufferposition zu schreiben.

Autor: Uwe Grosse-Wortmann (uwegw) Status: Testphase, keine Garantie fuer ordnungsgemaesse Funktion! letze Aenderungen: 23.03.07 Makros fuer TWCR eingefuegt. Abbruch des Sendens, wenn der

TXbuffer komplett gesendet wurde. 24.03.07 verbotene Buffergroessen abgefangen 25.03.07 noetige externe Bibliotheken eingebunden

Abgefangene Fehlbedienung durch den Master:

- Lesen ueber die Grenze des txbuffers hinaus
- Schreiben ueber die Grenzen des rxbuffers hinaus
- Angabe einer ungueltigen Schreib/Lese-Adresse
- Lesezugriff, ohne vorher Leseadresse geschrieben zu haben

Author:

Uwe Grosse-Wortmann (uwegw), reformatted by Ronald Schaten <ronald@schatenseite.de>

Version:

Id

twislave.h,v 1.1 2008/07/16 05:44:45 rschaten Exp

License: GNU GPL v2 (see License.txt)

Definition in file twislave.h.

3.3.2 Define Documentation

3.3.2.1 #define buffer_size 9

Groesse der Buffer in Byte (2..254).

Definition at line 52 of file twislave.h.

Referenced by ISR().

3.3.3 Function Documentation

3.3.3.1 void init_twi_slave (uint8_t *adr*)

Initalisierung des TWI-Interface.

Muss zu Beginn aufgerufen werden, sowie bei einem Wechsel der Slave Adresse.

Parameters:

adr gewuenschte Slave-Adresse

Definition at line 37 of file twislave.c.

References buffer_adr.

Referenced by main().

3.3.4 Variable Documentation

3.3.4.1 volatile uint8_t rxbuffer[buffer_size]

Der Buffer, in dem die empfangenen Daten gespeichert werden.

Der Slave funktioniert aehnlich wie ein normales Speicher-IC [I2C-EEPROM], man sendet die Adresse, an die man schreiben will, dann die Daten, die interne Speicher-Adresse wird dabei automatisch hochgezaehlt.

Definition at line 61 of file twislave.h.

Referenced by ISR(), and main().

3.3.4.2 volatile uint8_t txbuffer[buffer_size]

Der Sendebuffer, der vom Master ausgelesen werden kann.

Definition at line 66 of file twislave.h.

Referenced by ISR().

Index

buffer_adr
    twislave.c, 12
buffer_size
    twislave.h, 14

init_ports
    main.c, 8
init_twi_slave
    twislave.c, 11
    twislave.h, 14
ISR
    twislave.c, 11

main
    main.c, 8
main.c, 7
    init_ports, 8
    main, 8
    selectDigit, 8
    showByte, 8
    showDigitByte, 9

rxbuffer
    twislave.h, 15

selectDigit
    main.c, 8
showByte
    main.c, 8
showDigitByte
    main.c, 9

TWCR_ACK
    twislave.c, 11
TWCR_NACK
    twislave.c, 11
TWCR_RESET
    twislave.c, 11
twislave.c, 10
    buffer_adr, 12
    init_twi_slave, 11
    ISR, 11
    TWCR_ACK, 11
    TWCR_NACK, 11
    TWCR_RESET, 11
twislave.h, 13
    buffer_size, 14
    init_twi_slave, 14
    rxbuffer, 15
    txbuffer, 15
txbuffer
    twislave.h, 15