/\*

u8g\_com\_i2c.c

generic i2c interface

Universal 8bit Graphics Library

Copyright (c) 2011, olikraus@gmail.com

All rights reserved.

Redistribution and use in source and binary forms, with or without modification,

are permitted provided that the following conditions are met:

\* Redistributions of source code must retain the above copyright notice, this list

of conditions and the following disclaimer.

\* Redistributions in binary form must reproduce the above copyright notice, this

list of conditions and the following disclaimer in the documentation and/or other

materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND

CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,

INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF

MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE

DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR

CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,

SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT

NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;

LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER

CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,

STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)

ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF

ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

\*/

#include "u8g.h"

//#define U8G\_I2C\_WITH\_NO\_ACK

static uint8\_t u8g\_i2c\_err\_code;

static uint8\_t u8g\_i2c\_opt; /\* U8G\_I2C\_OPT\_NO\_ACK, SAM: U8G\_I2C\_OPT\_DEV\_1 \*/

/\*

position values

1: start condition

2: sla transfer

\*/

static uint8\_t u8g\_i2c\_err\_pos;

void u8g\_i2c\_clear\_error(void)

{

u8g\_i2c\_err\_code = U8G\_I2C\_ERR\_NONE;

u8g\_i2c\_err\_pos = 0;

}

uint8\_t u8g\_i2c\_get\_error(void)

{

return u8g\_i2c\_err\_code;

}

uint8\_t u8g\_i2c\_get\_err\_pos(void)

{

return u8g\_i2c\_err\_pos;

}

static void u8g\_i2c\_set\_error(uint8\_t code, uint8\_t pos)

{

if ( u8g\_i2c\_err\_code > 0 )

return;

u8g\_i2c\_err\_code |= code;

u8g\_i2c\_err\_pos = pos;

}

#if defined(\_\_AVR\_\_)

#define U8G\_ATMEGA\_HW\_TWI

/\* remove the definition for attiny \*/

#if \_\_AVR\_ARCH\_\_ == 2

#undef U8G\_ATMEGA\_HW\_TWI

#endif

#if \_\_AVR\_ARCH\_\_ == 25

#undef U8G\_ATMEGA\_HW\_TWI

#endif

#endif

#if defined(U8G\_ATMEGA\_HW\_TWI)

#include <avr/io.h>

#include <util/twi.h>

void u8g\_i2c\_init(uint8\_t options)

{

/\*

TWBR: bit rate register

TWSR: status register (contains preselector bits)

prescalar

0 1

1 4

2 16

3 64

f = F\_CPU/(16+2\*TWBR\*prescalar)

F\_CPU = 16MHz

TWBR = 152;

TWSR = 0;

--> 50KHz

TWBR = 72;

TWSR = 0;

--> 100KHz

TWBR = 12;

TWSR = 0;

--> 400KHz

F\_CPU/(2\*100000)-8 --> calculate TWBR value for 100KHz

\*/

u8g\_i2c\_opt = options;

TWSR = 0;

if ( options & U8G\_I2C\_OPT\_FAST )

{

TWBR = F\_CPU/(2\*400000)-8;

}

else

{

TWBR = F\_CPU/(2\*100000)-8;

}

u8g\_i2c\_clear\_error();

}

uint8\_t u8g\_i2c\_wait(uint8\_t mask, uint8\_t pos)

{

volatile uint16\_t cnt = 2000; /\* timout value should be > 280 for 50KHz Bus and 16 Mhz CPU, however the start condition might need longer \*/

while( !(TWCR & mask) )

{

if ( cnt == 0 )

{

if ( u8g\_i2c\_opt & U8G\_I2C\_OPT\_NO\_ACK )

{

return 1; /\* all ok \*/

}

else

{

u8g\_i2c\_set\_error(U8G\_I2C\_ERR\_TIMEOUT, pos);

return 0; /\* error \*/

}

}

cnt--;

}

return 1; /\* all ok \*/

}

/\* sla includes all 8 bits (with r/w bit), assums master transmit \*/

uint8\_t u8g\_i2c\_start(uint8\_t sla)

{

register uint8\_t status;

/\* send start \*/

TWCR = \_BV(TWINT) | \_BV(TWSTA) | \_BV(TWEN);

/\* wait \*/

if ( u8g\_i2c\_wait(\_BV(TWINT), 1) == 0 )

return 0;

status = TW\_STATUS;

/\* check status after start \*/

if ( status != TW\_START && status != TW\_REP\_START )

{

u8g\_i2c\_set\_error(U8G\_I2C\_ERR\_BUS, 1);

return 0;

}

/\* set slave address \*/

TWDR = sla;

/\* enable sla transfer \*/

TWCR = \_BV(TWINT) | \_BV(TWEN);

/\* wait \*/

if ( u8g\_i2c\_wait(\_BV(TWINT), 2) == 0 )

return 0;

if ( u8g\_i2c\_opt & U8G\_I2C\_OPT\_NO\_ACK )

{

/\* do not check for ACK \*/

}

else

{

status = TW\_STATUS;

/\* check status after sla \*/

if ( status != TW\_MT\_SLA\_ACK )

{

u8g\_i2c\_set\_error(U8G\_I2C\_ERR\_BUS, 2);

return 0;

}

}

return 1;

}

uint8\_t u8g\_i2c\_send\_byte(uint8\_t data)

{

register uint8\_t status;

TWDR = data;

TWCR = \_BV(TWINT) | \_BV(TWEN);

if ( u8g\_i2c\_wait(\_BV(TWINT), 3) == 0 )

return 0;

if ( u8g\_i2c\_opt & U8G\_I2C\_OPT\_NO\_ACK )

{

/\* do not check for ACK \*/

}

else

{

status = TW\_STATUS;

if ( status != TW\_MT\_DATA\_ACK )

{

u8g\_i2c\_set\_error(U8G\_I2C\_ERR\_BUS, 3);

return 0;

}

}

return 1;

}

void u8g\_i2c\_stop(void)

{

/\* write stop \*/

TWCR = \_BV(TWINT) | \_BV(TWEN) | \_BV(TWSTO);

/\* no error is checked for the stop condition \*/

u8g\_i2c\_wait(\_BV(TWSTO), 4);

}

/\*

void twi\_send(uint8\_t adr, uint8\_t data1, uint8\_t data2)

{

u8g\_i2c\_start(adr<<1);

u8g\_i2c\_send\_byte(data1);

u8g\_i2c\_send\_byte(data2);

u8g\_i2c\_stop();

}

\*/

#elif defined(ARDUINO) && defined(\_\_SAM3X8E\_\_)

/\* Arduino Due \*/

#include "Arduino.h"

#include "sam.h"

/\*

Controller

TWI0 TWCK0 PA18 A DUE PCB: SCL1

TWI0 TWD0 PA17 A DUE PCB: SDA1

TWI1 TWCK1 PB13 A DUE PCB: SCL 21

TWI1 TWD1 PB12 A DUE PCB: SDA 20

Arduino definitions

#define PIN\_WIRE\_SDA (20u)

#define PIN\_WIRE\_SCL (21u)

#define WIRE\_INTERFACE TWI1

#define WIRE\_INTERFACE\_ID ID\_TWI1

#define WIRE\_ISR\_HANDLER TWI1\_Handler

#define PIN\_WIRE1\_SDA (70u)

#define PIN\_WIRE1\_SCL (71u)

#define WIRE1\_INTERFACE TWI0

#define WIRE1\_INTERFACE\_ID ID\_TWI0

#define WIRE1\_ISR\_HANDLER TWI0\_Handler

\*/

static void i2c\_400KHz\_delay(void)

{

/\* should be at least 4 \*/

/\* should be 5 for 100KHz transfer speed \*/

/\*

Arduino Due

0x NOP: 470KHz

4x NOP: 450KHz

8x NOP: 430KHz

16x NOP: 400KHz

\*/

\_\_NOP();

\_\_NOP();

\_\_NOP();

\_\_NOP();

\_\_NOP();

\_\_NOP();

\_\_NOP();

\_\_NOP();

\_\_NOP();

\_\_NOP();

\_\_NOP();

\_\_NOP();

\_\_NOP();

\_\_NOP();

\_\_NOP();

\_\_NOP();

}

static void i2c\_100KHz\_delay(void)

{

/\*

1x u8g\_MicroDelay() ca. 130KHz

2x u8g\_MicroDelay() ca. 80KHz

\*/

u8g\_MicroDelay();

u8g\_MicroDelay();

}

uint32\_t i2c\_started = 0;

uint32\_t i2c\_scl\_pin = 0;

uint32\_t i2c\_sda\_pin = 0;

void (\*i2c\_delay)(void) = i2c\_100KHz\_delay;

const PinDescription \*i2c\_scl\_pin\_desc;

const PinDescription \*i2c\_sda\_pin\_desc;

/\* maybe this can be optimized \*/

static void i2c\_init(void)

{

i2c\_sda\_pin\_desc = &(g\_APinDescription[i2c\_sda\_pin]);

i2c\_scl\_pin\_desc = &(g\_APinDescription[i2c\_scl\_pin]);

pinMode(i2c\_sda\_pin, OUTPUT);

digitalWrite(i2c\_sda\_pin, HIGH);

pinMode(i2c\_scl\_pin, OUTPUT);

digitalWrite(i2c\_scl\_pin, HIGH);

PIO\_Configure( i2c\_sda\_pin\_desc->pPort, PIO\_OUTPUT\_0, i2c\_sda\_pin\_desc->ulPin, PIO\_OPENDRAIN );

PIO\_Configure( i2c\_scl\_pin\_desc->pPort, PIO\_OUTPUT\_0, i2c\_scl\_pin\_desc->ulPin, PIO\_OPENDRAIN );

PIO\_Clear( i2c\_sda\_pin\_desc->pPort, i2c\_sda\_pin\_desc->ulPin) ;

PIO\_Clear( i2c\_scl\_pin\_desc->pPort, i2c\_scl\_pin\_desc->ulPin) ;

PIO\_Configure( i2c\_sda\_pin\_desc->pPort, PIO\_INPUT, i2c\_sda\_pin\_desc->ulPin, PIO\_DEFAULT ) ;

PIO\_Configure( i2c\_scl\_pin\_desc->pPort, PIO\_INPUT, i2c\_scl\_pin\_desc->ulPin, PIO\_DEFAULT ) ;

i2c\_delay();

}

/\* actually, the scl line is not observed, so this procedure does not return a value \*/

static void i2c\_read\_scl\_and\_delay(void)

{

uint32\_t dwMask = i2c\_scl\_pin\_desc->ulPin;

//PIO\_Configure( i2c\_scl\_pin\_desc->pPort, PIO\_INPUT, i2c\_scl\_pin\_desc->ulPin, PIO\_DEFAULT ) ;

//PIO\_SetInput( i2c\_scl\_pin\_desc->pPort, i2c\_scl\_pin\_desc->ulPin, PIO\_DEFAULT ) ;

/\* set as input \*/

i2c\_scl\_pin\_desc->pPort->PIO\_ODR = dwMask ;

i2c\_scl\_pin\_desc->pPort->PIO\_PER = dwMask ;

i2c\_delay();

}

static void i2c\_clear\_scl(void)

{

uint32\_t dwMask = i2c\_scl\_pin\_desc->ulPin;

/\* set open collector and drive low \*/

//PIO\_Configure( i2c\_scl\_pin\_desc->pPort, PIO\_OUTPUT\_0, i2c\_scl\_pin\_desc->ulPin, PIO\_OPENDRAIN );

//PIO\_SetOutput( i2c\_scl\_pin\_desc->pPort, i2c\_scl\_pin\_desc->ulPin, 0, 1, 0);

/\* open drain, zero default output \*/

i2c\_scl\_pin\_desc->pPort->PIO\_MDER = dwMask;

i2c\_scl\_pin\_desc->pPort->PIO\_CODR = dwMask;

i2c\_scl\_pin\_desc->pPort->PIO\_OER = dwMask;

i2c\_scl\_pin\_desc->pPort->PIO\_PER = dwMask;

//PIO\_Clear( i2c\_scl\_pin\_desc->pPort, i2c\_scl\_pin\_desc->ulPin) ;

}

static uint8\_t i2c\_read\_sda(void)

{

uint32\_t dwMask = i2c\_sda\_pin\_desc->ulPin;

//PIO\_Configure( i2c\_sda\_pin\_desc->pPort, PIO\_INPUT, i2c\_sda\_pin\_desc->ulPin, PIO\_DEFAULT ) ;

//PIO\_SetInput( i2c\_sda\_pin\_desc->pPort, i2c\_sda\_pin\_desc->ulPin, PIO\_DEFAULT ) ;

/\* set as input \*/

i2c\_sda\_pin\_desc->pPort->PIO\_ODR = dwMask ;

i2c\_sda\_pin\_desc->pPort->PIO\_PER = dwMask ;

return 1;

}

static void i2c\_clear\_sda(void)

{

uint32\_t dwMask = i2c\_sda\_pin\_desc->ulPin;

/\* set open collector and drive low \*/

//PIO\_Configure( i2c\_sda\_pin\_desc->pPort, PIO\_OUTPUT\_0, i2c\_sda\_pin\_desc->ulPin, PIO\_OPENDRAIN );

//PIO\_SetOutput( i2c\_sda\_pin\_desc->pPort, i2c\_sda\_pin\_desc->ulPin, 0, 1, 0);

/\* open drain, zero default output \*/

i2c\_sda\_pin\_desc->pPort->PIO\_MDER = dwMask ;

i2c\_sda\_pin\_desc->pPort->PIO\_CODR = dwMask ;

i2c\_sda\_pin\_desc->pPort->PIO\_OER = dwMask ;

i2c\_sda\_pin\_desc->pPort->PIO\_PER = dwMask ;

//PIO\_Clear( i2c\_sda\_pin\_desc->pPort, i2c\_sda\_pin\_desc->ulPin) ;

}

static void i2c\_start(void)

{

if ( i2c\_started != 0 )

{

/\* if already started: do restart \*/

i2c\_read\_sda(); /\* SDA = 1 \*/

i2c\_delay();

i2c\_read\_scl\_and\_delay();

}

i2c\_read\_sda();

/\*

if (i2c\_read\_sda() == 0)

{

// do something because arbitration is lost

}

\*/

/\* send the start condition, both lines go from 1 to 0 \*/

i2c\_clear\_sda();

i2c\_delay();

i2c\_clear\_scl();

i2c\_started = 1;

}

static void i2c\_stop(void)

{

/\* set SDA to 0 \*/

i2c\_clear\_sda();

i2c\_delay();

/\* now release all lines \*/

i2c\_read\_scl\_and\_delay();

/\* set SDA to 1 \*/

i2c\_read\_sda();

i2c\_delay();

i2c\_started = 0;

}

static void i2c\_write\_bit(uint8\_t val)

{

if (val)

i2c\_read\_sda();

else

i2c\_clear\_sda();

i2c\_delay();

i2c\_read\_scl\_and\_delay();

i2c\_clear\_scl();

}

static uint8\_t i2c\_read\_bit(void)

{

uint8\_t val;

/\* do not drive SDA \*/

i2c\_read\_sda();

i2c\_delay();

i2c\_read\_scl\_and\_delay();

val = i2c\_read\_sda();

i2c\_delay();

i2c\_clear\_scl();

return val;

}

static uint8\_t i2c\_write\_byte(uint8\_t b)

{

i2c\_write\_bit(b & 128);

i2c\_write\_bit(b & 64);

i2c\_write\_bit(b & 32);

i2c\_write\_bit(b & 16);

i2c\_write\_bit(b & 8);

i2c\_write\_bit(b & 4);

i2c\_write\_bit(b & 2);

i2c\_write\_bit(b & 1);

/\* read ack from client \*/

/\* 0: ack was given by client \*/

/\* 1: nothing happend during ack cycle \*/

return i2c\_read\_bit();

}

void u8g\_i2c\_init(uint8\_t options)

{

u8g\_i2c\_opt = options;

u8g\_i2c\_clear\_error();

if ( u8g\_i2c\_opt & U8G\_I2C\_OPT\_FAST )

{

i2c\_delay = i2c\_400KHz\_delay;

}

else

{

i2c\_delay = i2c\_100KHz\_delay;

}

if ( u8g\_i2c\_opt & U8G\_I2C\_OPT\_DEV\_1 )

{

i2c\_scl\_pin = PIN\_WIRE1\_SCL;

i2c\_sda\_pin = PIN\_WIRE1\_SDA;

//REG\_PIOA\_PDR = PIO\_PB12A\_TWD1 | PIO\_PB13A\_TWCK1;

}

else

{

i2c\_scl\_pin = PIN\_WIRE\_SCL;

i2c\_sda\_pin = PIN\_WIRE\_SDA;

//REG\_PIOA\_PDR = PIO\_PA17A\_TWD0 | PIO\_PA18A\_TWCK0;

}

i2c\_init();

}

/\* sla includes also the r/w bit \*/

uint8\_t u8g\_i2c\_start(uint8\_t sla)

{

i2c\_start();

i2c\_write\_byte(sla);

return 1;

}

uint8\_t u8g\_i2c\_send\_byte(uint8\_t data)

{

return i2c\_write\_byte(data);

}

void u8g\_i2c\_stop(void)

{

i2c\_stop();

}

#elif defined(U8G\_RASPBERRY\_PI)

#include <wiringPi.h>

#include <wiringPiI2C.h>

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#define I2C\_SLA 0x3c

static int fd=-1;

static uint8\_t i2cMode = 0;

void u8g\_i2c\_init(uint8\_t options) {

u8g\_i2c\_clear\_error();

u8g\_i2c\_opt = options;

if (wiringPiSetup() == -1) {

printf("wiringPi-Error\n");

exit(1);

}

fd = wiringPiI2CSetup(I2C\_SLA);

if (fd < 0) {

printf ("Unable to open I2C device 0: %s\n", strerror (errno)) ;

exit (1) ;

}

//u8g\_SetPIOutput(u8g, U8G\_PI\_RESET);

//u8g\_SetPIOutput(u8g, U8G\_PI\_A0);

}

uint8\_t u8g\_i2c\_start(uint8\_t sla) {

u8g\_i2c\_send\_mode(0);

return 1;

}

void u8g\_i2c\_stop(void) {

}

uint8\_t u8g\_i2c\_send\_mode(uint8\_t mode) {

i2cMode = mode;

}

uint8\_t u8g\_i2c\_send\_byte(uint8\_t data) {

wiringPiI2CWriteReg8(fd, i2cMode, data);

return 1;

}

uint8\_t u8g\_i2c\_wait(uint8\_t mask, uint8\_t pos)

{

return 1;

}

#else

/\* empty interface \*/

void u8g\_i2c\_init(uint8\_t options)

{

u8g\_i2c\_clear\_error();

}

uint8\_t u8g\_i2c\_wait(uint8\_t mask, uint8\_t pos)

{

return 1;

}

uint8\_t u8g\_i2c\_start(uint8\_t sla)

{

return 1;

}

uint8\_t u8g\_i2c\_send\_byte(uint8\_t data)

{

return 1;

}

void u8g\_i2c\_stop(void)

{

}

#endif