/\*

u8g\_com\_io.c

abstraction layer for low level i/o

Universal 8bit Graphics Library

Copyright (c) 2012, 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.

Update for ATOMIC operation done (01 Jun 2013)

U8G\_ATOMIC\_OR(ptr, val)

U8G\_ATOMIC\_AND(ptr, val)

U8G\_ATOMIC\_START();

U8G\_ATOMIC\_END();

uint8\_t u8g\_Pin(uint8\_t port, uint8\_t bitpos) Convert to internal number: AVR: port\*8+bitpos, ARM: port\*16+bitpos

void u8g\_SetPinOutput(uint8\_t internal\_pin\_number)

void u8g\_SetPinInput(uint8\_t internal\_pin\_number)

void u8g\_SetPinLevel(uint8\_t internal\_pin\_number, uint8\_t level)

uint8\_t u8g\_GetPinLevel(uint8\_t internal\_pin\_number)

\*/

#include "u8g.h"

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

#include <avr/interrupt.h>

#include <avr/io.h>

typedef volatile uint8\_t \* IO\_PTR;

/\* create internal pin number \*/

uint8\_t u8g\_Pin(uint8\_t port, uint8\_t bitpos)

{

port <<= 3;

port += bitpos;

return port;

}

const IO\_PTR u8g\_avr\_ddr\_P[] PROGMEM = {

#ifdef DDRA

&DDRA,

#else

0,

#endif

&DDRB,

#ifdef DDRC

&DDRC,

#ifdef DDRD

&DDRD,

#ifdef DDRE

&DDRE,

#ifdef DDRF

&DDRF,

#ifdef DDRG

&DDRG,

#ifdef DDRH

&DDRH,

#endif

#endif

#endif

#endif

#endif

#endif

};

const IO\_PTR u8g\_avr\_port\_P[] PROGMEM = {

#ifdef PORTA

&PORTA,

#else

0,

#endif

&PORTB,

#ifdef PORTC

&PORTC,

#ifdef PORTD

&PORTD,

#ifdef PORTE

&PORTE,

#ifdef PORTF

&PORTF,

#ifdef PORTG

&PORTG,

#ifdef PORTH

&PORTH,

#endif

#endif

#endif

#endif

#endif

#endif

};

const IO\_PTR u8g\_avr\_pin\_P[] PROGMEM = {

#ifdef PINA

&PINA,

#else

0,

#endif

&PINB,

#ifdef PINC

&PINC,

#ifdef PIND

&PIND,

#ifdef PINE

&PINE,

#ifdef PINF

&PINF,

#ifdef PING

&PING,

#ifdef PINH

&PINH,

#endif

#endif

#endif

#endif

#endif

#endif

};

static volatile uint8\_t \*u8g\_get\_avr\_io\_ptr(const IO\_PTR \*base, uint8\_t offset)

{

volatile uint8\_t \* tmp;

base += offset;

memcpy\_P(&tmp, base, sizeof(volatile uint8\_t \* PROGMEM));

return tmp;

}

/\* set direction to output of the specified pin (internal pin number) \*/

void u8g\_SetPinOutput(uint8\_t internal\_pin\_number)

{

\*u8g\_get\_avr\_io\_ptr(u8g\_avr\_ddr\_P, internal\_pin\_number>>3) |= \_BV(internal\_pin\_number&7);

}

void u8g\_SetPinInput(uint8\_t internal\_pin\_number)

{

\*u8g\_get\_avr\_io\_ptr(u8g\_avr\_ddr\_P, internal\_pin\_number>>3) &= ~\_BV(internal\_pin\_number&7);

}

void u8g\_SetPinLevel(uint8\_t internal\_pin\_number, uint8\_t level)

{

volatile uint8\_t \* tmp = u8g\_get\_avr\_io\_ptr(u8g\_avr\_port\_P, internal\_pin\_number>>3);

if ( level == 0 )

{

U8G\_ATOMIC\_AND(tmp, ~\_BV(internal\_pin\_number&7));

// \*tmp &= ~\_BV(internal\_pin\_number&7);

}

else

{

U8G\_ATOMIC\_OR(tmp, \_BV(internal\_pin\_number&7));

//\*tmp |= \_BV(internal\_pin\_number&7);

}

}

uint8\_t u8g\_GetPinLevel(uint8\_t internal\_pin\_number)

{

volatile uint8\_t \* tmp = u8g\_get\_avr\_io\_ptr(u8g\_avr\_pin\_P, internal\_pin\_number>>3);

if ( ((\*tmp) & \_BV(internal\_pin\_number&7)) != 0 )

return 1;

return 0;

}

#elif defined(U8G\_RASPBERRY\_PI)

#include <wiringPi.h>

//#include "/usr/local/include/wiringPi.h"

void u8g\_SetPinOutput(uint8\_t internal\_pin\_number) {

pinMode(internal\_pin\_number, OUTPUT);

}

void u8g\_SetPinInput(uint8\_t internal\_pin\_number) {

pinMode(internal\_pin\_number, INPUT);

}

void u8g\_SetPinLevel(uint8\_t internal\_pin\_number, uint8\_t level) {

digitalWrite(internal\_pin\_number, level);

}

uint8\_t u8g\_GetPinLevel(uint8\_t internal\_pin\_number) {

return digitalRead(internal\_pin\_number);

}

#else

/\* convert "port" and "bitpos" to internal pin number \*/

uint8\_t u8g\_Pin(uint8\_t port, uint8\_t bitpos)

{

port <<= 3;

port += bitpos;

return port;

}

void u8g\_SetPinOutput(uint8\_t internal\_pin\_number)

{

}

void u8g\_SetPinInput(uint8\_t internal\_pin\_number)

{

}

void u8g\_SetPinLevel(uint8\_t internal\_pin\_number, uint8\_t level)

{

}

uint8\_t u8g\_GetPinLevel(uint8\_t internal\_pin\_number)

{

return 0;

}

#endif

#if defined(U8G\_WITH\_PINLIST)

void u8g\_SetPIOutput(u8g\_t \*u8g, uint8\_t pi)

{

uint8\_t pin;

pin = u8g->pin\_list[pi];

if ( pin != U8G\_PIN\_NONE )

u8g\_SetPinOutput(pin);

}

void u8g\_SetPILevel(u8g\_t \*u8g, uint8\_t pi, uint8\_t level)

{

uint8\_t pin;

pin = u8g->pin\_list[pi];

if ( pin != U8G\_PIN\_NONE )

u8g\_SetPinLevel(pin, level);

}

#else /\* defined(U8G\_WITH\_PINLIST) \*/

void u8g\_SetPIOutput(u8g\_t \*u8g, uint8\_t pi)

{

}

void u8g\_SetPILevel(u8g\_t \*u8g, uint8\_t pi, uint8\_t level)

{

}

#endif /\* defined(U8G\_WITH\_PINLIST) \*/