/\*

u8g\_arduino\_sw\_spi.c

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.

Update for ATOMIC operation done (01 Jun 2013)

U8G\_ATOMIC\_OR(ptr, val)

U8G\_ATOMIC\_AND(ptr, val)

U8G\_ATOMIC\_START();

U8G\_ATOMIC\_END();

\*/

#include "u8g.h"

#if defined(ARDUINO)

#if ARDUINO < 100

#include <WProgram.h>

#include "wiring\_private.h"

#include "pins\_arduino.h"

#else

#include <Arduino.h>

#include "wiring\_private.h"

#endif

/\*=========================================================\*/

/\* Arduino, AVR \*/

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

uint8\_t u8g\_bitData, u8g\_bitNotData;

uint8\_t u8g\_bitClock, u8g\_bitNotClock;

volatile uint8\_t \*u8g\_outData;

volatile uint8\_t \*u8g\_outClock;

static void u8g\_com\_arduino\_init\_shift\_out(uint8\_t dataPin, uint8\_t clockPin)

{

u8g\_outData = portOutputRegister(digitalPinToPort(dataPin));

u8g\_outClock = portOutputRegister(digitalPinToPort(clockPin));

u8g\_bitData = digitalPinToBitMask(dataPin);

u8g\_bitClock = digitalPinToBitMask(clockPin);

u8g\_bitNotClock = u8g\_bitClock;

u8g\_bitNotClock ^= 0x0ff;

u8g\_bitNotData = u8g\_bitData;

u8g\_bitNotData ^= 0x0ff;

}

static void u8g\_com\_arduino\_do\_shift\_out\_msb\_first(uint8\_t val) U8G\_NOINLINE;

static void u8g\_com\_arduino\_do\_shift\_out\_msb\_first(uint8\_t val)

{

uint8\_t cnt = 8;

uint8\_t bitData = u8g\_bitData;

uint8\_t bitNotData = u8g\_bitNotData;

uint8\_t bitClock = u8g\_bitClock;

uint8\_t bitNotClock = u8g\_bitNotClock;

volatile uint8\_t \*outData = u8g\_outData;

volatile uint8\_t \*outClock = u8g\_outClock;

U8G\_ATOMIC\_START();

do

{

if ( val & 128 )

\*outData |= bitData;

else

\*outData &= bitNotData;

\*outClock |= bitClock;

val <<= 1;

cnt--;

\*outClock &= bitNotClock;

} while( cnt != 0 );

U8G\_ATOMIC\_END();

}

/\*=========================================================\*/

/\* Arduino, Chipkit \*/

#elif defined(\_\_18CXX) || defined(\_\_PIC32MX)

uint16\_t dog\_bitData, dog\_bitNotData;

uint16\_t dog\_bitClock, dog\_bitNotClock;

volatile uint32\_t \*dog\_outData;

volatile uint32\_t \*dog\_outClock;

volatile uint32\_t dog\_pic32\_spi\_tmp;

static void u8g\_com\_arduino\_init\_shift\_out(uint8\_t dataPin, uint8\_t clockPin)

{

dog\_outData = portOutputRegister(digitalPinToPort(dataPin));

dog\_outClock = portOutputRegister(digitalPinToPort(clockPin));

dog\_bitData = digitalPinToBitMask(dataPin);

dog\_bitClock = digitalPinToBitMask(clockPin);

dog\_bitNotClock = dog\_bitClock;

dog\_bitNotClock ^= 0x0ffff;

dog\_bitNotData = dog\_bitData;

dog\_bitNotData ^= 0x0ffff;

}

static void u8g\_com\_arduino\_do\_shift\_out\_msb\_first(uint8\_t val)

{

uint8\_t cnt = 8;

U8G\_ATOMIC\_START();

do

{

if ( val & 128 )

\*dog\_outData |= dog\_bitData;

else

\*dog\_outData &= dog\_bitNotData;

val <<= 1;

/\*

There must be some delay here. However

fetching the adress dog\_outClock is enough delay, so

do not place dog\_outClock in a local variable. This will

break the procedure

\*/

\*dog\_outClock |= dog\_bitClock;

cnt--;

\*dog\_outClock &= dog\_bitNotClock;

/\*

little additional delay after clk pulse, done by 3x32bit reads

from I/O. Optimized for PIC32 with 80 MHz.

\*/

dog\_pic32\_spi\_tmp = \*dog\_outClock;

dog\_pic32\_spi\_tmp = \*dog\_outClock;

dog\_pic32\_spi\_tmp = \*dog\_outClock;

} while( cnt != 0 );

U8G\_ATOMIC\_END();

}

/\*=========================================================\*/

/\* Arduino Due \*/

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

/\* Due \*/

void u8g\_digital\_write\_sam\_high(uint8\_t pin)

{

PIO\_Set( g\_APinDescription[pin].pPort, g\_APinDescription[pin].ulPin) ;

}

void u8g\_digital\_write\_sam\_low(uint8\_t pin)

{

PIO\_Clear( g\_APinDescription[pin].pPort, g\_APinDescription[pin].ulPin) ;

}

static uint8\_t u8g\_sam\_data\_pin;

static uint8\_t u8g\_sam\_clock\_pin;

static void u8g\_com\_arduino\_init\_shift\_out(uint8\_t dataPin, uint8\_t clockPin)

{

u8g\_sam\_data\_pin = dataPin;

u8g\_sam\_clock\_pin = clockPin;

}

static void u8g\_com\_arduino\_do\_shift\_out\_msb\_first(uint8\_t val)

{

uint8\_t i = 8;

do

{

if ( val & 128 )

u8g\_digital\_write\_sam\_high(u8g\_sam\_data\_pin);

else

u8g\_digital\_write\_sam\_low(u8g\_sam\_data\_pin);

val <<= 1;

//u8g\_MicroDelay();

u8g\_digital\_write\_sam\_high(u8g\_sam\_clock\_pin);

u8g\_MicroDelay();

u8g\_digital\_write\_sam\_low(u8g\_sam\_clock\_pin);

u8g\_MicroDelay();

i--;

} while( i != 0 );

}

#else

/\* empty interface \*/

static void u8g\_com\_arduino\_init\_shift\_out(uint8\_t dataPin, uint8\_t clockPin)

{

}

static void u8g\_com\_arduino\_do\_shift\_out\_msb\_first(uint8\_t val)

{

}

#endif

uint8\_t u8g\_com\_arduino\_sw\_spi\_fn(u8g\_t \*u8g, uint8\_t msg, uint8\_t arg\_val, void \*arg\_ptr)

{

switch(msg)

{

case U8G\_COM\_MSG\_INIT:

u8g\_com\_arduino\_assign\_pin\_output\_high(u8g);

u8g\_com\_arduino\_digital\_write(u8g, U8G\_PI\_SCK, LOW);

u8g\_com\_arduino\_digital\_write(u8g, U8G\_PI\_MOSI, LOW);

u8g\_com\_arduino\_init\_shift\_out(u8g->pin\_list[U8G\_PI\_MOSI], u8g->pin\_list[U8G\_PI\_SCK]);

break;

case U8G\_COM\_MSG\_STOP:

break;

case U8G\_COM\_MSG\_RESET:

if ( u8g->pin\_list[U8G\_PI\_RESET] != U8G\_PIN\_NONE )

u8g\_com\_arduino\_digital\_write(u8g, U8G\_PI\_RESET, arg\_val);

break;

case U8G\_COM\_MSG\_CHIP\_SELECT:

if ( arg\_val == 0 )

{

/\* disable \*/

u8g\_com\_arduino\_digital\_write(u8g, U8G\_PI\_CS, HIGH);

}

else

{

/\* enable \*/

u8g\_com\_arduino\_digital\_write(u8g, U8G\_PI\_SCK, LOW);

u8g\_com\_arduino\_digital\_write(u8g, U8G\_PI\_CS, LOW);

/\* issue 227 \*/

u8g\_com\_arduino\_init\_shift\_out(u8g->pin\_list[U8G\_PI\_MOSI], u8g->pin\_list[U8G\_PI\_SCK]);

}

break;

case U8G\_COM\_MSG\_WRITE\_BYTE:

u8g\_com\_arduino\_do\_shift\_out\_msb\_first( arg\_val );

//u8g\_arduino\_sw\_spi\_shift\_out(u8g->pin\_list[U8G\_PI\_MOSI], u8g->pin\_list[U8G\_PI\_SCK], arg\_val);

break;

case U8G\_COM\_MSG\_WRITE\_SEQ:

{

register uint8\_t \*ptr = arg\_ptr;

while( arg\_val > 0 )

{

u8g\_com\_arduino\_do\_shift\_out\_msb\_first(\*ptr++);

// u8g\_arduino\_sw\_spi\_shift\_out(u8g->pin\_list[U8G\_PI\_MOSI], u8g->pin\_list[U8G\_PI\_SCK], \*ptr++);

arg\_val--;

}

}

break;

case U8G\_COM\_MSG\_WRITE\_SEQ\_P:

{

register uint8\_t \*ptr = arg\_ptr;

while( arg\_val > 0 )

{

u8g\_com\_arduino\_do\_shift\_out\_msb\_first( u8g\_pgm\_read(ptr) );

//u8g\_arduino\_sw\_spi\_shift\_out(u8g->pin\_list[U8G\_PI\_MOSI], u8g->pin\_list[U8G\_PI\_SCK], u8g\_pgm\_read(ptr));

ptr++;

arg\_val--;

}

}

break;

case U8G\_COM\_MSG\_ADDRESS: /\* define cmd (arg\_val = 0) or data mode (arg\_val = 1) \*/

u8g\_com\_arduino\_digital\_write(u8g, U8G\_PI\_A0, arg\_val);

break;

}

return 1;

}

#else /\* ARDUINO \*/

uint8\_t u8g\_com\_arduino\_sw\_spi\_fn(u8g\_t \*u8g, uint8\_t msg, uint8\_t arg\_val, void \*arg\_ptr)

{

return 1;

}

#endif /\* ARDUINO \*/