/\*

u8g\_com\_arduino\_hw\_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.

SPI Clock Cycle Type

SSD1351 50ns 20 MHz

SSD1322 300ns 3.3 MHz

SSD1327 300ns

SSD1306 300ns

ST7565 400ns 2.5 MHz

ST7920 400ns

Arduino DUE

PA25 MISO

PA26 MOSI 75

PA27 SCLK 76

typedef struct {

WoReg SPI\_CR; (Spi Offset: 0x00) Control Register

RwReg SPI\_MR; (Spi Offset: 0x04) Mode Register

RoReg SPI\_RDR; (Spi Offset: 0x08) Receive Data Register

WoReg SPI\_TDR; (Spi Offset: 0x0C) Transmit Data Register

RoReg SPI\_SR; (Spi Offset: 0x10) Status Register

WoReg SPI\_IER; (Spi Offset: 0x14) Interrupt Enable Register

WoReg SPI\_IDR; (Spi Offset: 0x18) Interrupt Disable Register

RoReg SPI\_IMR; (Spi Offset: 0x1C) Interrupt Mask Register

RoReg Reserved1[4];

RwReg SPI\_CSR[4]; (Spi Offset: 0x30) Chip Select Register

RoReg Reserved2[41];

RwReg SPI\_WPMR; (Spi Offset: 0xE4) Write Protection Control Register

RoReg SPI\_WPSR; (Spi Offset: 0xE8) Write Protection Status Register

} Spi;

Power Management Controller (PMC)

arduino-1.5.2/hardware/arduino/sam/system/CMSIS/Device/ATMEL/sam3xa/include/instance/instance\_pmc.h

- enable PIO

REG\_PMC\_PCER0 = 1UL << ID\_PIOA

- enable SPI

REG\_PMC\_PCER0 = 1UL << ID\_SPI0

- enable PIOA and SPI0

REG\_PMC\_PCER0 = (1UL << ID\_PIOA) | (1UL << ID\_SPI0);

Parallel Input/Output Controller (PIO)

arduino-1.5.2/hardware/arduino/sam/system/CMSIS/Device/ATMEL/sam3xa/include/instance/instance\_pioa.h

- enable special function of the pin: disable PIO on A26 and A27:

REG\_PIOA\_PDR = 0x0c000000

PIOA->PIO\_PDR = 0x0c000000

SPI

SPI0->SPI\_CR = SPI\_CR\_SPIDIS

SPI0->SPI\_CR = SPI\_CR\_SWRST ;

SPI0->SPI\_CR = SPI\_CR\_SWRST ;

SPI0->SPI\_CR = SPI\_CR\_SPIEN

Bit 0: Master Mode = 1 (active)

Bit 1: Peripheral Select = 0 (fixed)

Bit 2: Chip Select Decode Mode = 1 (4 to 16)

Bit 4: Mode Fault Detection = 1 (disabled)

Bit 5: Wait Data Read = 0 (disabled)

Bit 7: Loop Back Mode = 0 (disabled)

Bit 16-19: Peripheral Chip Select = 0 (chip select 0)

SPI0->SPI\_MR = SPI\_MR\_MSTR | SPI\_MR\_PCSDEC | SPI\_MR\_MODFDIS

Bit 0: Clock Polarity = 0

Bit 1: Clock Phase = 0

Bit 4-7: Bits = 0 (8 Bit)

Bit 8-15: SCBR = 1

SPI0->SPI\_CSR[0] = SPI\_CSR\_SCBR(x) Serial Baud Rate

SCBR / 84000000 > 50 / 1000000000

SCBR / 84 > 5 / 100

SCBR > 50 \*84 / 1000 --> SCBR=5

SCBR > 300\*84 / 1000 --> SCBR=26

SCBR > 400\*84 / 1000 --> SCBR=34

Arduino Due test code:

REG\_PMC\_PCER0 = (1UL << ID\_PIOA) | (1UL << ID\_SPI0);

REG\_PIOA\_PDR = 0x0c000000;

SPI0->SPI\_CR = SPI\_CR\_SPIDIS;

SPI0->SPI\_CR = SPI\_CR\_SWRST;

SPI0->SPI\_CR = SPI\_CR\_SWRST;

SPI0->SPI\_CR = SPI\_CR\_SPIEN;

SPI0->SPI\_MR = SPI\_MR\_MSTR | SPI\_MR\_PCSDEC | SPI\_MR\_MODFDIS;

SPI0->SPI\_CSR[0] = SPI\_CSR\_SCBR(30);

for(;;)

{

while( (SPI0->SPI\_SR & SPI\_SR\_TDRE) == 0 )

;

SPI0->SPI\_TDR = 0x050;

}

\*/

#include "u8g.h"

#if defined(ARDUINO)

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

#define U8G\_ARDUINO\_ATMEGA\_HW\_SPI

/\* remove the definition for attiny \*/

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

#undef U8G\_ARDUINO\_ATMEGA\_HW\_SPI

#endif

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

#undef U8G\_ARDUINO\_ATMEGA\_HW\_SPI

#endif

#endif

#if defined(U8G\_ARDUINO\_ATMEGA\_HW\_SPI)

#include <avr/interrupt.h>

#include <avr/io.h>

#if ARDUINO < 100

#include <WProgram.h>

/\* fixed pins \*/

#if defined(\_\_AVR\_ATmega644P\_\_) || defined(\_\_AVR\_ATmega1284P\_\_) // Sanguino.cc board

#define PIN\_SCK 7

#define PIN\_MISO 6

#define PIN\_MOSI 5

#define PIN\_CS 4

#else // Arduino Board

#define PIN\_SCK 13

#define PIN\_MISO 12

#define PIN\_MOSI 11

#define PIN\_CS 10

#endif // (\_\_AVR\_ATmega644P\_\_) || defined(\_\_AVR\_ATmega1284P\_\_)

#else

#include <Arduino.h>

/\* use Arduino pin definitions \*/

#define PIN\_SCK SCK

#define PIN\_MISO MISO

#define PIN\_MOSI MOSI

#define PIN\_CS SS

#endif

//static uint8\_t u8g\_spi\_out(uint8\_t data) U8G\_NOINLINE;

static uint8\_t u8g\_spi\_out(uint8\_t data)

{

/\* unsigned char x = 100; \*/

/\* send data \*/

SPDR = data;

/\* wait for transmission \*/

while (!(SPSR & (1<<SPIF)))

;

/\* clear the SPIF flag by reading SPDR \*/

return SPDR;

}

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

{

switch(msg)

{

case U8G\_COM\_MSG\_STOP:

break;

case U8G\_COM\_MSG\_INIT:

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

pinMode(PIN\_SCK, OUTPUT);

digitalWrite(PIN\_SCK, LOW);

pinMode(PIN\_MOSI, OUTPUT);

digitalWrite(PIN\_MOSI, LOW);

/\* pinMode(PIN\_MISO, INPUT); \*/

pinMode(PIN\_CS, OUTPUT); /\* system chip select for the atmega board \*/

digitalWrite(PIN\_CS, HIGH);

/\*

SPR1 SPR0

0 0 fclk/4

0 1 fclk/16

1 0 fclk/64

1 1 fclk/128

\*/

SPCR = 0;

SPCR = (1<<SPE) | (1<<MSTR)|(0<<SPR1)|(0<<SPR0)|(0<<CPOL)|(0<<CPHA);

#ifdef U8G\_HW\_SPI\_2X

SPSR = (1 << SPI2X); /\* double speed, issue 89 \*/

#else

if ( arg\_val <= U8G\_SPI\_CLK\_CYCLE\_50NS )

{

SPSR = (1 << SPI2X); /\* double speed, issue 89 \*/

}

#endif

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;

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);

}

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\_WRITE\_BYTE:

u8g\_spi\_out(arg\_val);

break;

case U8G\_COM\_MSG\_WRITE\_SEQ:

{

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

while( arg\_val > 0 )

{

u8g\_spi\_out(\*ptr++);

arg\_val--;

}

}

break;

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

{

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

while( arg\_val > 0 )

{

u8g\_spi\_out(u8g\_pgm\_read(ptr));

ptr++;

arg\_val--;

}

}

break;

}

return 1;

}

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

#elif defined(\_\_SAM3X8E\_\_) // Arduino Due, maybe we should better check for \_\_SAM3X8E\_\_

#include <Arduino.h>

/\* use Arduino pin definitions \*/

#define PIN\_SCK SCK

#define PIN\_MISO MISO

#define PIN\_MOSI MOSI

#define PIN\_CS SS

static uint8\_t u8g\_spi\_out(uint8\_t data)

{

/\* wait until tx register is empty \*/

while( (SPI0->SPI\_SR & SPI\_SR\_TDRE) == 0 )

;

/\* send data \*/

SPI0->SPI\_TDR = (uint32\_t)data;

return data;

}

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

{

switch(msg)

{

case U8G\_COM\_MSG\_STOP:

break;

case U8G\_COM\_MSG\_INIT:

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

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

/\* Arduino Due specific code \*/

/\* enable PIOA and SPI0 \*/

REG\_PMC\_PCER0 = (1UL << ID\_PIOA) | (1UL << ID\_SPI0);

/\* disable PIO on A26 and A27 \*/

REG\_PIOA\_PDR = 0x0c000000;

/\* reset SPI0 (from sam lib) \*/

SPI0->SPI\_CR = SPI\_CR\_SPIDIS;

SPI0->SPI\_CR = SPI\_CR\_SWRST;

SPI0->SPI\_CR = SPI\_CR\_SWRST;

SPI0->SPI\_CR = SPI\_CR\_SPIEN;

u8g\_MicroDelay();

/\* master mode, no fault detection, chip select 0 \*/

SPI0->SPI\_MR = SPI\_MR\_MSTR | SPI\_MR\_PCSDEC | SPI\_MR\_MODFDIS;

/\* Polarity, Phase, 8 Bit data transfer, baud rate \*/

/\* x \* 1000 / 84 --> clock cycle in ns

5 \* 1000 / 84 = 58 ns

SCBR > 50 \*84 / 1000 --> SCBR=5

SCBR > 300\*84 / 1000 --> SCBR=26

SCBR > 400\*84 / 1000 --> SCBR=34

\*/

if ( arg\_val <= U8G\_SPI\_CLK\_CYCLE\_50NS )

{

SPI0->SPI\_CSR[0] = SPI\_CSR\_SCBR(5) | 1;

}

else if ( arg\_val <= U8G\_SPI\_CLK\_CYCLE\_300NS )

{

SPI0->SPI\_CSR[0] = SPI\_CSR\_SCBR(26) | 1;

}

else if ( arg\_val <= U8G\_SPI\_CLK\_CYCLE\_400NS )

{

SPI0->SPI\_CSR[0] = SPI\_CSR\_SCBR(34) | 1;

}

else

{

SPI0->SPI\_CSR[0] = SPI\_CSR\_SCBR(84) | 1;

}

u8g\_MicroDelay();

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);

u8g\_MicroDelay();

break;

case U8G\_COM\_MSG\_CHIP\_SELECT:

if ( arg\_val == 0 )

{

/\* disable \*/

u8g\_MicroDelay(); /\* this delay is required to avoid that the display is switched off too early --> DOGS102 with DUE \*/

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

u8g\_MicroDelay();

}

else

{

/\* enable \*/

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

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

u8g\_MicroDelay();

}

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\_WRITE\_BYTE:

u8g\_spi\_out(arg\_val);

u8g\_MicroDelay();

break;

case U8G\_COM\_MSG\_WRITE\_SEQ:

{

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

while( arg\_val > 0 )

{

u8g\_spi\_out(\*ptr++);

arg\_val--;

}

}

break;

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

{

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

while( arg\_val > 0 )

{

u8g\_spi\_out(u8g\_pgm\_read(ptr));

ptr++;

arg\_val--;

}

}

break;

}

return 1;

}

#else /\* U8G\_ARDUINO\_ATMEGA\_HW\_SPI \*/

#endif /\* U8G\_ARDUINO\_ATMEGA\_HW\_SPI \*/

#else /\* ARDUINO \*/

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

{

return 1;

}

#endif /\* ARDUINO \*/