/\*

u8g\_delay.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.

void u8g\_Delay(uint16\_t val) Delay by "val" milliseconds

void u8g\_MicroDelay(void) Delay be one microsecond

void u8g\_10MicroDelay(void) Delay by 10 microseconds

\*/

#include "u8g.h"

/\*==== Part 1: Derive suitable delay procedure ====\*/

#if defined(ARDUINO)

# if ARDUINO < 100

# include <WProgram.h>

# else

# include <Arduino.h>

# endif

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

# define USE\_AVR\_DELAY

# elif defined(\_\_PIC32MX)

# define USE\_PIC32\_DELAY

# elif defined(\_\_arm\_\_) /\* Arduino Due & Teensy \*/

# define USE\_ARDUINO\_DELAY

# else

# define USE\_ARDUINO\_DELAY

# endif

#elif defined(U8G\_RASPBERRY\_PI)

# define USE\_RASPBERRYPI\_DELAY

#elif defined(\_\_AVR\_\_)

# define USE\_AVR\_DELAY

#elif defined(\_\_18CXX)

# define USE\_PIC18\_DELAY

#elif defined(\_\_arm\_\_)

/\* do not define anything, all procedures are expected to be defined outside u8glib \*/

/\*

void u8g\_Delay(uint16\_t val);

void u8g\_MicroDelay(void);

void u8g\_10MicroDelay(void);

\*/

#else

# define USE\_DUMMY\_DELAY

#endif

/\*==== Part 2: Definition of the delay procedures ====\*/

/\*== Raspberry Pi Delay ==\*/

#if defined (USE\_RASPBERRYPI\_DELAY)

#include <wiringPi.h>

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

void u8g\_Delay(uint16\_t val) {

//delay(val);

//usleep((uint32\_t)val\*(uint32\_t)1000);

delayMicroseconds((uint32\_t)val\*(uint32\_t)1000);

}

void u8g\_MicroDelay(void)

{

usleep(1);

}

void u8g\_10MicroDelay(void)

{

usleep(10);

}

#endif

/\*== AVR Delay ==\*/

#if defined(USE\_AVR\_DELAY)

#include <avr/interrupt.h>

#include <avr/io.h>

#include <util/delay.h>

/\*

Delay by the provided number of milliseconds.

Thus, a 16 bit value will allow a delay of 0..65 seconds

Makes use of the \_delay\_loop\_2

\_delay\_loop\_2 will do a delay of n \* 4 prozessor cycles.

with f = F\_CPU cycles per second,

n = f / (1000 \* 4 )

with f = 16000000 the result is 4000

with f = 1000000 the result is 250

the millisec loop, gcc requires the following overhead:

- movev 1

- subwi 2x2

- bne i 2

==> 7 cycles

==> must be devided by 4, rounded up 7/4 = 2

\*/

void u8g\_Delay(uint16\_t val)

{

/\* old version did a call to the arduino lib: delay(val); \*/

while( val != 0 )

{

\_delay\_loop\_2( (F\_CPU / 4000 ) -2);

val--;

}

}

/\* delay by one micro second \*/

void u8g\_MicroDelay(void)

{

#if (F\_CPU / 4000000 ) > 0

\_delay\_loop\_2( (F\_CPU / 4000000 ) );

#endif

}

/\* delay by 10 micro seconds \*/

void u8g\_10MicroDelay(void)

{

#if (F\_CPU / 400000 ) > 0

\_delay\_loop\_2( (F\_CPU / 400000 ) );

#endif

}

#endif

/\*== Delay for PIC18 (not tested) ==\*/

#if defined(USE\_PIC18\_DELAY)

#include <delays.h>

#define GetSystemClock() (64000000ul) // Hz

#define GetInstructionClock() (GetSystemClock()/4)

void u8g\_Delay(uint16\_t val)

{/\*

unsigned int \_iTemp = (val);

while(\_iTemp--)

Delay1KTCYx((GetInstructionClock()+999999)/1000000);

\*/

}

void u8g\_MicroDelay(void)

{

/\* not implemented \*/

}

void u8g\_10MicroDelay(void)

{

/\* not implemented \*/

}

#endif

/\*== Arduino Delay ==\*/

#if defined(USE\_ARDUINO\_DELAY)

void u8g\_Delay(uint16\_t val)

{

#if defined(\_\_arm\_\_)

delayMicroseconds((uint32\_t)val\*(uint32\_t)1000);

#else

delay(val);

#endif

}

void u8g\_MicroDelay(void)

{

delayMicroseconds(1);

}

void u8g\_10MicroDelay(void)

{

delayMicroseconds(10);

}

#endif

#if defined(USE\_PIC32\_DELAY)

/\*

Assume chipkit here with F\_CPU correctly defined

The problem was, that u8g\_Delay() is called within the constructor.

It seems that the chipkit is not fully setup at this time, so a

call to delay() will not work. So here is my own implementation.

\*/

#define CPU\_COUNTS\_PER\_SECOND (F\_CPU/2UL)

#define TICKS\_PER\_MILLISECOND (CPU\_COUNTS\_PER\_SECOND/1000UL)

#include "plib.h"

void u8g\_Delay(uint16\_t val)

{

uint32\_t d;

uint32\_t s;

d = val;

d \*= TICKS\_PER\_MILLISECOND;

s = ReadCoreTimer();

while ( (uint32\_t)(ReadCoreTimer() - s) < d )

;

}

void u8g\_MicroDelay(void)

{

uint32\_t d;

uint32\_t s;

d = TICKS\_PER\_MILLISECOND/1000;

s = ReadCoreTimer();

while ( (uint32\_t)(ReadCoreTimer() - s) < d )

;

}

void u8g\_10MicroDelay(void)

{

uint32\_t d;

uint32\_t s;

d = TICKS\_PER\_MILLISECOND/100;

s = ReadCoreTimer();

while ( (uint32\_t)(ReadCoreTimer() - s) < d )

;

}

#endif

/\*== Any other systems: Dummy Delay ==\*/

#if defined(USE\_DUMMY\_DELAY)

void u8g\_Delay(uint16\_t val)

{

/\* do not know how to delay... \*/

}

void u8g\_MicroDelay(void)

{

}

void u8g\_10MicroDelay(void)

{

}

#endif