/\*

u8g\_ll\_api.c

low level api

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 <stddef.h>

#include "u8g.h"

uint8\_t u8g\_call\_dev\_fn(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t msg, void \*arg)

{

return dev->dev\_fn(u8g, dev, msg, arg);

}

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

uint8\_t u8g\_InitLL(u8g\_t \*u8g, u8g\_dev\_t \*dev)

{

uint8\_t r;

u8g->state\_cb(U8G\_STATE\_MSG\_BACKUP\_ENV);

r = u8g\_call\_dev\_fn(u8g, dev, U8G\_DEV\_MSG\_INIT, NULL);

u8g->state\_cb(U8G\_STATE\_MSG\_BACKUP\_U8G);

u8g->state\_cb(U8G\_STATE\_MSG\_RESTORE\_ENV);

return r;

}

void u8g\_FirstPageLL(u8g\_t \*u8g, u8g\_dev\_t \*dev)

{

u8g->state\_cb(U8G\_STATE\_MSG\_BACKUP\_ENV);

u8g->state\_cb(U8G\_STATE\_MSG\_RESTORE\_U8G);

u8g\_call\_dev\_fn(u8g, dev, U8G\_DEV\_MSG\_PAGE\_FIRST, NULL);

u8g\_call\_dev\_fn(u8g, dev, U8G\_DEV\_MSG\_GET\_PAGE\_BOX, &(u8g->current\_page));

u8g->state\_cb(U8G\_STATE\_MSG\_RESTORE\_ENV);

}

uint8\_t u8g\_NextPageLL(u8g\_t \*u8g, u8g\_dev\_t \*dev)

{

uint8\_t r;

u8g->state\_cb(U8G\_STATE\_MSG\_BACKUP\_ENV);

u8g->state\_cb(U8G\_STATE\_MSG\_RESTORE\_U8G);

r = u8g\_call\_dev\_fn(u8g, dev, U8G\_DEV\_MSG\_PAGE\_NEXT, NULL);

if ( r != 0 )

{

u8g\_call\_dev\_fn(u8g, dev, U8G\_DEV\_MSG\_GET\_PAGE\_BOX, &(u8g->current\_page));

}

u8g->state\_cb(U8G\_STATE\_MSG\_RESTORE\_ENV);

return r;

}

uint8\_t u8g\_SetContrastLL(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t contrast)

{

return u8g\_call\_dev\_fn(u8g, dev, U8G\_DEV\_MSG\_CONTRAST, &contrast);

}

void u8g\_DrawPixelLL(u8g\_t \*u8g, u8g\_dev\_t \*dev, u8g\_uint\_t x, u8g\_uint\_t y)

{

u8g\_dev\_arg\_pixel\_t \*arg = &(u8g->arg\_pixel);

arg->x = x;

arg->y = y;

u8g\_call\_dev\_fn(u8g, dev, U8G\_DEV\_MSG\_SET\_PIXEL, arg);

}

void u8g\_Draw8PixelLL(u8g\_t \*u8g, u8g\_dev\_t \*dev, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t dir, uint8\_t pixel)

{

u8g\_dev\_arg\_pixel\_t \*arg = &(u8g->arg\_pixel);

arg->x = x;

arg->y = y;

arg->dir = dir;

arg->pixel = pixel;

u8g\_call\_dev\_fn(u8g, dev, U8G\_DEV\_MSG\_SET\_8PIXEL, arg);

}

void u8g\_Draw4TPixelLL(u8g\_t \*u8g, u8g\_dev\_t \*dev, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t dir, uint8\_t pixel)

{

u8g\_dev\_arg\_pixel\_t \*arg = &(u8g->arg\_pixel);

arg->x = x;

arg->y = y;

arg->dir = dir;

arg->pixel = pixel;

u8g\_call\_dev\_fn(u8g, dev, U8G\_DEV\_MSG\_SET\_4TPIXEL, arg);

}

#ifdef U8G\_DEV\_MSG\_IS\_BBX\_INTERSECTION

uint8\_t u8g\_IsBBXIntersectionLL(u8g\_t \*u8g, u8g\_dev\_t \*dev, u8g\_uint\_t x, u8g\_uint\_t y, u8g\_uint\_t w, u8g\_uint\_t h)

{

return u8g\_call\_dev\_fn(u8g, dev, U8G\_DEV\_MSG\_IS\_BBX\_INTERSECTION, &arg);

}

#endif

u8g\_uint\_t u8g\_GetWidthLL(u8g\_t \*u8g, u8g\_dev\_t \*dev)

{

u8g\_uint\_t r;

u8g\_call\_dev\_fn(u8g, dev, U8G\_DEV\_MSG\_GET\_WIDTH, &r);

return r;

}

u8g\_uint\_t u8g\_GetHeightLL(u8g\_t \*u8g, u8g\_dev\_t \*dev)

{

u8g\_uint\_t r;

u8g\_call\_dev\_fn(u8g, dev, U8G\_DEV\_MSG\_GET\_HEIGHT, &r);

return r;

}

u8g\_uint\_t u8g\_GetModeLL(u8g\_t \*u8g, u8g\_dev\_t \*dev)

{

return u8g\_call\_dev\_fn(u8g, dev, U8G\_DEV\_MSG\_GET\_MODE, NULL);

}

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

void u8g\_UpdateDimension(u8g\_t \*u8g)

{

u8g->width = u8g\_GetWidthLL(u8g, u8g->dev);

u8g->height = u8g\_GetHeightLL(u8g, u8g->dev);

u8g->mode = u8g\_GetModeLL(u8g, u8g->dev);

/\* 9 Dec 2012: u8g\_scale.c requires update of current page \*/

u8g\_call\_dev\_fn(u8g, u8g->dev, U8G\_DEV\_MSG\_GET\_PAGE\_BOX, &(u8g->current\_page));

}

static void u8g\_init\_data(u8g\_t \*u8g)

{

u8g->font = NULL;

u8g->cursor\_font = NULL;

u8g->cursor\_bg\_color = 0;

u8g->cursor\_fg\_color = 1;

u8g->cursor\_encoding = 34;

u8g->cursor\_fn = (u8g\_draw\_cursor\_fn)0;

#if defined(U8G\_WITH\_PINLIST)

{

uint8\_t i;

for( i = 0; i < U8G\_PIN\_LIST\_LEN; i++ )

u8g->pin\_list[i] = U8G\_PIN\_NONE;

}

#endif

u8g\_SetColorIndex(u8g, 1);

u8g\_SetFontPosBaseline(u8g);

u8g->font\_height\_mode = U8G\_FONT\_HEIGHT\_MODE\_XTEXT;

u8g->font\_ref\_ascent = 0;

u8g->font\_ref\_descent = 0;

u8g->font\_line\_spacing\_factor = 64; /\* 64 = 1.0, 77 = 1.2 line spacing factor \*/

u8g->line\_spacing = 0;

u8g->state\_cb = u8g\_state\_dummy\_cb;

}

uint8\_t u8g\_Begin(u8g\_t \*u8g)

{

/\* call and init low level driver and com device \*/

if ( u8g\_InitLL(u8g, u8g->dev) == 0 )

return 0;

/\* fetch width and height from the low level \*/

u8g\_UpdateDimension(u8g);

return 1;

}

uint8\_t u8g\_Init(u8g\_t \*u8g, u8g\_dev\_t \*dev)

{

u8g\_init\_data(u8g);

u8g->dev = dev;

/\* On the Arduino Environment this will lead to two calls to u8g\_Begin(), the following line will be called first (by U8glib constructors) \*/

/\* if - in future releases - this is removed, then still call u8g\_UpdateDimension() \*/

/\* if Arduino call u8g\_UpdateDimension else u8g\_Begin \*/

/\* issue 146 \*/

return u8g\_Begin(u8g);

}

/\* special init for pure ARM systems \*/

uint8\_t u8g\_InitComFn(u8g\_t \*u8g, u8g\_dev\_t \*dev, u8g\_com\_fnptr com\_fn)

{

u8g\_init\_data(u8g);

#if defined(U8G\_WITH\_PINLIST)

{

uint8\_t i;

for( i = 0; i < U8G\_PIN\_LIST\_LEN; i++ )

u8g->pin\_list[i] = U8G\_PIN\_DUMMY;

}

#endif

u8g->dev = dev;

/\* replace the device procedure with a custom communication procedure \*/

u8g->dev->com\_fn = com\_fn;

/\* On the Arduino Environment this will lead to two calls to u8g\_Begin(), the following line will be called first (by U8glib constructors) \*/

/\* if - in future releases - this is removed, then still call u8g\_UpdateDimension() \*/

/\* if Arduino call u8g\_UpdateDimension else u8g\_Begin \*/

/\* issue 146 \*/

return u8g\_Begin(u8g);

}

#if defined(U8G\_WITH\_PINLIST)

uint8\_t u8g\_InitSPI(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t sck, uint8\_t mosi, uint8\_t cs, uint8\_t a0, uint8\_t reset)

{

/\* fill data structure with some suitable values \*/

u8g\_init\_data(u8g);

u8g->dev = dev;

/\* assign user pins \*/

u8g->pin\_list[U8G\_PI\_SCK] = sck;

u8g->pin\_list[U8G\_PI\_MOSI] = mosi;

u8g->pin\_list[U8G\_PI\_CS] = cs;

u8g->pin\_list[U8G\_PI\_A0] = a0;

u8g->pin\_list[U8G\_PI\_RESET] = reset;

/\* On the Arduino Environment this will lead to two calls to u8g\_Begin(), the following line will be called first (by U8glib constructors) \*/

/\* if - in future releases - this is removed, then still call u8g\_UpdateDimension() \*/

/\* if Arduino call u8g\_UpdateDimension else u8g\_Begin \*/

/\* issue 146 \*/

return u8g\_Begin(u8g);

}

uint8\_t u8g\_InitHWSPI(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t cs, uint8\_t a0, uint8\_t reset)

{

/\* fill data structure with some suitable values \*/

u8g\_init\_data(u8g);

u8g->dev = dev;

/\* assign user pins \*/

u8g->pin\_list[U8G\_PI\_CS] = cs;

u8g->pin\_list[U8G\_PI\_A0] = a0;

u8g->pin\_list[U8G\_PI\_RESET] = reset;

return u8g\_Begin(u8g);

}

uint8\_t u8g\_InitI2C(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t options)

{

/\* fill data structure with some suitable values \*/

u8g\_init\_data(u8g);

u8g->dev = dev;

u8g->pin\_list[U8G\_PI\_I2C\_OPTION] = options;

return u8g\_Begin(u8g);

}

uint8\_t u8g\_Init8BitFixedPort(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t en, uint8\_t cs, uint8\_t di, uint8\_t rw, uint8\_t reset)

{

/\* fill data structure with some suitable values \*/

u8g\_init\_data(u8g);

u8g->dev = dev;

/\* assign user pins \*/

u8g->pin\_list[U8G\_PI\_EN] = en;

u8g->pin\_list[U8G\_PI\_CS] = cs;

u8g->pin\_list[U8G\_PI\_DI] = di;

u8g->pin\_list[U8G\_PI\_RW] = rw;

u8g->pin\_list[U8G\_PI\_RESET] = reset;

return u8g\_Begin(u8g);

}

uint8\_t u8g\_Init8Bit(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t d0, uint8\_t d1, uint8\_t d2, uint8\_t d3, uint8\_t d4, uint8\_t d5, uint8\_t d6, uint8\_t d7,

uint8\_t en, uint8\_t cs1, uint8\_t cs2, uint8\_t di, uint8\_t rw, uint8\_t reset)

{

/\* fill data structure with some suitable values \*/

u8g\_init\_data(u8g);

u8g->dev = dev;

/\* assign user pins \*/

u8g->pin\_list[U8G\_PI\_D0] = d0;

u8g->pin\_list[U8G\_PI\_D1] = d1;

u8g->pin\_list[U8G\_PI\_D2] = d2;

u8g->pin\_list[U8G\_PI\_D3] = d3;

u8g->pin\_list[U8G\_PI\_D4] = d4;

u8g->pin\_list[U8G\_PI\_D5] = d5;

u8g->pin\_list[U8G\_PI\_D6] = d6;

u8g->pin\_list[U8G\_PI\_D7] = d7;

u8g->pin\_list[U8G\_PI\_EN] = en;

u8g->pin\_list[U8G\_PI\_CS1] = cs1;

u8g->pin\_list[U8G\_PI\_CS2] = cs2;

u8g->pin\_list[U8G\_PI\_DI] = di;

u8g->pin\_list[U8G\_PI\_RW] = rw;

u8g->pin\_list[U8G\_PI\_RESET] = reset;

return u8g\_Begin(u8g);

}

/\*

PIN\_D0 8

PIN\_D1 9

PIN\_D2 10

PIN\_D3 11

PIN\_D4 4

PIN\_D5 5

PIN\_D6 6

PIN\_D7 7

PIN\_CS 14

PIN\_A0 15

PIN\_RESET 16

PIN\_WR 17

PIN\_RD 18

u8g\_InitRW8Bit(u8g, dev, d0, d1, d2, d3, d4, d5, d6, d7, cs, a0, wr, rd, reset)

u8g\_InitRW8Bit(u8g, dev, 8, 9, 10, 11, 4, 5, 6, 7, 14, 15, 17, 18, 16)

\*/

uint8\_t u8g\_InitRW8Bit(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t d0, uint8\_t d1, uint8\_t d2, uint8\_t d3, uint8\_t d4, uint8\_t d5, uint8\_t d6, uint8\_t d7,

uint8\_t cs, uint8\_t a0, uint8\_t wr, uint8\_t rd, uint8\_t reset)

{

/\* fill data structure with some suitable values \*/

u8g\_init\_data(u8g);

u8g->dev = dev;

/\* assign user pins \*/

u8g->pin\_list[U8G\_PI\_D0] = d0;

u8g->pin\_list[U8G\_PI\_D1] = d1;

u8g->pin\_list[U8G\_PI\_D2] = d2;

u8g->pin\_list[U8G\_PI\_D3] = d3;

u8g->pin\_list[U8G\_PI\_D4] = d4;

u8g->pin\_list[U8G\_PI\_D5] = d5;

u8g->pin\_list[U8G\_PI\_D6] = d6;

u8g->pin\_list[U8G\_PI\_D7] = d7;

u8g->pin\_list[U8G\_PI\_CS] = cs;

u8g->pin\_list[U8G\_PI\_A0] = a0;

u8g->pin\_list[U8G\_PI\_WR] = wr;

u8g->pin\_list[U8G\_PI\_RD] = rd;

u8g->pin\_list[U8G\_PI\_RESET] = reset;

return u8g\_Begin(u8g);

}

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

void u8g\_FirstPage(u8g\_t \*u8g)

{

u8g\_FirstPageLL(u8g, u8g->dev);

}

uint8\_t u8g\_NextPage(u8g\_t \*u8g)

{

if ( u8g->cursor\_fn != (u8g\_draw\_cursor\_fn)0 )

{

u8g->cursor\_fn(u8g);

}

return u8g\_NextPageLL(u8g, u8g->dev);

}

uint8\_t u8g\_SetContrast(u8g\_t \*u8g, uint8\_t contrast)

{

return u8g\_SetContrastLL(u8g, u8g->dev, contrast);

}

void u8g\_SleepOn(u8g\_t \*u8g)

{

u8g\_call\_dev\_fn(u8g, u8g->dev, U8G\_DEV\_MSG\_SLEEP\_ON, NULL);

}

void u8g\_SleepOff(u8g\_t \*u8g)

{

u8g\_call\_dev\_fn(u8g, u8g->dev, U8G\_DEV\_MSG\_SLEEP\_OFF, NULL);

}

void u8g\_DrawPixel(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y)

{

u8g\_DrawPixelLL(u8g, u8g->dev, x, y);

}

void u8g\_Draw8Pixel(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t dir, uint8\_t pixel)

{

u8g\_Draw8PixelLL(u8g, u8g->dev, x, y, dir, pixel);

}

void u8g\_Draw4TPixel(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t dir, uint8\_t pixel)

{

u8g\_Draw4TPixelLL(u8g, u8g->dev, x, y, dir, pixel);

}

/\* u8g\_IsBBXIntersection() has been moved to u8g\_clip.c \*/

#ifdef OBSOLETE\_CODE

uint8\_t u8g\_IsBBXIntersection(u8g\_t \*u8g, u8g\_uint\_t x, u8g\_uint\_t y, u8g\_uint\_t w, u8g\_uint\_t h)

{

/\* new code \*/

u8g\_dev\_arg\_bbx\_t arg;

arg.x = x;

arg.y = y;

arg.w = w;

arg.h = h;

return u8g\_is\_box\_bbx\_intersection(&(u8g->current\_page), &arg);

/\* old code \*/

//return u8g\_IsBBXIntersectionLL(u8g, u8g->dev, x, y, w, h);

}

#endif

/\*

idx: index for the palette entry (0..255)

r: value for red (0..255)

g: value for green (0..255)

b: value for blue (0..255)

\*/

void u8g\_SetColorEntry(u8g\_t \*u8g, uint8\_t idx, uint8\_t r, uint8\_t g, uint8\_t b)

{

u8g\_dev\_arg\_irgb\_t irgb;

irgb.idx = idx;

irgb.r = r;

irgb.g = g;

irgb.b = b;

u8g\_call\_dev\_fn(u8g, u8g->dev, U8G\_DEV\_MSG\_SET\_COLOR\_ENTRY, &irgb);

}

void u8g\_SetColorIndex(u8g\_t \*u8g, uint8\_t idx)

{

u8g->arg\_pixel.color = idx;

/\*u8g->color\_index = idx; \*/ /\* must be removed \*/

}

void u8g\_SetHiColor(u8g\_t \*u8g, uint16\_t rgb)

{

u8g->arg\_pixel.color = rgb&255;

u8g->arg\_pixel.hi\_color = rgb>>8;

/\*u8g->color\_index = idx; \*/ /\* must be removed \*/

}

void u8g\_SetHiColorByRGB(u8g\_t \*u8g, uint8\_t r, uint8\_t g, uint8\_t b)

{

r &= ~7;

g >>= 2;

b >>= 3;

u8g->arg\_pixel.color = b;

u8g->arg\_pixel.color |= (g & 7) << 5;

u8g->arg\_pixel.hi\_color = r;

u8g->arg\_pixel.hi\_color |= (g>>3) & 7;

//u8g\_SetHiColor(u8g, U8G\_GET\_HICOLOR\_BY\_RGB(r,g,b));

}

void u8g\_SetRGB(u8g\_t \*u8g, uint8\_t r, uint8\_t g, uint8\_t b)

{

if ( u8g->mode == U8G\_MODE\_R3G3B2 )

{

r &= 0x0e0;

g &= 0x0e0;

g >>= 3;

b >>= 6;

u8g->arg\_pixel.color = r | g | b;

}

else if ( u8g->mode == U8G\_MODE\_HICOLOR )

{

u8g\_SetHiColorByRGB(u8g, r,g,b);

}

else

{

u8g->arg\_pixel.color = r;

u8g->arg\_pixel.hi\_color = g;

u8g->arg\_pixel.blue = b;

}

}

uint8\_t u8g\_GetColorIndex(u8g\_t \*u8g)

{

return u8g->arg\_pixel.color;

}

uint8\_t u8g\_GetDefaultForegroundColor(u8g\_t \*u8g)

{

uint8\_t mode;

mode = u8g\_GetMode(u8g);

if ( mode == U8G\_MODE\_R3G3B2 )

return 255; /\* white \*/

else if ( u8g\_GetMode(u8g) == U8G\_MODE\_GRAY2BIT )

return 3; /\* max intensity \*/

else /\* if ( u8g.getMode() == U8G\_MODE\_BW ) \*/

return 1; /\* pixel on \*/

return 1;

}

void u8g\_SetDefaultForegroundColor(u8g\_t \*u8g)

{

if ( u8g->mode == U8G\_MODE\_HICOLOR )

{

u8g->arg\_pixel.color = 0x0ff;

u8g->arg\_pixel.hi\_color = 0x0ff;

}

else

{

u8g\_SetColorIndex(u8g, u8g\_GetDefaultForegroundColor(u8g));

}

}

uint8\_t u8g\_GetDefaultBackgroundColor(u8g\_t \*u8g)

{

return 0;

}

void u8g\_SetDefaultBackgroundColor(u8g\_t \*u8g)

{

u8g\_SetColorIndex(u8g, u8g\_GetDefaultBackgroundColor(u8g)); /\* pixel on / black \*/

}

uint8\_t u8g\_GetDefaultMidColor(u8g\_t \*u8g)

{

uint8\_t mode;

mode = u8g\_GetMode(u8g);

if ( mode == U8G\_MODE\_R3G3B2 )

return 0x06d; /\* gray: 01101101 \*/

else if ( u8g\_GetMode(u8g) == U8G\_MODE\_GRAY2BIT )

return 1; /\* low mid intensity \*/

else /\* if ( u8g.getMode() == U8G\_MODE\_BW ) \*/

return 1; /\* pixel on \*/

return 1; /\* default \*/

}

void u8g\_SetDefaultMidColor(u8g\_t \*u8g)

{

u8g\_SetColorIndex(u8g, u8g\_GetDefaultMidColor(u8g));

}