/\*

u8g\_pb16h1.c

2x 8bit height monochrom (1 bit) page buffer

byte has horizontal orientation

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.

total buffer size is limited to 2\*256 bytes because of the calculation inside the set pixel procedure

\*/

#include "u8g.h"

#include <string.h>

void u8g\_pb16h1\_Init(u8g\_pb\_t \*b, void \*buf, u8g\_uint\_t width) U8G\_NOINLINE;

void u8g\_pb16h1\_set\_pixel(u8g\_pb\_t \*b, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t color\_index) U8G\_NOINLINE;

void u8g\_pb16h1\_SetPixel(u8g\_pb\_t \*b, const u8g\_dev\_arg\_pixel\_t \* const arg\_pixel) U8G\_NOINLINE ;

void u8g\_pb16h1\_Set8PixelStd(u8g\_pb\_t \*b, u8g\_dev\_arg\_pixel\_t \*arg\_pixel) U8G\_NOINLINE;

uint8\_t u8g\_dev\_pb8h1\_base\_fn(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t msg, void \*arg);

void u8g\_pb16h1\_Clear(u8g\_pb\_t \*b)

{

uint8\_t \*ptr = (uint8\_t \*)b->buf;

uint8\_t \*end\_ptr = ptr;

end\_ptr += b->width\*2;

do

{

\*ptr++ = 0;

} while( ptr != end\_ptr );

}

void u8g\_pb16h1\_Init(u8g\_pb\_t \*b, void \*buf, u8g\_uint\_t width)

{

b->buf = buf;

b->width = width;

u8g\_pb16h1\_Clear(b);

}

/\* limitation: total buffer must not exceed 2\*256 bytes \*/

void u8g\_pb16h1\_set\_pixel(u8g\_pb\_t \*b, u8g\_uint\_t x, u8g\_uint\_t y, uint8\_t color\_index)

{

register uint8\_t mask;

u8g\_uint\_t tmp;

uint8\_t \*ptr = b->buf;

y -= b->p.page\_y0;

if ( y >= 8 )

{

ptr += b->width;

y &= 0x07;

}

tmp = b->width;

tmp >>= 3;

tmp \*= (uint8\_t)y;

ptr += tmp;

mask = 0x080;

mask >>= x & 7;

x >>= 3;

ptr += x;

if ( color\_index )

{

\*ptr |= mask;

}

else

{

mask ^=0xff;

\*ptr &= mask;

}

}

void u8g\_pb16h1\_SetPixel(u8g\_pb\_t \*b, const u8g\_dev\_arg\_pixel\_t \* const arg\_pixel)

{

if ( arg\_pixel->y < b->p.page\_y0 )

return;

if ( arg\_pixel->y > b->p.page\_y1 )

return;

if ( arg\_pixel->x >= b->width )

return;

u8g\_pb16h1\_set\_pixel(b, arg\_pixel->x, arg\_pixel->y, arg\_pixel->color);

}

void u8g\_pb16h1\_Set8PixelStd(u8g\_pb\_t \*b, u8g\_dev\_arg\_pixel\_t \*arg\_pixel)

{

register uint8\_t pixel = arg\_pixel->pixel;

do

{

if ( pixel & 128 )

{

u8g\_pb16h1\_SetPixel(b, arg\_pixel);

}

switch( arg\_pixel->dir )

{

case 0: arg\_pixel->x++; break;

case 1: arg\_pixel->y++; break;

case 2: arg\_pixel->x--; break;

case 3: arg\_pixel->y--; break;

}

pixel <<= 1;

} while( pixel != 0 );

}

void u8g\_pb16h1\_Set8PixelOpt2(u8g\_pb\_t \*b, u8g\_dev\_arg\_pixel\_t \*arg\_pixel)

{

register uint8\_t pixel = arg\_pixel->pixel;

u8g\_uint\_t dx = 0;

u8g\_uint\_t dy = 0;

switch( arg\_pixel->dir )

{

case 0: dx++; break;

case 1: dy++; break;

case 2: dx--; break;

case 3: dy--; break;

}

do

{

if ( pixel & 128 )

u8g\_pb16h1\_SetPixel(b, arg\_pixel);

arg\_pixel->x += dx;

arg\_pixel->y += dy;

pixel <<= 1;

} while( pixel != 0 );

}

uint8\_t u8g\_dev\_pb16h1\_base\_fn(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t msg, void \*arg)

{

u8g\_pb\_t \*pb = (u8g\_pb\_t \*)(dev->dev\_mem);

switch(msg)

{

case U8G\_DEV\_MSG\_SET\_8PIXEL:

if ( u8g\_pb\_Is8PixelVisible(pb, (u8g\_dev\_arg\_pixel\_t \*)arg) )

u8g\_pb16h1\_Set8PixelOpt2(pb, (u8g\_dev\_arg\_pixel\_t \*)arg);

break;

case U8G\_DEV\_MSG\_SET\_PIXEL:

u8g\_pb16h1\_SetPixel(pb, (u8g\_dev\_arg\_pixel\_t \*)arg);

break;

case U8G\_DEV\_MSG\_INIT:

break;

case U8G\_DEV\_MSG\_STOP:

break;

case U8G\_DEV\_MSG\_PAGE\_FIRST:

u8g\_pb16h1\_Clear(pb);

u8g\_page\_First(&(pb->p));

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

if ( u8g\_page\_Next(&(pb->p)) == 0 )

return 0;

u8g\_pb16h1\_Clear(pb);

break;

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

case U8G\_DEV\_MSG\_IS\_BBX\_INTERSECTION:

return u8g\_pb\_IsIntersection(pb, (u8g\_dev\_arg\_bbx\_t \*)arg);

#endif

case U8G\_DEV\_MSG\_GET\_PAGE\_BOX:

u8g\_pb\_GetPageBox(pb, (u8g\_box\_t \*)arg);

break;

case U8G\_DEV\_MSG\_GET\_WIDTH:

\*((u8g\_uint\_t \*)arg) = pb->width;

break;

case U8G\_DEV\_MSG\_GET\_HEIGHT:

\*((u8g\_uint\_t \*)arg) = pb->p.total\_height;

break;

case U8G\_DEV\_MSG\_SET\_COLOR\_ENTRY:

break;

case U8G\_DEV\_MSG\_SET\_XY\_CB:

break;

case U8G\_DEV\_MSG\_GET\_MODE:

return U8G\_MODE\_BW;

}

return 1;

}