/\*

u8g\_dev\_ssd1351\_128x128.c

Universal 8bit Graphics Library

Copyright (c) 2013, jamjardavies@gmail.com

Copyright (c) 2013, 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.

History:

Initial version 20 May 2013 jamjardavies@gmail.com

indexed device 22 May 2013 olikraus@gmail.com

\*/

#include "u8g.h"

#define WIDTH 128

#define HEIGHT 128

#define PAGE\_HEIGHT 8

static const uint8\_t u8g\_dev\_ssd1351\_128x128\_init\_seq[] PROGMEM = {

U8G\_ESC\_CS(0), /\* disable chip \*/

U8G\_ESC\_DLY(50),

U8G\_ESC\_ADR(0), /\* instruction mode \*/

U8G\_ESC\_RST(1), /\* do reset low pulse with (1\*16)+2 milliseconds \*/

U8G\_ESC\_CS(1), /\* enable chip \*/

U8G\_ESC\_DLY(50),

0xfd, /\* Command Lock \*/

U8G\_ESC\_ADR(1),

0x12,

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xfd,

U8G\_ESC\_ADR(1),

0xb1, /\* Command Lock \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xae, /\* Set Display Off \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xb3,

U8G\_ESC\_ADR(1),

0xf1, /\* Front Clock Div \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xca,

U8G\_ESC\_ADR(1),

0x7f, /\* Set Multiplex Ratio \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xa0,

U8G\_ESC\_ADR(1),

0xb4, /\* Set Colour Depth \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x15,

U8G\_ESC\_ADR(1),

0x00, 0x7f, /\* Set Column Address \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x75,

U8G\_ESC\_ADR(1),

0x00, 0x7f, /\* Set Row Address \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xa1,

U8G\_ESC\_ADR(1),

0x00, /\* Set Display Start Line \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xa2,

U8G\_ESC\_ADR(1),

0x00, /\* Set Display Offset \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xb5,

U8G\_ESC\_ADR(1),

0x00, /\* Set GPIO \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xab,

U8G\_ESC\_ADR(1),

0x01, /\* Set Function Selection \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xb1,

U8G\_ESC\_ADR(1),

0x32, /\* Set Phase Length \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xb4,

U8G\_ESC\_ADR(1),

0xa0, 0xb5, 0x55, /\* Set Segment Low Voltage \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xbb,

U8G\_ESC\_ADR(1),

0x17, /\* Set Precharge Voltage \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xbe,

U8G\_ESC\_ADR(1),

0x05, /\* Set VComH Voltage \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xc1,

U8G\_ESC\_ADR(1),

0xc8, 0x80, 0xc8, /\* Set Contrast \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xc7,

U8G\_ESC\_ADR(1),

0x0f, /\* Set Master Contrast \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xb6,

U8G\_ESC\_ADR(1),

0x01, /\* Set Second Precharge Period \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xa6, /\* Set Display Mode Reset \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xb8, /\* Set CMD Grayscale Lookup \*/

U8G\_ESC\_ADR(1),

0x05,

0x06,

0x07,

0x08,

0x09,

0x0a,

0x0b,

0x0c,

0x0D,

0x0E,

0x0F,

0x10,

0x11,

0x12,

0x13,

0x14,

0x15,

0x16,

0x18,

0x1a,

0x1b,

0x1C,

0x1D,

0x1F,

0x21,

0x23,

0x25,

0x27,

0x2A,

0x2D,

0x30,

0x33,

0x36,

0x39,

0x3C,

0x3F,

0x42,

0x45,

0x48,

0x4C,

0x50,

0x54,

0x58,

0x5C,

0x60,

0x64,

0x68,

0x6C,

0x70,

0x74,

0x78,

0x7D,

0x82,

0x87,

0x8C,

0x91,

0x96,

0x9B,

0xA0,

0xA5,

0xAA,

0xAF,

0xB4,

U8G\_ESC\_ADR(0),

0xaf, /\* Set Display On \*/

0x5c,

U8G\_ESC\_DLY(50),

U8G\_ESC\_CS(0), /\* disable chip \*/

U8G\_ESC\_ADR(1),

U8G\_ESC\_END /\* end of sequence \*/

};

/\* set gpio to high \*/

static const uint8\_t u8g\_dev\_ssd1351\_128x128gh\_init\_seq[] PROGMEM = {

U8G\_ESC\_CS(0), /\* disable chip \*/

U8G\_ESC\_DLY(50),

U8G\_ESC\_ADR(0), /\* instruction mode \*/

U8G\_ESC\_RST(1), /\* do reset low pulse with (1\*16)+2 milliseconds \*/

U8G\_ESC\_CS(1), /\* enable chip \*/

U8G\_ESC\_DLY(50),

0xfd, /\* Command Lock \*/

U8G\_ESC\_ADR(1),

0x12,

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xfd,

U8G\_ESC\_ADR(1),

0xb1, /\* Command Lock \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xae, /\* Set Display Off \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xb3,

U8G\_ESC\_ADR(1),

0xf1, /\* Front Clock Div \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xca,

U8G\_ESC\_ADR(1),

0x7f, /\* Set Multiplex Ratio \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xa0,

U8G\_ESC\_ADR(1),

0xb4, /\* Set Colour Depth \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x15,

U8G\_ESC\_ADR(1),

0x00, 0x7f, /\* Set Column Address \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0x75,

U8G\_ESC\_ADR(1),

0x00, 0x7f, /\* Set Row Address \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xa1,

U8G\_ESC\_ADR(1),

0x00, /\* Set Display Start Line \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xa2,

U8G\_ESC\_ADR(1),

0x00, /\* Set Display Offset \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xb5,

U8G\_ESC\_ADR(1),

0x03, /\* Set GPIO to High Level \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xab,

U8G\_ESC\_ADR(1),

0x01, /\* Set Function Selection \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xb1,

U8G\_ESC\_ADR(1),

0x32, /\* Set Phase Length \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xb4,

U8G\_ESC\_ADR(1),

0xa0, 0xb5, 0x55, /\* Set Segment Low Voltage \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xbb,

U8G\_ESC\_ADR(1),

0x17, /\* Set Precharge Voltage \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xbe,

U8G\_ESC\_ADR(1),

0x05, /\* Set VComH Voltage \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xc1,

U8G\_ESC\_ADR(1),

0xc8, 0x80, 0xc8, /\* Set Contrast \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xc7,

U8G\_ESC\_ADR(1),

0x0f, /\* Set Master Contrast \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xb6,

U8G\_ESC\_ADR(1),

0x01, /\* Set Second Precharge Period \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xa6, /\* Set Display Mode Reset \*/

U8G\_ESC\_ADR(0), /\* instruction mode \*/

0xb8, /\* Set CMD Grayscale Lookup \*/

U8G\_ESC\_ADR(1),

0x05,

0x06,

0x07,

0x08,

0x09,

0x0a,

0x0b,

0x0c,

0x0D,

0x0E,

0x0F,

0x10,

0x11,

0x12,

0x13,

0x14,

0x15,

0x16,

0x18,

0x1a,

0x1b,

0x1C,

0x1D,

0x1F,

0x21,

0x23,

0x25,

0x27,

0x2A,

0x2D,

0x30,

0x33,

0x36,

0x39,

0x3C,

0x3F,

0x42,

0x45,

0x48,

0x4C,

0x50,

0x54,

0x58,

0x5C,

0x60,

0x64,

0x68,

0x6C,

0x70,

0x74,

0x78,

0x7D,

0x82,

0x87,

0x8C,

0x91,

0x96,

0x9B,

0xA0,

0xA5,

0xAA,

0xAF,

0xB4,

U8G\_ESC\_ADR(0),

0xaf, /\* Set Display On \*/

0x5c,

U8G\_ESC\_DLY(50),

U8G\_ESC\_CS(0), /\* disable chip \*/

U8G\_ESC\_ADR(1),

U8G\_ESC\_END /\* end of sequence \*/

};

#define u8g\_dev\_ssd1351\_128x128\_init\_seq u8g\_dev\_ssd1351\_128x128\_init\_seq

static const uint8\_t u8g\_dev\_ssd1351\_128x128\_column\_seq[] PROGMEM = {

U8G\_ESC\_CS(1),

U8G\_ESC\_ADR(0), 0x15,

U8G\_ESC\_ADR(1), 0x00, 0x7f,

U8G\_ESC\_ADR(0), 0x75,

U8G\_ESC\_ADR(1), 0x00, 0x7f,

U8G\_ESC\_ADR(0), 0x5c,

U8G\_ESC\_ADR(1),

U8G\_ESC\_CS(0),

U8G\_ESC\_END

};

#define RGB332\_STREAM\_BYTES 8

static uint8\_t u8g\_ssd1351\_stream\_bytes[RGB332\_STREAM\_BYTES\*3];

void u8g\_ssd1351\_to\_stream(uint8\_t \*ptr)

{

uint8\_t cnt = RGB332\_STREAM\_BYTES;

uint8\_t val;

uint8\_t \*dest = u8g\_ssd1351\_stream\_bytes;

for( cnt = 0; cnt < RGB332\_STREAM\_BYTES; cnt++ )

{

val = \*ptr++;

\*dest++ = ((val & 0xe0) >> 2);

\*dest++ = ((val & 0x1c) << 1);

\*dest++ = ((val & 0x03) << 4);

}

}

#ifdef OBSOLETE

// Convert the internal RGB 332 to R

static uint8\_t u8g\_ssd1351\_get\_r(uint8\_t colour)

{

//return ((colour & 0xe0) >> 5) \* 9;

//return ((colour & 0xe0) >> 5) \* 8;

return ((colour & 0xe0) >> 2) ;

}

// Convert the internal RGB 332 to G

static uint8\_t u8g\_ssd1351\_get\_g(uint8\_t colour)

{

//return ((colour & 0x1c) >> 2) \* 9;

//return ((colour & 0x1c) >> 2) \* 8;

return ((colour & 0x1c) << 1);

}

// Convert the internal RGB 332 to B

static uint8\_t u8g\_ssd1351\_get\_b(uint8\_t colour)

{

//return (colour & 0x03) \* 21;

return (colour & 0x03) \* 16;

}

#endif

uint8\_t u8g\_dev\_ssd1351\_128x128\_332\_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\_INIT:

u8g\_InitCom(u8g, dev, U8G\_SPI\_CLK\_CYCLE\_50NS);

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1351\_128x128\_init\_seq);

break;

case U8G\_DEV\_MSG\_STOP:

break;

case U8G\_DEV\_MSG\_PAGE\_FIRST:

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1351\_128x128\_column\_seq);

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

{

u8g\_uint\_t x;

uint8\_t page\_height;

uint8\_t i;

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

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

u8g\_SetChipSelect(u8g, dev, 1);

page\_height = pb->p.page\_y1;

page\_height -= pb->p.page\_y0;

page\_height++;

for( i = 0; i < page\_height; i++ )

{

for (x = 0; x < pb->width; x+=RGB332\_STREAM\_BYTES)

{

u8g\_ssd1351\_to\_stream(ptr);

u8g\_WriteSequence(u8g, dev, RGB332\_STREAM\_BYTES\*3, u8g\_ssd1351\_stream\_bytes);

ptr += RGB332\_STREAM\_BYTES;

}

}

u8g\_SetChipSelect(u8g, dev, 0);

}

break;

case U8G\_DEV\_MSG\_GET\_MODE:

return U8G\_MODE\_R3G3B2;

}

return u8g\_dev\_pb8h8\_base\_fn(u8g, dev, msg, arg);

}

uint8\_t u8g\_dev\_ssd1351\_128x128gh\_332\_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\_INIT:

u8g\_InitCom(u8g, dev, U8G\_SPI\_CLK\_CYCLE\_50NS);

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1351\_128x128gh\_init\_seq);

break;

case U8G\_DEV\_MSG\_STOP:

break;

case U8G\_DEV\_MSG\_PAGE\_FIRST:

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1351\_128x128\_column\_seq);

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

{

u8g\_uint\_t x;

uint8\_t page\_height;

uint8\_t i;

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

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

u8g\_SetChipSelect(u8g, dev, 1);

page\_height = pb->p.page\_y1;

page\_height -= pb->p.page\_y0;

page\_height++;

for( i = 0; i < page\_height; i++ )

{

for (x = 0; x < pb->width; x+=RGB332\_STREAM\_BYTES)

{

u8g\_ssd1351\_to\_stream(ptr);

u8g\_WriteSequence(u8g, dev, RGB332\_STREAM\_BYTES\*3, u8g\_ssd1351\_stream\_bytes);

ptr += RGB332\_STREAM\_BYTES;

}

}

u8g\_SetChipSelect(u8g, dev, 0);

}

break;

case U8G\_DEV\_MSG\_GET\_MODE:

return U8G\_MODE\_R3G3B2;

}

return u8g\_dev\_pb8h8\_base\_fn(u8g, dev, msg, arg);

}

static uint8\_t u8g\_dev\_ssd1351\_128x128\_r[256];

static uint8\_t u8g\_dev\_ssd1351\_128x128\_g[256];

static uint8\_t u8g\_dev\_ssd1351\_128x128\_b[256];

uint8\_t u8g\_dev\_ssd1351\_128x128\_idx\_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\_INIT:

u8g\_InitCom(u8g, dev, U8G\_SPI\_CLK\_CYCLE\_50NS);

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1351\_128x128\_init\_seq);

break;

case U8G\_DEV\_MSG\_STOP:

break;

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

u8g\_dev\_ssd1351\_128x128\_r[ ((u8g\_dev\_arg\_irgb\_t \*)arg)->idx ] = ((u8g\_dev\_arg\_irgb\_t \*)arg)->r;

u8g\_dev\_ssd1351\_128x128\_g[ ((u8g\_dev\_arg\_irgb\_t \*)arg)->idx ] = ((u8g\_dev\_arg\_irgb\_t \*)arg)->g;

u8g\_dev\_ssd1351\_128x128\_b[ ((u8g\_dev\_arg\_irgb\_t \*)arg)->idx ] = ((u8g\_dev\_arg\_irgb\_t \*)arg)->b;

break;

case U8G\_DEV\_MSG\_PAGE\_FIRST:

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1351\_128x128\_column\_seq);

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

{

int x;

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

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

u8g\_SetChipSelect(u8g, dev, 1);

for (x = 0; x < pb->width; x++)

{

u8g\_WriteByte(u8g, dev, u8g\_dev\_ssd1351\_128x128\_r[(\*ptr)>>2]);

u8g\_WriteByte(u8g, dev, u8g\_dev\_ssd1351\_128x128\_g[(\*ptr)>>2]);

u8g\_WriteByte(u8g, dev, u8g\_dev\_ssd1351\_128x128\_b[(\*ptr)>>2]);

ptr++;

}

u8g\_SetChipSelect(u8g, dev, 0);

}

break;

case U8G\_DEV\_MSG\_GET\_MODE:

return U8G\_MODE\_INDEX;

}

return u8g\_dev\_pb8h8\_base\_fn(u8g, dev, msg, arg);

}

void u8g\_ssd1351\_hicolor\_to\_stream(uint8\_t \*ptr)

{

register uint8\_t cnt = RGB332\_STREAM\_BYTES;

register uint8\_t low, high, r, g, b;

uint8\_t \*dest = u8g\_ssd1351\_stream\_bytes;

for( cnt = 0; cnt < RGB332\_STREAM\_BYTES; cnt++ )

{

low = \*ptr++;

high = \*ptr++;

r = high & ~7;

r >>= 2;

b = low & 31;

b <<= 1;

g = high & 7;

g <<= 3;

g |= (low>>5)&7;

\*dest++ = r;

\*dest++ = g;

\*dest++ = b;

}

}

uint8\_t u8g\_dev\_ssd1351\_128x128\_hicolor\_fn(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t msg, void \*arg)

{

switch(msg)

{

case U8G\_DEV\_MSG\_INIT:

u8g\_InitCom(u8g, dev, U8G\_SPI\_CLK\_CYCLE\_50NS);

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1351\_128x128\_init\_seq);

break;

case U8G\_DEV\_MSG\_STOP:

break;

case U8G\_DEV\_MSG\_PAGE\_FIRST:

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1351\_128x128\_column\_seq);

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

{

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

uint8\_t i, j;

uint8\_t page\_height;

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

u8g\_SetChipSelect(u8g, dev, 1);

page\_height = pb->p.page\_y1;

page\_height -= pb->p.page\_y0;

page\_height++;

for( j = 0; j < page\_height; j++ )

{

for (i = 0; i < pb->width; i+=RGB332\_STREAM\_BYTES)

{

u8g\_ssd1351\_hicolor\_to\_stream(ptr);

u8g\_WriteSequence(u8g, dev, RGB332\_STREAM\_BYTES\*3, u8g\_ssd1351\_stream\_bytes);

ptr += RGB332\_STREAM\_BYTES\*2;

}

}

u8g\_SetChipSelect(u8g, dev, 0);

}

break; /\* continue to base fn \*/

case U8G\_DEV\_MSG\_GET\_MODE:

return U8G\_MODE\_HICOLOR;

}

return u8g\_dev\_pbxh16\_base\_fn(u8g, dev, msg, arg);

}

uint8\_t u8g\_dev\_ssd1351\_128x128gh\_hicolor\_fn(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t msg, void \*arg)

{

switch(msg)

{

case U8G\_DEV\_MSG\_INIT:

u8g\_InitCom(u8g, dev, U8G\_SPI\_CLK\_CYCLE\_50NS);

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1351\_128x128gh\_init\_seq);

break;

case U8G\_DEV\_MSG\_STOP:

break;

case U8G\_DEV\_MSG\_PAGE\_FIRST:

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1351\_128x128\_column\_seq);

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

{

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

uint8\_t i, j;

uint8\_t page\_height;

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

u8g\_SetChipSelect(u8g, dev, 1);

page\_height = pb->p.page\_y1;

page\_height -= pb->p.page\_y0;

page\_height++;

for( j = 0; j < page\_height; j++ )

{

for (i = 0; i < pb->width; i+=RGB332\_STREAM\_BYTES)

{

u8g\_ssd1351\_hicolor\_to\_stream(ptr);

u8g\_WriteSequence(u8g, dev, RGB332\_STREAM\_BYTES\*3, u8g\_ssd1351\_stream\_bytes);

ptr += RGB332\_STREAM\_BYTES\*2;

}

}

u8g\_SetChipSelect(u8g, dev, 0);

}

break; /\* continue to base fn \*/

case U8G\_DEV\_MSG\_GET\_MODE:

return U8G\_MODE\_HICOLOR;

}

return u8g\_dev\_pbxh16\_base\_fn(u8g, dev, msg, arg);

}

uint8\_t u8g\_dev\_ssd1351\_128x128\_byte\_buf[WIDTH\*PAGE\_HEIGHT] U8G\_NOCOMMON ;

u8g\_pb\_t u8g\_dev\_ssd1351\_128x128\_byte\_pb = { {PAGE\_HEIGHT, HEIGHT, 0, 0, 0}, WIDTH, u8g\_dev\_ssd1351\_128x128\_byte\_buf};

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128\_332\_sw\_spi = { u8g\_dev\_ssd1351\_128x128\_332\_fn, &u8g\_dev\_ssd1351\_128x128\_byte\_pb, U8G\_COM\_SW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128\_332\_hw\_spi = { u8g\_dev\_ssd1351\_128x128\_332\_fn, &u8g\_dev\_ssd1351\_128x128\_byte\_pb, U8G\_COM\_HW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128gh\_332\_sw\_spi = { u8g\_dev\_ssd1351\_128x128gh\_332\_fn, &u8g\_dev\_ssd1351\_128x128\_byte\_pb, U8G\_COM\_SW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128gh\_332\_hw\_spi = { u8g\_dev\_ssd1351\_128x128gh\_332\_fn, &u8g\_dev\_ssd1351\_128x128\_byte\_pb, U8G\_COM\_HW\_SPI };

//u8g\_dev\_t u8g\_dev\_ssd1351\_128x128\_idx\_sw\_spi = { u8g\_dev\_ssd1351\_128x128\_idx\_fn, &u8g\_dev\_ssd1351\_128x128\_byte\_pb, U8G\_COM\_SW\_SPI };

//u8g\_dev\_t u8g\_dev\_ssd1351\_128x128\_idx\_hw\_spi = { u8g\_dev\_ssd1351\_128x128\_idx\_fn, &u8g\_dev\_ssd1351\_128x128\_byte\_pb, U8G\_COM\_HW\_SPI };

/\* only half of the height, because two bytes are needed for one pixel \*/

u8g\_pb\_t u8g\_dev\_ssd1351\_128x128\_hicolor\_byte\_pb = { {PAGE\_HEIGHT/2, HEIGHT, 0, 0, 0}, WIDTH, u8g\_dev\_ssd1351\_128x128\_byte\_buf};

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128\_hicolor\_sw\_spi = { u8g\_dev\_ssd1351\_128x128\_hicolor\_fn, &u8g\_dev\_ssd1351\_128x128\_hicolor\_byte\_pb, U8G\_COM\_SW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128\_hicolor\_hw\_spi = { u8g\_dev\_ssd1351\_128x128\_hicolor\_fn, &u8g\_dev\_ssd1351\_128x128\_hicolor\_byte\_pb, U8G\_COM\_HW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128gh\_hicolor\_sw\_spi = { u8g\_dev\_ssd1351\_128x128gh\_hicolor\_fn, &u8g\_dev\_ssd1351\_128x128\_hicolor\_byte\_pb, U8G\_COM\_SW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128gh\_hicolor\_hw\_spi = { u8g\_dev\_ssd1351\_128x128gh\_hicolor\_fn, &u8g\_dev\_ssd1351\_128x128\_hicolor\_byte\_pb, U8G\_COM\_HW\_SPI };

uint8\_t u8g\_dev\_ssd1351\_128x128\_4x\_byte\_buf[WIDTH\*PAGE\_HEIGHT\*4] U8G\_NOCOMMON ;

u8g\_pb\_t u8g\_dev\_ssd1351\_128x128\_4x\_332\_byte\_pb = { {PAGE\_HEIGHT\*4, HEIGHT, 0, 0, 0}, WIDTH, u8g\_dev\_ssd1351\_128x128\_4x\_byte\_buf};

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128\_4x\_332\_sw\_spi = { u8g\_dev\_ssd1351\_128x128\_332\_fn, &u8g\_dev\_ssd1351\_128x128\_4x\_332\_byte\_pb, U8G\_COM\_SW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128\_4x\_332\_hw\_spi = { u8g\_dev\_ssd1351\_128x128\_332\_fn, &u8g\_dev\_ssd1351\_128x128\_4x\_332\_byte\_pb, U8G\_COM\_HW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128gh\_4x\_332\_sw\_spi = { u8g\_dev\_ssd1351\_128x128gh\_332\_fn, &u8g\_dev\_ssd1351\_128x128\_4x\_332\_byte\_pb, U8G\_COM\_SW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128gh\_4x\_332\_hw\_spi = { u8g\_dev\_ssd1351\_128x128gh\_332\_fn, &u8g\_dev\_ssd1351\_128x128\_4x\_332\_byte\_pb, U8G\_COM\_HW\_SPI };

u8g\_pb\_t u8g\_dev\_ssd1351\_128x128\_4x\_hicolor\_byte\_pb = { {PAGE\_HEIGHT/2\*4, HEIGHT, 0, 0, 0}, WIDTH, u8g\_dev\_ssd1351\_128x128\_4x\_byte\_buf};

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128\_4x\_hicolor\_sw\_spi = { u8g\_dev\_ssd1351\_128x128\_hicolor\_fn, &u8g\_dev\_ssd1351\_128x128\_4x\_hicolor\_byte\_pb, U8G\_COM\_SW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128\_4x\_hicolor\_hw\_spi = { u8g\_dev\_ssd1351\_128x128\_hicolor\_fn, &u8g\_dev\_ssd1351\_128x128\_4x\_hicolor\_byte\_pb, U8G\_COM\_HW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128gh\_4x\_hicolor\_sw\_spi = { u8g\_dev\_ssd1351\_128x128gh\_hicolor\_fn, &u8g\_dev\_ssd1351\_128x128\_4x\_hicolor\_byte\_pb, U8G\_COM\_SW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1351\_128x128gh\_4x\_hicolor\_hw\_spi = { u8g\_dev\_ssd1351\_128x128gh\_hicolor\_fn, &u8g\_dev\_ssd1351\_128x128\_4x\_hicolor\_byte\_pb, U8G\_COM\_HW\_SPI };

/\*

U8G\_PB\_DEV(u8g\_dev\_ssd1351\_128x128\_332\_sw\_spi, WIDTH, HEIGHT, PAGE\_HEIGHT, u8g\_dev\_ssd1351\_128x128\_332\_fn, U8G\_COM\_SW\_SPI);

U8G\_PB\_DEV(u8g\_dev\_ssd1351\_128x128\_332\_hw\_spi, WIDTH, HEIGHT, PAGE\_HEIGHT, u8g\_dev\_ssd1351\_128x128\_332\_fn, U8G\_COM\_HW\_SPI);

U8G\_PB\_DEV(u8g\_dev\_ssd1351\_128x128\_idx\_sw\_spi, WIDTH, HEIGHT, PAGE\_HEIGHT, u8g\_dev\_ssd1351\_128x128\_idx\_fn, U8G\_COM\_SW\_SPI);

U8G\_PB\_DEV(u8g\_dev\_ssd1351\_128x128\_idx\_hw\_spi, WIDTH, HEIGHT, PAGE\_HEIGHT, u8g\_dev\_ssd1351\_128x128\_idx\_fn, U8G\_COM\_HW\_SPI);

\*/