/\*

u8g\_dev\_ssd1325\_nhd27oled\_bw.c

1-Bit (BW) Driver for SSD1325 Controller (OLED Display)

Horizontal architecture, completly rewritten

Tested with NHD-2.7-12864UCY3

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.

SSD130x Monochrom OLED Controller

SSD131x Character OLED Controller

SSD132x Graylevel OLED Controller

SSD1331 Color OLED Controller

\*/

#include "u8g.h"

/\* width must be multiple of 8, largest value is 248 unless u8g 16 bit mode is enabled \*/

#define WIDTH 128

#define HEIGHT 64

/\* http://www.newhavendisplay.com/app\_notes/OLED\_2\_7\_12864.txt \*/

static const uint8\_t u8g\_dev\_ssd1325\_nhd\_27\_12864\_init\_seq[] PROGMEM = {

U8G\_ESC\_DLY(10), /\* delay 10 ms \*/

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

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 \*/

0x0ae, /\* display off, sleep mode \*/

0x0b3, 0x091, /\* set display clock divide ratio/oscillator frequency (set clock as 135 frames/sec) \*/

0x0a8, 0x03f, /\* multiplex ratio: 0x03f \* 1/64 duty \*/

0x0a2, 0x04c, /\* display offset, shift mapping ram counter \*/

0x0a1, 0x000, /\* display start line \*/

0x0ad, 0x002, /\* master configuration: disable embedded DC-DC, enable internal VCOMH \*/

0x0a0, 0x052, /\* remap configuration, horizontal address increment (bit 2 = 0), enable nibble remap (upper nibble is left, bit 1 = 1) \*/

0x086, /\* full current range (0x084, 0x085, 0x086) \*/

0x0b8, /\* set gray scale table \*/

0x01, 0x011, 0x022, 0x032, 0x043, 0x054, 0x065, 0x076,

0x081, 0x070, /\* contrast, brightness, 0..128, Newhaven: 0x040 \*/

0x0b2, 0x051, /\* frame frequency (row period) \*/

0x0b1, 0x055, /\* phase length \*/

0x0bc, 0x010, /\* pre-charge voltage level \*/

0x0b4, 0x002, /\* set pre-charge compensation level (not documented in the SDD1325 datasheet, but used in the NHD init seq.) \*/

0x0b0, 0x028, /\* enable pre-charge compensation (not documented in the SDD1325 datasheet, but used in the NHD init seq.) \*/

0x0be, 0x01c, /\* VCOMH voltage \*/

0x0bf, 0x002|0x00d, /\* VSL voltage level (not documented in the SDD1325 datasheet, but used in the NHD init seq.) \*/

0x0a4, /\* normal display mode \*/

0x0af, /\* display on \*/

U8G\_ESC\_DLY(50), /\* delay 50 ms \*/

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

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

};

static const uint8\_t u8g\_dev\_ssd1325\_prepare\_row\_seq[] PROGMEM = {

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

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

0x015, /\* column address... \*/

0x000, /\* start at column 0 \*/

0x03f, /\* end at column 63 (which is y == 127), because there are two pixel in one column \*/

0x075, /\* row address... \*/

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

};

static void u8g\_dev\_ssd1325\_prepare\_row(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t delta\_row)

{

uint8\_t row = ((u8g\_pb\_t \*)(dev->dev\_mem))->p.page;

row \*= ((u8g\_pb\_t \*)(dev->dev\_mem))->p.page\_height;

row += delta\_row;

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1325\_prepare\_row\_seq);

u8g\_WriteByte(u8g, dev, row); /\* start at the selected row \*/

u8g\_WriteByte(u8g, dev, row+1); /\* end within the selected row \*/

//u8g\_SetAddress(u8g, dev, 0); /\* instruction mode mode \*/

//u8g\_WriteByte(u8g, dev, 0x05c); /\* write to ram \*/

u8g\_SetAddress(u8g, dev, 1); /\* data mode \*/

}

static const uint8\_t u8g\_dev\_ssd13xx\_sleep\_on[] PROGMEM = {

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

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

0x0ae, /\* display off \*/

U8G\_ESC\_CS(0), /\* disable chip, bugfix 12 nov 2014 \*/

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

};

static const uint8\_t u8g\_dev\_ssd13xx\_sleep\_off[] PROGMEM = {

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

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

0x0af, /\* display on \*/

U8G\_ESC\_DLY(50), /\* delay 50 ms \*/

U8G\_ESC\_CS(0), /\* disable chip, bugfix 12 nov 2014 \*/

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

};

static uint8\_t u8g\_dev\_ssd1325\_nhd27oled\_bw\_fn(u8g\_t \*u8g, u8g\_dev\_t \*dev, uint8\_t msg, void \*arg)

{

switch(msg)

{

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

// return u8g\_pb\_IsIntersection((u8g\_pb\_t \*)(dev->dev\_mem), (u8g\_dev\_arg\_bbx\_t \*)arg);

case U8G\_DEV\_MSG\_INIT:

u8g\_InitCom(u8g, dev, U8G\_SPI\_CLK\_CYCLE\_300NS);

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1325\_nhd\_27\_12864\_init\_seq);

break;

case U8G\_DEV\_MSG\_STOP:

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

{

uint8\_t i;

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

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

u8g\_uint\_t cnt;

cnt = pb->width;

cnt >>= 3;

for( i = 0; i < pb->p.page\_height; i++ )

{

u8g\_dev\_ssd1325\_prepare\_row(u8g, dev, i); /\* this will also enable chip select \*/

u8g\_WriteSequenceBWTo16GrDevice(u8g, dev, cnt, p);

u8g\_SetChipSelect(u8g, dev, 0);

p+=cnt;

}

}

break;

case U8G\_DEV\_MSG\_CONTRAST:

u8g\_SetChipSelect(u8g, dev, 1);

u8g\_SetAddress(u8g, dev, 0); /\* instruction mode \*/

u8g\_WriteByte(u8g, dev, 0x081);

u8g\_WriteByte(u8g, dev, (\*(uint8\_t \*)arg) >> 1);

u8g\_SetChipSelect(u8g, dev, 0);

break;

case U8G\_DEV\_MSG\_SLEEP\_ON:

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd13xx\_sleep\_on);

return 1;

case U8G\_DEV\_MSG\_SLEEP\_OFF:

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd13xx\_sleep\_off);

return 1;

}

return u8g\_dev\_pb8h1\_base\_fn(u8g, dev, msg, arg);

}

static uint8\_t u8g\_dev\_ssd1325\_nhd27oled\_2x\_bw\_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\_300NS);

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd1325\_nhd\_27\_12864\_init\_seq);

break;

case U8G\_DEV\_MSG\_STOP:

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

{

uint8\_t i;

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

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

u8g\_uint\_t cnt;

cnt = pb->width;

cnt >>= 3;

for( i = 0; i < pb->p.page\_height; i++ )

{

u8g\_dev\_ssd1325\_prepare\_row(u8g, dev, i); /\* this will also enable chip select \*/

u8g\_WriteSequenceBWTo16GrDevice(u8g, dev, cnt, p);

u8g\_SetChipSelect(u8g, dev, 0);

p+=cnt;

}

}

break;

case U8G\_DEV\_MSG\_CONTRAST:

u8g\_SetChipSelect(u8g, dev, 1);

u8g\_SetAddress(u8g, dev, 0); /\* instruction mode \*/

u8g\_WriteByte(u8g, dev, 0x081);

u8g\_WriteByte(u8g, dev, (\*(uint8\_t \*)arg) >> 1);

u8g\_SetChipSelect(u8g, dev, 0);

break;

case U8G\_DEV\_MSG\_SLEEP\_ON:

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd13xx\_sleep\_on);

return 1;

case U8G\_DEV\_MSG\_SLEEP\_OFF:

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_ssd13xx\_sleep\_off);

return 1;

}

return u8g\_dev\_pb16h1\_base\_fn(u8g, dev, msg, arg);

}

U8G\_PB\_DEV(u8g\_dev\_ssd1325\_nhd27oled\_bw\_sw\_spi , WIDTH, HEIGHT, 8, u8g\_dev\_ssd1325\_nhd27oled\_bw\_fn, U8G\_COM\_SW\_SPI);

U8G\_PB\_DEV(u8g\_dev\_ssd1325\_nhd27oled\_bw\_hw\_spi , WIDTH, HEIGHT, 8, u8g\_dev\_ssd1325\_nhd27oled\_bw\_fn, U8G\_COM\_HW\_SPI);

U8G\_PB\_DEV(u8g\_dev\_ssd1325\_nhd27oled\_bw\_parallel , WIDTH, HEIGHT, 8, u8g\_dev\_ssd1325\_nhd27oled\_bw\_fn, U8G\_COM\_FAST\_PARALLEL);

uint8\_t u8g\_dev\_ssd1325\_nhd27oled\_2x\_bw\_buf[WIDTH\*2] U8G\_NOCOMMON ;

u8g\_pb\_t u8g\_dev\_ssd1325\_nhd27oled\_2x\_bw\_pb = { {16, HEIGHT, 0, 0, 0}, WIDTH, u8g\_dev\_ssd1325\_nhd27oled\_2x\_bw\_buf};

u8g\_dev\_t u8g\_dev\_ssd1325\_nhd27oled\_2x\_bw\_sw\_spi = { u8g\_dev\_ssd1325\_nhd27oled\_2x\_bw\_fn, &u8g\_dev\_ssd1325\_nhd27oled\_2x\_bw\_pb, U8G\_COM\_SW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1325\_nhd27oled\_2x\_bw\_hw\_spi = { u8g\_dev\_ssd1325\_nhd27oled\_2x\_bw\_fn, &u8g\_dev\_ssd1325\_nhd27oled\_2x\_bw\_pb, U8G\_COM\_HW\_SPI };

u8g\_dev\_t u8g\_dev\_ssd1325\_nhd27oled\_2x\_bw\_parallel = { u8g\_dev\_ssd1325\_nhd27oled\_2x\_bw\_fn, &u8g\_dev\_ssd1325\_nhd27oled\_2x\_bw\_pb, U8G\_COM\_FAST\_PARALLEL };