/\*

u8g\_dev\_uc1610\_dogxl160.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.

\*/

#include "u8g.h"

#define WIDTH 160

#define HEIGHT 104

static const uint8\_t u8g\_dev\_uc1610\_dogxl160\_init\_seq[] PROGMEM = {

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 \*/

0x0f1, /\* set display height-1 \*/

0x067, /\* \*/

0x0c0, /\* SEG & COM normal \*/

0x040, /\* set display start line \*/

0x050, /\* \*/

0x02b, /\* set panelloading \*/

0x0eb, /\* set bias 1/2 \*/

0x081, /\* set contrast \*/

0x05f, /\* \*/

0x089, /\* set auto increment \*/

0x0a6, /\* normal pixel mode \*/

0x0d3, /\* 0xd3=40% RMS separation for gray levels \*/

0x0af, /\* display on \*/

U8G\_ESC\_DLY(100), /\* delay 100 ms \*/

0x0a5, /\* display all points, ST7565, UC1610 \*/

U8G\_ESC\_DLY(100), /\* delay 100 ms \*/

U8G\_ESC\_DLY(100), /\* delay 100 ms \*/

0x0a4, /\* normal display \*/

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

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

};

static const uint8\_t u8g\_dev\_uc1610\_dogxl160\_data\_start[] PROGMEM = {

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

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

0x010, /\* set upper 4 bit of the col adr to 0 \*/

0x000, /\* set lower 4 bit of the col adr to 0 \*/

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

};

static uint8\_t u8g\_dev\_1to2(uint8\_t n)

{

register uint8\_t a,b,c;

a = n;

a &= 1;

n <<= 1;

b = n;

b &= 4;

n <<= 1;

c = n;

c &= 16;

n <<= 1;

n &= 64;

n |= a;

n |= b;

n |= c;

n |= n << 1;

return n;

}

uint8\_t u8g\_dev\_uc1610\_dogxl160\_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\_uc1610\_dogxl160\_init\_seq);

break;

case U8G\_DEV\_MSG\_STOP:

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

{

int i;

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

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_uc1610\_dogxl160\_data\_start);

u8g\_WriteByte(u8g, dev, 0x060 | (pb->p.page\*2) ); /\* select current page 1/2 (UC1610) \*/

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

for( i = 0; i < WIDTH; i++ )

{

u8g\_WriteByte(u8g, dev, u8g\_dev\_1to2( ((uint8\_t \*)(pb->buf))[i] ) );

}

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_uc1610\_dogxl160\_data\_start);

u8g\_WriteByte(u8g, dev, 0x060 | (pb->p.page\*2+1) ); /\* select current page 2/2 (UC1610) \*/

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

for( i = 0; i < WIDTH; i++ )

{

u8g\_WriteByte(u8g, dev, u8g\_dev\_1to2( ((uint8\_t \*)(pb->buf))[i] >> 4 ) );

}

u8g\_SetChipSelect(u8g, dev, 0);

}

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);

return 1;

}

return u8g\_dev\_pb8v1\_base\_fn(u8g, dev, msg, arg);

}

uint8\_t u8g\_dev\_uc1610\_dogxl160\_gr\_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\_uc1610\_dogxl160\_init\_seq);

break;

case U8G\_DEV\_MSG\_STOP:

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

{

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

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_uc1610\_dogxl160\_data\_start);

u8g\_WriteByte(u8g, dev, 0x060 | (pb->p.page) ); /\* select current page (UC1610) \*/

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

if ( u8g\_pb\_WriteBuffer(pb, u8g, dev) == 0 )

return 0;

u8g\_SetChipSelect(u8g, dev, 0);

}

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);

return 1;

}

return u8g\_dev\_pb8v2\_base\_fn(u8g, dev, msg, arg);

}

uint8\_t u8g\_dev\_uc1610\_dogxl160\_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\_uc1610\_dogxl160\_init\_seq);

break;

case U8G\_DEV\_MSG\_STOP:

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

{

int i;

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

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_uc1610\_dogxl160\_data\_start);

u8g\_WriteByte(u8g, dev, 0x060 | (pb->p.page\*4) ); /\* select current page 1/2 (UC1610) \*/

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

for( i = 0; i < WIDTH; i++ )

{

u8g\_WriteByte(u8g, dev, u8g\_dev\_1to2( ((uint8\_t \*)(pb->buf))[i] ) );

}

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_uc1610\_dogxl160\_data\_start);

u8g\_WriteByte(u8g, dev, 0x060 | (pb->p.page\*4+1) ); /\* select current page 2/2 (UC1610) \*/

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

for( i = 0; i < WIDTH; i++ )

{

u8g\_WriteByte(u8g, dev, u8g\_dev\_1to2( ((uint8\_t \*)(pb->buf))[i] >> 4 ) );

}

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_uc1610\_dogxl160\_data\_start);

u8g\_WriteByte(u8g, dev, 0x060 | (pb->p.page\*4+2) ); /\* select current page 1/2 (UC1610) \*/

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

for( i = 0; i < WIDTH; i++ )

{

u8g\_WriteByte(u8g, dev, u8g\_dev\_1to2( ((uint8\_t \*)((uint8\_t \*)(pb->buf)+WIDTH))[i] ) );

}

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_uc1610\_dogxl160\_data\_start);

u8g\_WriteByte(u8g, dev, 0x060 | (pb->p.page\*4+3) ); /\* select current page 2/2 (UC1610) \*/

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

for( i = 0; i < WIDTH; i++ )

{

u8g\_WriteByte(u8g, dev, u8g\_dev\_1to2( ((uint8\_t \*)((uint8\_t \*)(pb->buf)+WIDTH))[i] >> 4 ) );

}

u8g\_SetChipSelect(u8g, dev, 0);

}

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);

return 1;

}

return u8g\_dev\_pb16v1\_base\_fn(u8g, dev, msg, arg);

}

uint8\_t u8g\_dev\_uc1610\_dogxl160\_2x\_gr\_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\_uc1610\_dogxl160\_init\_seq);

break;

case U8G\_DEV\_MSG\_STOP:

break;

case U8G\_DEV\_MSG\_PAGE\_NEXT:

{

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

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_uc1610\_dogxl160\_data\_start);

u8g\_WriteByte(u8g, dev, 0x060 | (pb->p.page\*2) ); /\* select current page (UC1610) \*/

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

if ( u8g\_WriteSequence(u8g, dev, WIDTH, pb->buf) == 0 )

return 0;

u8g\_WriteEscSeqP(u8g, dev, u8g\_dev\_uc1610\_dogxl160\_data\_start);

u8g\_WriteByte(u8g, dev, 0x060 | (pb->p.page\*2+1) ); /\* select current page (UC1610) \*/

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

if ( u8g\_WriteSequence(u8g, dev, WIDTH, (uint8\_t \*)(pb->buf)+WIDTH) == 0 )

return 0;

u8g\_SetChipSelect(u8g, dev, 0);

}

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);

return 1;

}

return u8g\_dev\_pb16v2\_base\_fn(u8g, dev, msg, arg);

}

U8G\_PB\_DEV(u8g\_dev\_uc1610\_dogxl160\_bw\_sw\_spi, WIDTH, HEIGHT, 8, u8g\_dev\_uc1610\_dogxl160\_bw\_fn, U8G\_COM\_SW\_SPI);

U8G\_PB\_DEV(u8g\_dev\_uc1610\_dogxl160\_bw\_hw\_spi, WIDTH, HEIGHT, 8, u8g\_dev\_uc1610\_dogxl160\_bw\_fn, U8G\_COM\_HW\_SPI);

U8G\_PB\_DEV(u8g\_dev\_uc1610\_dogxl160\_gr\_sw\_spi, WIDTH, HEIGHT, 4, u8g\_dev\_uc1610\_dogxl160\_gr\_fn, U8G\_COM\_SW\_SPI);

U8G\_PB\_DEV(u8g\_dev\_uc1610\_dogxl160\_gr\_hw\_spi, WIDTH, HEIGHT, 4, u8g\_dev\_uc1610\_dogxl160\_gr\_fn, U8G\_COM\_HW\_SPI);

uint8\_t u8g\_dev\_uc1610\_dogxl160\_2x\_bw\_buf[WIDTH\*2] U8G\_NOCOMMON ;

u8g\_pb\_t u8g\_dev\_uc1610\_dogxl160\_2x\_bw\_pb = { {16, HEIGHT, 0, 0, 0}, WIDTH, u8g\_dev\_uc1610\_dogxl160\_2x\_bw\_buf};

u8g\_dev\_t u8g\_dev\_uc1610\_dogxl160\_2x\_bw\_sw\_spi = { u8g\_dev\_uc1610\_dogxl160\_2x\_bw\_fn, &u8g\_dev\_uc1610\_dogxl160\_2x\_bw\_pb, U8G\_COM\_SW\_SPI };

u8g\_dev\_t u8g\_dev\_uc1610\_dogxl160\_2x\_bw\_hw\_spi = { u8g\_dev\_uc1610\_dogxl160\_2x\_bw\_fn, &u8g\_dev\_uc1610\_dogxl160\_2x\_bw\_pb, U8G\_COM\_HW\_SPI };

uint8\_t u8g\_dev\_uc1610\_dogxl160\_2x\_gr\_buf[WIDTH\*2] U8G\_NOCOMMON ;

u8g\_pb\_t u8g\_dev\_uc1610\_dogxl160\_2x\_gr\_pb = { {8, HEIGHT, 0, 0, 0}, WIDTH, u8g\_dev\_uc1610\_dogxl160\_2x\_gr\_buf};

u8g\_dev\_t u8g\_dev\_uc1610\_dogxl160\_2x\_gr\_sw\_spi = { u8g\_dev\_uc1610\_dogxl160\_2x\_gr\_fn, &u8g\_dev\_uc1610\_dogxl160\_2x\_gr\_pb, U8G\_COM\_SW\_SPI };

u8g\_dev\_t u8g\_dev\_uc1610\_dogxl160\_2x\_gr\_hw\_spi = { u8g\_dev\_uc1610\_dogxl160\_2x\_gr\_fn, &u8g\_dev\_uc1610\_dogxl160\_2x\_gr\_pb, U8G\_COM\_HW\_SPI };