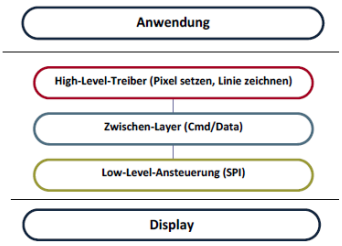
DOGS102 DISPLAY ANSTEUERUNG - TEXT UND GRAFIK

DOGS102 Display Ansteuerung

hawxgamer

Info: dontsendamail@live.de
Schnittstellen und Bus-Systeme
Sommersemester 2014 © Mustafa™



Übung 1 Beschreibung Zeichenfunktionen

Linie, Rechteck,... :

Für die Linie wird das Bresenham-Algorithmus benutzt.

Für das Rechteck liest man vier Koordinatenwerte ein und schreibt symmetrisch die Rechtecksachsen (vertikale Linien und horizontale Linien).

Beim Pixelsetzen lesen wir zwei Koordinatenwerte. Die y Koordinate wird durch die Pageanzahl dividiert. Dadurch wissen wir in welchem page das Pixel liegt. Weiter berechnen wir einen Modulowert aus y Koordinate und page Anzahl und erhalten die Restpixel. Die Restpixel werden anschließend in der richtigen Cachearrayposition mit den Restpixeln verodert und die WriteCache()-Funktion aufgerufen.

Übung 2 Beschreibung Zeichenfunktionen

Text mit direkter y-Wert als Parameterübergabe, invertierte Textdarstellung:

Zuerst muss die y Koordinate durch die page-Anzahl dividiert werden um uns im richtigen page zu positionieren. Anschließend muss die y Koordinate modulo der page-Anzahl berechnet werden um die Restpixel bzw. das Offset zu erhalten. Ich shifte dann meine Daten um den Offset, dabei muss ich auf die beim Shiften verloren gegangenen Bits auch achten.

Für die invertierte Textausgabe mache ich einfach eine xor-Verknüpfung bei meinem Datenbyte.

`z.B:`

```
11111111 xor 00110000 = 11001111
```

Dabei wird auch das Hintergrund invertiert und es passt somit auch zur Anforderung.

Displayausgabe

Source Code

```
#ifndef DOGS102_H
#define DOGS102_H

#include <avr/pgmspace.h>
#include "font.h"

//***************************************************************************
// Definitions
//***************************************************************************
#define DOGS102_PIXELX      102
#define DOGS102_PIXELY      64
#define DOGS102_PAGESIZE    8
#define DOGS102_PAGECNT     (DOGS102_PIXELY / DOGS102_PAGESIZE)
#define DOGS102_CACHE_SIZE    (DOGS102_PIXELX * DOGS102_PAGECNT)

//***************************************************************************
// Prototypes
//***************************************************************************
void DOGS102_Init(void);
void DOGS102_Clear(void);
void DOGS102_SetInvert(unsigned char InvertMode);
void DOGS102_PutStr(unsigned char PosX, unsigned char Line, char * pdata, tFont font
    );
void DOGS102_SetContrast(unsigned char Contrast);
void DrawPixel(unsigned char x, unsigned char y, unsigned char on);
void DrawRect(unsigned char startX, unsigned char startY, unsigned char endX,
    unsigned char endY);
void DrawLine_gbham(int xstart,int ystart,int xend,int yend, unsigned char on);
void DrawrasterCircle(int x0, int y0, int radius, unsigned char on);
void DrawEllipse(int xm, int ym, int a, int b,unsigned char on);
void DrawMyLogo(unsigned char PosX, unsigned char Page);
void DOGS102_Fill(void);



#endif
```

```
#include <avr/io.h>
#include <util/delay.h>
#include "dogs102.h"

//***************************************************************************
// TODO: Dogs102 SPI Command Definitions
//***************************************************************************
#define SET_SCROLL_LINE 0x40
#define SET_SEG_DIRECTION_NORMAL 0xA0
#define SET_SEG_DIRECTION_MIRROR 0xA1
#define SET_COM_DIRECTION_NORMAL 0xC0
#define SET_COM_DIRECTION_MIRROR 0xC1
#define SET_ALL_PIXEL_ON_SRAM 0xA4
#define SET_INVERSE_DISPLAY_NORMAL 0xA6
#define SET_INVERSE_DISPLAY_SRAM_CONTENT 0xA7
```

```
16 #define SET_LCD_BIAS_RATIO 0xA2
17 #define SET_POWER_CONTROL_ALL_ON 0x2F
18 #define SET_VLCD_RESISTOR_RATIO_ALL_ON 0x27
19 #define SET_ELECTRONIC_VOLUME_FIRST 0x81
20 #define SET_ELECTRONIC_VOLUME_SECOND 0x10
21 #define SET_ADV_CONTROL_PROGRAM_0_FIRST 0xFA
22 #define SET_ADV_CONTROL_PROGRAM_0_TC_1_WC_WP_ZERO 0x90
23 #define SET_DISPLAY_ENABLE 0xAF
24 #define ZERO 0x00
25 #define SET_PAGE_ADDRESS 0xB0
26 #define SET_COLUMN_ADDRESS_LSB_bm 0x00  //see uc1701.pdf page 12
27 #define SET_COLUMN_ADDRESS_MSB_bm 0x10
28 #define TAKE_LSB 0x0F
29 #define TAKE_PAGE 0x07
30 #define MY_LOGO_LENGTH 15
31
32 //SPI BMS AND BPS
33 #define SPI_CS PIN4_bp
34 #define SPI_DDR PORTC_DIR
35 #define SPI_MOSI PIN5_bp
36 #define SPI_PORT PORTC
37 #define SPI_SCK PIN7_bp
38 #define SPI_PORT_OUT PORTC_OUT
39 #define CS_PULLUP PORTC_PIN4CTRL=PORT_OPC_WIREDANDPULL_gc
40
41
42 //***************************************************************************
43 // TODO: IO Definitions
44 //***************************************************************************
45 /* RESET pin*/
46 #define RESET_INIT {PORTC.DIRSET=PIN1_bm;}
47 #define RESET_ACTIVE() { PORTC.DIRSET=PIN1_bm; PORTC.OUTSET=PIN1_bm;}
48 #define RESET_INACTIVE() { PORTC.DIRSET=PIN1_bm; PORTC.OUTCLR=PIN1_bm;}
49
50
51 /* Chip Select pins */
52 #define CS_HIGH { PORTC.DIRSET=PIN4_bm; PORTC.OUTSET=PIN4_bm;}
53 #define CS_LOW { PORTC.DIRSET=PIN4_bm; PORTC.OUTCLR=PIN4_bm;}
54
55
56 /* Command and Data pins */
57 #define CD_INIT() { PORTC_DIRSET=PIN0_bm; }
58 #define CD_ACTIVE() { PORTC_DIRSET=PIN0_bm; PORTC_OUTSET=PIN0_bm; }
59 #define CD_INACTIVE() { PORTC_DIRSET=PIN0_bm; PORTC_OUTCLR=PIN0_bm; }
60
61
62 #define DELAY_STABILIZE 10
63 #define MAX_CONTRAST 63
64 #define ASCII_FONTARRAY_SPACE_OFFSET 32
65 //#define F_CPU 8000000
66
67
68 //***************************************************************************
69 // global variables
70 //***************************************************************************
71 static unsigned char LCDCache[DOGS102_CACHE_SIZE];
72
73 static void WriteData(unsigned char data);
74 static void WriteCommand(unsigned char cmd);
75 static void MoveTo_XY(uint8_t PosX, uint8_t Page);
76 static void WriteCache(unsigned char PosX, unsigned char Page, unsigned int Len);
77 static int sgn(int x);  //für das Bresenhamalgorithmus
78 void SPI_Init(void);
79 void SPI_SendByte(unsigned char data);
80 void SPI_CS_Low();
81 void SPI_CS_High();
82
83 //***************************************************************************
84 // Init the display
85 //***************************************************************************
86 void DOGS102_Init(void)
87 {
```

```
88    CS_PULLUP;
89    // init SPI
90    SPI_Init();
91
92    // TODO: Init IOs (Reset, CD) and reset LCD
93
94    //RESET_ACTIVE();
95    RESET_INIT;
96    //RESET_ACTIVE();
97    SPI_CS_High();
98    CS_HIGH;
99    CD_INIT();
100   RESET_INACTIVE();
101   CD_INACTIVE();
102   RESET_ACTIVE();
103
104   _delay_ms(DELAY_STABILIZE);
105
106   WriteCommand(SET_SCROLL_LINE);
107   WriteCommand(SET_SEG_DIRECTION_MIRROR);
108   WriteCommand(SET_COM_DIRECTION_NORMAL);
109   WriteCommand(SET_ALL_PIXEL_ON_SRAM);
110   WriteCommand(SET_INVERSE_DISPLAY_NORMAL);
111   WriteCommand(SET_LCD_BIAS_RATIO);
112   WriteCommand(SET_POWER_CONTROL_ALL_ON);
113   WriteCommand(SET_VLCD_RESISTOR_RATIO_ALL_ON);
114   WriteCommand(SET_ELECTRONIC_VOLUME_FIRST);
115   WriteCommand(SET_ELECTRONIC_VOLUME_SECOND);
116   WriteCommand(SET_ADV_CONTROL_PROGRAM_0_FIRST);
117   WriteCommand(SET_ADV_CONTROL_PROGRAM_0_TC_1_WC_WP_ZERO);
118   WriteCommand(SET_DISPLAY_ENABLE);
119   // TODO: init LCD and then clear it
120   DOGS102_Clear();
121 }
122
123 //*****************************************************************************
124 // Clear the whole display cache (504 Byte)
125 //*****************************************************************************
126 void DOGS102_Clear(void)
127 {
128
129   //delete by colums moving on x-axis by one to delete all the pixels on the pages
        below the x coordinate
130   for (int i=0; i< DOGS102_PAGECNT; i++)
131   {
132     for (int j=0; j<DOGS102_PIXELX;j++)
133     {
134       MoveTo_XY(j,i);
135       WriteData(ZERO);
136     }
137   }
138
139   memset(LCDCache, 0, sizeof(LCDCache));
140   WriteCache(0, 0, DOGS102_CACHE_SIZE);
141 }
142
143 void DOGS102_Fill(void)
144 {
145
146   //delete by colums moving on x-axis by one to delete all the pixels on the pages
        below the x coordinate
147   for (int i=0; i< DOGS102_PAGECNT; i++)
148   {
149     for (int j=0; j<DOGS102_PIXELX;j++)
150     {
151       MoveTo_XY(j,i);
152       WriteData(ZERO);
153     }
154   }
155
156   memset(LCDCache, -1, sizeof(LCDCache));
157   WriteCache(0, 0, DOGS102_CACHE_SIZE);
```

```
158 }
159
160 //*****************************************************************************
161 // Invert display if InvertMode = 1
162 // else enable normal display mode
163 //*****************************************************************************
164 void DOGS102_SetInvert(unsigned char InvertMode)
165 {
166   if(InvertMode==0)
167   WriteCommand(SET_INVERSE_DISPLAY_NORMAL);
168   else if (InvertMode==1)
169   WriteCommand(SET_INVERSE_DISPLAY_SRAM_CONTENT); //invert display
170 }
171
172 //*****************************************************************************
173 // Write a text
174 //*****************************************************************************
175 void DOGS102_PutStr(unsigned char PosX, unsigned char Line, char * pdata, tFont font
      )
176 {
177   unsigned int Len = 0;
178   unsigned char Data = 0;
179   unsigned char l;
180   unsigned char FontIndex;
181   unsigned char ZeroLine;
182
183   while (*pdata)
184   {
185     unsigned char w;
186     FontIndex = 0;
187     for (w = 0; w < font.Width; w++)
188     {
189       ZeroLine = 0;
190       for (l = 0; l < font.Lines; l++)
191       {
192         //Data = // TODO: Zeichen einlesen mit pgm_read_byte
193         Data = pgm_read_byte(&(font.font[(*pdata -ASCII_FONTARRAY_SPACE_OFFSET)*font
       .CharSize+FontIndex])); //das FontIndex benutzen um einfach auf das nächste byte
        zu zeigen
194
195         FontIndex++;
196         ZeroLine |= Data;
197         LCDCache[(DOGS102_PIXELX * (Line + l)) + PosX + Len] = Data;
198       }
199       Len++;
200       if (ZeroLine == 0 && w > 1 && font.Prop == 0)
201       break;
202
203       // falls ein Zeichen genau font.Width Bytes benoetigt, dann am Ende noch
204       // eine Leerspalte einfuegen
205       if (w == font.Width - 1)
206       {
207         Data = 0x00;
208
209         for (l = 0; l < font.Lines; l++)
210         LCDCache[(DOGS102_PIXELX * Line) + PosX + Len] = Data;
211         Len++;
212       }
213     }
214     pdata++;
215   }
216
217   // write all lines needed for the font
218   for (l = 0; l < font.Lines; l++)
219   {
220     WriteCache(PosX, Line + l, Len);
221   }
222 }
223
224 //*****************************************************************************
225 // Adjust the contrast
226 //*****************************************************************************
```

```
227 void DOGS102_SetContrast(unsigned char Contrast)
228 {
229
230   WriteCommand(SET_ELECTRONIC_VOLUME_FIRST);
231   if(Contrast>MAX_CONTRAST)
232   Contrast=MAX_CONTRAST;
233   WriteCommand(Contrast);
234
235 }
236
237 //*****************************************************************************
238 //*****************************************************************************
239 // static functions
240 //*****************************************************************************
241 //*****************************************************************************
242
243 //*****************************************************************************
244 // Write a data byte
245 //*****************************************************************************
246 static void WriteData(unsigned char data)
247 {
248   SPI_CS_Low();
249   CD_ACTIVE();
250   SPI_SendByte(data);
251   SPI_CS_High();
252 }
253
254 //*****************************************************************************
255 // Write a command byte
256 //*****************************************************************************
257 static void WriteCommand(unsigned char cmd)
258 {
259
260   SPI_CS_Low();
261   CD_INACTIVE();
262   SPI_SendByte(cmd);
263   SPI_CS_High();
264 }
265
266 static void MoveTo_XY(uint8_t PosX, uint8_t Page)
267 {
268   // Set Page
269   if(Page>(DOGS102_PAGESIZE-1))
270   Page=DOGS102_PAGESIZE-1;
271
272   WriteCommand(SET_PAGE_ADDRESS | (Page & TAKE_PAGE));
273   //WriteCommand(SET_PAGE_ADDRESS | (Page));
274
275   // Set Column Address
276   if (PosX>DOGS102_PIXELX-1)
277   PosX=DOGS102_PIXELX-1;
278
279   WriteCommand(SET_COLUMN_ADDRESS_LSB_bm | (PosX & TAKE_LSB));
280   WriteCommand(SET_COLUMN_ADDRESS_MSB_bm | (PosX >>4));
281 }
282
283 //*****************************************************************************
284 // write cache to display
285 //*****************************************************************************
286 static void WriteCache(unsigned char PosX, unsigned char Page, unsigned int Len)
287 {
288   int temp;
289   if(Page>(DOGS102_PAGESIZE-1))
290   Page=DOGS102_PAGESIZE-1;
291
292   if (PosX>DOGS102_PIXELX-1)
293   PosX=DOGS102_PIXELX-1;
294
295   MoveTo_XY(PosX,Page);
296   temp=PosX;  //Startposition an temp weitergeben
297   for(unsigned int i=0;i<Len;i++)
298   {
```

```
299       if(temp==(DOGS102_PIXELX-1))
300       {
301         Page++; //erhöhe aufs nächste page
302         if(Page>(DOGS102_PAGECNT-1))
303         Page = (DOGS102_PAGECNT-1); //eigentlich unschön
304         temp=0;
305         MoveTo_XY(0,Page);  //Neupositionierung im neuen page
306       }
307       WriteData(LCDCache[(DOGS102_PIXELX * Page) + temp]);
308       temp++;
309     }
310   }
311
312   /* signum function */
313   static int sgn(int x){
314     return (x > 0) ? 1 : (x < 0) ? -1 : 0;
315   }
316
317   void DrawPixel(unsigned char x, unsigned char y, unsigned char on)  //on==1 Pixel
          schreiben on==0 Pixel löschen
318   {
319     unsigned char page, pageBit, byte;
320     if(y>(DOGS102_PIXELY-1))
321     y=(DOGS102_PIXELY-1);
322
323     if (x>DOGS102_PIXELX-1)
324     x=DOGS102_PIXELX-1;
325     page   = y / (DOGS102_PAGECNT); //konvertiere y Koordinate in pages
326     pageBit = y % (DOGS102_PAGECNT);  //die Restpixel
327
328     if (on == 0) LCDCache[(DOGS102_PIXELX * page) + x] &= ~(1<<pageBit);
329     else         LCDCache[(DOGS102_PIXELX * page) + x] |=  (1<<pageBit);
330
331     WriteCache(x, page, 1);
332     //WriteCache(0,0,DOGS102_CACHE_SIZE);
333   }
334
335   void DrawRect(unsigned char startX, unsigned char startY, unsigned char endX,
          unsigned char endY)
336   {
337     for(uint8_t i=startY; i<endY; i++)
338     {
339       DrawPixel(startX,i,1);
340       DrawPixel(endX,i,1);
341     }
342
343     for(uint8_t i=startX; i<endX; i++)
344     {
345       DrawPixel(i,startY,1);
346       DrawPixel(i,endY,1);
347     }
348   }
349
350   void DrawMyLogo(unsigned char PosX, unsigned char Page)
351   {
352     //unsigned int arr[]={0x3FFC,0x0333,0x0333,0x0333,0x0333,0x0333,0x0333,0x0CCF}; //
          Copyright R
353     unsigned int arr[]={0x03,0x01,0xFF,0xFF,0x80,0x80,0xFF,0xFF,0x80,0x80,0xFF,0xFF,0
          x80,0x80};
354     //unsigned int arr[]={0x80,0x80,0xFF,0xFF,0x80,0x80,0xFF,0xFF,0x80,0x80,0xFF,0xFF
          ,0x01,0x03};
355
356     if(Page>(DOGS102_PAGESIZE-1))
357     Page=DOGS102_PAGECNT-1;
358
359     if (PosX>DOGS102_PIXELX-MY_LOGO_LENGTH)
360     PosX=DOGS102_PIXELX-MY_LOGO_LENGTH;
361
362     for(int i=0; i<=(sizeof(arr)/sizeof(unsigned int));i++)
363     {
364       LCDCache[(DOGS102_PIXELX*Page)+PosX+i]=arr[i];
365     }
```

```
366   WriteCache(PosX,Page,(sizeof(arr)/sizeof(unsigned int)));
367   //DrawrasterCircle(PosX+MY_LOGO_LENGTH/4,Page*DOGS102_PAGESIZE+2,8,1);
368 }
369
370 //***************************************************************************
371 // Bresenham algorithm functions
372 //***************************************************************************
373 //Beschreibung siehe Wikipedia
374 void DrawLine_gbham(int xstart,int ystart,int xend,int yend, unsigned char on)
375 /*-------------------------------------------------------------
376 * Bresenham-Algorithmus: Linien auf Rastergeräten zeichnen
377 *
378 * Eingabeparameter:
379 *    int xstart, ystart        = Koordinaten des Startpunkts
380 *    int xend, yend            = Koordinaten des Endpunkts
381 *
382 * Ausgabe:
383 *    void SetPixel(int x, int y) setze ein Pixel in der Grafik
384 *         (wird in dieser oder aehnlicher Form vorausgesetzt)
385 *-------------------------------------------------------------
386 */
387 {
388   int x, y, t, dx, dy, incx, incy, pdx, pdy, ddx, ddy, es, el, err;
389
390   /* Entfernung in beiden Dimensionen berechnen */
391   dx = xend - xstart;
392   dy = yend - ystart;
393
394   /* Vorzeichen des Inkrements bestimmen */
395   incx = sgn(dx);
396   incy = sgn(dy);
397   if(dx<0) dx = -dx;
398   if(dy<0) dy = -dy;
399
400   /* feststellen, welche Entfernung größer ist */
401   if (dx>dy)
402   {
403     /* x ist schnelle Richtung */
404     pdx=incx; pdy=0;    /* pd. ist Parallelschritt */
405     ddx=incx; ddy=incy; /* dd. ist Diagonalschritt */
406     es =dy;   el =dx;   /* Fehlerschritte schnell, langsam */
407   } else
408   {
409     /* y ist schnelle Richtung */
410     pdx=0;    pdy=incy; /* pd. ist Parallelschritt */
411     ddx=incx; ddy=incy; /* dd. ist Diagonalschritt */
412     es =dx;   el =dy;   /* Fehlerschritte schnell, langsam */
413   }
414
415   /* Initialisierungen vor Schleifenbeginn */
416   x = xstart;
417   y = ystart;
418   err = el/2;
419   DrawPixel(x,y,on);
420
421   /* Pixel berechnen */
422   for(t=0; t<el; ++t) /* t zaehlt die Pixel, el ist auch Anzahl */
423   {
424     /* Aktualisierung Fehlerterm */
425     err -= es;
426     if(err<0)
427     {
428       /* Fehlerterm wieder positiv (>=0) machen */
429       err += el;
430       /* Schritt in langsame Richtung, Diagonalschritt */
431       x += ddx;
432       y += ddy;
433     } else
434     {
435       /* Schritt in schnelle Richtung, Parallelschritt */
436       x += pdx;
437       y += pdy;
```

```
438       }
439       DrawPixel(x,y,on);
440     }
441     } /* gbham() */
442
443     void DrawrasterCircle(int x0, int y0, int radius, unsigned char on)
444     {
445       int f = 1 - radius;
446       int ddF_x = 0;
447       int ddF_y = -2 * radius;
448       int x = 0;
449       int y = radius;
450
451       DrawPixel(x0, y0 + radius,on);
452       DrawPixel(x0, y0 - radius,on);
453       DrawPixel(x0 + radius, y0,on);
454       DrawPixel(x0 - radius, y0,on);
455
456       while(x < y)
457       {
458         if(f >= 0)
459         {
460           y--;
461           ddF_y += 2;
462           f += ddF_y;
463         }
464         x++;
465         ddF_x += 2;
466         f += ddF_x + 1;
467
468         DrawPixel(x0 + x, y0 + y,on);
469         DrawPixel(x0 - x, y0 + y,on);
470         DrawPixel(x0 + x, y0 - y,on);
471         DrawPixel(x0 - x, y0 - y,on);
472         DrawPixel(x0 + y, y0 + x,on);
473         DrawPixel(x0 - y, y0 + x,on);
474         DrawPixel(x0 + y, y0 - x,on);
475         DrawPixel(x0 - y, y0 - x,on);
476       }
477     }
478
479     void DrawEllipse(int xm, int ym, int a, int b,unsigned char on)
480     {
481       int dx = 0, dy = b; /* im I. Quadranten von links oben nach rechts unten */
482       long a2 = a*a, b2 = b*b;
483       long err = b2-(2*b-1)*a2, e2; /* Fehler im 1. Schritt */
484
485       do {
486         DrawPixel(xm+dx, ym+dy,on); /* I. Quadrant */
487         DrawPixel(xm-dx, ym+dy,on); /* II. Quadrant */
488         DrawPixel(xm-dx, ym-dy,on); /* III. Quadrant */
489         DrawPixel(xm+dx, ym-dy,on); /* IV. Quadrant */
490
491         e2 = 2*err;
492         if (e2 <  (2*dx+1)*b2) { dx++; err += (2*dx+1)*b2; }
493         if (e2 > -(2*dy-1)*a2) { dy--; err -= (2*dy-1)*a2; }
494       } while (dy >= 0);
495
496       while (dx++ < a) { /* fehlerhafter Abbruch bei flachen Ellipsen (b=1) */
497         DrawPixel(xm+dx, ym,on); /* -> Spitze der Ellipse vollenden */
498         DrawPixel(xm-dx, ym,on);
499       }
500     }
501
502     //************************************************************************
503     // Init the hardware SPI
504     //************************************************************************
505     void SPI_Init(void)
506     {
507       // Init Serial Ports
508       SPI_DDR |= (1<<SPI_SCK)|(1<<SPI_MOSI)|(1<<SPI_CS);
509       SPI_PORT_OUT |= (1<<SPI_SCK); //anfangs SPI_PORT
```

```
510
511     // SPI Modus wählen, Vorteiler wählen , SPI aktivieren, Double Speed aktivieren,
          SPI Master Modus aktivieren
512     SPIC.CTRL =  SPI_MODE_0_gc | SPI_PRESCALER_DIV4_gc | (1<< SPI_ENABLE_bp)|(1<<
         SPI_CLK2X_bp) |(1<< SPI_MASTER_bp);
513
514     // Interrupt ausschalten
515     SPIC.INTCTRL = SPI_INTLVL_OFF_gc;
516     // SPI Status auf 0 setzen
517     SPIC.STATUS = 0;
518   }
519
520   //*****************************************************************************
521   // Send a byte via SPI
522   //*****************************************************************************
523   void SPI_SendByte(unsigned char data)
524   {
525     SPIC.DATA = data;   //Sendet ein Byte
526     while(!(SPIC.STATUS & (1<<SPI_IF_bp))); //Wartet bis Byte gesendet wurde
527   }
528
529   void SPI_CS_Low()
530   {
531     CS_LOW;
532   }
533
534   void SPI_CS_High()
535   {
536     CS_HIGH;
537   }
```

```
1   /*
2   * DisplayTreiber_DOGS102.c
3   *
4   * Created: 10.03.2014 21:19:42
5   *  Author: hawxgamer
6   */
7
8
9   #include <avr/io.h>
10  #include "dogs102.h"
11  #include "font5x7.h"
12  #include "font12x16.h"
13
14
15  int main(void)
16  {
17
18    DOGS102_Init();
19    DOGS102_Clear();
20
21    //DrawPixel(50, 30,1);
22
23
24    //DOGS102_SetInvert(1);
25
26    tFont bla=Font5x7;
27    DOGS102_Clear();
28
29
30    DOGS102_PutStr(40,4,"Assassin's Creed World", bla );
31    bla=Font12x16;
32    DOGS102_PutStr(40,6,"Brotherhood", bla );
33
34    DrawPixel(10,10,1);
35    DrawPixel(10,11,1);
36    DrawPixel(10,12,0);
37    DrawPixel(10,13,1);
38    DrawRect(20,20,60,30);
```

```
39    DrawLine_gbham(20,20,60,30,1);
40
41    DrawrasterCircle(30,30,20,1);
42    DrawEllipse(40,40,10,30,1);
43    //DOGS102_SetContrast(30);
44    DrawMyLogo(80,2);
45
46
47    while(1)
48    {
49      //TODO:: Please write your application code
50    }
51  }
```