```c
/**
 ******************************************************************************
  * C Library for the FT800 EVE module
 ******************************************************************************
  * @author  Samuel Ruhl, Alexander Meier
  * @date    2017-04-04
  * @file    ft800.c
  * @brief   FT800 Display Controller Library
  *          This file contains the initialization and functions for
  *          the FT800 EVE module. Inspired by Akos Pasztor.
  * @info    http://www.ftdichip.com/Products/ICs/FT800.html
 ******************************************************************************
**/

#include <Dave.h>

#include "spi.h"
#include "ft800.h"

#include <stdlib.h>
#include <string.h>

/*
    Function: HOST_MEM_READ_STR
    ARGS:     addr: 24 Bit Command Address
              pnt:  output buffer for read data
              len:  length of bytes to be read

    Description: Reads len(n) bytes of data, starting at addr into pnt (buffer)
*/
void HOST_MEM_READ_STR(uint32_t addr, uint8_t *pnt, uint8_t len)
{
  FT_spi_select();
  SPI_send(((addr>>16)&0x3F) );         // Send out bits 23:16 of addr, bits 7:6 of this byte must be 00
  SPI_send(((addr>>8)&0xFF));           // Send out bits 15:8 of addr
  SPI_send((addr&0xFF));                // Send out bits 7:0 of addr

  SPI_send(0);                          // Send out DUMMY (0) byte

  while(len--){                         // While Len > 0 Read out n bytes
    *pnt++ = SPI_send(0);
  }
  FT_spi_deselect();
}

```

```
46  /*
47      Function: HOST_MEM_WR_STR
48      ARGS:     addr: 24 Bit Command Address
49                pnt:  input buffer of data to send
50                len:  length of bytes to be send
51
52      Description: Writes len(n) bytes of data from pnt (buffer) to addr
53  */
54  void HOST_MEM_WR_STR(uint32_t addr, uint8_t *pnt, uint8_t len)
55  {
56    FT_spi_select();
57    SPI_send(((addr>>16)&0x3F)|0x80);     // Send out 23:16 of addr, bits
        7:6 of this byte must be 10
58    SPI_send(((addr>>8)&0xFF));           // Send out bits 15:8 of addr
59    SPI_send((addr&0xFF));                // Send out bits 7:0 of addr
60
61    while(len--)                          // While Len > 0 Write *pnt (then
        increment pnt)
62      SPI_send(*pnt++);
63
64    FT_spi_deselect();
65  }
66
67  /*
68      Function: HOST_CMD_WRITE
69      ARGS:     CMD:  5 bit Command
70
71      Description: Writes Command to FT800
72  */
73  void HOST_CMD_WRITE(uint8_t CMD)
74  {
75    FT_spi_select();
76    SPI_send((uint8_t)(CMD|0x40));        // Send out Command, bits 7:6 must
         be 01
77    SPI_send(0x00);
78    SPI_send(0x00);
79    FT_spi_deselect();
80  }
81
82  void HOST_CMD_ACTIVE(void)
83  {
84    FT_spi_select();
85    SPI_send(0x00);
86    SPI_send(0x00);
87    SPI_send(0x00);
88    FT_spi_deselect();
89  }
90
91  /*
92      Function: HOST_MEM_WR8
93      ARGS:     addr: 24 Bit Command Address
94                data: 8bit Data Byte
95
```

```
 96      Description: Writes 1 byte of data to addr
 97  */
 98  void HOST_MEM_WR8(uint32_t addr, uint8_t data)
 99  {
100    FT_spi_select();
101    SPI_send((addr>>16)|0x80);
102    SPI_send(((addr>>8)&0xFF));
103    SPI_send((addr&0xFF));
104
105    SPI_send(data);
106
107    FT_spi_deselect();
108  }
109
110  /*
111      Function: HOST_MEM_WR16
112      ARGS:     addr: 24 Bit Command Address
113                data: 16bit (2 bytes)
114
115      Description: Writes 2 bytes of data to addr
116  */
117  void HOST_MEM_WR16(uint32_t addr, uint32_t data)
118  {
119    FT_spi_select();
120    SPI_send((addr>>16)|0x80);
121    SPI_send(((addr>>8)&0xFF));
122    SPI_send((addr&0xFF));
123
124    /* Little-Endian: Least Significant Byte to: smallest address */
125    SPI_send( (uint8_t)((data&0xFF)) );    //byte 0
126    SPI_send( (uint8_t)((data>>8)) );      //byte 1
127
128    FT_spi_deselect();
129  }
130
131  /*
132      Function: HOST_MEM_WR32
133      ARGS:     addr: 24 Bit Command Address
134                data: 32bit (4 bytes)
135
136      Description: Writes 4 bytes of data to addr
137  */
138  void HOST_MEM_WR32(uint32_t addr, uint32_t data)
139  {
140    FT_spi_select();
141    SPI_send((addr>>16)|0x80);
142    SPI_send(((addr>>8)&0xFF));
143    SPI_send((addr&0xFF));
144
145    SPI_send( (uint8_t)(data&0xFF) );
146    SPI_send( (uint8_t)((data>>8)&0xFF) );
147    SPI_send( (uint8_t)((data>>16)&0xFF) );
148    SPI_send( (uint8_t)((data>>24)&0xFF) );
```

```
149
150   FT_spi_deselect();
151 }
152
153 /*
154     Function: HOST_MEM_RD8
155     ARGS:     addr: 24 Bit Command Address
156
157     Description: Returns 1 byte of data from addr
158 */
159 uint8_t HOST_MEM_RD8(uint32_t addr)
160 {
161   uint8_t data_in;
162
163   FT_spi_select();
164   SPI_send((uint8_t)((addr>>16)&0x3F));
165   SPI_send((uint8_t)((addr>>8)&0xFF));
166   SPI_send((uint8_t)(addr));
167   SPI_send(0);
168
169   data_in = SPI_rec();
170
171   FT_spi_deselect();
172   return data_in;
173 }
174
175 /*
176     Function: HOST_MEM_RD16
177     ARGS:     addr: 24 Bit Command Address
178
179     Description: Returns 2 byte of data from addr in a 32bit variable
180 */
181 uint32_t HOST_MEM_RD16(uint32_t addr)
182 {
183   uint8_t data_in = 0;
184   uint32_t data = 0;
185   uint8_t i;
186
187   FT_spi_select();
188   SPI_send(((addr>>16)&0x3F));
189   SPI_send(((addr>>8)&0xFF));
190   SPI_send((addr&0xFF));
191   SPI_send(0);
192
193   for(i=0;i<2;i++)
194   {
195     data_in = SPI_rec();
196     data |= ( ((uint32_t)data_in) << (8*i) );
197   }
198
199   FT_spi_deselect();
200   return data;
201 }
```

```
202
203 /*
204     Function: HOST_MEM_RD32
205     ARGS:     addr: 24 Bit Command Address
206
207     Description: Returns 4 byte of data from addr in a 32bit variable
208 */
209 uint32_t HOST_MEM_RD32(uint32_t addr)
210 {
211   uint8_t data_in = 0;
212   uint32_t data = 0;
213   uint8_t i;
214
215   FT_spi_select();
216   SPI_send(((addr>>16)&0x3F));
217   SPI_send(((addr>>8)&0xFF));
218   SPI_send((addr&0xFF));
219   SPI_send(0);
220
221   for(i=0;i<4;i++)
222   {
223     data_in = SPI_rec();
224     data |= ( ((uint32_t)data_in) << (8*i) );
225   }
226
227   FT_spi_deselect();
228   return data;
229 }
230
231 /*** CMD Functions
     ***************************************************************/
232 uint8_t cmd_execute(uint32_t data)
233 {
234     uint32_t cmdBufferRd = 0;
235     uint32_t cmdBufferWr = 0;
236
237     cmdBufferRd = HOST_MEM_RD32(REG_CMD_READ);
238     cmdBufferWr = HOST_MEM_RD32(REG_CMD_WRITE);
239
240     uint32_t cmdBufferDiff = cmdBufferWr-cmdBufferRd;
241
242     if( (4096-cmdBufferDiff) > 4)
243     {
244         HOST_MEM_WR32(RAM_CMD + cmdBufferWr, data);
245         HOST_MEM_WR32(REG_CMD_WRITE, cmdBufferWr + 4);
246         return 1;
247     }
248     return 0;
249 }
250
251 uint8_t cmd(uint32_t data)
252 {
253     uint8_t tryCount = 255;
```

```
254     for(tryCount = 255; tryCount > 0; --tryCount)
255     {
256         if(cmd_execute(data)) { return 1; }
257     }
258     return 0;
259 }
260
261 uint8_t cmd_ready(void)
262 {
263     uint32_t cmdBufferRd = HOST_MEM_RD32(REG_CMD_READ);
264     uint32_t cmdBufferWr = HOST_MEM_RD32(REG_CMD_WRITE);
265
266     return (cmdBufferRd == cmdBufferWr) ? 1 : 0;
267 }
268
269 /*** Track ↵
      ********************************************************************** ↵
      /
270 void cmd_track(int16_t x, int16_t y, int16_t w, int16_t h, int16_t tag)
271 {
272     cmd(CMD_TRACK);
273     cmd( ((uint32_t)y<<16)|(x & 0xffff) );
274     cmd( ((uint32_t)h<<16)|(w & 0xffff) );
275     cmd( (uint32_t)tag );
276 }
277
278 /*** Draw Spinner ↵
      ********************************************************************/
279 void cmd_spinner(int16_t x, int16_t y, uint16_t style, uint16_t scale)
280 {
281     cmd(CMD_SPINNER);
282     cmd( ((uint32_t)y<<16)|(x & 0xffff) );
283     cmd( ((uint32_t)scale<<16)|style );
284
285 }
286
287 /*** Draw Slider ↵
      ********************************************************************/
288 void cmd_slider(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t ↵
      options, uint16_t val, uint16_t range)
289 {
290     cmd(CMD_SLIDER);
291     cmd( ((uint32_t)y<<16)|(x & 0xffff) );
292     cmd( ((uint32_t)h<<16)|(w & 0xffff) );
293     cmd( ((uint32_t)val<<16)|(options & 0xffff) );
294     cmd( (uint32_t)range );
295 }
296
297 /*** Draw Text ↵
      ********************************************************************/
298 void cmd_text(int16_t x, int16_t y, int16_t font, uint16_t options, const ↵
      char* str)
299 {
```

```
300     /*
301         i: data pointer
302         q: str  pointer
303         j: loop counter
304     */
305
306     uint16_t i,j,q;
307     const uint16_t length = strlen(str);
308     if(!length) return ;
309
310     uint32_t* data = (uint32_t*) calloc((length/4)+1, sizeof(uint32_t));
311
312     q = 0;
313     for(i=0; i<(length/4); ++i, q=q+4)
314     {
315         data[i] = (uint32_t)str[q+3]<<24 | (uint32_t)str[q+2]<<16 |
              (uint32_t)str[q+1]<<8 | (uint32_t)str[q];
316     }
317     for(j=0; j<(length%4); ++j, ++q)
318     {
319         data[i] |= (uint32_t)str[q] << (j*8);
320     }
321
322     cmd(CMD_TEXT);
323     cmd( ((uint32_t)y<<16)|(x & 0xffff) );
324     cmd( ((uint32_t)options<<16)|(font & 0xffff) );
325     for(j=0; j<(length/4)+1; ++j)
326     {
327         cmd(data[j]);
328     }
329     free(data);
330 }
331
332 /*** Draw Button
      ************************************************************/
333 void cmd_button(int16_t x, int16_t y, int16_t w, int16_t h, int16_t font,
      uint16_t options, const char* str)
334 {
335     /*
336         i: data pointer
337         q: str  pointer
338         j: loop counter
339     */
340
341     uint16_t i,j,q;
342     const uint16_t length = strlen(str);
343     if(!length) return ;
344
345     uint32_t* data = (uint32_t*) calloc((length/4)+1, sizeof(uint32_t));
346
347     q = 0;
348     for(i=0; i<(length/4); ++i, q=q+4)
349     {
```

```
350         data[i] = (uint32_t)str[q+3]<<24 | (uint32_t)str[q+2]<<16 |
              (uint32_t)str[q+1]<<8 | (uint32_t)str[q];
351     }
352     for(j=0; j<(length%4); ++j, ++q)
353     {
354         data[i] |= (uint32_t)str[q] << (j*8);
355     }
356 
357     cmd(CMD_BUTTON);
358     cmd( ((uint32_t)y<<16)|(x & 0xffff) );
359     cmd( ((uint32_t)h<<16)|(w & 0xffff) );
360     cmd( ((uint32_t)options<<16)|(font & 0xffff) );
361     for(j=0; j<(length/4)+1; ++j)
362     {
363         cmd(data[j]);
364     }
365     free(data);
366 }
367 
368 /*** Draw Keyboard
      **********************************************************************/
369 void cmd_keys(int16_t x, int16_t y, int16_t w, int16_t h, int16_t font,
      uint16_t options, const char* str)
370 {
371     /*
372         i: data pointer
373         q: str  pointer
374         j: loop counter
375     */
376 
377     uint16_t i,j,q;
378     const uint16_t length = strlen(str);
379     if(!length) return ;
380 
381     uint32_t* data = (uint32_t*) calloc((length/4)+1, sizeof(uint32_t));
382 
383     q = 0;
384     for(i=0; i<(length/4); ++i, q=q+4)
385     {
386         data[i] = (uint32_t)str[q+3]<<24 | (uint32_t)str[q+2]<<16 |
              (uint32_t)str[q+1]<<8 | (uint32_t)str[q];
387     }
388     for(j=0; j<(length%4); ++j, ++q)
389     {
390         data[i] |= (uint32_t)str[q] << (j*8);
391     }
392 
393     cmd(CMD_KEYS);
394     cmd( ((uint32_t)y<<16)|(x & 0xffff) );
395     cmd( ((uint32_t)h<<16)|(w & 0xffff) );
396     cmd( ((uint32_t)options<<16)|(font & 0xffff) );
397     for(j=0; j<(length/4)+1; ++j)
398     {
```

```
399             cmd(data[j]);
400         }
401         free(data);
402     }
403
404     /*** Write zero to a block of memory
          ********************************************/
405     void cmd_memzero(uint32_t ptr, uint32_t num)
406     {
407         cmd(CMD_MEMZERO);
408         cmd(ptr);
409         cmd(num);
410     }
411
412     /*** Set FG color
          ********************************************************************/
413     void cmd_fgcolor(uint32_t c)
414     {
415         cmd(CMD_FGCOLOR);
416         cmd(c);
417     }
418
419     /*** Set BG color
          ********************************************************************/
420     void cmd_bgcolor(uint32_t c)
421     {
422         cmd(CMD_BGCOLOR);
423         cmd(c);
424     }
425
426     /*** Set Gradient color
          ************************************************************/
427     void cmd_gradcolor(uint32_t c)
428     {
429         cmd(CMD_GRADCOLOR);
430         cmd(c);
431     }
432
433     /*** Draw Gradient
          ************************************************************************/
434     void cmd_gradient(int16_t x0, int16_t y0, uint32_t rgb0, int16_t x1,
          int16_t y1, uint32_t rgb1)
435     {
436         cmd(CMD_GRADIENT);
437         cmd( ((uint32_t)y0<<16)|(x0 & 0xffff) );
438         cmd(rgb0);
439         cmd( ((uint32_t)y1<<16)|(x1 & 0xffff) );
440         cmd(rgb1);
441     }
442
443     /*** Matrix Functions
          ************************************************************************/
444     void cmd_loadidentity(void)
```

```
445 {
446     cmd(CMD_LOADIDENTITY);
447 }
448 
449 void cmd_setmatrix(void)
450 {
451     cmd(CMD_SETMATRIX);
452 }
453 
454 void cmd_rotate(int32_t angle)
455 {
456     cmd(CMD_ROTATE);
457     cmd(angle);
458 }
459 
460 void cmd_translate(int32_t tx, int32_t ty)
461 {
462     cmd(CMD_TRANSLATE);
463     cmd(tx);
464     cmd(ty);
465 }
466 
```