```
1
2   #include "clock_init.h"
3   #include "io_init.h"
4   #include "lcd_init.h"
5   #include "adc_init.h"
6   #include "timer_init.h"
7   #include "interrupt_init.h"
8
9   const uint32_t LCD_REFRESH = 250;       //LCD Refresh time (ms)
10  const uint32_t LED_REFRESH = 1000;      //LED Flash time (ms)
11  const uint16_t ZFC_PERIOD = 833;        //ZFC period (60Hz * 2) (ms)
12
13  typedef enum {
14    no_fault,
15    EIC_00, //Internal Software Fault
16    EIC_01, //Startup Sequence Fault
17    EIC_10, //User Input Fault
18    EIC_11, //Setpoint Out of Bounds Fault
19    EIC_20, //Reserved
20    EIC_30, //Reserved
21    EIC_40, //Reserved
22    EIC_50, //Reserved
23    EOC_00, //Faulted Drive
24    EOC_01, //ZFC Input not Detected
25    EOC_10, //Overcurrent
26    EOC_20, //Reserved
27    EOC_30, //Reserved
28    EOC_40, //Reserved
29    EOC_50  //Reserved
30  } fault_code_type;
31
32  const char fault_strings[9][6] =  {"      ",
33                                     "EIC_00",
34                                     "EIC_01",
35                                     "EIC_10",
36                                     "EIC_11",
37                                     "EOC_00",
38                                     "EOC_01",
39                                     "EOC_10",
40                                     "EOC_11"};
41
42
43  typedef struct {
44    fault_code_type fault_code;
45  } fault_type;
46
47  typedef enum {
48    init,
49    idle_1,
50    run_1,
51    run_2,
52    faulted
53  } menu_state_type;
54  menu_state_type menu_state = init;
55
56  typedef enum {
57    change,
58    set
59  } menu_set_type;
60  menu_set_type menu_set = set;
61
62  //ADC variables
63  volatile uint32_t adc_channel = 2;
64  volatile uint32_t adc_eoc = 1;
65  volatile uint32_t raw_setpoint = 0;
66
67  //Timing functions and variables
68  volatile uint32_t tick = 0;
69
70  //Menu functions and variables
71  menu_set_type state_change(menu_state_type *menu_state, menu_state_type state);
72  uint32_t lcd_last_refresh = 0;
```

```
 73  uint32_t led_last_refresh = 0;
 74  char lcd_line1[17];
 75  char lcd_line2[17];
 76
 77  //String formatting functions
 78  void string_format(char* lcd_line, char* string);
 79  void setpoint_format(char* lcd_line, uint32_t raw_setpoint);
 80  void voltage_format(char* lcd_line, uint32_t raw_setpoint);
 81  void power_format(char* lcd_line, uint32_t raw_setpoint);
 82  void current_format(char* lcd_line, uint32_t raw_setpoint);
 83  void fault_format(char* lcd_line, fault_type fault);
 84  void empty_format(char* lcd_line);
 85
 86  //Scaling functions
 87  uint16_t setpoint_scale(uint32_t raw_setpoint);
 88
 89  //Fault functions and variables
 90  void set_fault(fault_type* fault, fault_code_type fault_code);
 91  fault_type fault;
 92
 93  int main(void) {
 94
 95    clock_init();
 96    io_init();
 97    lcd_init();
 98    timer_init(240, ZFC_PERIOD, 1, 1);
 99    interrupt_init();
100    SysTick_init(24000000/1000); //Ticks every ms
101    adc_init();
102    lcd_clear();
103
104    fault.fault_code = no_fault;
105
106    while(1) {
107
108      switch (menu_state) { //Inputs
109
110        case init:
111          state_change(&menu_state, idle_1);
112          break;
113
114        case idle_1:
115
116          if (btn_read(2)) {
117              menu_set = state_change(&menu_state, run_1);
118          } else {
119            if (btn_read_all()) menu_set = change; else menu_set = set;
120          }
121
122          break;
123
124        case run_1:
125
126          if (menu_set) {
127            if (btn_read(0)) {
128              menu_set = state_change(&menu_state, run_2);
129            } else if (btn_read(1)) {
130              menu_set = state_change(&menu_state, run_2);
131            } else if (btn_read(3)) {
132              menu_set = state_change(&menu_state, idle_1);
133            }
134          } else {
135            if (btn_read_all()) menu_set = change; else menu_set = set;
136          }
137          break;
138
139        case run_2:
140
141          if (menu_set) {
142            if (btn_read(0)) {
143              menu_set = state_change(&menu_state, run_1);
144            } else if (btn_read(1)) {
```

```
145               menu_set = state_change(&menu_state, run_1);
146             } else if (btn_read(3)) {
147               menu_set = state_change(&menu_state, idle_1);
148             }
149           } else {
150             if (btn_read_all()) menu_set = change; else menu_set = set;
151           }
152           break;
153
154         case faulted:
155
156           if (fault.fault_code == no_fault) set_fault(&fault, EIC_00);
157           menu_set = set;
158           break;
159
160         default:
161           menu_set = state_change(&menu_state, faulted);
162           break;
163
164       }
165
166       if (sw_read(0) && !(menu_state == faulted)) menu_set = state_change(&menu_state, faulted);  //Fault
    Test
167       if (sw_read(1) && !(menu_state == faulted)) raw_setpoint = 0xFFFFF;
168
169       if (raw_setpoint > 0xFFF) {
170         set_fault(&fault, EIC_11);
171         menu_set = state_change(&menu_state, faulted);
172       }
173
174       if (tick >= (lcd_last_refresh + LCD_REFRESH)) { //Menu logic
175
176         switch (menu_state){
177
178           case init:
179             break;
180           case idle_1:
181             string_format(lcd_line1, "Idle            ;");
182             setpoint_format(lcd_line2, raw_setpoint);
183             break;
184           case run_1:
185             current_format(lcd_line1, raw_setpoint);
186             setpoint_format(lcd_line2, raw_setpoint);
187             break;
188           case run_2:
189             voltage_format(lcd_line1, raw_setpoint);
190             power_format(lcd_line2, raw_setpoint);
191             break;
192           case faulted:
193             fault_format(lcd_line1, fault);
194             empty_format(lcd_line2);
195             break;
196           default:
197             menu_set = state_change(&menu_state, faulted);
198             break;
199
200         }
201
202         if (menu_set == change) lcd_clear();
203
204         lcd_display(lcd_line1, 0, 0);
205         lcd_display(lcd_line2, 1, 0);
206         lcd_last_refresh = tick;
207
208         if (tick >= 1000000) {
209           tick = 0;
210           led_last_refresh = 0;
211           lcd_last_refresh = 0;
212         }
213         adc_eoc = 1;
214       }
215
```

```
216       switch (menu_state) { //Outputs
217         case init:
218           disable_zfc();
219           timer_disable();
220           break;
221         case idle_1:
222           disable_zfc();
223           timer_disable();
224           adc_start(2);
225           adc_channel = 2;
226           GPIOA->BSRR |= GPIO_BSRR_BS9;
227           break;
228         case run_1:
229           timer_pulse(setpoint_scale(raw_setpoint));  //Changes timer pulse length based on potentiometer
230           enable_zfc();
231           timer_enable();
232           GPIOA->BSRR |= GPIO_BSRR_BR9;
233           break;
234         case run_2:
235           timer_pulse(setpoint_scale(raw_setpoint));  //Changes timer pulse length based on potentiometer
236           enable_zfc();
237           timer_enable();
238           GPIOA->BSRR |= GPIO_BSRR_BR9;
239           break;
240         case faulted:
241           if (tick >= (led_last_refresh + LED_REFRESH)) {
242             GPIOA->ODR ^= GPIO_ODR_ODR9;
243             led_last_refresh = tick;
244           }
245           disable_zfc();
246           timer_disable();
247           break;
248         default:
249           menu_set = state_change(&menu_state, faulted);
250           break;
251
252       }
253
254       if (adc_eoc == 1) { //Starts ADC Conversion
255         raw_setpoint = adc_get();
256         adc_start(2);
257         adc_eoc = 0;
258       }
259
260     }
261   }
262   //Change menu type, returns menu change
263   menu_set_type state_change(menu_state_type *menu_state, menu_state_type state){
264
265     (*menu_state) = state;
266     return change;
267
268   }
269
270
271   void set_fault(fault_type* fault, fault_code_type fault_code){
272     fault->fault_code = fault_code;
273   }
274
275   void string_format(char* lcd_line, char* string){
276
277     int i = 0;
278     for (; i<17; i++) lcd_line[i] = string[i];
279
280   }
281
282   void setpoint_format(char* lcd_line, uint32_t raw_setpoint){
283
284     uint32_t setpoint = 10000 - ((raw_setpoint * 10000) / 4095);
285     char value[3];
286     char string[17] = "Setpoint:  XX.X%;";
287
```

```
288     uint32_t i = 0, rem = setpoint;
289     for (; i < 4; i++) {
290       value[3 - i] = (char)((rem % 10) + '0');
291       rem /= 10;
292     }
293     if ((value[0] == '0') && (value[1] == '0') && setpoint > 100) string[10] = '1';
294     string[11] = value[0];
295     string[12] = value[1];
296     string[14] = value[2];
297     for (i = 0; i<17; i++) lcd_line[i] = string[i];
298
299
300   }
301
302   void voltage_format(char* lcd_line, uint32_t raw_setpoint){
303
304     uint32_t voltage = (((0xFFF - raw_setpoint) * 1200) / 4095);
305     char value[3];
306     char string[17] = "Voltage:    XXXV;";
307
308     uint32_t i = 0, rem = voltage;
309     for (; i < 4; i++) {
310       value[3 - i] = (char)((rem % 10) + '0');
311       rem /= 10;
312     }
313
314     string[12] = ((value[0] == '0') ? ' ' : value[0]);
315     string[13] = (((value[1] == '0') && (value[0] == '0')) ? ' ' : value[1]);
316     string[14] = value[2];
317     for (i = 0; i<17; i++) lcd_line[i] = string[i];
318
319   }
320
321   void power_format(char* lcd_line, uint32_t raw_setpoint){
322
323     uint32_t power = (((0xFFF - raw_setpoint) * 400) / 4095);
324     char value[3];
325     char string[17] = "Power:      XXXW;";
326
327     uint32_t i = 0, rem = power;
328     for (; i < 4; i++) {
329       value[3 - i] = (char)((rem % 10) + '0');
330       rem /= 10;
331     }
332
333     string[12] = ((value[0] == '0') ? ' ' : value[0]);
334     string[13] = (((value[1] == '0') && (value[0] == '0')) ? ' ' : value[1]);
335     string[14] = value[2];
336     for (i = 0; i<17; i++) lcd_line[i] = string[i];
337
338   }
339   void current_format(char* lcd_line, uint32_t raw_setpoint){
340
341     char value[3];
342     char string[17] = "Current:   X.XXA;";
343     uint32_t current = (((0xFFF - raw_setpoint) * 360) / 4095);
344
345     uint32_t i = 0;
346     uint32_t rem = current;
347     for (; i < 4; i++) {
348       value[3 - i] = (char)((rem % 10) + '0');
349       rem /= 10;
350     }
351
352     string[11] = value[0];
353     string[13] = value[1];
354     string[14] = value[2];
355
356     for (i = 0; i<17; i++) lcd_line[i] = string[i];
357
358   }
359
```

```
360   void fault_format(char* lcd_line, fault_type fault){
361
362     uint32_t i = 0;
363     char string[17] = "Faulted:  XXX_XX;";
364     for (; i<6; i++) string[i + 10] = fault_strings[fault.fault_code][i];
365     for (i = 0; i<17; i++) lcd_line[i] = string[i];
366
367   }
368
369   void empty_format(char* lcd_line){
370     uint32_t i = 0;
371     for (i = 0; i<17; i++) lcd_line[i] = ' ';
372   }
373
374   uint16_t setpoint_scale(uint32_t raw_setpoint){
375
376     return (raw_setpoint * ZFC_PERIOD) / 4095;
377
378   }
379
380   void EXTI0_IRQHandler(void){
381     EXTI->PR |= EXTI_PR_PR0;
382     TIM1->CNT = (uint16_t)0x0;
383     TIM1->CR1 |= TIM_CR1_CEN;
384   }
385
386   void ADC1_IRQHandler(void){
387     ADC1->SR &= ~ADC_SR_EOC;
388     adc_eoc = 1;
389   }
390
391   void SysTick_Handler(void)  {
392     tick++;
393   }
394
395   //EOF
396
```

```c
/*
Filename:
  adc_init.h
Description:
  Contains function declarations for all ADC
  initialization and read/write operations
Author:
  Brant Geddes
  200350415

*/

#include "stm32f10x.h"

void adc_init(void);
void adc_start(uint32_t channel);
uint32_t adc_get(void);

//EOF

```

```c
/*
Filename:
  adc_init.c
Description:
  Contains function definitions for all ADC
  initialization and access functions
Author:
  Brant Geddes
  200350415

*/

#include "adc_init.h"
#include "clock_init.h"
#include "stm32f10x.h"

void adc_init(void) {

  //Enable clocks for PORT A and ADC1
  RCC->APB2ENR |= RCC_APB2ENR_ADC1EN | RCC_APB2ENR_IOPAEN;

  //Power ADC1 on
  ADC1->CR2 |= ADC_CR2_ADON;
  delay(10);
  ADC1->CR2 |= ADC_CR2_CAL;
  while((ADC1->CR2 & ADC_CR2_CAL) == ADC_CR2_CAL);

  //Configure PORT A Pin 1, 2, 3 as analog inputs
  GPIOA->CRL &= ~(GPIO_CRL_CNF1) & ~(GPIO_CRL_CNF2) & ~(GPIO_CRL_CNF3) &
                ~(GPIO_CRL_MODE1) & ~(GPIO_CRL_MODE2) & ~(GPIO_CRL_MODE3);

  //Enable interrupt
  ADC1->CR1 |= ADC_CR1_EOCIE;

  //Select channels to convert
  ADC1->SQR1 &= ~ADC_SQR1_L;
  ADC1->SQR3 |= ADC_SQR3_SQ1_1;

  //Select channel sample time
  ADC1->SMPR2 &= ~ADC_SMPR2_SMP0;

  //Continuous conversion
  //ADC1->CR2 |= ADC_CR2_CONT;

  //Start ADC
  //ADC1->CR2 |= ADC_CR2_ADON;

}

void adc_start(uint32_t channel) {

  ADC1->SQR3 &= ~ADC_SQR3_SQ1;
  ADC1->SQR3 |= ((channel == 2) ? ADC_SQR3_SQ1_1 : (ADC_SQR3_SQ1_1 & ADC_SQR3_SQ1_0));

  ADC1->CR2 |= ADC_CR2_ADON;

}

uint32_t adc_get(void) {

  return ADC1->DR;

}

//EOF

```

```
1   /*
2   Filename:
3     clock_init.h
4   Description:
5     Contains function declarations for all clock
6     initialization and general purpose functions
7   Author:
8     Brant Geddes
9     200350415
10
11  */
12
13  void clock_init(void);
14
15  void delay(int);
16  //EOF
17
```

```c
/*
Filename:
  clock_init.c
Description:
  Contains function definitions for all clock
  initialization and general purpose functions
Author:
  Brant Geddes
  200350415

*/

#include "clock_init.h"
#include "stm32f10x.h"

void clock_init(void) {

  RCC->CFGR |= RCC_CFGR_PLLMULL3 | RCC_CFGR_PLLSRC_PREDIV1;

  RCC->CR |= RCC_CR_PLLON | RCC_CR_HSEON | RCC_CR_HSION;

  while((RCC->CR & RCC_CR_PLLRDY) != RCC_CR_PLLRDY);


}

void delay(int delay_time){

  int i;
  for (i = 0; i < (delay_time * 100); i++);

}

//EOF

```

```c
/*
Filename:
  interrupt_init.h
Description:
  Contains function declarations for all interupt
  initialization and control operations
Author:
  Brant Geddes
  200350415

*/

#include "stm32f10x.h"

void interrupt_init(void);
void disable_zfc(void);
void enable_zfc(void);
void SysTick_init(uint32_t counts);

//EOF

```

```c
/*
Filename:
  interrupt_init.c
Description:
  Contains function definitions for all interupt
  initialization and control operations
Author:
  Brant Geddes
  200350415

*/

#include "interrupt_init.h"
#include "stm32f10x.h"

void interrupt_init(void) {

  RCC->APB2ENR |= RCC_APB2ENR_AFIOEN | RCC_APB2ENR_IOPAEN;
  GPIOA->CRL &= ~GPIO_CRL_MODE0;
  GPIOA->CRL &= ~GPIO_CRL_CNF0_1;
  GPIOA->CRL |= GPIO_CRL_CNF0_0;
  AFIO->EXTICR[0] |= AFIO_EXTICR1_EXTI0_PA;
  EXTI->IMR |= EXTI_IMR_MR0;
  EXTI->FTSR |= EXTI_FTSR_TR0;
  EXTI->RTSR |= EXTI_RTSR_TR0;
  NVIC->ISER[0] |= NVIC_ISER_SETENA_6; //EXTI0 on PA0
  NVIC->ISER[0] |= NVIC_ISER_SETENA_18; //ADC1 EOC flag

}

void disable_zfc(void) {
  EXTI->IMR &= ~EXTI_IMR_MR0;
}

void enable_zfc(void) {
  EXTI->IMR |= EXTI_IMR_MR0;
}

void SysTick_init(uint32_t counts) {

  SysTick->CTRL = 0x0;
  SysTick->VAL = 0x0;
  SysTick->LOAD = counts;
  SysTick->CTRL |= SysTick_CTRL_ENABLE | SysTick_CTRL_CLKSOURCE | SysTick_CTRL_TICKINT;

}

//EOF

```

```
1   /*
2   Filename:
3     b.h
4   Description:
5     Contains function declarations for all IO
6     initialization and read/write operations
7   Author:
8     Brant Geddes
9     200350415
10
11  */
12
13  #include "stm32f10x.h"
14
15  void io_init(void);
16
17  void led_set(int led);
18
19  void led_rset(int led);
20
21  int sw_read(int sw);
22  //
23  //sw_read_all
24  //
25  //  description:
26  //      Returns an int with the first four bits set to
27  //      the status of the four dip-switches
28  //  return type:
29  //      int
30  //
31  int sw_read_all(void);
32
33  int btn_read(int btn);
34
35  int btn_read_all(void);
36
37  //EOF
38
```

```c
/*
Filename:
  io_init.c
Description:
  Contains function definitions for all IO
  initialization and read/write operations
Author:
  Brant Geddes
  200350415

*/

#include "io_init.h"
#include "stm32f10x.h"


void io_init(void){

  RCC->APB2ENR |= RCC_APB2ENR_IOPAEN | RCC_APB2ENR_IOPBEN | RCC_APB2ENR_IOPCEN;

  //Enable PA9, PA10, PA11, and PA12 as general purpose output push-pull, max speed 50MHz
  GPIOA->CRH |= GPIO_CRH_MODE9 | GPIO_CRH_MODE10 | GPIO_CRH_MODE11 | GPIO_CRH_MODE12;
  GPIOA->CRH &= ~GPIO_CRH_CNF9 & ~GPIO_CRH_CNF10 & ~GPIO_CRH_CNF11 & ~GPIO_CRH_CNF12;

  GPIOC->CRH |= GPIO_CRH_MODE8 | GPIO_CRH_MODE9;
  GPIOC->CRH &= ~GPIO_CRH_CNF8 & ~GPIO_CRH_CNF9;

  //Enable PA6, PA7, PC10, PC11 as input with pull up/down
  GPIOA->CRL &= ~GPIO_CRL_MODE5 & ~GPIO_CRL_MODE6 & ~GPIO_CRL_MODE7;
  GPIOA->CRL &= ~GPIO_CRL_CNF5_0 & ~GPIO_CRL_CNF6_0 & ~GPIO_CRL_CNF7_0;
  GPIOA->CRL |= GPIO_CRL_CNF5_1 | GPIO_CRL_CNF6_1 | GPIO_CRL_CNF7_1;
  GPIOB->CRH &= ~GPIO_CRH_MODE8 & ~GPIO_CRH_MODE9;
  GPIOB->CRH &= ~GPIO_CRH_CNF8_0 & ~GPIO_CRH_CNF9_0;
  GPIOB->CRH |= GPIO_CRH_CNF8_1 | GPIO_CRH_CNF9_1;
  GPIOC->CRH &= ~GPIO_CRH_MODE10 & ~GPIO_CRH_MODE11 & ~GPIO_CRH_MODE12;
  GPIOC->CRH &= ~GPIO_CRH_CNF10_0 & ~GPIO_CRH_CNF11_0 & ~GPIO_CRH_CNF12_0;
  GPIOC->CRH |= GPIO_CRH_CNF10_1 | GPIO_CRH_CNF11_1 | GPIO_CRH_CNF12_1;

  GPIOA->ODR |= 0x1E00;

}

void led_set(int led){

  if ((led < 0) | (led > 3)) {
    return;
  } else {
    GPIOA->BSRR |= GPIO_BSRR_BR9<<led;
  }

}

void led_rset(int led) {

  if ((led < 0) | (led > 3)) {
    return;
  } else {
    GPIOA->BSRR |= GPIO_BSRR_BS9<<led;
  }

}

int sw_read(int sw) {

  return !!((1<<sw) & sw_read_all());

}

int sw_read_all(void){

  return  0xF & ~(((GPIOA->IDR & GPIO_IDR_IDR6) |
          (GPIOA->IDR & GPIO_IDR_IDR7))>>6 |
```

```
73              ((GPIOC->IDR & GPIO_IDR_IDR10) |
74              (GPIOC->IDR & GPIO_IDR_IDR11))>>8);
75
76    }
77
78    int btn_read(int btn) {
79
80      return !!((1<<btn) & btn_read_all());
81
82    }
83
84    int btn_read_all(void) {
85
86      return 0xF &
87              ~((((GPIOB->IDR & GPIO_IDR_IDR9) |
88              (GPIOB->IDR & GPIO_IDR_IDR8))>>8) |
89              ((GPIOC->IDR & GPIO_IDR_IDR12)>>10) |
90              ((GPIOA->IDR & GPIO_IDR_IDR5)>>2));
91
92    }
93
94    //EOF
95
```

```
/*
Filename:
  lcd_init.h
Description:
  Contains function declarations for all LCD
  initialization and access functions
Author:
  Brant Geddes
  200350415

*/

#include "stm32f10x.h"

void lcd_init(void);

void lcd_start(void);
//
//lcd_display
//
//Description:
//  Takes a pointer to a character array delimited with ';',
//  a row, and an offset and writes to the lcd
//
void lcd_display(char *, uint32_t, uint32_t);

void lcd_clear(void);

void lcd_latch(void);

void lcd_data_latch(uint32_t);

void lcd_instruction_latch(uint32_t);

void lcd_configure_db_read(void);

void lcd_configure_db_write(void);

void lcd_delay(void);

//EOF

```

```c
/*
Filename:
  lcd_init.c
Description:
  Contains function definitions for all LCD
  initialization and access functions
Author:
  Brant Geddes
  200350415

*/


#include "lcd_init.h"
#include "clock_init.h"
#include "stm32f10x.h"

//#define LCD_CHECK_BUSY_FLAG //not working yet

#define LCD_CLEAR_DISPLAY   ((uint32_t)0x01)
#define LCD_RETURN_HOME     ((uint32_t)0x02)
#define LCD_DISPLAY_SET     ((uint32_t)0x0E)
#define LCD_ENTRY_MODE      ((uint32_t)0x06)
#define LCD_FUNCTION_SET    ((uint32_t)0x38)
#define LCD_DDRAM_SET       ((uint32_t)0x80)

#define LCD_ENA_ON          (GPIO_BSRR_BS1)
#define LCD_RS_ON           (GPIO_BSRR_BS0)
#define LCD_RW_ON           (GPIO_BSRR_BS5)
#define LCD_ENA_OFF         (GPIO_BSRR_BR1)
#define LCD_RS_OFF          (GPIO_BSRR_BR0)
#define LCD_RW_OFF          (GPIO_BSRR_BR5)

void lcd_init(void) {

  //Initialize clocks for PORT B, C
  RCC->APB2ENR |= RCC_APB2ENR_IOPBEN | RCC_APB2ENR_IOPCEN;

  lcd_configure_db_write();

  //Initialize LCD_RS, _RW, and _ENA as general purpose push-pull outputs, 50 MHz
  GPIOB->CRL |= GPIO_CRL_MODE0 | GPIO_CRL_MODE1 | GPIO_CRL_MODE5;
  GPIOB->CRL &= ~(GPIO_CRL_CNF0) & ~(GPIO_CRL_CNF1) & ~(GPIO_CRL_CNF5);

  GPIOB->BSRR |= LCD_ENA_ON | LCD_RS_OFF | LCD_RW_OFF;

  lcd_start();

}

void lcd_start(void) {

  delay(1000);
  GPIOC->BRR |= 0xFF;
  GPIOC->BSRR |= LCD_FUNCTION_SET & 0xFF;
  GPIOB->BSRR |= LCD_ENA_ON; delay(10); GPIOB->BSRR |= LCD_ENA_OFF; delay(10);
  GPIOB->BSRR |= LCD_ENA_ON; delay(10); GPIOB->BSRR |= LCD_ENA_OFF; delay(10);
  GPIOB->BSRR |= LCD_ENA_ON; delay(10); GPIOB->BSRR |= LCD_ENA_OFF; delay(10);

  lcd_instruction_latch(LCD_DISPLAY_SET);
  lcd_instruction_latch(LCD_ENTRY_MODE);
  lcd_instruction_latch(LCD_CLEAR_DISPLAY);

}

void lcd_display(char *message, uint32_t row, uint32_t offset) {

  int i = 0, j = 0, ddram = 0;

  if (offset > 15) return;
  if (row > 1) return;

```

```
73      while (message[i] != ';') {
74        i++;
75        if (i > 16) return;
76      }
77
78      if (offset > 16) return;
79      if (row > 1) return;
80      if (offset + i > 16) return;
81
82      ddram = ((row * 0x40) + offset) | LCD_DDRAM_SET;
83
84      lcd_instruction_latch(ddram);
85
86      for (j = 0; j < i; j++) lcd_data_latch(message[j]);
87
88    }
89
90    void lcd_clear(void) {
91
92      lcd_instruction_latch(LCD_CLEAR_DISPLAY);
93
94    }
95
96    void lcd_latch(void) {
97      GPIOB->BSRR |= LCD_ENA_ON; lcd_delay(); GPIOB->BSRR |= LCD_ENA_OFF;
98    }
99
100   void lcd_data_latch(uint32_t data) {
101
102     GPIOB->BSRR |= LCD_RS_ON;
103
104     GPIOC->BRR |= 0xff;
105     GPIOC->BSRR |= (data) & ((uint32_t)0xFF);
106     lcd_latch();
107
108   }
109
110   void lcd_instruction_latch(uint32_t instruction) {
111
112     GPIOB->BSRR |= LCD_RS_OFF;
113
114     GPIOC->BRR |= 0xFF;
115     GPIOC->BSRR = (instruction) & ((uint32_t)0xFF);
116     lcd_latch();
117
118   }
119
120   void lcd_configure_db_read(void) {
121
122     //Initialize data bus as floating inputs
123     GPIOC->CRL &= ~(GPIO_CRL_MODE0) & ~(GPIO_CRL_MODE1) & ~(GPIO_CRL_MODE2) & ~(GPIO_CRL_MODE3) &
124                   ~(GPIO_CRL_MODE4) & ~(GPIO_CRL_MODE5) & ~(GPIO_CRL_MODE6) & ~(GPIO_CRL_MODE7);
125     GPIOC->CRL &= ~(GPIO_CRL_CNF0) & ~(GPIO_CRL_CNF1) & ~(GPIO_CRL_CNF2) & ~(GPIO_CRL_CNF3) &
126                   ~(GPIO_CRL_CNF4) & ~(GPIO_CRL_CNF5) & ~(GPIO_CRL_CNF6) & ~(GPIO_CRL_CNF7);
127     GPIOC->CRL |= GPIO_CRL_CNF0_0 | GPIO_CRL_CNF1_0 | GPIO_CRL_CNF2_0 | GPIO_CRL_CNF3_0 |
128                   GPIO_CRL_CNF4_0 | GPIO_CRL_CNF5_0 | GPIO_CRL_CNF6_0 | GPIO_CRL_CNF7_0;
129     //Set RW to read
130     GPIOB->BSRR |= LCD_RW_ON;
131
132   }
133
134   void lcd_configure_db_write(void) {
135
136     //Initialize data bus as general purpose push-pull outputs, 50MHz
137     GPIOC->CRL |= GPIO_CRL_MODE0 | GPIO_CRL_MODE1 | GPIO_CRL_MODE2 | GPIO_CRL_MODE3 |
138                   GPIO_CRL_MODE4 | GPIO_CRL_MODE5 | GPIO_CRL_MODE6 | GPIO_CRL_MODE7;
139     GPIOC->CRL &= ~(GPIO_CRL_CNF0) & ~(GPIO_CRL_CNF1) & ~(GPIO_CRL_CNF2) & ~(GPIO_CRL_CNF3) &
140                   ~(GPIO_CRL_CNF4) & ~(GPIO_CRL_CNF5) & ~(GPIO_CRL_CNF6) & ~(GPIO_CRL_CNF7);
141     //Set RW to write
142     GPIOB->BSRR |= LCD_RW_OFF;
143
144   }
```

```
145
146   #ifdef LCD_CHECK_BUSY_FLAG
147
148   void lcd_delay(void) {
149
150     lcd_configure_db_read();
151
152     GPIOB->BSRR |= LCD_RS_OFF;
153     GPIOB->BSRR |= LCD_ENA_OFF;
154     while ((GPIOC->IDR & GPIO_IDR_IDR7) == GPIO_IDR_IDR7) { GPIOB->BSRR |= LCD_ENA_OFF; delay(10);
      GPIOB->BSRR |= LCD_ENA_ON; delay(10); }
155     lcd_configure_db_write();
156
157   }
158
159   #else
160
161   void lcd_delay(void) {
162
163     delay(100);
164
165   }
166
167   #endif
168
169   //EOF
170
```

```c
/*
Filename:
  timer_init.h
Description:
  Contains function declarations for all timer
  initialization and control operations
Author:
  Brant Geddes
  200350415

*/

#include "stm32f10x.h"

void timer_init(uint16_t prescaler, uint16_t period, uint16_t pulse, uint16_t cycles);
void timer_pulse(uint16_t pulse);
void timer_start(void);
void timer_disable(void);
void timer_enable(void);

//EOF

```

```c
/*
Filename:
  timer_init.c
Description:
  Contains function definitions for all timer
  initialization and control operations
Author:
  Brant Geddes
  200350415

*/

#include "timer_init.h"
#include "stm32f10x.h"

void timer_init(uint16_t prescaler, uint16_t period, uint16_t pulse, uint16_t cycles){

  RCC->APB2ENR |= RCC_APB2ENR_TIM1EN; //Start TIM1 clock

  GPIOA->CRH |= GPIO_CRH_MODE8;
  GPIOA->CRH &= ~GPIO_CRH_CNF8;
  GPIOA->CRH |= GPIO_CRH_CNF8_1; //Configure GPIO as alternate function
                                 //Maps timer output to pin PA9

  TIM1->CR1 &= ~(TIM_CR1_DIR | TIM_CR1_CMS); //Select up counter mode

  TIM1->CR1 &= ~TIM_CR1_CKD;

  TIM1->ARR = period; //Set the Autoreload value (period)
  TIM1->CCR2 = pulse; //Set the Pulse value
  TIM1->PSC = prescaler; //Set the Prescaler value
  TIM1->RCR = cycles - 1; //Pulse repeat
  TIM1->EGR = TIM_EGR_UG; //Generate an update event to reload the prescaler

  TIM1->SMCR = 0x0; //Configure slave mode

  TIM1->CR1 |= TIM_CR1_OPM; //Select one-shot pulse mode
  TIM1->CCMR1 &= (uint16_t)~TIM_CCMR1_OC1M;
  TIM1->CCMR1 &= (uint16_t)~TIM_CCMR1_CC1S;
  TIM1->CCMR1 |= TIM_CCMR1_OC1M; //PWM2

  TIM1->CCER &= (uint16_t)~TIM_CCER_CC1P; //Select Channel 1 Output Compare

  TIM1->CCER = TIM_CCER_CC1E; //Enable the Compare output channel 2
  //TIM1->BDTR |= TIM_BDTR_MOE; //Enable the Timer main Output

}

void timer_pulse(uint16_t pulse){

  if (pulse > 825) pulse = 825;
  TIM1->CCR1 = pulse;

}

void timer_start(void) {
  if (!(TIM1->CR1 & TIM_CR1_CEN)) TIM1->CR1 |= TIM_CR1_CEN;
}

void timer_disable(void) {
  TIM1->BDTR &= ~TIM_BDTR_MOE;
}

void timer_enable(void) {
  TIM1->BDTR |= TIM_BDTR_MOE;
}

//EOF

```