

```

1  /*-----
2  * Name:      main.c
3  * Purpose:
4  * Note(s):
5  *-----
6  *
7  *-----*/
8
9  #include "STM32F4xx.h"
10 #include "stm32f4_discovery.h"
11 #include "main_2.h"
12 #include "LED.h"
13 #include "SWT.h"
14 #include "LCD.h"
15 #include "Sqaure.h"
16 #include "DAC.h"
17 #include "DDS.h"
18 #include "FreqMeter.h"
19 #include "hd44780.h"
20 #include "ArbitoryFunc.h"
21 #include "FSK.h"
22 #include <stdio.h>
23
24 volatile uint32_t msTicks;                /* counts 1ms timeTicks */
25 volatile double currentFrequency = 1000;
26 volatile double increment = 1;
27 volatile int function = WAVE_GENERATION;
28 volatile int freqRange = HUNDRED_TO_10K;
29 volatile unsigned char updateFlag = 1;
30 volatile int dutyCycle = 50;
31
32 /*-----
33  MAIN function
34  *-----*/
35 int main (void) {
36
37     __disable_irq();
38     SystemCoreClockUpdate();                /* Get Core Clock Frequency */
39
40     if (SysTick_Config(SystemCoreClock / 1680)) { /* SysTick 1 msec interrupts */
41         while (1);                            /* Capture error */
42     }
43     __enable_irq();
44
45     // Initialise Required Pins
46     BTN_Init();
47     SWTS_Init();
48     LED_Init();
49     init_lcd_driver();
50     hd44780_init(GPIOD, GPIOB, GPIO_Pin_0, GPIO_Pin_1, GPIO_Pin_2, GPIO_Pin_4,
51                 GPIO_Pin_5, GPIO_Pin_6, GPIO_Pin_7, HD44780_LINES_2, HD44780_FONT_5x8);
52     DDS_Init();
53     DACs_Init();
54
55     //Initialise components to defaults
56     DDS_Default_Init();
57     Pulse_Config();
58
59     // Turn on LCD display
60     hd44780_display(true, false, false);
61
62     // Set up interrupts for the blue user button - ie the menu
63     //STM_EVAL_PBInit(BUTTON_USER, BUTTON_MODE_EXTI);
64     Config_menu_interrupt();
65
66     while(1)
67     {
68
69         while(function == WAVE_GENERATION)
70         {
71             uint32_t switchsState;
72
73             if(updateFlag == 1)
74             {
75                 updateFlag = 0;
76                 hd44780_clear();
77                 hd44780_position(0, 0);
78                 hd44780_print("WAVE GENERATION");

```

```

79     }
80
81     // Check for switch presses to change DDS frequency
82     switchsState = SWT_Get();
83
84     if (switchsState == (1UL << 8)) {
85         LED_All_Off();
86         LED_On(0);
87         increment = 0.01;
88         hd44780_print_lines("WAVE GENERATION", "Inc = 0.01      Hz");
89     }
90     else if (switchsState == (1UL << 9)) {
91         LED_All_Off();
92         LED_On(1);
93         increment = 1;
94         hd44780_print_lines("WAVE GENERATION", "Inc = 1          Hz");
95     }
96     else if (switchsState == (1UL << 10)) {
97         LED_All_Off();
98         LED_On(2);
99         increment = 100;
100        hd44780_print_lines("WAVE GENERATION", "Inc = 100        Hz");
101    }
102    else if (switchsState == (1UL << 11)) {
103        LED_All_Off();
104        LED_On(3);
105        increment = 1000;
106        hd44780_print_lines("WAVE GENERATION", "Inc = 1000       Hz");
107    }
108    }
109    else if (switchsState == (1UL << 12)) {
110        LED_All_Off();
111        LED_On(4);
112        increment = 100000;
113        hd44780_print_lines("WAVE GENERATION", "Inc = 10000      Hz");
114    }
115    else if (switchsState == (1UL << 13)) {
116        LED_All_Off();
117        LED_On(5);
118        increment = 1000000;
119        hd44780_print_lines("WAVE GENERATION", "Inc = 1000000 Hz");
120    }
121    else if (switchsState == (1UL << 14)) {
122        char tmp_string[15];
123
124        LED_On(6);
125
126        currentFrequency = currentFrequency - increment;
127        if(currentFrequency < 0.01)
128            currentFrequency = 0.01;
129        DDS_Set(currentFrequency);
130
131        sprintf(tmp_string, "Freq = %.2f", currentFrequency);
132        hd44780_print_lines("WAVE GENERATION", tmp_string);
133
134        LED_Off(6);
135    }
136    else if (switchsState == (1UL << 15)) {
137        char tmp_string[15];
138
139        LED_On(7);
140
141        currentFrequency = currentFrequency + increment;
142        if(currentFrequency > 35000000)
143            currentFrequency = 35000000;
144        DDS_Set(currentFrequency);
145
146        sprintf(tmp_string, "Freq = %.2f ", currentFrequency);
147        hd44780_print_lines("WAVE GENERATION", tmp_string);
148
149        LED_Off(7);
150    }
151 }
152
153 while(function == FREQUENCY_METER)
154 {
155     uint32_t switchsState;
156

```

```

157     if(updateFlag == 1)
158     {
159         Freq_Meter_Init();
160         updateFlag = 0;
161         hd44780_clear();
162         hd44780_position(0, 0);
163         hd44780_print("FREQUENCY METER");
164     }
165
166
167     // Check for switch presses to capture frequency meter value
168     switchsState = SWT_Get();
169
170     if (switchsState == (1UL << 8)) {
171         LED_All_Off();
172         LED_On(0);
173         freqRange = LESS_THAN_1;
174         TIM4->PSC = 0xF000;
175         hd44780_print_lines("FREQUENCY METER", "Range = <1");
176     }
177     else if (switchsState == (1UL << 9)) {
178         LED_All_Off();
179         LED_On(1);
180         freqRange = ONE_TO_100;
181         TIM4->PSC = 0x0F00;
182         hd44780_print_lines("FREQUENCY METER", "Range = 1-100");
183     }
184     else if (switchsState == (1UL << 10)) {
185         LED_All_Off();
186         LED_On(2);
187         freqRange = HUNDRED_TO_10K;
188         TIM4->PSC = 0x000F;
189         hd44780_print_lines("FREQUENCY METER", "Range = 100-10K");
190     }
191     else if (switchsState == (1UL << 11)) {
192         LED_All_Off();
193         LED_On(3);
194         freqRange = MORE_THAN_10K;
195         TIM4->PSC = 0x0000;
196         hd44780_print_lines("FREQUENCY METER", "Range = > 10K");
197     }
198     else if (switchsState == (1UL << 15)) {
199         char Freq_Tmp[15];
200         char DC_Tmp[15];
201
202         LED_On(7);
203
204         if(freqRange == LESS_THAN_1){
205             sprintf(Freq_Tmp, "Freq = %.2f", low_Frequency);
206         }
207         else {
208             sprintf(Freq_Tmp, "Freq = %d", Frequency);
209         }
210         sprintf(DC_Tmp, "Duty = %d", DutyCycle);
211         hd44780_print_lines(Freq_Tmp, DC_Tmp);
212
213         LED_Off(7);
214     }
215 }
216
217 while(function == NOISE_GENERATION)
218 {
219     if(updateFlag == 1)
220     {
221         DAC_Ch1_NoiseConfig();
222         DAC_Noise_On();
223         updateFlag = 0;
224         hd44780_clear();
225         hd44780_position(0, 0);
226         hd44780_print("NOISE GENERATION");
227     }
228 }
229
230
231 while(function == ARBITRARY_FUNCTION)
232 {
233     if(updateFlag == 1)
234     {

```

```

235     DAC_Ch1_ArbitraryConfig();
236     DAC_Arbitrary_On();
237     updateFlag = 0;
238     hd44780_clear();
239     hd44780_position(0, 0);
240     hd44780_print("ARBITRARY FUNC");
241 }
242
243 }
244
245 while(function == PULSE_GENERATOR)
246 {
247     uint32_t switchsState;
248
249     if(updateFlag == 1)
250     {
251         updateFlag = 0;
252         hd44780_clear();
253         hd44780_position(0, 0);
254         hd44780_print("PULSE GENERATOR");
255     }
256
257     // Check for switch presses to change duty cycle
258     switchsState = SWT_Get();
259
260     if (switchsState == (1UL << 14)) {
261         char tmp_string[15];
262
263         LED_On(6);
264
265         dutyCycle--;
266
267         if(dutyCycle < 0)
268             dutyCycle = 0;
269
270         PWM_SetDC(dutyCycle);
271
272         sprintf(tmp_string, "Duty = %d %%", dutyCycle);
273         hd44780_print_lines("PULSE GENERATOR", tmp_string);
274
275         LED_Off(6);
276     }
277     else if (switchsState == (1UL << 15)) {
278         char tmp_string[15];
279
280         LED_On(7);
281
282         dutyCycle++;
283
284         if(dutyCycle > 100)
285             dutyCycle = 100;
286
287         PWM_SetDC(dutyCycle);
288
289         sprintf(tmp_string, "Duty = %d %%", dutyCycle);
290         hd44780_print_lines("PULSE GENERATOR", tmp_string);
291
292         LED_Off(7);
293     }
294 }
295
296 while(function == FREQUENCY_KEY_SHIFT)
297 {
298     if(updateFlag == 1)
299     {
300         FSK_Init();
301         updateFlag = 0;
302         hd44780_clear();
303         hd44780_position(0, 0);
304         hd44780_print("FREQ KEY SHIFT");
305     }
306
307     if(FSK_Change == true)
308     {
309         if(FSK_Freq == HIGH)
310         {
311             DDS_Set(1000);           //Output 1KHz wave if input wave is "high"
312         }
313     }
314 }

```

```

313         else if(FSK_Freq == LOW)
314         {
315             DDS_Set(1000000);           //Output 100Hz wave if input wave is "low"
316         }
317         FSK_Change = false;
318     }
319 }
320 }
321 }
322
323 /*-----
324     SysTick_Handler
325     -----*/
326 void SysTick_Handler(void) {
327     msTicks++;
328 }
329
330 /*-----
331     delays number of tick Systicks (happens every 1 ms)
332     -----*/
333 void Delay (uint32_t dlyTicks) {
334     uint32_t curTicks;
335     curTicks = msTicks;
336
337     while ((msTicks - curTicks) < dlyTicks);
338 }
339
340 void Config_menu_interrupt_2 (void) {
341     EXTI_InitTypeDef EXTI_InitStructure;
342     NVIC_InitTypeDef NVIC_InitStructure;
343     GPIO_InitTypeDef GPIO_InitStructure;
344
345     RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
346     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
347
348     /* Configure GPIOs as as inputs */
349     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15 | GPIO_Pin_14 | GPIO_Pin_13 | GPIO_Pin_12 | GPIO_Pin_11 |
GPIO_Pin_10;
350     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
351     GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
352     GPIO_Init(GPIOB, &GPIO_InitStructure);
353
354     /* Connect EXTI Lines 10-15 to GPIOB Pins 10-15*/
355     SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource10);
356     SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource11);
357     SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource12);
358     SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource13);
359     SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource14);
360     SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource15);
361
362     /* Configure EXTI lines 8-15 */
363     EXTI_InitStructure.EXTI_Line = EXTI_Line10 | EXTI_Line11 | EXTI_Line12 | EXTI_Line13 | EXTI_Line14
| EXTI_Line15;
364     EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
365     EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
366     EXTI_InitStructure.EXTI_LineCmd = ENABLE;
367     EXTI_Init(&EXTI_InitStructure);
368
369     /* Enable and set EXTI Lines 8-15 Interrupt to the lowest priority */
370     NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
371     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
372     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
373     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
374     NVIC_Init(&NVIC_InitStructure);
375 }
376
377 void Config_menu_interrupt(void) {
378     EXTI_InitTypeDef EXTI_InitStructure;
379     NVIC_InitTypeDef NVIC_InitStructure;
380
381     RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
382
383     /* Connect EXTI Line0 to GPIOA Pin 0*/
384     SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);
385
386     /* Configure EXTI line0 */
387     EXTI_InitStructure.EXTI_Line = EXTI_Line0;
388     EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;

```

```

389     EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
390     EXTI_InitStructure.EXTI_LineCmd = ENABLE;
391     EXTI_Init(&EXTI_InitStructure);
392
393     /* Enable and set EXTI Line0 Interrupt to the lowest priority */
394     NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
395     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1; // changed from 0x01
396     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1; // changed from 0x01
397     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
398     NVIC_Init(&NVIC_InitStructure);
399 }
400
401 void EXTI0_IRQHandler(void) {
402
403     LED_All_Off();
404     updateFlag = 1;
405
406     if (function == WAVE_GENERATION)
407     {
408         function = FREQUENCY_METER;
409     }
410     else if (function == FREQUENCY_METER)
411     {
412         function = NOISE_GENERATION;
413     }
414     else if (function == NOISE_GENERATION)
415     {
416         function = ARBITORY_FUNCTION;
417         DAC_Noise_Off();
418     }
419     else if (function == ARBITORY_FUNCTION)
420     {
421         function = PULSE_GENERATOR;
422         DAC_Arbitrary_Off();
423     }
424     else if (function == PULSE_GENERATOR)
425     {
426         function = FREQUENCY_KEY_SHIFT;
427     }
428     else if (function == FREQUENCY_KEY_SHIFT)
429     {
430         function = WAVE_GENERATION;
431     }
432     else
433     {
434         function = WAVE_GENERATION;
435         DAC_Noise_Off();
436         DAC_Arbitrary_Off();
437     }
438
439     EXTI_ClearITPendingBit(EXTI_Line0); // Clear the pending bit to signal IRQ finished
440 }
441
442 void EXTI15_10_IRQHandler(void) {
443
444     ITStatus line10, line11, line12, line13, line14, line15;
445
446     LED_All_Off();
447     updateFlag = 1;
448
449     line10 = EXTI_GetITStatus(EXTI_Line10);
450     line11 = EXTI_GetITStatus(EXTI_Line11);
451     line12 = EXTI_GetITStatus(EXTI_Line12);
452     line13 = EXTI_GetITStatus(EXTI_Line13);
453     line14 = EXTI_GetITStatus(EXTI_Line14);
454     line15 = EXTI_GetITStatus(EXTI_Line15);
455
456     if(line10 == SET) {
457         function = WAVE_GENERATION;
458     }
459     else if(line11 == SET) {
460         function = FREQUENCY_METER;
461     }
462     else if(line12 == SET) {
463         function = NOISE_GENERATION;
464     }
465     else if(line13 == SET) {
466         function = ARBITORY_FUNCTION;

```

```
467     }
468     else if(line14 == SET) {
469         function = PULSE_GENERATOR;
470     }
471     else if(line15 == SET) {
472         function = FREQUENCY_KEY_SHIFT;
473     }
474
475     DAC_Noise_Off();
476     DAC_Arbitrary_Off();
477
478 }
479
```