

```

1  /*-----
2  * Name:      ArbitraryFunc.c
3  * Purpose:   Code to output a specified resolution of a wave table to the DAC, using
4              DMA requests to free up CPU time.
5  * Note(s):  adapted from example code found at
6  *           http://00xnor.blogspot.co.uk/2014/01/6-stm32-f4-dac-dma-waveform-generator.html
7  *-----
8  *
9  *-----*/
10
11 #include "STM32F4xx.h"
12 #include "main_2.h"
13
14 #define DAC_DHR12R1_ADDR 0x40007408
15 #define OUT_FREQ          5000 // Output waveform frequency
16 #define WAVE_RES          128 // Waveform resolution
17 #define CNT_FREQ          84000000 // TIM6 counter clock (prescaled
18                                APB1)
19 #define TIM_PERIOD        ((CNT_FREQ)/((WAVE_RES)*(OUT_FREQ))) // Autoreload reg value
20
21 // Sinc function
22 const uint16_t waveForm[WAVE_RES] = { 3995, 3987, 3964, 3925, 3872, 3805, 3725, 3633, 3531, 3419,
23                                         3300, 3176, 3047, 2915, 2784, 2653, 2524, 2400, 2282, 2171,
24                                         2068, 1975, 1891, 1819, 1758, 1708, 1670, 1644, 1629, 1624,
25                                         1630, 1646, 1669, 1700, 1738, 1780, 1827, 1876, 1926, 1977,
26                                         2027, 2075, 2120, 2161, 2198, 2229, 2255, 2275, 2289, 2296,
27                                         2297, 2293, 2282, 2267, 2247, 2223, 2195, 2165, 2134, 2101,
28                                         2068, 2036, 2005, 1976, 1950, 1927, 1907, 1891, 1880, 1873,
29                                         1870, 1871, 1877, 1886, 1899, 1916, 1935, 1956, 1979, 2003,
30                                         2027, 2051, 2075, 2097, 2118, 2136, 2152, 2165, 2175, 2182,
31                                         2185, 2185, 2182, 2175, 2166, 2154, 2140, 2124, 2106, 2087,
32                                         2068, 2049, 2030, 2011, 1994, 1979, 1965, 1954, 1945, 1939,
33                                         1935, 1935, 1937, 1941, 1948, 1957, 1969, 1981, 1996, 2011,
34                                         2027, 2043, 2059, 2074, 2089, 2102, 2114, 212 };
35
36 void TIM5_Config(void)
37 {
38     TIM_TimeBaseInitTypeDef TIM5_TimeBase;
39
40     /* TIM5 Periph clock enable */
41     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM5, ENABLE);
42
43     /* Time base configuration */
44     TIM_TimeBaseStructInit(&TIM5_TimeBase);
45     TIM5_TimeBase.TIM_Period = (uint16_t)TIM_PERIOD;
46     TIM5_TimeBase.TIM_Prescaler = 0;
47     TIM5_TimeBase.TIM_ClockDivision = 0;
48     TIM5_TimeBase.TIM_CounterMode = TIM_CounterMode_Up;
49     TIM_TimeBaseInit(TIM6, &TIM5_TimeBase);
50
51     /* TIM5 TRGO selection */
52     TIM_SelectOutputTrigger(TIM5, TIM_TRGOSource_Update);
53
54     /* TIM5 enable counter */
55     TIM_Cmd(TIM5, ENABLE);
56 }
57
58 void DAC_Ch1_ArbitraryConfig(void)
59 {
60     DAC_InitTypeDef DAC_INIT;
61     DMA_InitTypeDef DMA_INIT;
62
63     /* DAC channel1 Configuration */
64     DAC_INIT.DAC_Trigger = DAC_Trigger_T5_TRGO;
65     DAC_INIT.DAC_WaveGeneration = DAC_WaveGeneration_None;
66     DAC_INIT.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
67     DAC_Init(DAC_Channel_1, &DAC_INIT);
68
69     /* DMA1_Stream5 channel7 configuration *****/
70     DMA_DeInit(DMA1_Stream5);
71     DMA_INIT.DMA_Channel = DMA_Channel_7;
72     DMA_INIT.DMA_PeripheralBaseAddr = (uint32_t)DAC_DHR12R1_ADDR;
73     DMA_INIT.DMA_Memory0BaseAddr = (uint32_t)&waveForm;
74     DMA_INIT.DMA_DIR = DMA_DIR_MemoryToPeripheral;
75     DMA_INIT.DMA_BufferSize = WAVE_RES;
76     DMA_INIT.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
77     DMA_INIT.DMA_MemoryInc = DMA_MemoryInc_Enable;
78     DMA_INIT.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;

```

```
78     DMA_INIT.DMA_MemoryDataSize    = DMA_MemoryDataSize_HalfWord;
79     DMA_INIT.DMA_Mode               = DMA_Mode_Circular;
80     DMA_INIT.DMA_Priority           = DMA_Priority_High;
81     DMA_INIT.DMA_FIFOMode           = DMA_FIFOMode_Disable;
82     DMA_INIT.DMA_FIFOThreshold      = DMA_FIFOThreshold_HalfFull;
83     DMA_INIT.DMA_MemoryBurst        = DMA_MemoryBurst_Single;
84     DMA_INIT.DMA_PeripheralBurst    = DMA_PeripheralBurst_Single;
85     DMA_Init(DMA1_Stream5, &DMA_INIT);
86
87     /* Enable DMA1_Stream5 */
88     DMA_Cmd(DMA1_Stream5, ENABLE);
89
90     /* Enable DAC Channel1 */
91     DAC_Cmd(DAC_Channel_1, ENABLE);
92
93     /* Enable DMA for DAC Channel1 */
94     DAC_DMAMCmd(DAC_Channel_1, ENABLE);
95 }
96
97 void DAC_Arbitory_On(void)
98 {
99     /* Enable DAC Channel1 */
100     DAC_Cmd(DAC_Channel_1, ENABLE);
101 }
102
103 void DAC_Arbitory_Off(void)
104 {
105     /* Disable DAC Channel1 */
106     DAC_Cmd(DAC_Channel_1, DISABLE);
107 }
108
```

```

1  /*-----*/
2  * Name:      DAC.c
3  * Purpose:   Functions to initilise the DAC, and subsequently provide triangle wave
4              functionality, noise generation, and supports arbitrary function generation.
5  * Note(s):  Example code taken from STMicroElectronics Application Teams,
6              DAC_SignalsGeneration example project
7  *-----*/
8  *
9  *-----*/
10
11 #include "STM32F4xx.h"
12 #include "main_2.h"
13 #include "DAC.h"
14 #include "ArbitraryFunc.h"
15
16 // CMSIS data structure for DAC
17 DAC_InitTypeDef  DAC_InitStructure;
18
19 void DACs_Init(void)
20 {
21     /* Preconfiguration before using DAC-----*/
22     GPIO_InitTypeDef GPIO_InitStructure;
23
24     /* DMA1 clock and GPIOA clock enable (to be used with DAC) */
25     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1 | RCC_AHB1Periph_GPIOA, ENABLE);
26
27     /* DAC Periph clock enable */
28     RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
29
30     /* DAC channel 1 & 2 (DAC_OUT1 = PA.4) (DAC_OUT2 = PA.5) configuration */
31     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4 | GPIO_Pin_5;
32     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
33     GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
34     GPIO_Init(GPIOA, &GPIO_InitStructure);
35
36     /* TIM Configuration -----*/
37     TIM6_Config();
38     TIM5_Config();
39
40     /* Set DAC registers to default values */
41     DAC_DeInit();
42 }
43
44 void TIM6_Config(void)
45 {
46     TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
47
48     /* TIM6 Periph clock enable */
49     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);
50
51     /* Time base configuration */
52     TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
53     TIM_TimeBaseStructure.TIM_Period = 1;
54     TIM_TimeBaseStructure.TIM_Prescaler = 0;
55     TIM_TimeBaseStructure.TIM_ClockDivision = 0;
56     TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
57     TIM_TimeBaseInit(TIM6, &TIM_TimeBaseStructure);
58
59     /* TIM6 TRGO selection */
60     TIM_SelectOutputTrigger(TIM6, TIM_TRGOSource_Update);
61
62     /* TIM6 enable counter */
63     TIM_Cmd(TIM6, ENABLE);
64 }
65
66 void DAC_Ch2_TriangleConfig(void)
67 {
68     /* DAC channel2 Configuration */
69     DAC_InitStructure.DAC_Trigger = DAC_Trigger_T6_TRGO;
70     DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_Triangle;
71     DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude = DAC_TriangleAmplitude_255;
72     DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
73     DAC_Init(DAC_Channel_2, &DAC_InitStructure);
74
75     /* Set DAC channel2 DHR12RD register */
76     DAC_SetChannel2Data(DAC_Align_12b_R, 0x100);
77 }
78

```

```
79 void DAC_Ch1_NoiseConfig(void)
80 {
81     /* DAC channell1 Configuration */
82     DAC_InitStructure.DAC_Trigger = DAC_Trigger_T6_TRGO;
83     DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_Noise;
84     DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude = DAC_LFSRUnmask_Bits11_0;    // Max bits unmasked
85     DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
86     DAC_Init(DAC_Channel_1, &DAC_InitStructure);
87
88     /* Set DAC Channell1 DHR12L register */
89     DAC_SetChannell1Data(DAC_Align_12b_L, 0x7FF0);
90 }
91
92 void DAC_Noise_On(void)
93 {
94     /* Enable DAC Channell1 */
95     DAC_Cmd(DAC_Channel_1, ENABLE);
96 }
97
98 void DAC_Noise_Off(void)
99 {
100     /* Disable DAC Channell1 */
101     DAC_Cmd(DAC_Channel_1, DISABLE);
102 }
103
104 void DAC_Triangle_On(void) {
105     /* Enable DAC Channel2 */
106     DAC_Cmd(DAC_Channel_2, ENABLE);
107 }
108
109 void DAC_Traingle_Off(void) {
110     /* Disable DAC Channel2 */
111     DAC_Cmd(DAC_Channel_2, DISABLE);
112 }
113
```

```

1  /*-----*/
2  * Name:      DDS.c
3  * Purpose:   Functions to initialise the DDS, set default data, and accept new
4              frequencies from the user.
5  * Note(s):
6  *-----*/
7
8  #include "STM32F4xx.h"
9  #include "DDS.h"
10 #include "main_2.h"
11 #include <math.h>
12
13 #define DDS_CLOCK 125000000
14 #define CLOCK 4 /* W_CLK pin */
15 #define LOAD 5 /* FQ_UP pin*/
16 #define DATA 3 /* DATA pin */
17
18 /*-----*/
19 initialize DDS for serial communication
20 *-----*/
21 void DDS_Init (void) {
22
23     RCC->AHB1ENR |= ((1UL << 4)); /* Enable GPIOE clock */
24
25     GPIOE->MODER  &= ~( (3UL << 2* 3) |
26                        (3UL << 2* 4) |
27                        (3UL << 2* 5) ); /* PE.0,3-4 are outputs */
28     GPIOE->MODER  |= ( (1UL << 2* 3) |
29                        (1UL << 2* 4) |
30                        (1UL << 2* 5) );
31     GPIOE->OTYPER  &= ~( (1UL << 3) |
32                         (1UL << 4) |
33                         (1UL << 5) ); /* PE.0,3-4 are output Push-Pull */
34     GPIOE->OSPEEDR &= ~( (3UL << 2* 3) |
35                         (3UL << 2* 4) |
36                         (3UL << 2* 5) ); /* PE.0,3-4 are 50MHz Fast Speed */
37     GPIOE->OSPEEDR |= ( (2UL << 2* 3) |
38                         (2UL << 2* 4) |
39                         (2UL << 2* 5) );
40     GPIOE->PUPDR  &= ~( (3UL << 2* 3) |
41                         (3UL << 2* 4) |
42                         (3UL << 2* 5) ); /* PE.0,3-4 are Pull up */
43     GPIOE->PUPDR  |= ( (1UL << 2* 3) |
44                         (1UL << 2* 4) |
45                         (1UL << 2* 5) );
46 }
47
48 void Pulse_Clock() {
49     GPIOE->ODR |= (1 << CLOCK);
50     Delay(1);
51     GPIOE->ODR &= ~(1 << CLOCK);
52 }
53
54 void Pulse_Frequency() {
55     GPIOE->ODR |= (1 << LOAD);
56     Delay(1);
57     GPIOE->ODR &= ~(1 << LOAD);
58 }
59
60 void Data_Low() {
61     GPIOE->ODR &= ~(1 << DATA);
62 }
63
64 void DDS_Write_Data(int input_data) {
65     GPIOE->ODR |= (input_data << DATA);
66 }
67
68 /*-----*/
69 Function that set the DDS output to a default value
70 *-----*/
71 void DDS_Default_Init (void) {
72
73     int i = 0;
74     int Default_Data[40] = {0,0,0,1,1,1,0,0,0,1,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}; //1KHz
75
76     Pulse_Clock();
77     Delay(1);

```

```
78     Pulse_Frequency();
79     Delay(1);
80
81     // Send the data array 1 bit at a time to the DDS
82     for(i = 0; i <40; i++){
83         Data_Low();
84         DDS_Write_Data(Default_Data[i]);
85         Delay(1);
86         Pulse_Clock();
87     }
88
89     Pulse_Frequency();
90     Delay(1);
91     Data_Low();
92
93 }
94 /*-----
95     Function that sets the DDS frequency to a user provided value
96     -----*/
97 void DDS_Set (double frequency) {
98
99     int j = 0;
100     int k = 0;
101     int tuningWord = 0;
102     int Send_Data[40];
103
104     // Calculate the new tuning word
105     tuningWord = (int) ((frequency * pow(2, 32))/DDS_CLOCK);
106
107     // Construct the data array ready to be sent to DDS
108     for (j = 0; j < 40; j++) {
109         // calculate each array position by bitwise anding the tuning word with 1
110         Send_Data[j] = tuningWord & (1 << j) ? 1 : 0;
111     }
112
113     Pulse_Clock();
114     Delay(1);
115     Pulse_Frequency();
116     Delay(1);
117
118     // Send the data array 1 bit at a time to the DDS
119     for(k = 0; k <40; k++){
120         Data_Low();
121         DDS_Write_Data(Send_Data[k]);
122         Delay(1);
123         Pulse_Clock();
124     }
125
126     Pulse_Frequency();
127     Delay(1);
128     Data_Low();
129 }
130
```

```
35      /* TIM4 configuration: PWM Input mode -----
```

```

1  /*-----
2  * Name:      FSK.c
3  * Purpose:   Functions to provide frequency shift keying for an input waveform,
4  *            by setting 2 different DDS frequencies appropriately.
5  * Note(s):   Example code taken from STMicroElectronics Application Teams,
6  *            TIM_PWM_Input eample project.
7  *-----
8  *
9  *-----*/
10
11 #include "STM32F4xx.h"
12 #include "main_2.h"
13 #include "FSK.h"
14 #include "DDS.h"
15
16 void FSK_Init(void)
17 {
18     TIM_ICInitTypeDef  TIM_ICInitStructure;
19
20     GPIO_InitTypeDef GPIO_InitStructure;
21     NVIC_InitTypeDef  NVIC_InitStructure;
22
23     /* TIM4 clock enable */
24     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
25
26     /* GPIOB clock enable */
27     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
28
29     /* TIM4 chennel2 configuration : PB.07 */
30     GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_7;
31     GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF;
32     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
33     GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
34     GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP ;
35     GPIO_Init(GPIOB, &GPIO_InitStructure);
36
37     /* Connect TIM pin to AF2 */
38     GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_TIM4);
39
40     /* Enable the TIM4 global Interrupt */
41     NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
42     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
43     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
44     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
45     NVIC_Init(&NVIC_InitStructure);
46
47     /* TIM4 configuration: PWM Input mode */
48     TIM_ICInitStructure.TIM_Channel = TIM_Channel_2;
49     TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_BothEdge;
50     TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
51     TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
52     TIM_ICInitStructure.TIM_ICFilter = 0x0;
53
54     TIM_PWMConfig(TIM4, &TIM_ICInitStructure);
55
56     /* Select the TIM4 Input Trigger: TI2FP2 */
57     TIM_SelectInputTrigger(TIM4, TIM_TS_TI2FP2);
58
59     /* Select the slave Mode: Reset Mode */
60     TIM_SelectSlaveMode(TIM4, TIM_SlaveMode_Reset);
61     TIM_SelectMasterSlaveMode(TIM4, TIM_MasterSlaveMode_Enable);
62
63     /* TIM enable counter */
64     TIM_Cmd(TIM4, ENABLE);
65
66     /* Enable the CC2 Interrupt Request */
67     TIM_ITConfig(TIM4, TIM_IT_CC2, ENABLE);
68 }

```



```

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 fequency
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
```

```

1  /*-----
2  * Name:      Square.c
3  * Purpose:   Pulse gerator, with a duty cyle variable by the user.
4  * Note(s):  Code modified from ST MicroElectronics Application Teams,
5             TIM_PWM_Output example project.
6  *-----
7  *
8  *-----*/
9
10 #include "STM32F4xx.h"
11 #include "LCD.h"
12 #include "Sqaure.h"
13
14 #define TIM3_CLK_OUT 42000
15 #define TIM3_CNT_CLK 28000000
16 #define TIM3_ARR 665 //((TIM3_CNT_CLK / TIM3_CLK_OUT) - 1)
17 #define FIFTY_PERCENT 333
18
19 void Pulse_Config(void) {
20     // Run timer config and initialise pulses to 50:50 duty cycle
21     TIM3_Config();
22     PWM_Config(TIM3_ARR);
23 }
24
25 void TIM3_Config(void) {
26     GPIO_InitTypeDef GPIO_InitStructure;
27
28     /* TIM3 clock enable */
29     RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);
30
31     /* GPIOC clock enable */
32     RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
33
34     /* GPIOC Configuration: TIM3 CH1 (PC6) */
35     GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 ;
36     GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
37     GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
38     GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
39     GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;
40     GPIO_Init(GPIOC, &GPIO_InitStructure);
41
42     /* Connect TIM3 pins to AF2 */
43     GPIO_PinAFConfig(GPIOC, GPIO_PinSource6, GPIO_AF_TIM3);
44 }
45
46 void PWM_Config(int period)
47 {
48     TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure;
49     TIM_OCInitTypeDef TIM_OCInitStructure;
50     uint16_t PrescalerValue = 0;
51
52     /* Compute the prescaler value */
53     PrescalerValue = (uint16_t) ((SystemCoreClock /2) / 28000000) - 1;
54
55     /* Time base configuration */
56     TIM_TimeBaseStructure.TIM_Period = 665;
57     TIM_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
58     TIM_TimeBaseStructure.TIM_ClockDivision = 0;
59     TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
60
61     TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);
62
63     /* PWM1 Mode configuration: Channell */
64     TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
65     TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
66     TIM_OCInitStructure.TIM_Pulse = FIFTY_PERCENT; // 50:50 duty cyle
67     TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
68
69     TIM_OC1Init(TIM3, &TIM_OCInitStructure);
70
71     TIM_OC1PreloadConfig(TIM3, TIM_OCPreload_Enable);
72
73     TIM_ARRPreloadConfig(TIM3, ENABLE);
74
75     /* TIM3 enable counter */
76     TIM_Cmd(TIM3, ENABLE);
77 }
78

```



```
79 void PWM_SetDC(uint16_t dutycycle)
80 {
81     uint16_t newDutyCycle;
82
83     // Calculate the new duty cycle
84     newDutyCycle = (dutycycle * TIM3_ARR) / 100;
85
86     // set the new duty cycle into the capture compare register
87     TIM_SetCompare1(TIM3, newDutyCycle);
88 }
89
```