```c
/*----------------------------------------------------------------------------
 * Name:    main.c
 * Purpose:
 * Note(s):
 *----------------------------------------------------------------------------
 *
 *----------------------------------------------------------------------------*/

#include "STM32F4xx.h"
#include "stm32f4_discovery.h"
#include "main_2.h"
#include "LED.h"
#include "SWT.h"
#include "LCD.h"
#include "Sqaure.h"
#include "DAC.h"
#include "DDS.h"
#include "FreqMeter.h"
#include "hd44780.h"
#include "ArbitoryFunc.h"
#include "FSK.h"
#include <stdio.h>

volatile uint32_t msTicks;                        /* counts 1ms timeTicks       */
volatile double currentFrequency = 1000;
volatile double increment = 1;
volatile int function = WAVE_GENERATION;
volatile int freqRange = HUNDRED_TO_10K;
volatile unsigned char updateFlag = 1;
volatile int dutyCycle = 50;

/*----------------------------------------------------------------------------
  MAIN function
 *----------------------------------------------------------------------------*/
int main (void) {

  __disable_irq();
  SystemCoreClockUpdate();                        /* Get Core Clock Frequency   */

  if (SysTick_Config(SystemCoreClock / 1680)) {  /* SysTick 1 msec interrupts  */
    while (1);                                    /* Capture error              */
  }
  __enable_irq();

  // Initialise Required Pins
  BTN_Init();
  SWTS_Init();
  LED_Init();
  init_lcd_driver();
  hd44780_init(GPIOD, GPIOB, GPIO_Pin_0, GPIO_Pin_1, GPIO_Pin_2, GPIO_Pin_4,
              GPIO_Pin_5, GPIO_Pin_6, GPIO_Pin_7, HD44780_LINES_2, HD44780_FONT_5x8);
  DDS_Init();
  DACs_Init();

  //Initialise components to defaults
  DDS_Default_Init();
  Pulse_Config();

  // Turn on LCD display
  hd44780_display(true, false, false);

  // Set up intterupts for the blue user button - ie the menu
  //STM_EVAL_PBInit(BUTTON_USER, BUTTON_MODE_EXTI);
  Config_menu_interrupt();

  while(1)
  {

    while(function == WAVE_GENERATION)
    {
      uint32_t switchsState;

      if(updateFlag == 1)
      {
        updateFlag = 0;
        hd44780_clear();
        hd44780_position(0, 0);
        hd44780_print("WAVE GENERATION");
      }

      // Check for switch presses to change DDS fequency
      switchsState = SWT_Get();

      if (switchsState == (1UL << 8)) {
        LED_All_Off();
        LED_On(0);
        increment = 0.01;
        hd44780_print_lines("WAVE GENERATION", "Inc = 0.01    Hz");
      }
      else if (switchsState == (1UL << 9)) {
        LED_All_Off();
        LED_On(1);
        increment = 1;
        hd44780_print_lines("WAVE GENERATION", "Inc = 1       Hz");
      }
      else if (switchsState == (1UL << 10)) {
        LED_All_Off();
        LED_On(2);
        increment = 100;
        hd44780_print_lines("WAVE GENERATION", "Inc = 100     Hz");
      }
      else if (switchsState == (1UL << 11)) {
        LED_All_Off();
        LED_On(3);
        increment = 1000;
        hd44780_print_lines("WAVE GENERATION", "Inc = 1000    Hz");

      }
      else if (switchsState == (1UL << 12)) {
        LED_All_Off();
        LED_On(4);
        increment = 100000;
        hd44780_print_lines("WAVE GENERATION", "Inc = 10000   Hz");
      }
      else if (switchsState == (1UL << 13)) {
        LED_All_Off();
        LED_On(5);
        increment = 1000000;
        hd44780_print_lines("WAVE GENERATION", "Inc = 1000000 Hz");
      }
      else if (switchsState == (1UL << 14)) {
        char tmp_string[15];

        LED_On(6);

        currentFrequency = currentFrequency - increment;
        if(currentFrequency < 0.01)
          currentFrequency = 0.01;
        DDS_Set(currentFrequency);

        sprintf(tmp_string, "Freq = %.2f", currentFrequency);
        hd44780_print_lines("WAVE GENERATION", tmp_string);

        LED_Off(6);
      }
      else if (switchsState == (1UL << 15)) {
        char tmp_string[15];

        LED_On(7);

        currentFrequency = currentFrequency + increment;
        if(currentFrequency > 35000000)
            currentFrequency = 35000000;
        DDS_Set(currentFrequency);

        sprintf(tmp_string, "Freq = %.2f ", currentFrequency);
        hd44780_print_lines("WAVE GENERATION", tmp_string);

        LED_Off(7);
      }
    }

    while(function == FREQUENCY_METER)
    {
      uint32_t switchsState;

      if(updateFlag == 1)
      {
        Freq_Meter_Init();
        updateFlag = 0;
        hd44780_clear();
        hd44780_position(0, 0);
        hd44780_print("FREQUENCY METER");
      }


      // Check for switch presses to capture frequency meter value
      switchsState = SWT_Get();

      if (switchsState == (1UL << 8)) {
        LED_All_Off();
        LED_On(0);
        freqRange = LESS_THAN_1;
        TIM4->PSC = 0xF000;
        hd44780_print_lines("FREQUENCY METER", "Range = <1");
      }
      else if (switchsState == (1UL << 9)) {
        LED_All_Off();
        LED_On(1);
        freqRange = ONE_TO_100;
        TIM4->PSC = 0x0F00;
        hd44780_print_lines("FREQUENCY METER", "Range = 1-100");
      }
      else if (switchsState == (1UL << 10)) {
        LED_All_Off();
        LED_On(2);
        freqRange = HUNDRED_TO_10K;
        TIM4->PSC = 0x000F;
        hd44780_print_lines("FREQUENCY METER", "Range = 100-10K");
      }
      else if (switchsState == (1UL << 11)) {
        LED_All_Off();
        LED_On(3);
        freqRange = MORE_THAN_10K;
        TIM4->PSC = 0x0000;
        hd44780_print_lines("FREQUENCY METER", "Range = > 10K");
      }
      else if (switchsState == (1UL << 15)) {
        char Freq_Tmp[15];
        char DC_Tmp[15];

        LED_On(7);

        if(freqRange == LESS_THAN_1){
          sprintf(Freq_Tmp, "Freq = %.2f", low_Frequency);
        }
        else {
          sprintf(Freq_Tmp, "Freq = %d", Frequency);
        }
        sprintf(DC_Tmp, "Duty = %d", DutyCycle);
        hd44780_print_lines(Freq_Tmp, DC_Tmp);

        LED_Off(7);
      }
    }

    while(function == NOISE_GENERATION)
    {
      if(updateFlag == 1)
      {
        DAC_Ch1_NoiseConfig();
        DAC_Noise_On();
        updateFlag = 0;
        hd44780_clear();
        hd44780_position(0, 0);
        hd44780_print("NOISE GENERATION");
      }

    }

    while(function == ARBITORY_FUNCTION)
    {
      if(updateFlag == 1)
      {
        DAC_Ch1_ArbitoryConfig();
        DAC_Arbitory_On();
        updateFlag = 0;
        hd44780_clear();
        hd44780_position(0, 0);
        hd44780_print("ARBITRARY FUNC");
      }

    }

    while(function == PULSE_GENERATOR)
    {
      uint32_t switchsState;

      if(updateFlag == 1)
      {
        updateFlag = 0;
        hd44780_clear();
        hd44780_position(0, 0);
        hd44780_print("PULSE GENERATOR");
      }

      // Check for switch presses to change duty cycle
      switchsState = SWT_Get();

      if (switchsState == (1UL << 14)) {
        char tmp_string[15];

        LED_On(6);

        dutyCycle--;

        if(dutyCycle < 0)
          dutyCycle = 0;

        PWM_SetDC(dutyCycle);

        sprintf(tmp_string, "Duty = %d %%", dutyCycle);
        hd44780_print_lines("PULSE GENERATOR", tmp_string);

        LED_Off(6);
      }
      else if (switchsState == (1UL << 15)) {
        char tmp_string[15];

        LED_On(7);

        dutyCycle++;

        if(dutyCycle > 100)
          dutyCycle = 100;

        PWM_SetDC(dutyCycle);

        sprintf(tmp_string, "Duty = %d %%", dutyCycle);
        hd44780_print_lines("PULSE GENERATOR", tmp_string);

        LED_Off(7);
      }
    }

    while(function == FREQUENCY_KEY_SHIFT)
    {
      if(updateFlag == 1)
      {
        FSK_Init();
        updateFlag = 0;
        hd44780_clear();
        hd44780_position(0, 0);
        hd44780_print("FREQ KEY SHIFT");
      }

      if(FSK_Change == true)
      {
        if(FSK_Freq == HIGH)
        {
          DDS_Set(1000);          //Output 1KHz wave if input wave is "high"
        }
        else if(FSK_Freq == LOW)
        {
          DDS_Set(1000000);           //Output 100Hz wave if input wave is "low"
        }
        FSK_Change = false;
      }
    }
  }
}

/*----------------------------------------------------------------------------
  SysTick_Handler
 *----------------------------------------------------------------------------*/
void SysTick_Handler(void) {
  msTicks++;
}

/*----------------------------------------------------------------------------
  delays number of tick Systicks (happens every 1 ms)
 *----------------------------------------------------------------------------*/
void Delay (uint32_t dlyTicks) {
  uint32_t curTicks;
  curTicks = msTicks;

  while ((msTicks - curTicks) < dlyTicks);
}

void Config_menu_interrupt_2 (void) {
  EXTI_InitTypeDef EXTI_InitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;
  GPIO_InitTypeDef GPIO_InitStructure;

  RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

  /* Configure GPIOs as as inputs */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_15 | GPIO_Pin_14 | GPIO_Pin_13 | GPIO_Pin_12 | GPIO_Pin_11 |
 GPIO_Pin_10;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOB, &GPIO_InitStructure);

  /* Connect EXTI Lines 10-15 to GPIOB Pins 10-15*/
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource10);
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource11);
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource12);
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource13);
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource14);
  SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource15);

  /* Configure EXTI lines 8-15 */
  EXTI_InitStructure.EXTI_Line = EXTI_Line10 | EXTI_Line11 | EXTI_Line12 | EXTI_Line13 | EXTI_Line14
| EXTI_Line15;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  /* Enable and set EXTI Lines 8-15 Interrupt to the lowest priority */
  NVIC_InitStructure.NVIC_IRQChannel = EXTI15_10_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}

void Config_menu_interrupt(void) {
    EXTI_InitTypeDef EXTI_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

    /* Connect EXTI Line0 to GPIOA Pin 0*/
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);

    /* Configure EXTI line0 */
    EXTI_InitStructure.EXTI_Line = EXTI_Line0;
    EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_InitStructure.EXTI_LineCmd = ENABLE;
    EXTI_Init(&EXTI_InitStructure);

    /* Enable and set EXTI Line0 Interrupt to the lowest priority */
    NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1; // changed from 0x01
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;         // changed from 0x01
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}

 void EXTI0_IRQHandler(void) {

   LED_All_Off();
   updateFlag = 1;

   if (function == WAVE_GENERATION)
    {
      function = FREQUENCY_METER;
    }
  else if (function == FREQUENCY_METER)
    {
      function = NOISE_GENERATION;
    }
  else if (function == NOISE_GENERATION)
    {
      function = ARBITORY_FUNCTION;
      DAC_Noise_Off();
    }
  else if (function == ARBITORY_FUNCTION)
    {
      function = PULSE_GENERATOR;
      DAC_Arbitory_Off();
    }
  else if (function == PULSE_GENERATOR)
    {
      function = FREQUENCY_KEY_SHIFT;
    }
  else if (function == FREQUENCY_KEY_SHIFT)
    {
      function = WAVE_GENERATION;
    }
  else
    {
      function = WAVE_GENERATION;
      DAC_Noise_Off();
      DAC_Arbitory_Off();
    }

  EXTI_ClearITPendingBit(EXTI_Line0);       // Clear the pending bit to signal IRQ finished
}

void EXTI15_10_IRQHandler(void) {

  ITStatus line10, line11, line12, line13, line14, line15;

  LED_All_Off();
  updateFlag = 1;

  line10 = EXTI_GetITStatus(EXTI_Line10);
  line11 = EXTI_GetITStatus(EXTI_Line11);
  line12 = EXTI_GetITStatus(EXTI_Line12);
  line13 = EXTI_GetITStatus(EXTI_Line13);
  line14 = EXTI_GetITStatus(EXTI_Line14);
  line15 = EXTI_GetITStatus(EXTI_Line15);

  if(line10 == SET) {
    function = WAVE_GENERATION;
  }
  else if(line11 == SET) {
    function = FREQUENCY_METER;
  }
  else if(line12 == SET) {
    function = NOISE_GENERATION;
  }
  else if(line13 == SET) {
    function = ARBITORY_FUNCTION;
  }
  else if(line14 == SET) {
    function = PULSE_GENERATOR;
  }
  else if(line15 == SET) {
    function = FREQUENCY_KEY_SHIFT;
  }

  DAC_Noise_Off();
  DAC_Arbitory_Off();

}

```