```c
/*-------------------------------------------------------------------------------
 * Name:    FreqMeter.c
 * Purpose: Function to initialse timer 4 into PWM mode, allowing the duty cycle
            and frquency to be measured by the capture compare registers.
 * Note(s): Example code taken from STMicroElectronics Application Teams,
            TIM_PWM_Input eample project.
 *-------------------------------------------------------------------------------
 *
 *-------------------------------------------------------------------------------*/

#include "STM32F4xx.h"
#include "stm32f4_discovery.h"
#include <stdio.h>
#include "main_2.h"
#include "LCD.h"
#include "FreqMeter.h"
#include "FSK.h"

TIM_ICInitTypeDef  TIM_ICInitStructure;

volatile uint16_t DutyCycle;
volatile uint32_t Frequency;
volatile double low_Frequency;
volatile uint16_t IC2Value;

volatile bool FSK_Change = false;
volatile int FSK_Freq;
volatile int toggleBit = 1;

void Freq_Meter_Init(void)
{
  /* TIM Configuration */
  TIM_Config();

  /* TIM4 configuration: PWM Input mode ------------------------
     The external signal is connected to TIM4 CH2 pin (PB.07),
     The Rising edge is used as active edge,
     The TIM4 CCR2 is used to compute the frequency value
     The TIM4 CCR1 is used to compute the duty cycle value
  ------------------------------------------------------------ */

  TIM_ICInitStructure.TIM_Channel = TIM_Channel_2;
  TIM_ICInitStructure.TIM_ICPolarity = TIM_ICPolarity_Rising;
  TIM_ICInitStructure.TIM_ICSelection = TIM_ICSelection_DirectTI;
  TIM_ICInitStructure.TIM_ICPrescaler = TIM_ICPSC_DIV1;
  TIM_ICInitStructure.TIM_ICFilter = 0x0;

  TIM_PWMIConfig(TIM4, &TIM_ICInitStructure);

  /* Select the TIM4 Input Trigger: TI2FP2 */
  TIM_SelectInputTrigger(TIM4, TIM_TS_TI2FP2);

  /* Select the slave Mode: Reset Mode */
  TIM_SelectSlaveMode(TIM4, TIM_SlaveMode_Reset);
  TIM_SelectMasterSlaveMode(TIM4,TIM_MasterSlaveMode_Enable);

  /* TIM enable counter */
  TIM_Cmd(TIM4, ENABLE);

  /* Enable the CC2 Interrupt Request */
  TIM_ITConfig(TIM4, TIM_IT_CC2, ENABLE);
}

void TIM_Config(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;
  NVIC_InitTypeDef NVIC_InitStructure;

  /* TIM4 clock enable */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

  /* GPIOB clock enable */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

  /* TIM4 chennel2 configuration : PB.07 */
  GPIO_InitStructure.GPIO_Pin   = GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
```

```
  79     GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  80     GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_DOWN ;
  81     GPIO_Init(GPIOB, &GPIO_InitStructure);
  82
  83     /* Connect TIM pin to AF2 */
  84     GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_TIM4);
  85
  86     /* Enable the TIM4 global Interrupt */
  87     NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
  88     NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;
  89     NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  90     NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  91     NVIC_Init(&NVIC_InitStructure);
  92
  93   }
  94
  95   void TIM4_IRQHandler(void){
  96
  97     if(function == FREQUENCY_METER)
  98     {
  99       RCC_ClocksTypeDef RCC_Clocks;
 100       RCC_GetClocksFreq(&RCC_Clocks);
 101
 102       /* Get the Input Capture value */
 103       IC2Value = TIM_GetCapture2(TIM4);
 104
 105       if (IC2Value != 0)
 106       {
 107         /* Duty cycle computation */
 108         DutyCycle = (TIM_GetCapture1(TIM4) * 100) / IC2Value;
 109
 110         /* Frequency computation TIM4 counter clock = (RCC_Clocks.HCLK_Frequency)/2 */
 111         if(freqRange == LESS_THAN_1) {
 112           low_Frequency = (((RCC_Clocks.HCLK_Frequency)/2 / (double)IC2Value) / (61440 - 1));     /*
     0.06 - 1 hz */
 113         }
 114         else if(freqRange == ONE_TO_100) {
 115           Frequency = (((RCC_Clocks.HCLK_Frequency)/2 / IC2Value) / (3840 - 1));      /* 1 - 100 hz */
 116         }
 117         else if(freqRange == HUNDRED_TO_10K) {
 118           Frequency = (((RCC_Clocks.HCLK_Frequency)/2 / IC2Value) / (15 - 1));        /* 100 - 10000
     hz */
 119         }
 120         else if(freqRange == MORE_THAN_10K) {
 121           Frequency = (((RCC_Clocks.HCLK_Frequency)/2 / IC2Value) / 1);         /* 10000 - ~10M hz */
 122         }
 123         else {
 124           Frequency = ((RCC_Clocks.HCLK_Frequency)/2 / IC2Value);         /* DEFAULT - 1.28k - 1M hz */
 125         }
 126       }
 127       else
 128       {
 129         DutyCycle = 0;
 130         Frequency = 0;
 131       }
 132     }
 133     else if (function == FREQUENCY_KEY_SHIFT)
 134     {
 135       NVIC_InitTypeDef NVIC_InitStructure;
 136
 137       // Set the FSK_Chnage flag to true to signal DDS frequencies need updating
 138       FSK_Change = true;
 139
 140       // Based on the current toggleBit value, change the DDS frequency
 141       if (toggleBit == 1)
 142       {
 143         FSK_Freq = HIGH;
 144         toggleBit = 0;
 145       }
 146       else
 147       {
 148         FSK_Freq = LOW;
 149         toggleBit = 1;
 150       }
 151
 152       // Problems encounter with the timers global interrupt flag, so re-enable
 153       // the interrupt which appears to fix the problem for some reason.
 154       NVIC_InitStructure.NVIC_IRQChannel = TIM4_IRQn;
```

```
155       NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
156       NVIC_Init(&NVIC_InitStructure);
157     }
158 
159     /* Clear TIM4 Capture compare interrupt pending bit */
160     TIM_ClearITPendingBit(TIM4, TIM_IT_CC2);
161   }
162 
```