```c
/******************** (C) COPYRIGHT 2008 STMicroelectronics ********************
* File Name          : stm32f10x_it.c
* Author             : MCD Application Team
* Version            : V2.0.1
* Date               : 06/13/2008
* Description        : Main Interrupt Service Routines.
*                      This file provides template for all exceptions handler
*                      and peripherals interrupt service routine.
********************************************************************************
* THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS
* WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE TIME.
* AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY DIRECT,
* INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING FROM THE
* CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE CODING
* INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.
*******************************************************************************/

/* Includes ------------------------------------------------------------------*/
#include "stm32f10x_it.h"
#include "stm32_dsp.h"
#include "dimmers_storage.h"
#include "dimmers_wdg.h"
#include "dimmers_exti.h"
#include "dimmers_gpio.h"
#include "dimmers_tim.h"
#include "dimmers_bkp.h"
#include "dimmers_i2c.h"

#include "freeRTOS.h"
#include "task.h"
#include "semphr.h"

#define  I2C_EVENT_SLAVE_BYTE_RECEIVED_AND_STOP       ((u32)0x00000050)
#define  I2C_EVENT_SLAVE_BYTE_RECEIVED_WITH_BTF       ((u32)0x00020044)
#define  I2C_EVENT_MASTER_MODE_SB                     ((u32)0x00000001)
#define  I2C_EVENT_SLAVE_TRANSMITTER_STOP             ((u32)0x00060090)
#define  I2C_EVENT_SLAVE_TRANSMITTER_MATCHED          ((u32)0x00060080)
#define  I2C_EVENT_SLAVE_BYTE_TRANSMITTED_NO_BTF      ((u32)0x00060080)
#define  I2C_EVENT_MASTER_BYTE_RECEIVED_WITH_BTF      ((u32)0x00030044)
#define  I2C_EVENT_SLAVE_BYTE_NOT_TRANSMITTED_NACK    ((u32)0x00060480)

#define  REG_HARD_FAULT *((u32 *)0xE000ED2C)



/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/

/* Private functions ---------------------------------------------------------*/

/*******************************************************************************
* Function Name  : NMIException
* Description    : This function handles NMI exception.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void NMIException(void)
{
}

/*******************************************************************************
* Function Name  : HardFaultException
* Description    : This function handles Hard Fault exception.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
```

```c
void HardFaultException(void)
{
    static vu32 fault;

  /* Go to infinite loop when Hard Fault exception occurs */
  while (1)
  {
    fault = REG_HARD_FAULT & 0b11000000000000000000000000000010;

  }
}

/*******************************************************************************
* Function Name  : MemManageException
* Description    : This function handles Memory Manage exception.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void MemManageException(void)
{
  /* Go to infinite loop when Memory Manage exception occurs */
  while (1)
  {
  }
}

/*******************************************************************************
* Function Name  : BusFaultException
* Description    : This function handles Bus Fault exception.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void BusFaultException(void)
{
  /* Go to infinite loop when Bus Fault exception occurs */
  while (1)
  {
  }
}

/*******************************************************************************
* Function Name  : UsageFaultException
* Description    : This function handles Usage Fault exception.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void UsageFaultException(void)
{
  /* Go to infinite loop when Usage Fault exception occurs */
  while (1)
  {
  }
}

/*******************************************************************************
* Function Name  : DebugMonitor
* Description    : This function handles Debug Monitor exception.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void DebugMonitor(void)
{
}

/*******************************************************************************
* Function Name  : SVCHandler
* Description    : This function handles SVCall exception.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void SVCHandler(void)
{
}

/*******************************************************************************
* Function Name  : PendSVC
* Description    : This function handles PendSVC exception.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void PendSVC(void)
{
}

/*******************************************************************************
* Function Name  : SysTickHandler
* Description    : This function handles SysTick Handler.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void SysTickHandler(void)
{
}

/*******************************************************************************
* Function Name  : WWDG_IRQHandler
* Description    : This function handles WWDG interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void WWDG_IRQHandler(void)
{
    /* Update WWDG counter */
    WWDG_SetCounter(WDG_COUNTER_VALUE);
    /* Clear EWI flag */
    WWDG_ClearFlag();
}

/*******************************************************************************
* Function Name  : PVD_IRQHandler
* Description    : This function handles PVD interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void PVD_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : TAMPER_IRQHandler
* Description    : This function handles Tamper interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TAMPER_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : RTC_IRQHandler
* Description    : This function handles RTC global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void RTC_IRQHandler(void)
```

```c
{
    static portBASE_TYPE xFromHigher;
    xFromHigher = pdFALSE;

    if (RTC_GetITStatus(RTC_IT_SEC) != RESET)
    {

        if ( xTaskGetSchedulerState() == taskSCHEDULER_RUNNING )
            xSemaphoreGiveFromISR ( xRefreshSync , &xFromHigher );

        RTC_ClearITPendingBit(RTC_IT_SEC);

        if ( xFromHigher == pdTRUE ){
            vPortYieldFromISR();
        }
    }
}

/*******************************************************************************
* Function Name  : FLASH_IRQHandler
* Description    : This function handles Flash interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void FLASH_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : RCC_IRQHandler
* Description    : This function handles RCC interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void RCC_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : EXTI0_IRQHandler
* Description    : This function handles External interrupt Line 0 request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void EXTI0_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : EXTI1_IRQHandler
* Description    : This function handles External interrupt Line 1 request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void EXTI1_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : EXTI2_IRQHandler
* Description    : This function handles External interrupt Line 2 request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void EXTI2_IRQHandler(void)
{
}
```

```
/*******************************************************************************
* Function Name  : EXTI3_IRQHandler
* Description    : This function handles External interrupt Line 3 request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void EXTI3_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : EXTI4_IRQHandler
* Description    : This function handles External interrupt Line 4 request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void EXTI4_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : DMA1_Channel1_IRQHandler
* Description    : This function handles DMA1 Channel 1 interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void DMA1_Channel1_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : DMA1_Channel2_IRQHandler
* Description    : This function handles DMA1 Channel 2 interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void DMA1_Channel2_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : DMA1_Channel3_IRQHandler
* Description    : This function handles DMA1 Channel 3 interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void DMA1_Channel3_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : DMA1_Channel4_IRQHandler
* Description    : This function handles DMA1 Channel 4 interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void DMA1_Channel4_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : DMA1_Channel5_IRQHandler
* Description    : This function handles DMA1 Channel 5 interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
```

```c
void DMA1_Channel5_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : DMA1_Channel6_IRQHandler
* Description    : This function handles DMA1 Channel 6 interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void DMA1_Channel6_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : DMA1_Channel7_IRQHandler
* Description    : This function handles DMA1 Channel 7 interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void DMA1_Channel7_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : ADC1_2_IRQHandler
* Description    : This function handles ADC1 and ADC2 global interrupts requests.
* Input          : Xi,Co (global)
* Output         : Yi (global)
* Return         : None
* Timings        :
*                    5.6 µs mem shift   (measured stand alone)
*                    7.9 µs FIR MAC     (measured stand alone)
*                    1   µs Ct
*                   14.5 µs total
*
*                   15.2 µs all together
*                   500  ns overhead on interrupt latency and calls
*
*                   Coef from CONST out of flash gives 1.4µs extra overhead
*
* Samples        :
*                       pot meter at minimum with 1.24V pp:
*
*                       max = 2881
*                       min = 1464
*                       (max-min)/2 = 708
*                       zero = 708 + min = 2125 -> 805µV/level = 1.71V
*
*
* - boolean introduced to avoid multiple zero samples resetting the timer
*
*******************************************************************************/
void ADC1_2_IRQHandler(void)
{
    static bool tim = FALSE;

    ADC_ClearITPendingBit(ADC1, ADC_IT_EOC);
    fir_16by16_stm32(Yi , Xi , &co , 4);
    Yi[0] >>= 15;

    if ( TIM_GetCounter(TIM2) > TIM_CHECK_ON_HALF )
        tim = TRUE;

    if (    (Yi[0] < ( theZero + ZeroCrossTreshold ))
        &&  (Yi[0] > ( theZero - ZeroCrossTreshold )) )
    {
        if ( tim ) TimResetAll();
        tim = FALSE;
    }
    mem_shift(Xi,68);

    if (counterMinMax++ <= SAMPLES_MIN_MAX )
    {
        if (Xi[0] > theMax ) theMax = Xi[0];
        if (Xi[0] < theMin ) theMin = Xi[0];
        theZero = (theMax - theMin)/2 + theMin;
    }
}

/*******************************************************************************
* Function Name  : USB_HP_CAN_TX_IRQHandler
* Description    : This function handles USB High Priority or CAN TX interrupts
*                  requests.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void USB_HP_CAN_TX_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : USB_LP_CAN_RX0_IRQHandler
* Description    : This function handles USB Low Priority or CAN RX0 interrupts
*                  requests.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void USB_LP_CAN_RX0_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : CAN_RX1_IRQHandler
* Description    : This function handles CAN RX1 interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void CAN_RX1_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : CAN_SCE_IRQHandler
* Description    : This function handles CAN SCE interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void CAN_SCE_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : EXTI9_5_IRQHandler
* Description    : This function handles External lines 9 to 5 interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void EXTI9_5_IRQHandler(void)
{
    static portBASE_TYPE xFromHigher;
    xFromHigher = pdFALSE;

    if(EXTI_GetITStatus(EXTI_BUTTON_1) != RESET)
    {
        if ( xTaskGetSchedulerState() == taskSCHEDULER_RUNNING )
            xSemaphoreGiveFromISR ( xSemaphoreButtonArrayHandle[0] , &xFromHigher );
        EXTI_ClearITPendingBit(EXTI_Line6);

        if ( xFromHigher == pdTRUE ){
            vPortYieldFromISR();
            xFromHigher = pdFALSE;
        }
    }
    if(EXTI_GetITStatus(EXTI_BUTTON_2) != RESET)
    {
        if ( xTaskGetSchedulerState() == taskSCHEDULER_RUNNING )
            xSemaphoreGiveFromISR ( xSemaphoreButtonArrayHandle[1] , &xFromHigher );

        EXTI_ClearITPendingBit(EXTI_Line7);

        if ( xFromHigher == pdTRUE ){
            vPortYieldFromISR();
            xFromHigher = pdFALSE;
        }
    }
    if(EXTI_GetITStatus(EXTI_BUTTON_3) != RESET)
    {
        if ( xTaskGetSchedulerState() == taskSCHEDULER_RUNNING )
            xSemaphoreGiveFromISR ( xSemaphoreButtonArrayHandle[2] , &xFromHigher );

        EXTI_ClearITPendingBit(EXTI_Line8);

        if ( xFromHigher == pdTRUE ){
            vPortYieldFromISR();
            xFromHigher = pdFALSE;
        }
    }
    if(EXTI_GetITStatus(EXTI_BUTTON_4) != RESET)
    {
        if ( xTaskGetSchedulerState() == taskSCHEDULER_RUNNING )
            xSemaphoreGiveFromISR ( xSemaphoreButtonArrayHandle[3] , &xFromHigher );

        EXTI_ClearITPendingBit(EXTI_Line9);

        if ( xFromHigher == pdTRUE ){
            vPortYieldFromISR();
            xFromHigher = pdFALSE;
        }
    }
}

/*******************************************************************************
* Function Name  : TIM1_BRK_IRQHandler
* Description    : This function handles TIM1 Break interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM1_BRK_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : TIM1_UP_IRQHandler
* Description    : This function handles TIM1 overflow and update interrupt
*                  request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM1_UP_IRQHandler(void)
{
    TIM_ClearITPendingBit(TIM1, TIM_IT_Update );
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}

/*******************************************************************************
* Function Name  : TIM1_TRG_COM_IRQHandler
* Description    : This function handles TIM1 Trigger and commutation interrupts
*                  requests.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM1_TRG_COM_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : TIM1_CC_IRQHandler
* Description    : This function handles TIM1 capture compare interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM1_CC_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : TIM2_IRQHandler
* Description    : This function handles TIM2 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM2_IRQHandler(void)
{
    static bool forceHold1, forceHold2 = FALSE;

    if (TIM_GetITStatus(TIM2, TIM_IT_CC1) != RESET)
    {
        TIM_ClearITPendingBit(TIM2, TIM_IT_CC1);
        if ( Lights[LIGHT1_INDEX].value != LIGHT_OFF ){
            GPIO_SetBits(GPIOC, PIN_LIGHT_1);
            forceHold1 = TRUE;
        }
    }
    else if (TIM_GetITStatus(TIM2, TIM_IT_CC2) != RESET)
    {
        TIM_ClearITPendingBit(TIM2, TIM_IT_CC2);
        if ( (Lights[LIGHT1_INDEX].value != LIGHT_OFF) || forceHold1 ){
            GPIO_ResetBits(GPIOC, PIN_LIGHT_1);
            forceHold1 = FALSE;
        }
    }
    else if (TIM_GetITStatus(TIM2, TIM_IT_CC3) != RESET)
    {
        TIM_ClearITPendingBit(TIM2, TIM_IT_CC3);
        if ( Lights[LIGHT2_INDEX].value != LIGHT_OFF ){
            GPIO_SetBits(GPIOC, PIN_LIGHT_2);
            forceHold2 = TRUE;
        }
    }
    else if (TIM_GetITStatus(TIM2, TIM_IT_CC4) != RESET)
    {
        TIM_ClearITPendingBit(TIM2, TIM_IT_CC4);
        if ( (Lights[LIGHT2_INDEX].value != LIGHT_OFF) || forceHold2 ){
            GPIO_ResetBits(GPIOC, PIN_LIGHT_2);
            forceHold2 = FALSE;
        }
    }
}

/*******************************************************************************
* Function Name  : TIM3_IRQHandler
* Description    : This function handles TIM3 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM3_IRQHandler(void)
```

```c
{
    static bool forceHold3, forceHold4 = FALSE;

    if (TIM_GetITStatus(TIM3, TIM_IT_CC1) != RESET)
    {
        TIM_ClearITPendingBit(TIM3, TIM_IT_CC1);
        if ( Lights[LIGHT3_INDEX].value != LIGHT_OFF ){
            GPIO_SetBits(GPIOC, PIN_LIGHT_3);
            forceHold3 = TRUE;
        }
    }
    else if (TIM_GetITStatus(TIM3, TIM_IT_CC2) != RESET)
    {
        TIM_ClearITPendingBit(TIM3, TIM_IT_CC2);
        if ( (Lights[LIGHT3_INDEX].value != LIGHT_OFF) || forceHold3 ){
            GPIO_ResetBits(GPIOC, PIN_LIGHT_3);
            forceHold3 = FALSE;
        }
    }
    else if (TIM_GetITStatus(TIM3, TIM_IT_CC3) != RESET)
    {
        TIM_ClearITPendingBit(TIM3, TIM_IT_CC3);
        if ( Lights[LIGHT4_INDEX].value != LIGHT_OFF ){
            GPIO_SetBits(GPIOC, PIN_LIGHT_4);
            forceHold4 = TRUE;
        }
    }
    else if (TIM_GetITStatus(TIM3, TIM_IT_CC4) != RESET)
    {
        TIM_ClearITPendingBit(TIM3, TIM_IT_CC4);
        if ( (Lights[LIGHT4_INDEX].value != LIGHT_OFF) || forceHold4 ){
            GPIO_ResetBits(GPIOC, PIN_LIGHT_4);
            forceHold4 = FALSE;
        }
    }
}

/*******************************************************************************
* Function Name  : TIM4_IRQHandler
* Description    : This function handles TIM4 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM4_IRQHandler(void)
{
    static bool forceHold5, forceHold6 = FALSE;

    if (TIM_GetITStatus(TIM4, TIM_IT_CC1) != RESET)
    {
        TIM_ClearITPendingBit(TIM4, TIM_IT_CC1);
        if ( Lights[LIGHT5_INDEX].value != LIGHT_OFF ){
            GPIO_SetBits(GPIOC, PIN_LIGHT_5);
            forceHold5 = TRUE;
        }
    }
    else if (TIM_GetITStatus(TIM4, TIM_IT_CC2) != RESET)
    {
        TIM_ClearITPendingBit(TIM4, TIM_IT_CC2);
        if ( (Lights[LIGHT5_INDEX].value != LIGHT_OFF) || forceHold5 ){
            GPIO_ResetBits(GPIOC, PIN_LIGHT_5);
            forceHold5 = FALSE;
        }
    }
    else if (TIM_GetITStatus(TIM4, TIM_IT_CC3) != RESET)
    {
        TIM_ClearITPendingBit(TIM4, TIM_IT_CC3);
        if ( Lights[LIGHT6_INDEX].value != LIGHT_OFF ){
            GPIO_SetBits(GPIOC, PIN_LIGHT_6);
            forceHold6 = TRUE;
        }
    }
    else if (TIM_GetITStatus(TIM4, TIM_IT_CC4) != RESET)
    {
        TIM_ClearITPendingBit(TIM4, TIM_IT_CC4);
        if ( (Lights[LIGHT6_INDEX].value != LIGHT_OFF) || forceHold6 ){
            GPIO_ResetBits(GPIOC, PIN_LIGHT_6);
            forceHold6 = FALSE;
        }
    }
}

/*******************************************************************************
* Function Name  : I2C1_EV_IRQHandler
* Description    : This function handles I2C1 Event interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void I2C1_EV_IRQHandler(void)
{
    static u32 evt;

    evt =  I2C_GetLastEvent(I2C1);

    switch (evt)
    {
        case I2C_EVENT_SLAVE_TRANSMITTER_ADDRESS_MATCHED:
             TxIdx = 0;
             I2C_ITConfig(I2C1, I2C_IT_BUF , ENABLE);
             I2C_SendData(I2C1, InternData.values[TxIdx++]);
        break;
        case I2C_EVENT_SLAVE_BYTE_TRANSMITTED:
        case I2C_EVENT_SLAVE_BYTE_TRANSMITTED_NO_BTF:
            if ( TxIdx <  sizeof (TBKPData) )
                I2C_SendData(I2C1, InternData.values[TxIdx++]);
        break;
        case I2C_EVENT_SLAVE_ACK_FAILURE:
        //case I2C_EVENT_SLAVE_BYTE_NOT_TRANSMITTED_NACK:
            I2C_ClearFlag(I2C1, I2C_IT_AF);
        break;
        default:
            GPIO_WriteBit(GPIOA, GPIO_Pin_4, (BitAction)((1-GPIO_ReadOutputDataBit(GPIOA,   ↙
    GPIO_Pin_4))));
        break;
  }
}

/*******************************************************************************
* Function Name  : I2C1_ER_IRQHandler
* Description    : This function handles I2C1 Error interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void I2C1_ER_IRQHandler(void)
{
  if (I2C_GetITStatus(I2C1, I2C_IT_AF))
  {
    I2C_ClearITPendingBit(I2C1, I2C_IT_AF);

  }
  if (I2C_GetITStatus(I2C1, I2C_IT_BERR))
  {
    I2C_ClearITPendingBit(I2C1, I2C_IT_BERR);
  }
  if (I2C_GetITStatus(I2C1, I2C_IT_ARLO))
  {
    I2C_ClearITPendingBit(I2C1, I2C_IT_ARLO);
  }
  if (I2C_GetITStatus(I2C1, I2C_IT_OVR))
  {
    I2C_ClearITPendingBit(I2C1, I2C_IT_OVR);
  }
}
```

```
/*******************************************************************************
* Function Name  : I2C2_EV_IRQHandler
* Description    : This function handles I2C2 Event interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void I2C2_EV_IRQHandler(void)
{
   static u32 evt;

    //evt =  I2C_GetLastEvent(I2C2);

    switch (I2C_GetLastEvent(I2C2)){
        case I2C_EVENT_MASTER_MODE_SELECT:
             I2C_AcknowledgeConfig(I2C2, ENABLE);
             I2C_Send7bitAddress(I2C2 , I2C1_SLAVE_ADDRESS7_CORTEX2 ,                    ↙
    I2C_Direction_Receiver);
        break;
        case I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED:
             RxIdx = 0;
        break;
        case I2C_EVENT_MASTER_BYTE_RECEIVED:
        case I2C_EVENT_MASTER_BYTE_RECEIVED_WITH_BTF:
            if (RxIdx < sizeof (TBKPData))
              ExternData.values[RxIdx++] = I2C_ReceiveData(I2C2);
            if( RxIdx == (sizeof (TBKPData) - 1) )
            {
                I2C_AcknowledgeConfig(I2C2, DISABLE);
                I2C_GenerateSTOP(I2C2, ENABLE);
                GPIO_WriteBit(GPIOC, GPIO_Pin_12, (BitAction)((1-GPIO_ReadOutputDataBit      ↙
    (GPIOC, GPIO_Pin_12))));
            }
        break;
        case I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED:
             TxIdx = 0;
             I2C_SendData(I2C2, InternData.values[TxIdx++]);
        break;
        case I2C_EVENT_MASTER_BYTE_TRANSMITTING:
        case I2C_EVENT_MASTER_BYTE_TRANSMITTED:
            if ( TxIdx <  sizeof (TBKPData) )
                I2C_SendData(I2C2, InternData.values[TxIdx++]);
            else
                I2C_GenerateSTOP(I2C2, ENABLE);
        break;
        case I2C_EVENT_SLAVE_RECEIVER_ADDRESS_MATCHED:
            RxIdx = 0;
            I2C_ITConfig(I2C2, I2C_IT_BUF , ENABLE);
        break;
        case I2C_EVENT_SLAVE_BYTE_RECEIVED:
        case I2C_EVENT_SLAVE_BYTE_RECEIVED_WITH_BTF:
           if (RxIdx < sizeof (TBKPData))
              ExternData.values[RxIdx++] = I2C_ReceiveData(I2C2);
        break;
        case I2C_EVENT_SLAVE_STOP_DETECTED:
        case I2C_EVENT_SLAVE_BYTE_RECEIVED_AND_STOP:
            I2C_ITConfig( I2C2, I2C_IT_BUF , DISABLE );
            if (RxIdx < sizeof (TBKPData))
                ExternData.values[RxIdx++] = I2C_ReceiveData(I2C2);
            (void)(I2C_GetITStatus(I2C2, I2C_IT_STOPF));
            I2C_Cmd(I2C2, ENABLE);
            I2CRecReady = TRUE;
        break;
        case I2C_EVENT_SLAVE_ACK_FAILURE:
            I2C_ClearFlag(I2C2, I2C_IT_AF);
        break;
        default:
//             if ( RxIdx == (sizeof (TBKPData)-1 ) )
//                ExternData.values[RxIdx++] = I2C_ReceiveData(I2C2);
        break;
    }
}
```

```
/*******************************************************************************
* Function Name  : I2C2_ER_IRQHandler
* Description    : This function handles I2C2 Error interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void I2C2_ER_IRQHandler(void)
{

  if (I2C_GetITStatus(I2C2, I2C_IT_AF))
  {
    I2C_ClearITPendingBit(I2C2, I2C_IT_AF);
  }
  if (I2C_GetITStatus(I2C2, I2C_IT_BERR))
  {
    I2C_ClearITPendingBit(I2C2, I2C_IT_BERR);
  }
  if (I2C_GetITStatus(I2C2, I2C_IT_ARLO))
  {
    I2C_ClearITPendingBit(I2C2, I2C_IT_ARLO);
  }
  if (I2C_GetITStatus(I2C2, I2C_IT_OVR))
  {
    I2C_ClearITPendingBit(I2C2, I2C_IT_OVR);
  }
}

/*******************************************************************************
* Function Name  : SPI1_IRQHandler
* Description    : This function handles SPI1 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void SPI1_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : SPI2_IRQHandler
* Description    : This function handles SPI2 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void SPI2_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : USART1_IRQHandler
* Description    : This function handles USART1 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void USART1_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : USART2_IRQHandler
* Description    : This function handles USART2 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void USART2_IRQHandler(void)
{
}

/*******************************************************************************
```

```
* Function Name  : USART3_IRQHandler
* Description    : This function handles USART3 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void USART3_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : EXTI15_10_IRQHandler
* Description    : This function handles External lines 15 to 10 interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void EXTI15_10_IRQHandler(void)
{
    static portBASE_TYPE xFromHigher;
    xFromHigher = pdFALSE;

    if(EXTI_GetITStatus(EXTI_BUTTON_5) != RESET)
    {
        if ( xTaskGetSchedulerState() == taskSCHEDULER_RUNNING )
            xSemaphoreGiveFromISR ( xSemaphoreButtonArrayHandle[4] , &xFromHigher );

        EXTI_ClearITPendingBit(EXTI_Line10);

        if ( xFromHigher == pdTRUE ){
            vPortYieldFromISR();
            xFromHigher = pdFALSE;
        }
    }

    if(EXTI_GetITStatus(EXTI_BUTTON_6) != RESET)
    {
        if ( xTaskGetSchedulerState() == taskSCHEDULER_RUNNING )
            xSemaphoreGiveFromISR ( xSemaphoreButtonArrayHandle[5] , &xFromHigher );

        EXTI_ClearITPendingBit(EXTI_Line11);

        if ( xFromHigher == pdTRUE ){
            vPortYieldFromISR();
            xFromHigher = pdFALSE;
        }
    }
}

/*******************************************************************************
* Function Name  : RTCAlarm_IRQHandler
* Description    : This function handles RTC Alarm interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void RTCAlarm_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : USBWakeUp_IRQHandler
* Description    : This function handles USB WakeUp interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void USBWakeUp_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : TIM8_BRK_IRQHandler
* Description    : This function handles TIM8 Break interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM8_BRK_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : TIM8_UP_IRQHandler
* Description    : This function handles TIM8 overflow and update interrupt
*                  request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM8_UP_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : TIM8_TRG_COM_IRQHandler
* Description    : This function handles TIM8 Trigger and commutation interrupts
*                  requests.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM8_TRG_COM_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : TIM8_CC_IRQHandler
* Description    : This function handles TIM8 capture compare interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM8_CC_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : ADC3_IRQHandler
* Description    : This function handles ADC3 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void ADC3_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : FSMC_IRQHandler
* Description    : This function handles FSMC global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void FSMC_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : SDIO_IRQHandler
* Description    : This function handles SDIO global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
```

```
void SDIO_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : TIM5_IRQHandler
* Description    : This function handles TIM5 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM5_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : SPI3_IRQHandler
* Description    : This function handles SPI3 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void SPI3_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : UART4_IRQHandler
* Description    : This function handles UART4 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void UART4_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : UART5_IRQHandler
* Description    : This function handles UART5 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void UART5_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : TIM6_IRQHandler
* Description    : This function handles TIM6 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM6_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : TIM7_IRQHandler
* Description    : This function handles TIM7 global interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void TIM7_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : DMA2_Channel1_IRQHandler
* Description    : This function handles DMA2 Channel 1 interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void DMA2_Channel1_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : DMA2_Channel2_IRQHandler
* Description    : This function handles DMA2 Channel 2 interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void DMA2_Channel2_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : DMA2_Channel3_IRQHandler
* Description    : This function handles DMA2 Channel 3 interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void DMA2_Channel3_IRQHandler(void)
{
}

/*******************************************************************************
* Function Name  : DMA2_Channel4_5_IRQHandler
* Description    : This function handles DMA2 Channel 4 and DMA2 Channel 5
*                  interrupt request.
* Input          : None
* Output         : None
* Return         : None
*******************************************************************************/
void DMA2_Channel4_5_IRQHandler(void)
{
}

/******************* (C) COPYRIGHT 2008 STMicroelectronics *****END OF FILE****/
```