```
#define INTR
#define ENABLE_UART
int Address = 0x69;

/**
      *
      * Brandon Mouser
      * U0962682
      *
******************************************************************************
      * File Name                          : main.c
      * Description                        : Main program body
******************************************************************************
      ** This notice applies to any and all portions of this file
      * that are not between comment pairs USER CODE BEGIN and
      * USER CODE END. Other portions of this file, whether
      * inserted by the user or by software development tools
      * are owned by their respective copyright owners.
      *
      * COPYRIGHT(c) 2018 STMicroelectronics
      *
      * Redistribution and use in source and binary forms, with or without
modification,
      * are permitted provided that the following conditions are met:
      *      1. Redistributions of source code must retain the above copyright
notice,
      *                 this list of conditions and the following disclaimer.
      *      2. Redistributions in binary form must reproduce the above copyright
notice,
      *                 this list of conditions and the following disclaimer in the
documentation
      *                 and/or other materials provided with the distribution.
      *      3. Neither the name of STMicroelectronics nor the names of its
contributors
      *                 may be used to endorse or promote products derived from
this software
      *                 without specific prior written permission.
      *
      * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
      * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
      * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE
      * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE
      * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
      * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
      * SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
      * CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY,
      * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE
USE
      * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
      *
******************************************************************************
      */
```

```
/* Includes ------------------------------------------------------------------*/
//#include "led.h"
//#include "irq.h"
//#include "timer.h"

#include "main.h"
#include "stm32f072xb.h"
#include "stm32f0xx_hal.h"
void _Error_Handler(char * file, int line);

/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
/* Private variables ---------------------------------------------------------*/

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes -----------------------------------------------*/

/* USER CODE END PFP */

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */
void EnableLEDPin(uint32_t PinNo)
{
      GPIOC->BSRR = PinNo;
}

void DisableLEDPin(uint32_t PinNo)
{
      const uint32_t UpperHalf = 16;
      GPIOC->BSRR = PinNo << UpperHalf;
}


void ToggleLEDPin(uint32_t PinNo)
{
      if ((GPIOC->ODR & PinNo) != 0X00u)
      {
            DisableLEDPin(PinNo);
      }
      else
      {
            EnableLEDPin(PinNo);
      }

}

void InitGPIOCPin(uint32_t PinIndex)
{
	const uint32_t Output = GPIO_MODE_OUTPUT_PP;
	const uint32_t Speed = GPIO_SPEED_FREQ_LOW;
	const uint32_t Pull = GPIO_PULLDOWN;

	/* Configure output type */
	uint32_t OutputMode = GPIOC->MODER;
	OutputMode &= ~(GPIO_MODER_MODER0 << (0x2 * PinIndex));
	OutputMode |= (Output & 0x03) << (0x2 * PinIndex);
	GPIOC->MODER = OutputMode;

	/* Configure i/o output type  */
	uint32_t TypeMode = GPIOC->OTYPER;
	TypeMode &= ~(GPIO_OTYPER_OT_0 << (0x2 * PinIndex));
	TypeMode |= (((GPIO_MODE_OUTPUT_PP & 0x10) >> 4U) << (0x2 * PinIndex));
	GPIOC->OTYPER = TypeMode;

	/* Configure i/o output speed */
	uint32_t SpeedMode = GPIOC->OSPEEDR;
	SpeedMode &= ~(GPIO_OSPEEDER_OSPEEDR0 << (0x2 * PinIndex));
	SpeedMode |= (Speed << (0x2 * PinIndex));
	GPIOC->OSPEEDR = SpeedMode;

	/* Setup pull-up or pull-down for this pin */
	uint32_t PullUpDownMode = GPIOC->PUPDR;
	PullUpDownMode &= ~(GPIO_PUPDR_PUPDR0 << (0x2 * PinIndex));
	PullUpDownMode |= ((Pull) << (0x2 * PinIndex));
	GPIOC->PUPDR = PullUpDownMode;
}

void InitBGPIOPin(uint32_t PinIndex)
{
	const uint32_t Output = GPIO_MODE_OUTPUT_PP;
	const uint32_t Speed = GPIO_SPEED_FREQ_LOW;
	const uint32_t Pull = GPIO_NOPULL;

	/* Configure output type */
	uint32_t OutputMode = GPIOB->MODER;
	OutputMode &= ~(GPIO_MODER_MODER0 << (0x2 * PinIndex));
	OutputMode |= (Output & 0x03) << (0x2 * PinIndex);
	GPIOB->MODER = OutputMode;

	/* Configure i/o output type  */
	uint32_t TypeMode = GPIOB->OTYPER;
	TypeMode &= ~(GPIO_OTYPER_OT_0 << (0x2 * PinIndex));
	TypeMode |= (((GPIO_MODE_OUTPUT_PP & 0x10) >> 4U) << (0x2 * PinIndex));
	GPIOB->OTYPER = TypeMode;

	/* Configure i/o output speed */
	uint32_t SpeedMode = GPIOB->OSPEEDR;
	SpeedMode &= ~(GPIO_OSPEEDER_OSPEEDR0 << (0x2 * PinIndex));
	SpeedMode |= (Speed << (0x2 * PinIndex));
	GPIOB->OSPEEDR = SpeedMode;

	/* Setup pull-up or pull-down for this pin */
	uint32_t PullUpDownMode = GPIOB->PUPDR;
	PullUpDownMode &= ~(GPIO_PUPDR_PUPDR0 << (0x2 * PinIndex));
	PullUpDownMode |= ((Pull) << (0x2 * PinIndex));
	GPIOB->PUPDR = PullUpDownMode;
}
```

```
void InitI2CGPIOPin(uint32_t PinIndex)
{
      const uint32_t Output = GPIO_MODE_AF_OD;
      const uint32_t Speed = GPIO_SPEED_FREQ_LOW;

      /* Configure output type */
      uint32_t OutputMode = GPIOC->MODER;
      OutputMode &= ~(GPIO_MODER_MODER0 << (0x2 * PinIndex));
      OutputMode |= (Output & 0x03) << (0x2 * PinIndex);
      GPIOC->MODER = OutputMode;

      /* Configure i/o output type  */
      uint32_t TypeMode = GPIOC->OTYPER;
      TypeMode &= ~(GPIO_OTYPER_OT_0 << (0x2 * PinIndex));
      TypeMode |= (((GPIO_MODE_OUTPUT_PP & 0x10) >> 4U) << (0x2 * PinIndex));
      GPIOC->OTYPER = TypeMode;

      /* Configure i/o output speed */
      uint32_t SpeedMode = GPIOC->OSPEEDR;
      SpeedMode &= ~(GPIO_OSPEEDER_OSPEEDR0 << (0x2 * PinIndex));
      SpeedMode |= (Speed << (0x2 * PinIndex));
      GPIOC->OSPEEDR = SpeedMode;
}

void InitGPIOCPinAlternate(uint32_t PinIndex)
{
      const uint32_t Output = GPIO_MODE_AF_PP;
      const uint32_t Speed = GPIO_SPEED_FREQ_LOW;
      const uint32_t Pull = GPIO_NOPULL;

      /* Configure output type */
      uint32_t OutputMode = GPIOC->MODER;
      OutputMode &= ~(GPIO_MODER_MODER0 << (0x2 * PinIndex));
      OutputMode |= (Output & 0x03) << (0x2 * PinIndex);
      GPIOC->MODER = OutputMode;

      /* Configure i/o output type  */
      uint32_t TypeMode = GPIOC->OTYPER;
      TypeMode &= ~(GPIO_OTYPER_OT_0 << (0x2 * PinIndex));
      TypeMode |= (((GPIO_MODE_OUTPUT_PP & 0x10) >> 4U) << (0x2 * PinIndex));
      GPIOC->OTYPER = TypeMode;

      /* Configure i/o output speed */
      uint32_t SpeedMode = GPIOC->OSPEEDR;
      SpeedMode &= ~(GPIO_OSPEEDER_OSPEEDR0 << (0x2 * PinIndex));
      SpeedMode |= (Speed << (0x2 * PinIndex));
      GPIOC->OSPEEDR = SpeedMode;

      /* Setup pull-up or pull-down for this pin */
      uint32_t PullUpDownMode = GPIOC->PUPDR;
      PullUpDownMode &= ~(GPIO_PUPDR_PUPDR0 << (0x2 * PinIndex));
      PullUpDownMode |= ((Pull) << (0x2 * PinIndex));
      GPIOC->PUPDR = PullUpDownMode;
}

#ifdef ENABLE_UART
void WriteCharRaw(USART_TypeDef *Def, char Cur)
{
      Def->TDR = Cur;
}

void WriteChar(USART_TypeDef *Def, char Cur)
{
      if (Cur == '\n')
      {
            WriteCharRaw(Def, '\r');
      }
      WriteCharRaw(Def, Cur);
      while ((Def->ISR & USART_ISR_TC) != USART_ISR_TC)
      {
      }
}

void FiniWrite()
{
      USART3->ICR |= USART_ICR_TCCF;
}

char RecvChar(USART_TypeDef *Def)
{
      for (;;)
      {
            if ((Def->ISR & USART_ISR_RXNE) == USART_ISR_RXNE)
            {
                  return Def->RDR;
            }
      }
}
#endif

void WriteString(USART_TypeDef *Def, const char *Str)
{
      #ifdef ENABLE_UART
      for (uint16_t Index = 0;; Index++)
      {
            char Cur = Str[Index];
            if (Cur == 0x00)
            {
                  break;
            }
            WriteChar(Def, Cur);
      }
      FiniWrite();
      #endif
}

#define SERIAL_LOG(x) WriteString(USART3, x)

void SendI2C(uint8_t Address, uint8_t Len, const uint8_t *Data)
{
      /* Clear all flags */
      I2C2->ICR = 0xFFFFFFFF;
      I2C2->CR2 &= ~((0x7F << 16) | (0x3FF << 0));
      I2C2->CR2 |= ((uint32_t)Len << 16) | ((uint32_t)Address << 1);

      /* Do not autoend! */
      I2C2->CR2 &= ~I2C_CR2_AUTOEND;
```

```
    /* Clear to write */
    I2C2->CR2 &= ~I2C_CR2_RD_WRN;

    /* Begin! */
    I2C2->CR2 |= I2C_CR2_START;

    /* Actually write. FIXME: handle NACKs */
    for (int Index = 0; Index < Len; ++Index)
    {
        /* Block until ready */
        while ((I2C2->ISR & I2C_ISR_TXIS) == 0){}
        I2C2->TXDR = Data[Index];
    }

    /* Block until done */
    while ((I2C2->ISR & I2C_ISR_TC) == 0){}
}

void ReadI2C(uint8_t Address, uint8_t Len, uint8_t *Data)
{
    /* Clear all flags */
    I2C2->ICR = 0xFFFFFFFF;
    I2C2->CR2 &= ~((0x7F << 16) | (0x3FF << 0));
    I2C2->CR2 |= ((uint32_t)Len << 16) | ((uint32_t)Address << 1);

    /* Do not autoend! */
    I2C2->CR2 &= ~I2C_CR2_AUTOEND;

    /* Set to read */
    I2C2->CR2 |= I2C_CR2_RD_WRN;

    /* Begin! */
    I2C2->CR2 |= I2C_CR2_START;

    /* Actually read */
    for (int Index = 0; Index < Len; ++Index)
    {
        /* Block until ready */
        while ((I2C2->ISR & I2C_ISR_RXNE) == 0){}
        Data[Index] = I2C2->RXDR;
    }
}

void StopI2C()
{
    I2C2->CR2 |= I2C_CR2_STOP;
}

void EnableGPIOBPin(uint32_t PinNo)
{
    GPIOB->BSRR = PinNo;
}

void EnableGPIOCPin(uint32_t PinNo)
{
    GPIOC->BSRR = PinNo;
}
```

```
void itoa16(int Num, char *Out)
{
      /* We don't have an RTOS to provide any kind of malloc, so hope it's big
enough.
       * Numbers have to be written in reverse order, since numbers are right-to-
left for significance and we
       * want to process things in a left-to-right order. (ie, this is a problem
for Arabic numberals)
       */
      int Index = 0;
      do
      {
            int NewNum = Num % 16;
            if (Num < 0)
            {
                  /* Fix negatives... C is a little weird with negative numbers
here. */
                  NewNum = 16 + NewNum;
            }
            Out[Index++] = (NewNum < 10) ? '0' + NewNum : 'A' + (NewNum - 10);
      } while ((Num /= 16) > 0);

      /* Flip the number so we get what we wanted */
      for (int Subindex = 0; Subindex < Index / 2; ++Subindex)
      {
            char Tmp = Out[Subindex];
            Out[Subindex] = Out[Index - Subindex - 1];
            Out[Index - Subindex - 1] = Tmp;
      }

      /* Be 10000% sure we have a null character. */
      Out[Index] = '\0';
}

int16_t GetX()
{
      /* Defined in gyro datasheet */
      const uint8_t XLowReg = 0x28;
      const uint8_t XHighReg = 0x29;

      uint8_t Data[2];

      /* Request low */
      SendI2C(Address, 1, &XLowReg);
      ReadI2C(Address, 1, &(Data[0]));

      /* Request high */
      SendI2C(Address, 1, &XHighReg);
      ReadI2C(Address, 1, &(Data[1]));

      /* Reemmber this is a little endian machine... Is this right? */
      int16_t Result = 0;

      int16_t Low = Data[0];
      int16_t High = Data[1];
      High <<= 8;

      Result = Low | High;
      return Result;
```

```
}

int16_t GetY()
{
      const uint8_t YLowReg = 0x2A;
      const uint8_t YHighReg = 0x2B;

      uint8_t Data[2];

      /* Request low */
      SendI2C(Address, 1, &YLowReg);
      ReadI2C(Address, 1, &(Data[0]));

      /* Request high */
      SendI2C(Address, 1, &YHighReg);
      ReadI2C(Address, 1, &(Data[1]));

      /* Reemmber this is a little endian machine... Is this right? */
      int16_t Result = 0;

      int16_t Low = Data[0];
      int16_t High = Data[1];
      High <<= 8;

      Result = Low | High;
      return Result;
}

void RunGyro()
{
      #define XEn (1 << 0)    /* Enable X-Axis */
      #define YEn (1 << 1)    /* Enable Y-Axis */
      #define ZDis (0 << 2)   /* Disable Z axis */
      #define PD (1 << 3)     /* "Normal or sleep" mode */

      const uint8_t Val = (XEn | YEn | ZDis | PD);

      /* Datasheet states these are on address 0x20.
       * To send, these need the on-device address, and then the content. */

      const uint8_t InitSequence[] = {0x20, Val};
      SendI2C(Address, 2, InitSequence);
      StopI2C();
      SERIAL_LOG("Gyro initialized!");

      for (;;)
      {
            int16_t XDir = GetX();
            int16_t YDir = GetY();

            char Log[10];
            SERIAL_LOG("x is: ");
            itoa16(XDir, Log);
            SERIAL_LOG(Log);
            SERIAL_LOG("\n");

            SERIAL_LOG("y is: ");
            itoa16(YDir, Log);
            SERIAL_LOG(Log);
            SERIAL_LOG("\n");

            StopI2C();

            /* Disable all LEDs */
            DisableLEDPin(GPIO_PIN_6);
            DisableLEDPin(GPIO_PIN_7);
            DisableLEDPin(GPIO_PIN_8);
            DisableLEDPin(GPIO_PIN_9);

            const int16_t Threshold = 0xA0;

            /* Handle the X direction */
            if (XDir > Threshold || XDir < -Threshold)
            {
                  if (XDir > 0)
                  {
                        EnableLEDPin(GPIO_PIN_9);
                  }
                  else
                  {
                        EnableLEDPin(GPIO_PIN_8);
                  }
            }

            /* Handle the Y direction */
            if (YDir > Threshold || YDir < -Threshold)
            {
                  if (YDir > 0)
                  {
                        EnableLEDPin(GPIO_PIN_6);
                  }
                  else
                  {
                        EnableLEDPin(GPIO_PIN_7);
                  }
            }

            /* Lazilly stall for 100ms */
            HAL_Delay(100);
      }
}

void SetupUART()
{
      EXTI->IMR = 0x01;
      EXTI->FTSR = 0x00;
      EXTI->RTSR = 0x01;

      NVIC_EnableIRQ(EXTI0_1_IRQn);
      NVIC_SetPriority(EXTI0_1_IRQn, 3);
      NVIC_SetPriority(SysTick_IRQn, 2);

      /* Get the right baud rate... */
      uint32_t DestBaud = 115200;
      uint32_t SrcClock = HAL_RCC_GetHCLKFreq();
      uint32_t BaudBRR = SrcClock / DestBaud;

      USART3->BRR = BaudBRR;
      USART3->CR3 = USART_CR3_CTSE | USART_CR3_RTSE;
      NVIC_EnableIRQ(USART3_4_IRQn);
      USART3->CR1 = USART_CR1_RXNEIE | USART_CR1_RE | USART_CR1_UE | USART_CR1_TE;
}

void SetupI2CTiming()
{
      /* Wipe out CR1 and CR2 first. */
      I2C2->CR1 = 0;
      I2C2->CR2 = 0;

      I2C2->CR1 = 0;
      I2C2->TIMINGR = 0x00;
      I2C2->TIMINGR |= (0x01 << I2C_TIMINGR_PRESC_Pos);
      I2C2->TIMINGR |= (0x02 << I2C_TIMINGR_SDADEL_Pos) | (0x04 <<
I2C_TIMINGR_SCLDEL_Pos);
      I2C2->TIMINGR |= (0x0F << I2C_TIMINGR_SCLH_Pos) | (0x13 <<
I2C_TIMINGR_SCLL_Pos);
      I2C2->CR1 |= I2C_CR1_PE;
}

void ConfigureRCC()
{
      RCC->AHBENR |= RCC_AHBENR_GPIOBEN | RCC_AHBENR_GPIOCEN;
      #ifdef ENABLE_UART
      RCC->APB1ENR |= RCC_APB1ENR_USART3EN;
      #endif
      RCC->APB1ENR |= RCC_APB2ENR_SYSCFGEN;
      RCC->APB1ENR |= RCC_APB1ENR_I2C2EN;
}

void ConfigureGPIOs()
{
      /* Configure 11, 13, and 15 */
      GPIOC->AFR[0] |= (0x01 << 16) | (0x01 << 20);
      GPIOB->AFR[1] |= (0x01 << 12) | (0x05 << 20);

      GPIOB->MODER |= GPIO_MODER_MODER14_0 | GPIO_MODER_MODER11_1 |
GPIO_MODER_MODER13_1;
      GPIOB->OTYPER |= GPIO_OTYPER_OT_11 | GPIO_OTYPER_OT_13;

      GPIOC->MODER |= GPIO_MODER_MODER0_0;
      GPIOC->OTYPER = 0;

      /* Debug LEDs */
      for (char Index = 6; Index <= 9; Index++)
      {
            InitGPIOCPin(Index);
      }

      #ifdef ENABLE_UART
      /* Setup UART */
      InitGPIOCPinAlternate(4);
      InitGPIOCPinAlternate(5);

      /* Get the right baud rate... */
      uint32_t DestBaud = 115200;
      uint32_t SrcClock = HAL_RCC_GetHCLKFreq();
      uint32_t BaudBRR = SrcClock / DestBaud;
```

```
	USART3->BRR = BaudBRR;
	USART3->CR3 = USART_CR3_CTSE | USART_CR3_RTSE;

	NVIC_EnableIRQ(USART3_4_IRQn);
	USART3->CR1 = USART_CR1_RXNEIE | USART_CR1_RE | USART_CR1_UE | USART_CR1_TE;
	#endif
}

int main(void)
{
	HAL_Init();
	SystemClock_Config();
	ConfigureRCC();
	ConfigureGPIOs();

	EnableGPIOCPin(GPIO_PIN_0);
	EnableGPIOBPin(GPIO_PIN_14);

	SetupI2CTiming();
	SERIAL_LOG("Finished with setup\n");
	SERIAL_LOG("Okay!\n");

	/* Now try to find the slave address. */
	uint8_t WhoAmI = 0x0F;
	SendI2C(Address, 1, &WhoAmI);

	SERIAL_LOG("Done sending!\n");

	uint8_t DevID = 0;
	ReadI2C(Address, 1, &DevID);

	SERIAL_LOG("Got a response: ID is: ");
	char Content[40];
	itoa16(DevID, Content);
	SERIAL_LOG(Content);
	SERIAL_LOG("\n");
	#define GYRO_INIT

	#if defined(GYRO_INIT)
	RunGyro();
	#endif

	for(;;){}
}

/** System Clock Configuration
*/
void SystemClock_Config(void)
{
	RCC_OscInitTypeDef RCC_OscInitStruct;
	RCC_ClkInitTypeDef RCC_ClkInitStruct;

		/**Initializes the CPU, AHB and APB busses clocks
		*/
	RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
	RCC_OscInitStruct.HSIState = RCC_HSI_ON;
	RCC_OscInitStruct.HSICalibrationValue = 16;
	RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
```

```
      if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
      {
            _Error_Handler(__FILE__, __LINE__);
      }

            /**Initializes the CPU, AHB and APB busses clocks
            */
      RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

      |RCC_CLOCKTYPE_PCLK1;
      RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
      RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
      RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

      if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
      {
            _Error_Handler(__FILE__, __LINE__);
      }

            /**Configure the Systick interrupt time
            */
      HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

            /**Configure the Systick
            */
      HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

      /* SysTick_IRQn interrupt configuration */
      HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
      * @brief    This function is executed in case of error occurrence.
      * @param    None
      * @retval None
      */
void _Error_Handler(char * file, int line)
{
      /* USER CODE BEGIN Error_Handler_Debug */
      /* User can add his own implementation to report the HAL error return state
*/
      while(1)
      {
      }
      /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT

/**
       * @brief Reports the name of the source file and the source line number
       * where the assert_param error has occurred.
       * @param file: pointer to the source file name
       * @param line: assert_param error line source number
       * @retval None
       */
void assert_failed(uint8_t* file, uint32_t line)
{
      /* USER CODE BEGIN 6 */
      /* User can add his own implementation to report the file name and line
number,
            ex: printf("Wrong parameters value: file %s on line %d\r\n", file,
line) */
      /* USER CODE END 6 */

}

#endif

/**
      * @}
      */

/**
      * @}
*/

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