```
/**
  ******************************************************************************
  * @file main.c
  * @brief This file contains the main function for Discover example.
  * @author STMicroelectronics - MCD Application Team
  * @version V1.0.0
  * @date 24/11/2011
  ******************************************************************************
  *
  * 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.
  * FOR MORE INFORMATION PLEASE READ CAREFULLY THE LICENSE AGREEMENT FILE LOCATED
  * IN THE ROOT DIRECTORY OF THIS FIRMWARE PACKAGE.
  *
  * <h2><center>&copy; COPYRIGHT 2009 STMicroelectronics</center></h2>
  * @image html logo.bmp
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "stm8s.h"
#include "main.h"
#include "stm8s_clk.h"

/**
  * @addtogroup TIM4_TimeBase_InterruptConfiguration
  * @{
  */

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/

//const u8 iii[1500];

u16 Counter;
u16 PeriodNumber = 0;

uint32_t Ticks_50uS;
uint16_t Ticks_1mS;
uint32_t Ticks_S;

//   10v to 26V
u8 TableInterval[17] = { 65,56,50,45,40,36,32,29,26,23,21,19,17,16,15,14,13};
#define IntervalSlope (57/2)

struct {
        uint8_t Enabled : 1;
        uint8_t Run : 1;
        uint8_t LastOrigin : 1;
        uint16_t Position;                    // Current position (pulse)
        uint16_t Target;                      // Target Position  (pulse)
        uint16_t Overrun;                     // Origin cal overrun (pulse)
        uint8_t Overrun2;                     // Valve hysteresis calibration overrun
        uint16_t ZeroOffset;
        uint8_t Phase;                        // motor coil phase
        uint16_t  MaxOverrun;
        uint16_t  ORGPosition;        // Hall IC origin position
        uint8_t State;
        uint16_t Interval ;                   // Drive interval (mS)
        uint16_t Ticks;
        uint16_t Timeout_1S;          // Drive Timeout
        u8 ExcitationType;
        u8 Origin;
        u8 NormalOpen;
} Drive;

struct {
        uint8_t ID;
        uint8_t ErrNo;
        uint16_t Step;
        uint16_t Pulse;
        uint16_t TmrDrive_1mS;
        uint8_t IsRxSend;                       // is Send Packet ?
        uint8_t DipSW;
        uint8_t PacketLen;
        uint8_t Mode;
} My;
// Run, Test, Test JIG


struct {
        u16 Value[MAX_ADC];
        u8 Idx;
} Adc;

/*
uint8_t POS = 5U;

uint8_t CRLF[2] = {0x0a, 0x0d};

uint8_t iLED = 0;
uint16_t RxLedDelay_1mS = 0;
uint8_t RxDelay_1mS;
uint16_t TmrTx_1mS=0;

uint8_t StateTest;
uint16_t DelayTest_mS;

uint16_t d1;

uint8_t i;
*/
uint8_t PowerDelay_1mS=499;

/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
void TIMER_Configuration(void);
void Blinking_StateMachine(void);

uint16_t CStep( uint16_t pulse );
uint16_t CPulse( uint16_t step );

void ExOff(void);

void Excitation_1Phase( void );
void Excitation_12Phase( void );
void Excitation_2Phase( void );
void DriveService(void);
void StartDriveOR( uint16_t Target, uint8_t overrun );
u8 Interval( u16 Vmon);


void OnTimer4(void);

void Clock_Config(void);
void Gpio_Config(void);
static void Adc_Config( void );
static void Timer4_Config(void);

/* Public functions ----------------------------------------------------------*/
```

```
/**
  * @brief Example firmware main entry point.
  * @par Parameters:
  * None
  * @retval
  * None
  */



void main(void)
{

        Clock_Config();
        Gpio_Config();

        Timer4_Config();

        Adc.Idx = 2;
        Adc_Config(  );


  /* Initialize the Interrupt sensitivity */
  //EXTI_SetExtIntSensitivity(EXTI_PORT_GPIOB, EXTI_SENSITIVITY_RISE_ONLY);

        //Drive.NormalOpen = 1;
        Drive.Enabled = ENABLED;
        Drive.MaxOverrun = OVERRUN;                             // Zero position over run
        Drive.Interval = PULSE_INTERVAL_mS;             // Pulse interval
        Drive.Timeout_1S =  VALVE_TIMEOUT_S;    // Valve timeout

        Drive.Position = CPulse(MAX_STEP);

        Drive.ExcitationType = 1;

        enableInterrupts();

  while ( PowerDelay_1mS )
                ;

        //StartDriveOR( Ou, Ou OVERRUN2 );                              // Goto Zero position
        Drive.Interval = Interval(Adc.Value[1]);
        StartDriveOR( Ou, Ou  );                                // Goto Zero position

        while (Drive.Run)                               // Wait while motor is running
                        ;


  while (1)
  {
   ;
//       Blinking_StateMachine();
  }
}


OnTimer_1S(void)
{
        //if ( ! Drive.Run )
        //      StartDriveOR( CPulse( Adc.Value[0]/5 ) , OVERRUN2  );
                //StartDriveOR( CPulse( Adc.Value[0]/5 ) , Ou  );


}



u16 pos;
```

```
OnTimer_1mS(void)
{
	//u16 pos;
	if ( PowerDelay_1mS )
		PowerDelay_1mS--;

	if (  Ticks_1mS < 999 )
		Ticks_1mS++;
	else
	{
		Ticks_1mS = 0;
		OnTimer_1S();
	}

	pos = Adc.Value[0] / 4;
	pos = pos *  4;
	pos = pos / 5 ;

	if  ( pos < 20 ) pos = 0;

	if ( ! (Ticks_1mS % 500) )
	{
		if ( ! Drive.Run )
		{
			Drive.Interval = Interval(Adc.Value[1]);
			StartDriveOR( CPulse( pos ) , 0u  );
		}
		GPIO_WriteReverse(LED_PORT, LED_PIN);
	}
		//StartDriveOR( CPulse( pos ) , OVERRUN2  );

}

/*   Timer Callback function every 50uS         */
void OnTimer4(void)
{

	Adc_Config(  );
	Drive.Origin =   ! GPIO_ReadInputPin(_ORG_PORT, _ORG_PIN);

	if (  Ticks_50uS < 19 )
		Ticks_50uS++;
	else
	{
		Ticks_50uS = 0;
		OnTimer_1mS();
	}


	if ( Drive.Ticks > 0 )
		Drive.Ticks--;

	if ( Drive.Enabled &&  ( Drive.Ticks == 0 ) )
	{
		DriveService();
		Drive.Ticks = Drive.Interval;
	}

}




//	Motor Excitation all  off
void ExOff(void)
{
	GPIO_WriteLow(_X1_PORT, _X1_PIN);
```

```
        GPIO_WriteLow(_Y1_PORT, _Y1_PIN);
        GPIO_WriteLow(_X2_PORT, _X2_PIN);
        GPIO_WriteLow(_Y2_PORT, _Y2_PIN);
}

//
//      OFF all Drive TR
//
void StopDrive(void)                    //TROff()
{
        ExOff();
}


//
//
//
void StartDriveOR( uint16_t Target, uint8_t overrun )
{
        /*
        if ( (Drive.Target==0) && Drive.Run )
                return;
        if ( Drive.Position == Target )
                return;
        */

        if ( Drive.Run ) return;
        if ( Drive.Position == Target ) return;

        // Drive.Overrun =      0;
        if ( Target == 0 )                              //  goto origin
        {
                Drive.Overrun = 0u;
                Drive.ZeroOffset = CPulse( OVER_STEP );
                //Drive.Position += CPulse( OVER_STEP );        // POSITION;
                Drive.Position += Drive.ZeroOffset;     // POSITION;
                Drive.Target = 0u;
                Drive.State = nMotorClose;              // 0;
        }
        else
        {
                if ( Target > MAX_POSITION )
                        Target = MAX_POSITION;

                if (Drive.Position >= Target)           // Closing
                {
                        Drive.Overrun = 0u;
                        Drive.State = nMotorClose;
                        Drive.Overrun2 = 0u;
                        Drive.Target = Target;
                        Drive.State = nMotorClose;              // 0;
                }
                else                                                            // Opening
                {
                        Drive.State = nMotorOpen;       //1;

                        //if (My.Mode == CMD_TESTJIG )
                        //{
                        //      Drive.Overrun2 = 0u;    //(uint8_t)OVERRUN2;            //100;
                        //      Drive.Target = Target;  // + OVERRUN2;  //100;
                        //}
                        //else
                        {
                                Drive.Overrun2 = overrun;               //100;
                                Drive.Target = Target + overrun;        //100;
                        }
                }
        }
        Drive.Run = TRUE;
        Drive.Ticks = Drive.Interval;
}


//
//
//
void MotorStep(MOTOR_DIR Dir)
{
        if ( Dir == nMotorClose )
        {
                if ( Drive.Position != 0 )
                {
                        Drive.Position--;
                        if ( Drive.NormalOpen )
                                Drive.Phase++;
                        else
                                Drive.Phase--;
                }
                //else
                if ( Drive.Position == 0 )
                {
                        Drive.Enabled = FALSE;
                        Drive.Overrun = 0u;
                        Drive.Run = FALSE;
                        Drive.ZeroOffset = 0u;
                        Drive.Position = 0u;
                        Drive.Target = 0u;
                        StopDrive();
                }
        }
        else                                                    // Find hall ic position
        {
                if (  Drive.Position < MAX_POSITION )
                        Drive.Position++;

                if ( Drive.NormalOpen )
                                Drive.Phase--;
                        else
                                Drive.Phase++;

                //Drive.Phase++;
                if ( Drive.Origin )
                {
                        Drive.ORGPosition = Drive.Position;
                        Drive.LastOrigin = TRUE;
                }
                else
                        if ( Drive.LastOrigin )
                        {
                                Drive.MaxOverrun  = Drive.ORGPosition - HALL_THRESHOLD;
                                if ( Drive.MaxOverrun > OVERRUN )
                                        Drive.MaxOverrun = OVERRUN;

                        }
                        //50 : hall IC threshold
        }

        if ( !Drive.Run ) return;
        if ( !Drive.Enabled ) return;
        switch ( Drive.ExcitationType )
        {

                case 0:
                        Excitation_1Phase();
                        break;
                case 1:
                        Excitation_12Phase();
                        break;
                case 2:
                        Excitation_2Phase();
                        break;

        }


}

//
//
//
void MotorClose(void)
{
        if ( Drive.Origin )             // Hall IC Sensing
        {
                if ( Drive.Overrun >= Drive.MaxOverrun ) //     max overrun
                {
                        Drive.Target = 0u;
                        Drive.Position = 0u;
                        Drive.Overrun = 0u;
                        Drive.ZeroOffset = 0u;
                        Drive.Run = FALSE;
                        StopDrive();
                }
                else                                            // overrun 1 step close
                {
                        Drive.Overrun++;
                        MotorStep(nMotorClose);
                }
        }
        else
                MotorStep(nMotorClose);         // move 1 step close
}

//
//
//
void DriveService(void)
{
        //      20150925
        /*
        X1 =  0;        //EX[phase % 4].X;
    Y1 =  0;    //EX[phase % 4].Y;
    X1_ = 0;    //EX[phase % 4].X_;
    Y1_ = 0;    //EX[phase % 4].Y_;
*/
        ExOff();

        if ( ! Drive.Run ) return;      //when Drive is stop

//      if ( My.PacketLen == 4 && My.Mode == CMD_GOTO && My.TmrDrive_1mS == 0 )
//              return;
//      if ( My.PacketLen == 4 && My.Mode == CMD_GOTOA && My.TmrDrive_1mS == 0 )
//              return;

        if ( Drive.Target > Drive.Position )
                MotorStep(nMotorOpen);
        else if ( Drive.Target < Drive.Position )
                MotorClose( );
        else if ( Drive.Target )        // on target position
                {
                        if ( Drive.Overrun2 )
                        {
                                Drive.Target = Drive.Target - Drive.Overrun2;   //   OVERRUN2;
//100;
                                Drive.Overrun2 = 0u;
                        }
                        else
                        {
                                Drive.ZeroOffset = 0u;
                                Drive.Overrun = 0u;
                                Drive.Run = FALSE;
                                StopDrive();
                        }
                }
                else                                    // Origin Error
                {
                        //Drive.Enabled = FALSE;
                        Drive.ZeroOffset = 0u;
                        Drive.Overrun = 0u;
                        Drive.Run = FALSE;
                        StopDrive();
                }
}


void Excitation_2Phase( void )
{

        switch (Drive.Phase%8)
        {
                case 0:
                case 1:
                        GPIO_WriteHigh(_X1_PORT, _X1_PIN);
                        GPIO_WriteHigh(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteLow(_X2_PORT, _X2_PIN);
                        GPIO_WriteLow(_Y2_PORT, _Y2_PIN);
                        break;
                case 2:
                case 3:
                        GPIO_WriteLow(_X1_PORT, _X1_PIN);
                        GPIO_WriteHigh(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteHigh(_X2_PORT, _X2_PIN);
                        GPIO_WriteLow(_Y2_PORT, _Y2_PIN);
                        break;
                case 4:
                case 5:
                        GPIO_WriteLow(_X1_PORT, _X1_PIN);
                        GPIO_WriteLow(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteHigh(_X2_PORT, _X2_PIN);
                        GPIO_WriteHigh(_Y2_PORT, _Y2_PIN);
                        break;
                case 6:
                case 7:
                        GPIO_WriteHigh(_X1_PORT, _X1_PIN);
                        GPIO_WriteLow(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteLow(_X2_PORT, _X2_PIN);
                        GPIO_WriteHigh(_Y2_PORT, _Y2_PIN);
                        break;

                default:
                        break;
        }

}


void Excitation_12Phase( void )
{

        switch (Drive.Phase%8)
        {
                case 0:
                        GPIO_WriteHigh(_X1_PORT, _X1_PIN);
                        GPIO_WriteHigh(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteLow(_X2_PORT, _X2_PIN);
                        GPIO_WriteLow(_Y2_PORT, _Y2_PIN);
                        break;
                case 1:
                        GPIO_WriteLow(_X1_PORT, _X1_PIN);
                        GPIO_WriteHigh(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteLow(_X2_PORT, _X2_PIN);
                        GPIO_WriteLow(_Y2_PORT, _Y2_PIN);
                        break;
                case 2:
                        GPIO_WriteLow(_X1_PORT, _X1_PIN);
                        GPIO_WriteHigh(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteHigh(_X2_PORT, _X2_PIN);
                        GPIO_WriteLow(_Y2_PORT, _Y2_PIN);
                        break;
                case 3:
                        GPIO_WriteLow(_X1_PORT, _X1_PIN);
                        GPIO_WriteLow(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteHigh(_X2_PORT, _X2_PIN);
                        GPIO_WriteLow(_Y2_PORT, _Y2_PIN);
                        break;
                case 4:
                        GPIO_WriteLow(_X1_PORT, _X1_PIN);
                        GPIO_WriteLow(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteHigh(_X2_PORT, _X2_PIN);
                        GPIO_WriteHigh(_Y2_PORT, _Y2_PIN);
                        break;
                case 5:
                        GPIO_WriteLow(_X1_PORT, _X1_PIN);
                        GPIO_WriteLow(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteLow(_X2_PORT, _X2_PIN);
                        GPIO_WriteHigh(_Y2_PORT, _Y2_PIN);
                        break;
                case 6:
                        GPIO_WriteHigh(_X1_PORT, _X1_PIN);
                        GPIO_WriteLow(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteLow(_X2_PORT, _X2_PIN);
                        GPIO_WriteHigh(_Y2_PORT, _Y2_PIN);
                        break;
                case 7:
                        GPIO_WriteHigh(_X1_PORT, _X1_PIN);
                        GPIO_WriteLow(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteLow(_X2_PORT, _X2_PIN);
                        GPIO_WriteLow(_Y2_PORT, _Y2_PIN);
                        break;

                default:
                        break;
        }

}

void Excitation_1Phase( void )
{


        switch (Drive.Phase%8)
        {
                case 0:
                case 1:
                        GPIO_WriteHigh(_X1_PORT, _X1_PIN);
                        GPIO_WriteLow(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteLow(_X2_PORT, _X2_PIN);
                        GPIO_WriteLow(_Y2_PORT, _Y2_PIN);
                        break;
                case 2:
                case 3:
                        GPIO_WriteLow(_X1_PORT, _X1_PIN);
                        GPIO_WriteHigh(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteLow(_X2_PORT, _X2_PIN);
                        GPIO_WriteLow(_Y2_PORT, _Y2_PIN);
                        break;
                case 4:
                case 5:
                        GPIO_WriteLow(_X1_PORT, _X1_PIN);
                        GPIO_WriteLow(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteHigh(_X2_PORT, _X2_PIN);
                        GPIO_WriteLow(_Y2_PORT, _Y2_PIN);
                        break;
                case 6:
                case 7:
                        GPIO_WriteLow(_X1_PORT, _X1_PIN);
                        GPIO_WriteLow(_Y1_PORT, _Y1_PIN);
                        GPIO_WriteLow(_X2_PORT, _X2_PIN);
                        GPIO_WriteHigh(_Y2_PORT, _Y2_PIN);
                        break;

                default:
                        break;
        }

}


u8 Interval( u16 Vmon)
{
        Vmon = Vmon / IntervalSlope;

        if (Vmon < 10 ) Vmon = 10;
        if (Vmon > 26 ) Vmon = 26;

        return ( TableInterval[ Vmon-10 ] );

        /*
        if ( Vmon < 250 ) return 80u;
        if ( Vmon < 287 ) return 65u;
        if ( Vmon < 340 ) return 50u;
        if ( Vmon < 430 ) return 38u;
        if ( Vmon < 515 ) return 28u;
        if ( Vmon < 600 ) return 20u;
        if ( Vmon < 688 ) return 15u;
        return 15u;
        */
}


uint16_t CStep( uint16_t pulse )
{
        uint32_t i;

        i = (uint32_t) pulse * 200u / (2400UL);
        i = (i+1)/2;                            // Round up  20150925
        return (uint16_t) i;
}


uint16_t CPulse( uint16_t step )
{
        uint32_t i;

    i= (uint32_t) step *(24UL);
        //i = (i+1)/2;
        return (uint16_t) i;
```

```
}

/*






*/

void Clock_Config(void)
{
        //      Clock configuration
        CLK_DeInit();
        CLK_HSICmd(ENABLE);
        CLK_LSICmd(DISABLE);
        CLK_HSECmd(DISABLE);

        CLK_HSIPrescalerConfig(CLK_PRESCALER_HSIDIV1);
        CLK_SYSCLKConfig(CLK_PRESCALER_CPUDIV1);
}


void Gpio_Config(void)
{
        // Configure LED as output push-pull low (led switched on)
  GPIO_Init(LED_PORT, LED_PIN, GPIO_MODE_OUT_PP_LOW_FAST);
        GPIO_WriteHigh(LED_PORT, LED_PIN);

        //   Motor port
        GPIO_Init(_X1_PORT, _X1_PIN, GPIO_MODE_OUT_PP_LOW_FAST);
        GPIO_Init(_X2_PORT, _X2_PIN, GPIO_MODE_OUT_PP_LOW_FAST);
        GPIO_Init(_Y1_PORT, _Y1_PIN, GPIO_MODE_OUT_PP_LOW_FAST);
        GPIO_Init(_Y2_PORT, _Y2_PIN, GPIO_MODE_OUT_PP_LOW_FAST);

        GPIO_WriteLow(_X1_PORT, _X1_PIN);
        GPIO_WriteLow(_Y1_PORT, _Y1_PIN);
        GPIO_WriteLow(_X2_PORT, _X2_PIN);
        GPIO_WriteLow(_Y2_PORT, _Y2_PIN);

        //  Hall sensor port
  GPIO_Init(_ORG_PORT, _ORG_PIN, GPIO_MODE_IN_FL_NO_IT);

        // for ADC3
        GPIO_Init(GPIOD, GPIO_PIN_2, GPIO_MODE_IN_FL_NO_IT );

        //GPIO_Init(GPIOD, GPIO_PIN_2, GPIO_MODE_IN_PU_NO_IT );
        GPIO_Init(GPIOC, GPIO_PIN_4, GPIO_MODE_IN_FL_NO_IT );
        GPIO_Init(GPIOD, GPIO_PIN_3, GPIO_MODE_IN_FL_NO_IT );

}


//      Timer 4 Configuration
void Timer4_Config(void)
{
  TIM4_DeInit();

  /* Time base configuration */
  //TIM4_TimeBaseInit(TIM4_PRESCALER_64, 0xFA );                //oxfa =  250    16,000,000 /
64/250 = 1000ticks = 1mS
        //TIM4_TimeBaseInit(TIM4_PRESCALER_8, 0xFA );
        //TIM4_TimeBaseInit(TIM4_PRESCALER_16, 0xFA );
        TIM4_TimeBaseInit(TIM4_PRESCALER_16, 0x32 );  //  16 * 0x32 = 50uS  Ticks

  /* Enable TIM4 IT UPDATE */
  TIM4_ITConfig(TIM4_IT_UPDATE, ENABLE);
  /* Enable TIM4 */
  TIM4_Cmd(ENABLE);
}

//      ADC configuration
static void Adc_Config( void )
{

        Adc.Value[Adc.Idx] = ADC1_GetConversionValue();
        ADC1_DeInit();

        switch (Adc.Idx)
        {
                case 0:

                        ADC1_Init(ADC1_CONVERSIONMODE_SINGLE , ADC1_CHANNEL_3,
ADC1_PRESSEL_FCPU_D2,
                                                                ADC1_EXTTRIG_TIM,DISABLE,
ADC1_ALIGN_RIGHT, ADC1_SCHMITTTRIG_CHANNEL3, DISABLE );
                        Adc.Idx = 1;
                        break;
                case 1:
                        ADC1_Init(ADC1_CONVERSIONMODE_SINGLE , ADC1_CHANNEL_4,
ADC1_PRESSEL_FCPU_D2,
                                                                ADC1_EXTTRIG_TIM,DISABLE,
ADC1_ALIGN_RIGHT, ADC1_SCHMITTTRIG_CHANNEL4, DISABLE );
                        Adc.Idx = 2;
                        break;

                case 2:
                        ADC1_Init(ADC1_CONVERSIONMODE_SINGLE , ADC1_CHANNEL_2,
ADC1_PRESSEL_FCPU_D2,
                                                                ADC1_EXTTRIG_TIM,DISABLE,
ADC1_ALIGN_RIGHT, ADC1_SCHMITTTRIG_CHANNEL2, DISABLE );
                        Adc.Idx = 0;
                        break;

                default:
                        Adc.Idx = 0;
                        break;
        }

                ADC1_StartConversion();
}



#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(u8* file, u32 line)
{
  /* 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) */

  /* Infinite loop */
  while (1)
  {}
}
#endif
```

```
/**
  * @}
  */

/******************* (C) COPYRIGHT 2009 STMicroelectronics *****END OF FILE****/
```