static *void* MX\_SPI2\_Init(*void*)

{

    /\* USER CODE BEGIN SPI2\_Init 0 \*/

    /\* USER CODE END SPI2\_Init 0 \*/

    /\* USER CODE BEGIN SPI2\_Init 1 \*/

    /\* USER CODE END SPI2\_Init 1 \*/

    /\* SPI2 parameter configuration\*/

    hspi2.Instance = SPI2;

    hspi2.Init.Mode = SPI\_MODE\_MASTER;

    hspi2.Init.Direction = SPI\_DIRECTION\_1LINE;

    hspi2.Init.DataSize = SPI\_DATASIZE\_4BIT;

    hspi2.Init.CLKPolarity = SPI\_POLARITY\_LOW;

    hspi2.Init.CLKPhase = SPI\_PHASE\_1EDGE;

    hspi2.Init.NSS = SPI\_NSS\_SOFT;

    hspi2.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_2;

    hspi2.Init.FirstBit = SPI\_FIRSTBIT\_MSB;

    hspi2.Init.TIMode = SPI\_TIMODE\_DISABLE;

    hspi2.Init.CRCCalculation = SPI\_CRCCALCULATION\_DISABLE;

    hspi2.Init.CRCPolynomial = 7;

    hspi2.Init.CRCLength = SPI\_CRC\_LENGTH\_DATASIZE;

    hspi2.Init.NSSPMode = SPI\_NSS\_PULSE\_ENABLE;

    if (HAL\_SPI\_Init(&hspi2) != HAL\_OK)

    {

        Error\_Handler();

    }

    /\* USER CODE BEGIN SPI2\_Init 2 \*/

    /\* USER CODE END SPI2\_Init 2 \*/

}

*extern DMA\_HandleTypeDef hdma\_spi1\_tx;*

*extern DMA\_HandleTypeDef hdma\_spi2\_tx;*

*void* HAL\_SPI\_MspInit(SPI\_HandleTypeDef\* *hspi*)

{

  GPIO\_InitTypeDef GPIO\_InitStruct = {0};

  if(hspi->Instance==SPI2)

  {

  /\* USER CODE BEGIN SPI2\_MspInit 0 \*/

  /\* USER CODE END SPI2\_MspInit 0 \*/

    /\* Peripheral clock enable \*/

    \_\_HAL\_RCC\_SPI2\_CLK\_ENABLE();

    \_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

    /\*\*SPI2 GPIO Configuration

    PB15     ------> SPI2\_MOSI

    PB13     ------> SPI2\_SCK

    \*/

    GPIO\_InitStruct.Pin = GPIO\_PIN\_15|GPIO\_PIN\_13;

    GPIO\_InitStruct.Mode = GPIO\_MODE\_AF\_PP;

    GPIO\_InitStruct.Pull = GPIO\_NOPULL;

    GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_VERY\_HIGH;

    GPIO\_InitStruct.Alternate = GPIO\_AF5\_SPI2;

    HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStruct);

/\* Peripheral DMA init\*/

hdma\_spi1\_tx.Instance = DMA2\_Stream2;

hdma\_spi1\_tx.Init.Channel = DMA\_CHANNEL\_2;

hdma\_spi1\_tx.Init.Direction = DMA\_MEMORY\_TO\_PERIPH;

hdma\_spi1\_tx.Init.PeriphInc = DMA\_PINC\_DISABLE;

hdma\_spi1\_tx.Init.MemInc = DMA\_MINC\_ENABLE;

hdma\_spi1\_tx.Init.PeriphDataAlignment = DMA\_PDATAALIGN\_BYTE;

hdma\_spi1\_tx.Init.MemDataAlignment = DMA\_MDATAALIGN\_BYTE;

hdma\_spi1\_tx.Init.Mode = DMA\_NORMAL;

hdma\_spi1\_tx.Init.Priority = DMA\_PRIORITY\_LOW;

hdma\_spi1\_tx.Init.FIFOMode = DMA\_FIFOMODE\_DISABLE;

if (HAL\_DMA\_Init(&hdma\_spi1\_tx) != HAL\_OK)

{

Error\_Handler();

}

\_\_HAL\_LINKDMA(hspi,hdmatx,hdma\_spi1\_tx);

  /\* USER CODE BEGIN SPI2\_MspInit 1 \*/

  /\* USER CODE END SPI2\_MspInit 1 \*/

  }

}

*void* MX\_UART5\_Init(*void*)

{

    /\* USER CODE BEGIN UART5\_Init 0 \*/

    /\* USER CODE END UART5\_Init 0 \*/

    /\* USER CODE BEGIN UART5\_Init 1 \*/

    /\* USER CODE END UART5\_Init 1 \*/

    huart5.Instance = UART5;

    huart5.Init.BaudRate = 115200;

    huart5.Init.WordLength = UART\_WORDLENGTH\_8B;

    huart5.Init.StopBits = UART\_STOPBITS\_1;

    huart5.Init.Parity = UART\_PARITY\_NONE;

    huart5.Init.Mode = UART\_MODE\_TX\_RX;

    huart5.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

    huart5.Init.OverSampling = UART\_OVERSAMPLING\_16;

    huart5.Init.OneBitSampling = UART\_ONE\_BIT\_SAMPLE\_DISABLE;

    huart5.AdvancedInit.AdvFeatureInit = UART\_ADVFEATURE\_NO\_INIT;

    if (HAL\_UART\_Init(&huart5) != HAL\_OK)

    {

        Error\_Handler();

    }

    /\* USER CODE BEGIN UART5\_Init 2 \*/

    /\* USER CODE END UART5\_Init 2 \*/

}

static *void* MX\_I2C1\_Init(*void*)

{

  /\* USER CODE BEGIN I2C1\_Init 0 \*/

  /\* USER CODE END I2C1\_Init 0 \*/

  /\* USER CODE BEGIN I2C1\_Init 1 \*/

  /\* USER CODE END I2C1\_Init 1 \*/

  hi2c1.Instance = I2C1;

  hi2c1.Init.ClockSpeed = 100000;

  hi2c1.Init.DutyCycle = I2C\_DUTYCYCLE\_2;

  hi2c1.Init.OwnAddress1 = 0;

  hi2c1.Init.AddressingMode = I2C\_ADDRESSINGMODE\_7BIT;

  hi2c1.Init.DualAddressMode = I2C\_DUALADDRESS\_DISABLE;

  hi2c1.Init.OwnAddress2 = 0;

  hi2c1.Init.GeneralCallMode = I2C\_GENERALCALL\_DISABLE;

  hi2c1.Init.NoStretchMode = I2C\_NOSTRETCH\_DISABLE;

  if (HAL\_I2C\_Init(&hi2c1) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\*\* Configure Analogue filter

  \*/

  if (HAL\_I2CEx\_ConfigAnalogFilter(&hi2c1, I2C\_ANALOGFILTER\_ENABLE) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\*\* Configure Digital filter

  \*/

  if (HAL\_I2CEx\_ConfigDigitalFilter(&hi2c1, 0) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\* USER CODE BEGIN I2C1\_Init 2 \*/

  /\* USER CODE END I2C1\_Init 2 \*/

}

*void* MX\_USART3\_UART\_Init(*void*)

{

  /\* USER CODE BEGIN USART3\_Init 0 \*/

  /\* USER CODE END USART3\_Init 0 \*/

  /\* USER CODE BEGIN USART3\_Init 1 \*/

  /\* USER CODE END USART3\_Init 1 \*/

  huart3.Instance = USART3;

  huart3.Init.BaudRate = 115200;

  huart3.Init.WordLength = UART\_WORDLENGTH\_8B;

  huart3.Init.StopBits = UART\_STOPBITS\_1;

  huart3.Init.Parity = UART\_PARITY\_NONE;

  huart3.Init.Mode = UART\_MODE\_TX\_RX;

  huart3.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

  huart3.Init.OverSampling = UART\_OVERSAMPLING\_16;

  if (HAL\_UART\_Init(&huart3) != HAL\_OK)

  {

    Error\_Handler();

  }

  /\* USER CODE BEGIN USART3\_Init 2 \*/

  /\* USER CODE END USART3\_Init 2 \*/

}

*void* SystemClock\_Config(*void*)

{

  RCC\_OscInitTypeDef RCC\_OscInitStruct;

  RCC\_ClkInitTypeDef RCC\_ClkInitStruct;

  \_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

  \_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

  RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_LSI|RCC\_OSCILLATORTYPE\_HSE;

  RCC\_OscInitStruct.HSEState = RCC\_HSE\_ON;

  RCC\_OscInitStruct.LSIState = RCC\_LSI\_ON;

  RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_NONE;

  if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

  {

    Error\_Handler();

  }

  RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

                              |RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

  RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_HSE;

  RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

  RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;

  RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

  if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_0) != HAL\_OK)

  {

    Error\_Handler();

  }

  HAL\_SYSTICK\_Config(HAL\_RCC\_GetHCLKFreq()/1000);

  HAL\_SYSTICK\_CLKSourceConfig(SYSTICK\_CLKSOURCE\_HCLK);

  /\* SysTick\_IRQn interrupt configuration \*/

  HAL\_NVIC\_SetPriority(SysTick\_IRQn, 0, 0);

}

static *void* MX\_ADC1\_Init(*void*)

{

  ADC\_ChannelConfTypeDef sConfig;

    /\*\*Configure the global features of the ADC (Clock, Resolution, Data Alignment and number of conversion)

    \*/

  hadc1.Instance = ADC1;

  hadc1.Init.ClockPrescaler = ADC\_CLOCK\_SYNC\_PCLK\_DIV2;

  hadc1.Init.Resolution = ADC\_RESOLUTION\_12B;

  hadc1.Init.ScanConvMode = DISABLE;

  hadc1.Init.ContinuousConvMode = DISABLE;

  hadc1.Init.DiscontinuousConvMode = DISABLE;

  hadc1.Init.ExternalTrigConvEdge = ADC\_EXTERNALTRIGCONVEDGE\_NONE;

  hadc1.Init.DataAlign = ADC\_DATAALIGN\_RIGHT;

  hadc1.Init.NbrOfConversion = 1;

  hadc1.Init.DMAContinuousRequests = DISABLE;

  hadc1.Init.EOCSelection = ADC\_EOC\_SINGLE\_CONV;

  if (HAL\_ADC\_Init(&hadc1) != HAL\_OK)

  {

    Error\_Handler();

  }

    /\*\*Configure for the selected ADC regular channel its corresponding rank in the sequencer and its sample time.

    \*/

  sConfig.Channel = ADC\_CHANNEL\_9;

  sConfig.Rank = 1;

  sConfig.SamplingTime = ADC\_SAMPLETIME\_3CYCLES;

  if (HAL\_ADC\_ConfigChannel(&hadc1, &sConfig) != HAL\_OK)

  {

    Error\_Handler();

  }

}

static *void* MX\_IWDG\_Init(*void*)

{

  hiwdg.Instance = IWDG;

  hiwdg.Init.Prescaler = IWDG\_PRESCALER\_4;

  hiwdg.Init.Reload = 4095;

  if (HAL\_IWDG\_Init(&hiwdg) != HAL\_OK)

  {

    Error\_Handler();

  }

}

static *void* MX\_TIM2\_Init(*void*)

{

  TIM\_ClockConfigTypeDef sClockSourceConfig;

  TIM\_MasterConfigTypeDef sMasterConfig;

  htim2.Instance = TIM2;

  htim2.Init.Prescaler = 2599;                  // 26MHz / 2600 = 10KHz     0.1ms

  htim2.Init.CounterMode = TIM\_COUNTERMODE\_UP;

  htim2.Init.Period = 9;                        // 0.1ms \* 10 = 1ms

  htim2.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

  if (HAL\_TIM\_Base\_Init(&htim2) != HAL\_OK)

  {

    Error\_Handler();

  }

  sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

  if (HAL\_TIM\_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL\_OK)

  {

    Error\_Handler();

  }

  sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

  sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

  if (HAL\_TIMEx\_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL\_OK)

  {

    Error\_Handler();

  }

}

static *void* MX\_DMA\_Init(*void*)

{

  /\* DMA controller clock enable \*/

  \_\_HAL\_RCC\_DMA1\_CLK\_ENABLE();

  \_\_HAL\_RCC\_DMA2\_CLK\_ENABLE();

  /\* DMA interrupt init \*/

  /\* DMA1\_Stream4\_IRQn interrupt configuration \*/

  HAL\_NVIC\_SetPriority(DMA1\_Stream4\_IRQn, 0, 0);

  HAL\_NVIC\_EnableIRQ(DMA1\_Stream4\_IRQn);

  /\* DMA2\_Stream2\_IRQn interrupt configuration \*/

  HAL\_NVIC\_SetPriority(DMA2\_Stream2\_IRQn, 0, 0);

  HAL\_NVIC\_EnableIRQ(DMA2\_Stream2\_IRQn);

}

*void* HAL\_MspInit(*void*)

{

  /\* USER CODE BEGIN MspInit 0 \*/

  /\* USER CODE END MspInit 0 \*/

  HAL\_NVIC\_SetPriorityGrouping(NVIC\_PRIORITYGROUP\_2);

  /\* System interrupt init\*/

  /\* MemoryManagement\_IRQn interrupt configuration \*/

  HAL\_NVIC\_SetPriority(MemoryManagement\_IRQn, 0, 0);

  /\* BusFault\_IRQn interrupt configuration \*/

  HAL\_NVIC\_SetPriority(BusFault\_IRQn, 0, 0);

  /\* UsageFault\_IRQn interrupt configuration \*/

  HAL\_NVIC\_SetPriority(UsageFault\_IRQn, 0, 0);

  /\* SVCall\_IRQn interrupt configuration \*/

  HAL\_NVIC\_SetPriority(SVCall\_IRQn, 0, 0);

  /\* DebugMonitor\_IRQn interrupt configuration \*/

  HAL\_NVIC\_SetPriority(DebugMonitor\_IRQn, 0, 0);

  /\* PendSV\_IRQn interrupt configuration \*/

  HAL\_NVIC\_SetPriority(PendSV\_IRQn, 0, 0);

  /\* SysTick\_IRQn interrupt configuration \*/

  HAL\_NVIC\_SetPriority(SysTick\_IRQn, 0, 0);

  /\* USER CODE BEGIN MspInit 1 \*/

  /\* USER CODE END MspInit 1 \*/

}

*void* HAL\_ADC\_MspInit(ADC\_HandleTypeDef\* *hadc*)

{

  GPIO\_InitTypeDef GPIO\_InitStruct;

  if(hadc->Instance==ADC1)

  {

  /\* USER CODE BEGIN ADC1\_MspInit 0 \*/

  /\* USER CODE END ADC1\_MspInit 0 \*/

    /\* Peripheral clock enable \*/

    \_\_HAL\_RCC\_ADC1\_CLK\_ENABLE();

    /\*\*ADC1 GPIO Configuration

    PB1     ------> ADC1\_IN9

    \*/

    GPIO\_InitStruct.Pin = GPIO\_PIN\_1;

    GPIO\_InitStruct.Mode = GPIO\_MODE\_ANALOG;

    GPIO\_InitStruct.Pull = GPIO\_NOPULL;

    HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStruct);

  /\* USER CODE BEGIN ADC1\_MspInit 1 \*/

  /\* USER CODE END ADC1\_MspInit 1 \*/

  }

}

*void* HAL\_I2C\_MspInit(I2C\_HandleTypeDef\* *hi2c*)

{

  GPIO\_InitTypeDef GPIO\_InitStruct;

  if(hi2c->Instance==I2C1)

  {

  /\* USER CODE BEGIN I2C1\_MspInit 0 \*/

  /\* USER CODE END I2C1\_MspInit 0 \*/

    /\*\*I2C1 GPIO Configuration

    PB8     ------> I2C1\_SCL

    PB9     ------> I2C1\_SDA

    \*/

    GPIO\_InitStruct.Pin = GPIO\_PIN\_8|GPIO\_PIN\_9;

    GPIO\_InitStruct.Mode = GPIO\_MODE\_AF\_OD;

    GPIO\_InitStruct.Pull = GPIO\_PULLUP;

    GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_VERY\_HIGH;

    GPIO\_InitStruct.Alternate = GPIO\_AF4\_I2C1;

    HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStruct);

    /\* Peripheral clock enable \*/

    \_\_HAL\_RCC\_I2C1\_CLK\_ENABLE();

  /\* USER CODE BEGIN I2C1\_MspInit 1 \*/

  /\* USER CODE END I2C1\_MspInit 1 \*/

  }

}

*void* HAL\_TIM\_Base\_MspInit(TIM\_HandleTypeDef\* *htim\_base*)

{

  if(htim\_base->Instance==TIM2)

  {

  /\* USER CODE BEGIN TIM2\_MspInit 0 \*/

  /\* USER CODE END TIM2\_MspInit 0 \*/

    /\* Peripheral clock enable \*/

    \_\_HAL\_RCC\_TIM2\_CLK\_ENABLE();

    /\* Peripheral interrupt init \*/

    HAL\_NVIC\_SetPriority(TIM2\_IRQn, 2, 2);

    HAL\_NVIC\_EnableIRQ(TIM2\_IRQn);

  /\* USER CODE BEGIN TIM2\_MspInit 1 \*/

  /\* USER CODE END TIM2\_MspInit 1 \*/

  }

}

/\*\*

\* *@brief* This function handles System tick timer.

\*/

*void* SysTick\_Handler(*void*)

{

  /\* USER CODE BEGIN SysTick\_IRQn 0 \*/

  /\* USER CODE END SysTick\_IRQn 0 \*/

  HAL\_IncTick();

  HAL\_SYSTICK\_IRQHandler();

  /\* USER CODE BEGIN SysTick\_IRQn 1 \*/

  /\* USER CODE END SysTick\_IRQn 1 \*/

}

/\*\*

\* *@brief* This function handles DMA1 stream4 global interrupt.

\*/

*void* DMA1\_Stream4\_IRQHandler(*void*)

{

  /\* USER CODE BEGIN DMA1\_Stream4\_IRQn 0 \*/

  /\* USER CODE END DMA1\_Stream4\_IRQn 0 \*/

  HAL\_DMA\_IRQHandler(&hdma\_spi2\_tx);

  /\* USER CODE BEGIN DMA1\_Stream4\_IRQn 1 \*/

  /\* USER CODE END DMA1\_Stream4\_IRQn 1 \*/

}

/\*\*

\* *@brief* This function handles TIM2 global interrupt.

\*/

*void* TIM2\_IRQHandler(*void*)

{

  /\* USER CODE BEGIN TIM2\_IRQn 0 \*/

  /\* USER CODE END TIM2\_IRQn 0 \*/

  HAL\_TIM\_IRQHandler(&htim2);

  /\* USER CODE BEGIN TIM2\_IRQn 1 \*/

  /\* USER CODE END TIM2\_IRQn 1 \*/

}

/\*\*

\* *@brief* This function handles EXTI line[15:10] interrupts.

\*/

*void* EXTI15\_10\_IRQHandler(*void*)

{

  /\* USER CODE BEGIN EXTI15\_10\_IRQn 0 \*/

  /\* USER CODE END EXTI15\_10\_IRQn 0 \*/

  HAL\_GPIO\_EXTI\_IRQHandler(GPIO\_PIN\_12);

  /\* USER CODE BEGIN EXTI15\_10\_IRQn 1 \*/

  /\* USER CODE END EXTI15\_10\_IRQn 1 \*/

}

/\*\*

\* *@brief* This function handles DMA2 stream2 global interrupt.

\*/

*void* DMA2\_Stream2\_IRQHandler(*void*)

{

  /\* USER CODE BEGIN DMA2\_Stream2\_IRQn 0 \*/

  /\* USER CODE END DMA2\_Stream2\_IRQn 0 \*/

  HAL\_DMA\_IRQHandler(&hdma\_spi1\_tx);

  /\* USER CODE BEGIN DMA2\_Stream2\_IRQn 1 \*/

  /\* USER CODE END DMA2\_Stream2\_IRQn 1 \*/

}

 \* *@fn*        Key\_Scan()

 \*

 \* *@brief*     按键扫描驱动

 \*

 \* *@param*     *None*

 \*

 \* *@return*    Key\_None：按键无效   Key\_Short：按键短按   Key\_Long：按键长按

 \*/

KeyStatus Key\_Scan(*void*)

{

    if(Key\_Up == 1)        // 有按键松起

    {

        Key\_Up = 0;

        if(Time\_1ms > Long\_Thre)

        {

            //Time\_1ms = 0;          // 启动定时器前已清零

            return Key\_Long;         // 按键按下时间大于2s      长按

        }

        else if(Time\_1ms > Short\_Thre)

        {

            //Time\_1ms = 0;

            return Key\_Short;        // 按键按下时间小于2s      短按

        }

        else

        {

            //Time\_1ms = 0;

            return Key\_None;         // 视作按键抖动

        }

    }

    return Key\_None;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* *@fn*        HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)()

 \*

 \* *@brief*     TIM2 detection callbacks.

 \*

 \* *@param*     *htim*：所用定时器结构体

 \*

 \* *@return*    None

 \*/

*void* HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \**htim*)

{

  /\* Prevent unused argument(s) compilation warning \*/

  UNUSED(htim);

  /\* NOTE : This function Should not be modified, when the callback is needed,

            the \_\_HAL\_TIM\_PeriodElapsedCallback could be implemented in the user file

   \*/

    Time\_1ms++;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

 \* *@fn*        HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)()

 \*

 \* *@brief*     EXTI line detection callbacks.

 \*

 \* *@param*     *GPIO\_Pin*: Specifies the pins connected EXTI line

 \*

 \* *@return*    None

 \*/

*void* HAL\_GPIO\_EXTI\_Callback(*uint16\_t* *GPIO\_Pin*)

{

  /\* Prevent unused argument(s) compilation warning \*/

  UNUSED(GPIO\_Pin);

  /\* NOTE: This function Should not be modified, when the callback is needed,

           the HAL\_GPIO\_EXTI\_Callback could be implemented in the user file

   \*/

    if( HAL\_GPIO\_ReadPin(GPIOA, GPIO\_PIN\_12) )     // 上升沿

    {

            HAL\_TIM\_Base\_Stop\_IT(&htim2);    // 按键松起，关闭定时器

            Key\_Up = 1;                      // 按键松起标志

    }

    else                                           // 下降沿

    {

        if(Key\_Up == 0)       // 上一次按键响应完成

        {

            Time\_1ms = 0;

            HAL\_TIM\_Base\_Start\_IT(&htim2);   // 按键按下，开启定时器

        }

    }

}