Preface

The series of PY32F031 microcontrollers features a high-performance 32-bit ARM® Cortex®-M0+ core, a wide voltage range MCU. They are embedded with up to 64 Kbytes of flash memory and 8 Kbytes of SRAM memory, operating at a maximum frequency of 72 MHz.

This application note will help users understand the attentions for the application of various modules in PY32F031 and quickly start development.

Table 1 Applicable Products

Type	Product Series
Microcontroller Series	PY32F031

Contents

1. SPI DMA to communicate with FLASH 3
2. PWR Configuration 4
3. I2C Configuration 4
4. LCD Configuration 4
5. GPIO Configuration 5
6. ADC Configuration 5
7. RCC Configuration 7
8. SPI Transmission and Reception 7
9. IAP Upgrade 7
10. FLASH Configuration 8
11. OPTION Configuration 8
12. LPTIM Configuration 10
13. DMA Configuration 10
14. Version History 11
Appendix 1 12
1 PY32F031 reads the Vreferint 1.2V actual value stored in the information area (see 6.3 for specific address). 12
Appendix 2 13
2. When using PLL48M as the system clock, IAP jump should disable the PLL 13

1. SPI DMA to communicate with FLASH

1.1 Attentions

- When using PY32's SPI module under the Divided-by-Two clock, in order to ensure the reliability fo the tansmission, it's necessary to consider the relationship between the SPI clock division and Datasize. Since the DMA operates under the condition of a SPI clock divided by two, using 8BIT Datasize to transmit data won't be timely, it is recommended to use a 16BIT Datasize,Table 1-1 gives the recommendations for SPI clock and Datasize configuration in different modes.

Table 1-1 Frequency Limits in Different Modes of SPI

Mode	Descriptions
Master	SCK frequency max. at PCLK/2 in master mode
	Datasize is set to 16BIT

1.2 Operation Process

- Define the SPI_HandleTypeDef type variable SpiHandle and initialise each SpiHandle member variable;
- For the BaudRatePrescaler member variable in SpiHandle, set it to SPI_BAUDRATEPRESCALER_2 (2 divisions);
- For the DataSize member variable in SpiHandle, set to SPI_DATASIZE_16BIT (16BIT);
- Initialize the SPI module;
- Communicate using SPI.

1.3 Code Example

```
  SPI_HandleTypeDef SpiHandle;
  …
  /* De-Initialize the SPI peripheral */
  Spi1Handle.Instance                = SPI1;                                  /* SPI1 */
  Spi1Handle.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2; /* prescaler :2 */
  Spi1Handle.Init.Direction          = SPI_DIRECTION_2LINES;          /* full duplex */
  Spi1Handle.Init.CLKPolarity        = SPI_POLARITY_LOW;     /* SPI Clock Polarity: low */
  Spi1Handle.Init.CLKPhase           = SPI_PHASE_1EDGE ;      /* Data sampling starts at the
first clock edge */
  Spi1Handle.Init.DataSize           = SPI_DATASIZE_16BIT;   /* SPI Data Size is 16 bit */
  Spi1Handle.Init.FirstBit           = SPI_FIRSTBIT_MSB;      /* SPI MSB Transmission */
  Spi1Handle.Init.NSS                = SPI_NSS_SOFT;          /* NSS Software Mode    */
  Spi1Handle.Init.Mode = SPI_MODE_MASTER;                      /* Configure as host */
  Spi1Handle.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;      /* The CRC check is
disabled */

    /* Initialize SPI peripheral */
    if (HAL_SPI_Init(&Spi1Handle) != HAL_OK)
    {
      APP_ErrorHandler();
    }
```

2. PWR Configuration

- To ensure system stability, it is essential to enable the watchdog function.
- It is recommended that customers enable the watchdog in the Option and set the watchdog overflow time according to actual conditions through software.
- The systick interrupt (HAL_SuspendTick()) needs to be turned off before the MCU enters Stop.
- If the CPU clock is split frequency and the EXTI module clock and CPU clock come from the same clock source but are split frequency, waking up the CPU with an event in Sleep mode will fail, and it is necessary to use an interrupt to wake up the CPU.
- When entering STOP mode, if the clock frequency of the HSE is lower than the system clock, which can cause the HSE clock to not turn off properly.
- The VCC fall rate must be slower than 30 us/V or the programme will be abnormal.

3. I2C Configuration

- When using the FM+ mode of the IIC, an extenal 1k ohm register is needed if the communication rate is going to be 1 MHz.

4. LCD Configuration

- When writing data to the same LCD_RAMx register, it is necessary to write the data within 2 lcd clk cycles, and then wait for 2 pclk + 1 lcd clk cycles before continuing to write the data. (refer to the following)

```
    #define   Delay   40*2
    LCD_HandleTypeDef LcdHandle;
    __IO uint32_t      RatioNops = 0;

int main()
{
    RatioNops = Delay * (SystemCoreClock / 1000000U) / 4;

    HAL_LCD_Write(&LcdHandle，  LCD_RAM_REGISTER0，  0x0f0f0f0f);

    APP_DelayNops(RatioNops);/*延迟 2 个 pclk+1 个 lcd clk 周期，约为 80us

    HAL_LCD_Write(&LcdHandle，  LCD_RAM_REGISTER0，  0xf0f0f0f0);
}

static void APP_DelayNops(uint32_t Nops)
{
    for(uint32_t i=0; i<Nops;i++)
    {
        __NOP();
    }
}
```

- After writing to RAM, you must wait for 2 LCD clock cycles before entering STOP mode. (refer to the following)

```
#define   Delay   40*2
LCD_HandleTypeDef LcdHandle;

int main()
{
RatioNops = Delay * (SystemCoreClock / 1000000U) / 4;

HAL_LCD_Write(&LcdHandle，  LCD_RAM_REGISTER0，  0x0f0f0f0f);

   APP_DelayNops(RatioNops);/* Delay 2 lcd clk cycles, about 80us
}

while(1)
{
/* Suspend SysTick interrupt */
HAL_SuspendTick();
/* Enter Stop Mode and Wakeup by WFI */
HAL_PWR_EnterSTOPMode(PWR_LOWPOWERREGULATOR_ON，
PWR_STOPENTRY_WFI);
/* Resume Systick */
HAL_ResumeTick();

}
static void APP_DelayNops(uint32_t Nops)
{
   for(uint32_t i=0; i<Nops;i++)
   {
      __NOP();
   }
}
```

5. GPIO Configuration

- All GPIOs must not have a negative voltage exceeding -0.3V; for example, in the intercom market, it is recommended to use an IO port without AD functionality for channel switching features.
- The input voltage of all GPIO pins must not exceed VCC+0.3V.
- Structures such as GPIO initialization must be assigned a value of 0 to avoid undefined initial values.

6. ADC Configuration

6.1 ADC Software Configuration

- Add ADC_FORCE_RESET before ADC initialization to ensure successful initialization.
- ADC channels must be configured before enabling; configuring after enabling will result in failure.

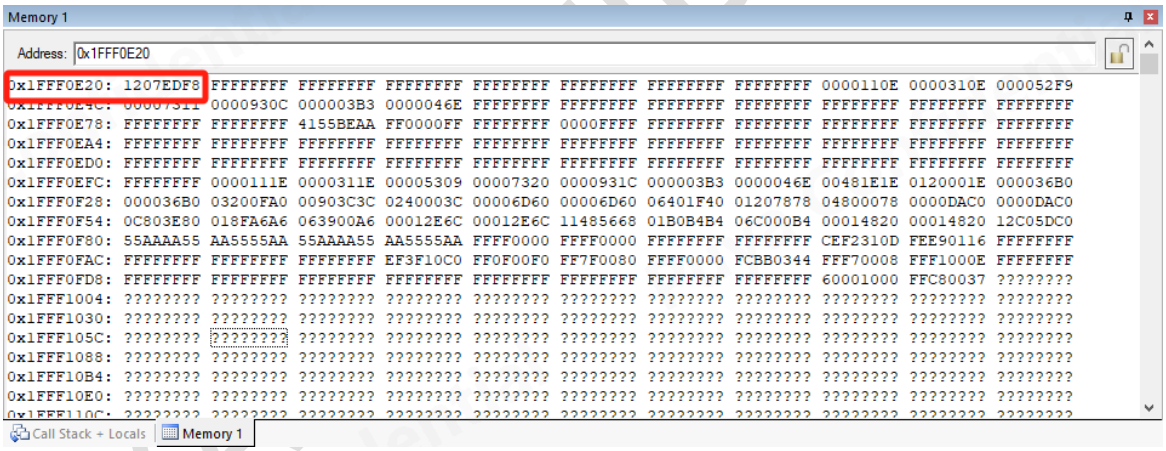
- The ADC clock must be configured to below 16MHz to ensure ADC sampling accuracy.
- An 8 ADC clock cycle delay must be added after ADC enabling before enabling conversion; otherwise, it will affect sampling accuracy.
- Directly driving high-power devices with GPIO can affect ADC sampling results (e.g., LED display. It is recommended not to sample ADC during LED display, or to insert a 10-100 ohm resistor in series on each GPIO of the LED, which can be adjusted according to actual conditions).

6.2 ADC Hardware Configuration

- The voltage of the ADC channel must not exceed VCC+0.3V (even if the ADC channel is not configured as an AD function), otherwise the ADC sampling will be inaccurate.

6.3 Vreferint 1.2V

- The actual value of the Vreferint 1.2V which the chip uses is plasced in the information area (0x1FFF0E20) in FLASH. (The high 16 bits are the actual value and the low 16 bits are the inverse code.) The procedure for reading Vreferint 1.2V is shown in Appendix 1;



- When sampling the internal reference voltage of 1.2V, the result calculated through the ADC sampling time conversion formula should be at least 20 microseconds. The methods are as follows:

a) Reduce the resolution.

b) Decrease the ADC clock frequency.

c) Increase the ADC sampling period.

The total conversion time is calculated as follows:

tCONV = Sampling Time 采样时间 + (Conversion Resolution+0.5) x ADC Clock Cycle

For example:

When ADC_CLK = 12MHz, the resolution is 12 bits, and the sampling time is 239.5

ADC clock cycles:

tCONV = (239.5 + 12.5) x ADC Clock Cycle=252 x ADC Clock Cycle= 21 us

7. RCC Configuration

- With the PLL enabled, FLASH_LATENCY needs to be set to 1.
- The PLL needs to be turned off before entering sleep mode, and the clock should be switched to HSI.
- At 48MHz, the PLL should be turned off during IAP jump (refer to Appendix 2 for examples).
- The HCLK crossover HPRE register bits cannot be toggled from 8 to 1-7 or the clock will hang.
- When the APB crossover frequency coefficient is large, after executing the module reset (on the APB bus), the module registers cannot be read or written immediately, if the operation is required, it is necessary to increase the __NOP() null instruction, and the number of null instructions should be larger than the number of APB crossover frequencies, such as APB 8 crossover frequencies, it is recommended to increase the number of __NOP instructions by more than 10.

8. SPI Transmission and Reception

- When SPI acts as a master and receives a series of data, it will have an extra byte, and the software needs to discard the first byte.
- When using SPI as a master for transmission, it is not recommended to use hardware chip select; instead, software chip select is recommended. To release the hardware chip select, SPI needs to be disabled.
- When SPI acts as a master and sends data by directly writing to the DR register, it is necessary to add four NOP() instructions after writing to DR to ensure successful transmission.
- It is recommended that customers use libraries directly for SPI operations.
- When SPI uses DMA mode, SPI should be started first, and then DMA should be enabled.

9. IAP Upgrade

- In PY32F031, when IAP jumps to the APP, interrupt remapping is required. The APP code and interrupt vectors are located at the address (0x80000000 + VECT_TAB_OFFSET), which is 0x80001000. Therefore, VECT_TAB_OFFSET is 0x1000, so it is necessary to define VECT_TAB_OFFSET as 0x1000 (#define VECT_TAB_OFFSET 0x1000).

- The BOOT program area needs to have write protection to prevent the BOOT area from being erased. (Write protection needs to be configured in the programmer).
- If there are write operations to FLASH in the program, write protection should be added to the program area to avoid accidental operations in the program area.

10. FLASH Configuration

- FLASH only supports Page erase and Page write, a Page is 256 bytes, the start address can only be Page aligned; (e.g. start address 0x08005000, 0x08005100, etc.)
- Every time Page write must be preceded by Page erase.

11. OPTION Configuration

- When mass production, Option operation needs to be configured in the Option Byte configuration of the porgammer, and the function that operates Option in the program should be blocked.
- It is recommended that the customer program enables write protection, which is set in Option, as shown in Figure 11-1 and Figure 11-2.

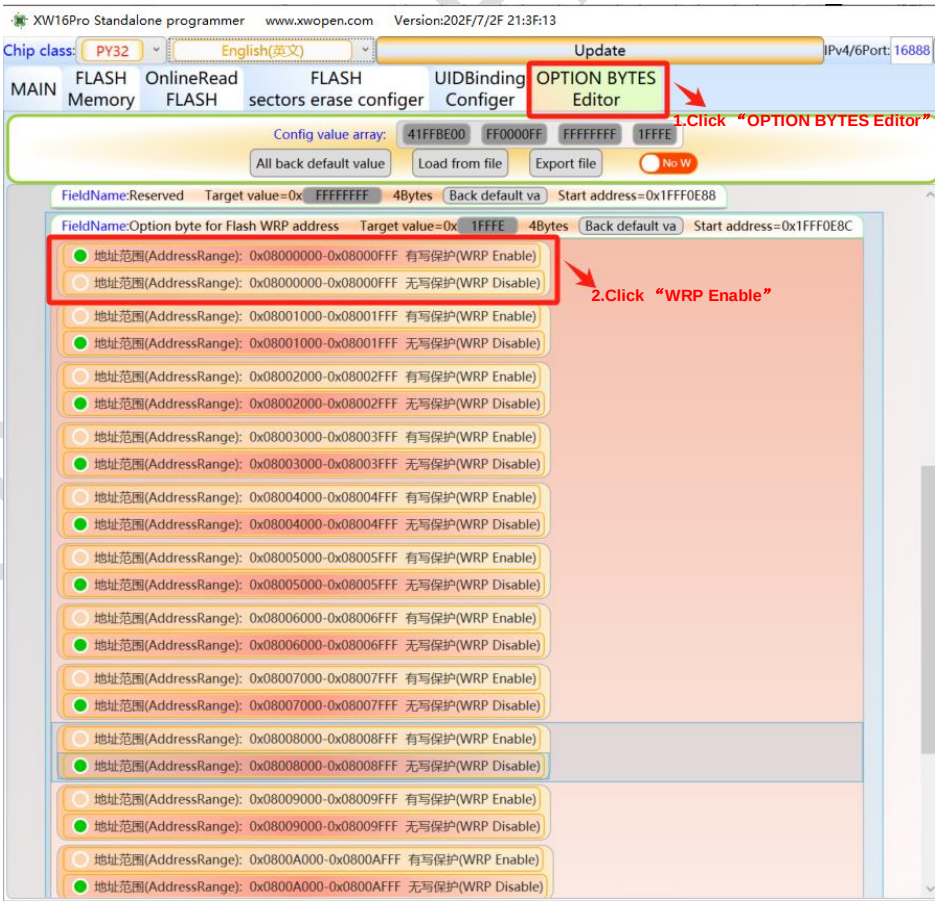


Figure 11-1 XW operation Option write protection

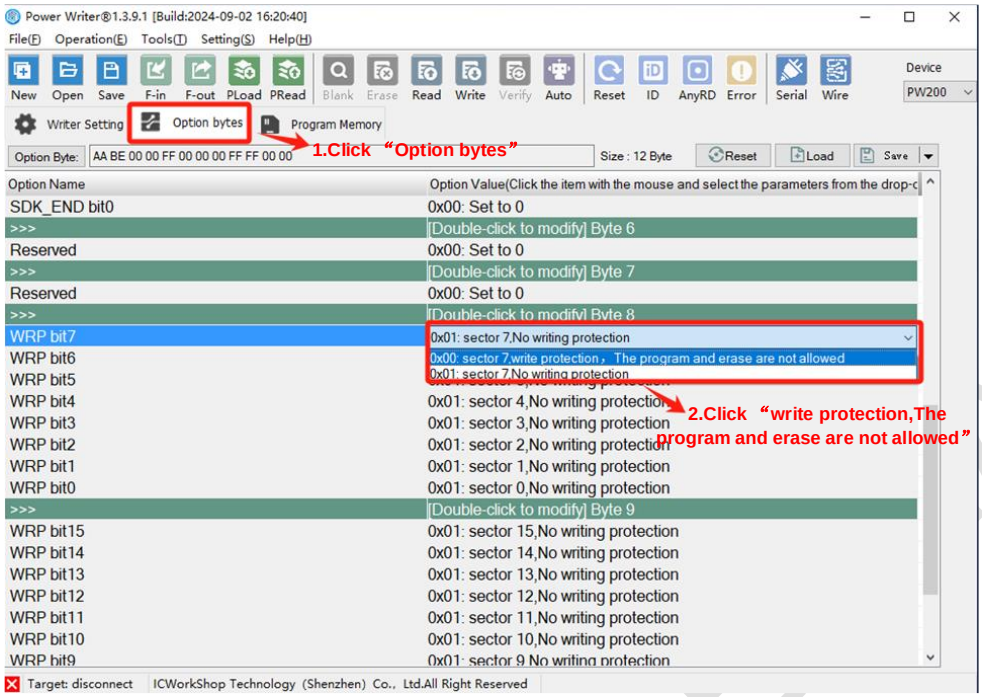


Figure 11-2 Power Write Operation Option Write Protection

- When configuring the Option of the programmer, you need to check the "Smart Reset" function or "Restart the chip after programming" (programmers typically have similar options that need to be selected). The specific steps are shown in Figures 11-3, Figure 11-4.

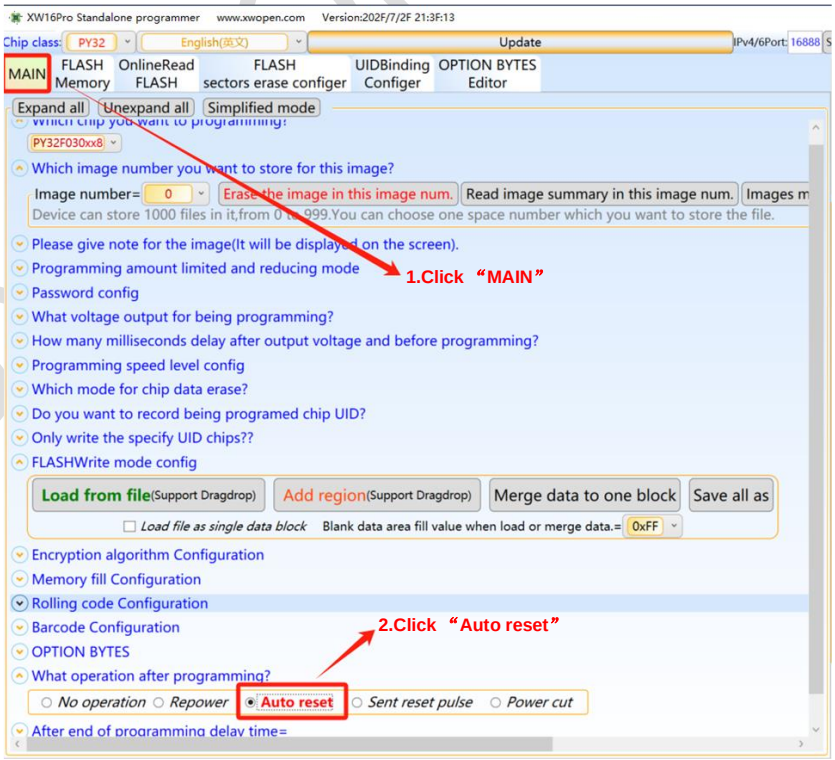


Figure 11-3 XW Operation 'Smart Reset'

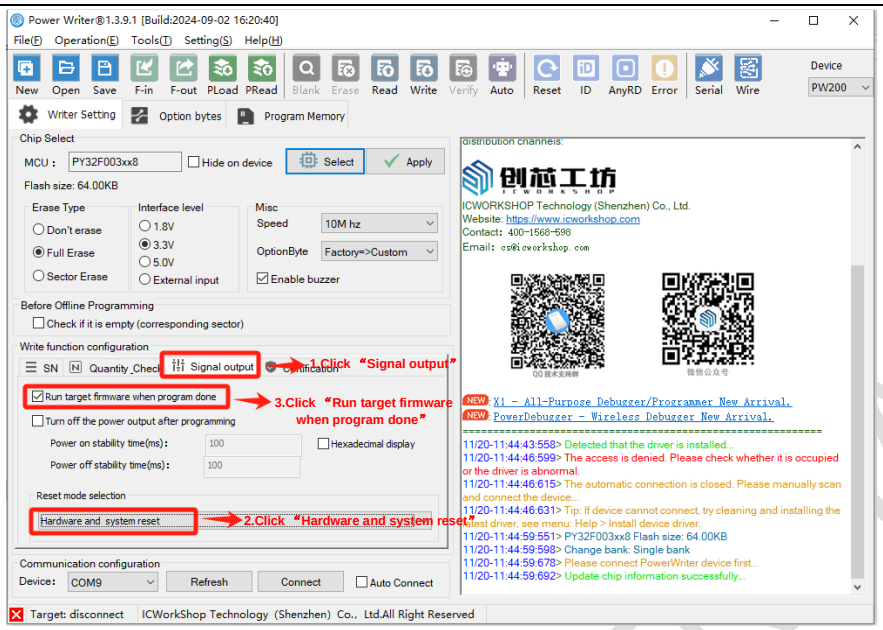


Figure 11-4 Power Write Operation Tick 'Reboot Chip After Programming''

12. LPTIM Configuration

- When LPTIM uses RCC_CCIPR->LPTIMSEL to select PCLK as the clock source, the pre-scaler cannot be set to 1, otherwise LPTIM has a probability of running abnormally.
- When LPTIM uses the RSTARE function, the interval between two reads of the CNT registers should satisfy 4 LSI clocks.

12.1 LPTIM Continuous Mode

- LPTIM continuous mode must clear ARRMCF each time before entering STOP and wait for 1 LSI clock cycle*PSC factor.(Approx. 40us*PSC including programme execution time.)
- Changing the LPTIM reload value requires a wait of 4 LSI clock cycles * PSC factor.(Approx. 160us*PSC including programme execution time.)

12.2 LPTIM Once Mode

- LPTIM once mode wake-up from STOP and wait 3 LSI clock cycles before entering STOP again. (Takes about 120us, including programme execution time.)
- Wait 4 LSI clock cycles when changing the reload value LPTIM_ARR. (Approx. 160us including programme execution time.)

13. DMA Configuration

- The DMA must ensure that the DMA enable is turned off only after the transfer is complete and not during the transfer, otherwise the DMA transfer will fail.

14. Version History

Version	Date	Update Records
V1.0	2023.10.09	Initial release
V1.1	2025.01.19	Add PWR、I2C、LCD、GPIO、ADC、RCC、SPI、IAP、FLASH、Option、LPTIM、DMA module content.
V1.2	2025.06.25	Modify COMP module contents

Puya Semiconductor Co.， Ltd.

Appendix 1

1 PY32F031 reads the Vreferint 1.2V actual value stored in the information area (see 6.3 for specific address).

```
#define HAL_VREF_INT                    (*(uint8_t *)(0x1fff0E23))
#define HAL_VREF_DEC                    (*(uint8_t *)(0x1fff0E22))
#define vref_int            (*(uint8_t *)(HAL_VREF_INT))
// Store the integer part of the reference voltage
#define vref_dec            (*(uint8_t *)(HAL_VREF_DEC))
// Store the fractional part of the reference voltage
float vref;                 //reference voltage

static uint8_t Bcd2ToByte(uint8_t Value)
{
    uint32_t tmp = 0U;
    tmp = ((uint8_t)(Value & (uint8_t)0xF0) >> (uint8_t)0x4) * 10U;
    return (tmp + (Value & (uint8_t)0x0F));
}

float read_1_2V(void)
{
   uint8_t data_vref_int,data_vref_dec;
   data_vref_int = Bcd2ToByte(HAL_VREF_INT);
   data_vref_dec = Bcd2ToByte(HAL_VREF_DEC);

   //Initialise all peripherals, flash interface, systick
    vref = data_vref_int/10;        //Calculating the reference voltage
    vref = vref + ((data_vref_int%10)*0.1 + data_vref_dec*0.001);
     return vref;
}
```

Appendix 2

2. When using PLL48M as the system clock, IAP jump should disable the PLL

```
LL_UTILS_ClkInitTypeDef UTILS_ClkInitStruct;
static void SYSCLK(void);
/**
  * @brief  The application entry point.
  * @param  None
  * @retval None
  */
int main(void)
{
  SYSCLK();
#ifdef JUMP_TO_APP_BY_USER_BUTTON
  /* Configure user Button */
  BSP_PB_Init(BUTTON_USER, BUTTON_MODE_GPIO);

  /* Check if the USER Button is pressed */
  if (BSP_PB_GetState(BUTTON_USER) == 0x00)
  {
    JumpToAddress(APP_ADDR);
  }
#endif

  APP_SystemClockConfig(LL_RCC_HSICALIBRATION_24MHz, 24000000);

  Bootloader_Init();

  /* Infinite loop */
  while (1)
  {
    Bootloader_ProtocolDetection();
  }
}
static void SYSCLK(void)
{
  /* Enable and initialize HSI */
  LL_RCC_HSI_Enable();
  LL_RCC_HSI_SetCalibFreq(LL_RCC_HSICALIBRATION_24MHz);
  while(LL_RCC_HSI_IsReady() != 1)
  {
  }

  LL_PLL_ConfigSystemClock_HSI(&UTILS_ClkInitStruct);

  /* Set AHB prescaler */
  LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);

  /* Configure HSISYS as system clock and initialize it */
  LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_PLL);
  while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_PLL)
  {
  }

  /* Set APB1 prescaler and initialize it */
  LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
```

```
    /* Update system clock global variable SystemCoreClock (can also be updated by calling
SystemCoreClockUpdate function) */
    LL_SetSystemCoreClock(48000000);
  }
  void APP_SystemClockConfig(uint32_t Value, uint32_t HCLKFrequency)
  {
    /* HSI Enable and Initialization */
    LL_RCC_HSI_Enable();
    LL_RCC_HSI_SetCalibFreq(Value);
    while(LL_RCC_HSI_IsReady() != 1)
    {
    }

    /*Setting AHB Clock Divider */
    LL_RCC_SetAHBPrescaler(LL_RCC_SYSCLK_DIV_1);

    /* Configuring HSISYS as the System Clock and Initialization */
    LL_RCC_SetSysClkSource(LL_RCC_SYS_CLKSOURCE_HSISYS);
    while(LL_RCC_GetSysClkSource() != LL_RCC_SYS_CLKSOURCE_STATUS_HSISYS)
    {
    }

    LL_FLASH_SetLatency(LL_FLASH_LATENCY_0);

    /* Setting APB1 Clock Divider and Initialization */
    LL_RCC_SetAPB1Prescaler(LL_RCC_APB1_DIV_1);
    /*Updating the global variable SystemCoreClock for the system clock (which can also be updated
through debugging the SystemCoreClockUpdate function) */
    LL_SetSystemCoreClock(HCLKFrequency);
  }
```