AN0082
Application Note

AT32F403A/407 CRM Quick Start Guide

Introduction

This application note mainly introduces:
1. How to configure and modify the clock source code based on the BSP_V2.x.x provided by Artery.
2. How to use the accessary clock configuration tools to set clock path and parameters to generate and use the corresponding clock code.

Applicable products:

Part number	AT32F403A series
	AT32F407 series

Contents

List of tables

List of figures

1 Overview

As the necessary condition for the correct and efficient running of chips, proper clock configuration is of great importance. The clock configuration of each AT32 series MCU may be slight different. This application note mainly introduces how to use BSP_V2.x.x provided by Artery to perform clock configuration of the corresponding AT32 MCU series.

The clock is configured with the following methods:

1. Manually write code to call the driver functions in BSP to implement clock configuration;
2. Use clock configuration tools to set and generate the corresponding source code file.

2 Clock tree

Before configuring the clock, it is necessary to have a comprehensive understanding of the clock tree, with focuses on the clock source, frequency multiplication and system clock.

Figure 1. AT32F403A/407 clock tree

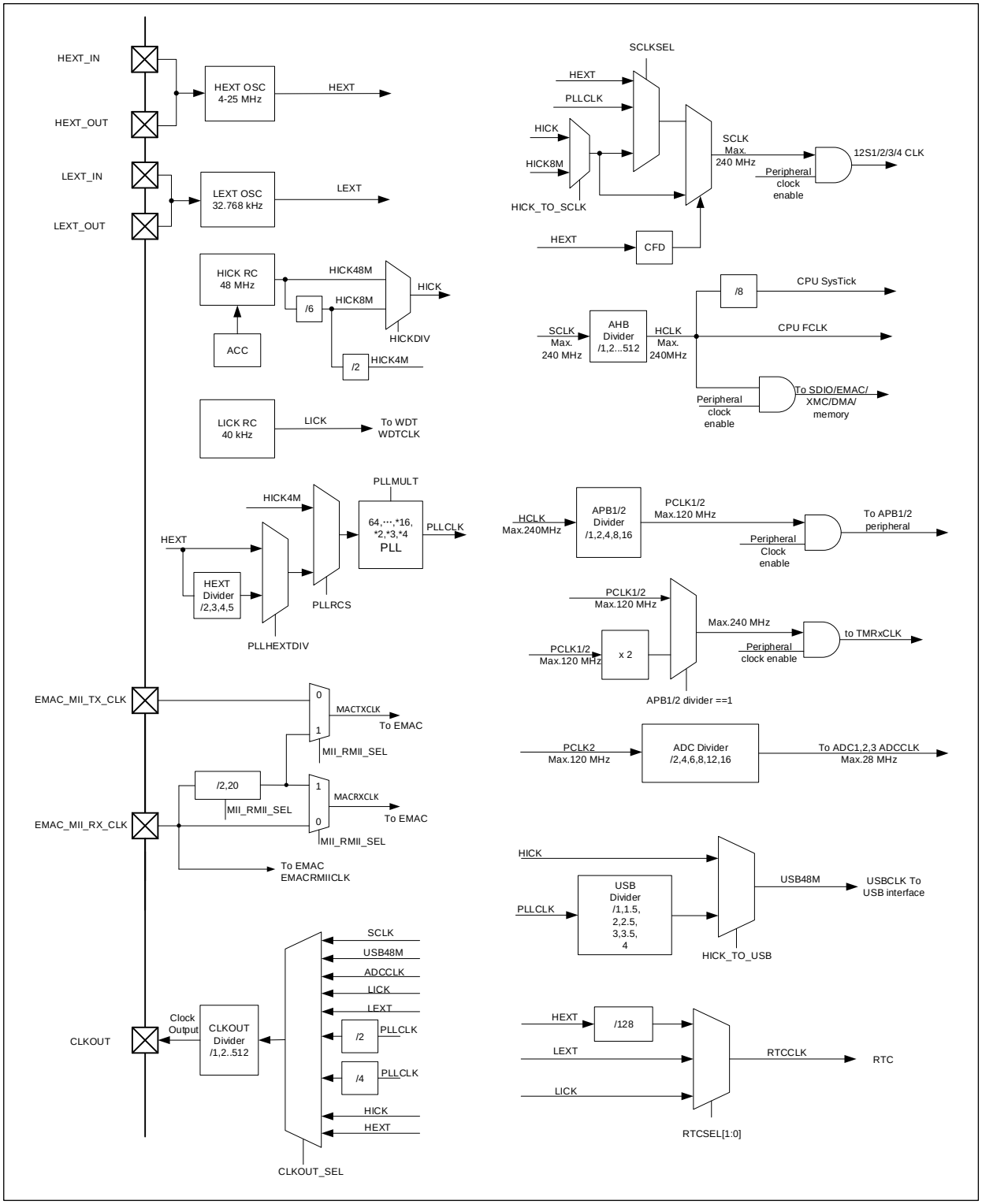


Figure 1 contains the following key elements:

1) SCLKSEL: The system clock can be selected from HEXT, PLLCLK and HICK.
2) HEXT: It is a high-speed external oscillator that is connected externally to a 4~25 MHz crystal or clock source.
3) HICK: It is a high-speed internal oscillator that is clocked by a high-speed RC, and the internal frequency of the HICK clock is 48 MHz. The HICKDIV bit is used to select HICK (48 MHz) or HICK/6 (if the HICK/6 is selected, the clock frequency is 8 MHz).
4) PLLCLK: PLL clock = PLL entry clock * PLL multiplication factor.
5) PLL entry clock: It is determined by PLLRCS and PLLHEXTDIV bits, and it can be selected from HICK 4 MHz, HEXT and HEXTDIV (derived from HEXT division, HEXT/2 by default).

3 Code configuration

This section introduces how to use library functions to configure the clock.

3.1 Functions

The interface functions for hardware clock configuration in BSP of the corresponding MCU series are encapsulated and can be called. Functions commonly used for clock configuration are listed below. Refer to *at32f403a_407_crm.c/.h* for details about parameters and return value of each function.

/* CRM reset function, which resets clock configuration to default*/ void crm_reset(void);
/* HEXT bypass enable function */ void crm_hext_bypass(confirm_state new_state);
/* Status flag get function, such as the stable status flag of PLL/HEXT/HICK */ flag_status crm_flag_get(uint32_t flag);
/* Wait for HEXT clock to stabilize*/ error_status crm_hext_stable_wait(void);
/* Clock source enable function, such as enable PLL/HEXT/HICK */ void crm_clock_source_enable(crm_clock_source_type source, confirm_state new_state);
/* Clock frequency division function when HEXT is selected as the PLL clock source */ void crm_hext_clock_div_set(crm_hext_div_type value);
/* PLL configuration function, including PLL clock source, PLL multiplication factor, frequency after multiplication*/ void crm_pll_config(crm_pll_clock_source_type clock_source, crm_pll_mult_type mult_value, crm_pll_output_range_type pll_range);
/* System clock switch function */ void crm_sysclk_switch(crm_sclk_type value);
/* Current system clock switch status get function */ crm_sclk_type crm_sysclk_switch_status_get(void);
/* Auto step-by-step system clock switch enable function; when PLL frequency > 108 MHz, enable auto step-by-step switch before switching the system clock to PLL */ void crm_auto_step_mode_enable(confirm_state new_state);
/* HICK/6 configuration function, which is mainly used to implement HICK 48 MHz as system clock and USB clock */ void crm_hick_divider_select(crm_hick_div_6_type value);
/* Frequency select function when HICK is selected as system clock; the frequency can be set to a fixed value (8 MHz) or HICK/6 (depending on the HICK divider select function) */ void crm_hick_sclk_frequency_select(crm_hick_sclk_frequency_type value);
/* System clock to AHB clock divider configuration function */ void crm_ahb_div_set(crm_ahb_div_type value);
/* AHB clock to APB1 clock divider configuration function */

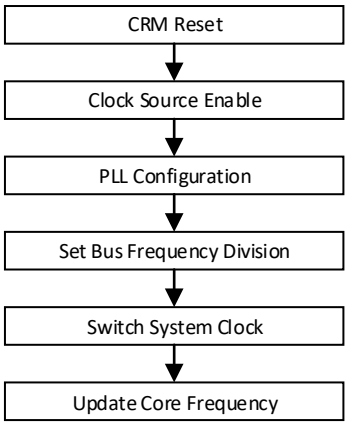
```
void crm_apb1_div_set(crm_apb1_div_type value);
```

```
/* AHB clock to APB2 clock divider configuration function */
void crm_apb2_div_set(crm_apb2_div_type value);
```

3.2 Clock configuration

Figure 2 shows the clock configuration process.

Figure 2. Clock configuration process



3.2.1 CRM reset

According to the configuration process, perform CRM reset firstly, which switches system clock to HICK, and write the default value to other system clock configuration registers. The code to reset CRM is as follows:

```
crm_reset();                    /* CRM reset */
```

3.2.2 Clock source configuration

The HEXT or HICK can be selected as the system clock source, which can also be used as a reference clock source of PLL. Before enabling PLL, enable the PLL reference clock source and wait until it becomes stable.

- HEXT

If HEXT is connected externally to an active clock, the HEXT bypass mode should be enabled. If the crystal oscillator is used, the HEXT bypass mode should be disabled. The bypass mode is set before enabling the HEXT clock source (disabled by default). The code to enable HEXT bypass mode is as follows:

```
crm_hext_bypass(TRUE);                                  /* HEXT bypass mode enable */
```

Enable HEXT clock source and wait until HEXT clock becomes stable. The code is as follows:

```
crm_clock_source_enable(CRM_CLOCK_SOURCE_HEXT, TRUE);   /* Enable HEXT clock source */
while(crm_hext_stable_wait() == ERROR)                  /* Wait until HEXT clock becomes stable */
{
}
```

- HICK

The HICK oscillator is clocked by a high-speed RC in the microcontroller. Enable HICK clock source and wait until HICK clock becomes stable. The code is as follows:

```
crm_clock_source_enable(CRM_CLOCK_SOURCE_HICK, TRUE);   /* Enable HICK clock source */
while(crm_flag_get(CRM_HICK_STABLE_FLAG) != SET)        /* Wait until HICK stable flag is set */
{
}
```

3.2.3 PLL configuration

PLL configuration includes PLL clock source, PLL multiplication factor and PLL frequency range after multiplication. The frequency multiplication formula is PLLCLK = PLL entry clock * PLL multiplication factor.

- PLL clock source

PLL clock source can be selected from HICK (4 MHz), HEXT and a divided HEXT. PLL clock source is enabled and becomes stable before enabling the PLL configuration. The corresponding parameters of these three clock sources in *crm_pll_config* function are as follows:

```
CRM_PLL_SOURCE_HICK
CRM_PLL_SOURCE_HEXT
CRM_PLL_SOURCE_HEXT_DIV
```

When the PLL is clocked by CRM_PLL_SOURCE_HEXT_DIV, the HEXT division factor is set by the *crm_hext_clock_div_set* function. The HEXT is divided by 2, by default.

- PLL multiplication factor

The PLL multiplication factor is set (from 2 to 64) according to the maximum frequency. For example, select CRM_PLL_MULT_8, indicating that the multiplication factor is 8.

- PLL frequency range

The frequency range is set according to PLLCLK after multiplication.

```
CRM_PLL_OUTPUT_RANGE_LE72MHZ        /* PLLCLK ≤ 72 MHz */
CRM_PLL_OUTPUT_RANGE_GT72MHZ        /* PLLCLK > 72 MHz */
```

After the PLL configuration is completed, enable PLL and wait until the PLL becomes stable. For example, select HEXT/2 (8 MHz/2= 4 MHz) as the PLL clock source, and the code to implement 240 MHz PLLCLK is as follows:

```
crm_pll_config(CRM_PLL_SOURCE_HEXT_DIV, CRM_PLL_MULT_60,
CRM_PLL_OUTPUT_RANGE_GT72MHZ);                  /* Configure PLL parameters */
crm_hext_clock_div_set(CRM_HEXT_DIV_2);         /* Configure HEXT division factor */
crm_clock_source_enable(CRM_CLOCK_SOURCE_PLL, TRUE);   /* Enable PLL clock source */
while(crm_flag_get(CRM_PLL_STABLE_FLAG) != SET)        /* Wait until PLL stable flag is set */
{
}
```

3.2.4 Set bus frequency division

The bus frequency division configuration includes SCLK to AHBCLK, AHBCLK to APB1CLK and AHBCLK to APB2CLK. The code to implement AHB bus (SCLK/1) and APB1/APB2 bus (AHBCLK/2) is as follow:

```
crm_ahb_div_set(CRM_AHB_DIV_1);          /* SCLK/1 is used as AHB bus clock */
crm_apb2_div_set(CRM_APB2_DIV_2);        /* AHBCLK/2 is used as APB2 bus clock */
crm_apb1_div_set(CRM_APB1_DIV_2);        /* AHBCLK/2 is used as APB1 bus clock */
```

3.2.5 Switch system clock

The system clock has three main sources: HICK, HEXT and PLLCLK. Before switching the system clock to any of the three clocks, the corresponding clock source must be stable.

◆ Auto step-by-step switch mode

The automatic frequency switch is designed to ensure a smooth and stable switch of system frequency. When the operational target is larger than 108 MHz, it is recommended to enable the auto step-by-step switch mode. This function is mainly used when PLLCLK is selected as system clock, which is enabled before switching the system clock and disabled after the switching is completed. The code is as follows:

```
crm_auto_step_mode_enable(TRUE);          /* Enable auto step-by-step switch mode */
crm_auto_step_mode_enable(FALSE);         /* Disable auto step-by-step switch mode */
```

◆ HICK

The HICK is used as system clock source by default after system reset. As shown in Figure 1, the HICK frequency is set to 8 MHz (by default) and can be set to 48 MHz.

The code to configure HICK 8 MHz as system clock is as follows:

```
crm_sysclk_switch(CRM_SCLK_HICK);                    /* Switch system clock source to HICK */
while(crm_sysclk_switch_status_get() != CRM_SCLK_HICK)   /* Wait until the system clock is switched to HICK */
{
}
```

The code of to configure HICK 48 MHz as system clock is as follows:

```
crm_hick_sclk_frequency_select (CRM_HICK_SCLK_48MHZ);   /* Select 48 MHz HICK */
crm_sysclk_switch(CRM_SCLK_HICK);                       /* Switch system clock source to HICK */
while(crm_sysclk_switch_status_get() != CRM_SCLK_HICK)  /* Wait until the system clock is switched to HICK */
{
}
```

◆ HEXT

When the HEXT is used as system clock source, the system clock frequency depends on the actual external clock frequency (range: 4~25 MHz). The code to configure HEXT as system clock source

is as follows:

```
crm_sysclk_switch(CRM_SCLK_HEXT);                    /* Switch system clock source to HEXT */
while(crm_sysclk_switch_status_get() != CRM_SCLK_HEXT)    /* Wait until the system clock is switched to HEXT */
{
}
```

- PLLCLK

When the PLLCLK is used as system clock source, the system clock frequency depends on the actual PLL frequency multiplication and the maximum frequency. The code to configure PLLCLK as system clock source is as follows:

```
crm_sysclk_switch(CRM_SCLK_PLL);                    /* Switch system clock source to PLL */
while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL)    /* Wait until the system clock is switched to PLL */
{
}
```

3.2.6 Update core frequency

The parameter “system_core_clock” indicating the system core frequency is reserved in the BSP code framework to save the CPU core operating frequency. It is updated each time the system clock is configured, so that peripherals can quickly get and use the current core operating frequency. The code is as follows:

```
system_core_clock_update();                    /* Update system core frequency system_core_clock */
```

3.3 Example of clock configuration

In this example, the 8 MHz HEXT is used as clock source, then select HEXT/2 and implement 240 MHz system clock through PLL multiplication; SCLK/1 is used as AHB bus clock, and AHBCLK/2 is used as APB1/APB2 bus clock. The *system_clock_config* function code is as follows:

```
void system_clock_config(void)
{
  crm_reset();                                    /* CRM reset */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_HEXT, TRUE);    /* Enable HEXT clock source */
  while(crm_hext_stable_wait() == ERROR)                   /* Wait until HEXT clock becomes stable */
  {
  }
  crm_pll_config(CRM_PLL_SOURCE_HEXT_DIV, CRM_PLL_MULT_60,
CRM_PLL_OUTPUT_RANGE_GT72MHZ);                    /* Configure PLL: select HEXT division factor, set multiplication factor = 60 and PLL clock output range > 72 MHz; formula: PLLCLK = 8 / 2 * 60 = 240 MHz */
  crm_hext_clock_div_set(CRM_HEXT_DIV_2);                  /* HEXT is divided by 2 */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_PLL, TRUE);     /* Enable PLL */
```

```
  while(crm_flag_get(CRM_PLL_STABLE_FLAG) != SET)          /* Wait until PLL becomes stable */
  {
  }
  crm_ahb_div_set(CRM_AHB_DIV_1);                          /* SCLK/1 is used as AHB bus clock */
  crm_apb2_div_set(CRM_APB2_DIV_2);                        /* AHBCLK/2 is used as APB2 bus clock */
  crm_apb1_div_set(CRM_APB1_DIV_2);                        /* AHBCLK/2 is used as APB1 bus clock */
  crm_auto_step_mode_enable(TRUE);                         /* PLLCLK 240 MHz >108 MHz, enable
auto step-by-step switch mode */
  crm_sysclk_switch(CRM_SCLK_PLL);                         /* Switch system clock source to PLL */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL)    /* Wait until the system clock is switched
to PLL */
  {
  }
  crm_auto_step_mode_enable(FALSE);                        /* Switch completed, disable auto step-by-
step switch mode */
  system_core_clock_update();                              /* Update system core frequency */
}
```

4 Clock configuration tools

The New Clock Configuration is a graphical configuration tool developed by Artery for configuring the clock of AT32 series MCUs, to help users have a better understanding of the clock path, configure the required frequency and generate source files.

4.1 Environment requirement

- **Software**

Windows7 and above

4.2 Installation

- **Software installation**

Run *AT32_New_Clock_Configuration.exe* directly.

4.3 Function overview

This section mainly introduces basic operations of this tool. The startup interface and configuration interface are shown below.

Figure 3. Startup interface

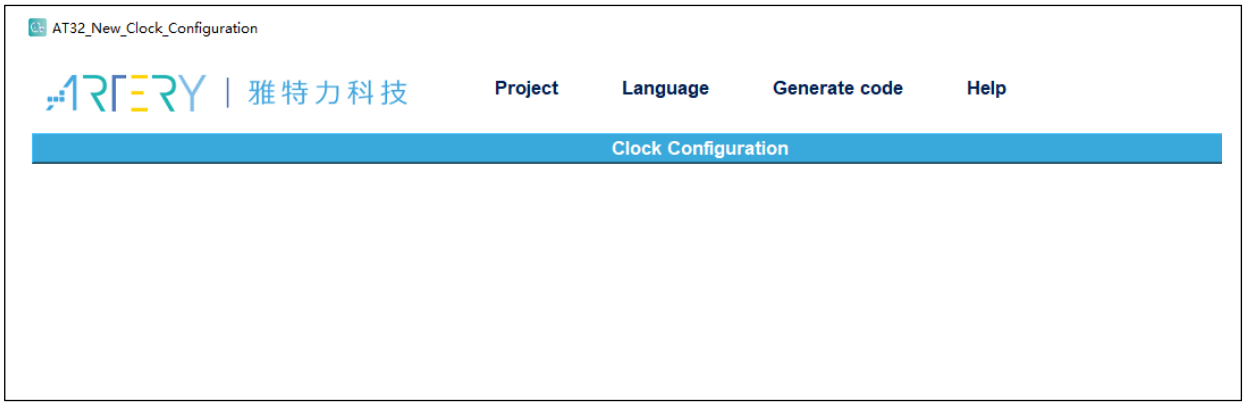
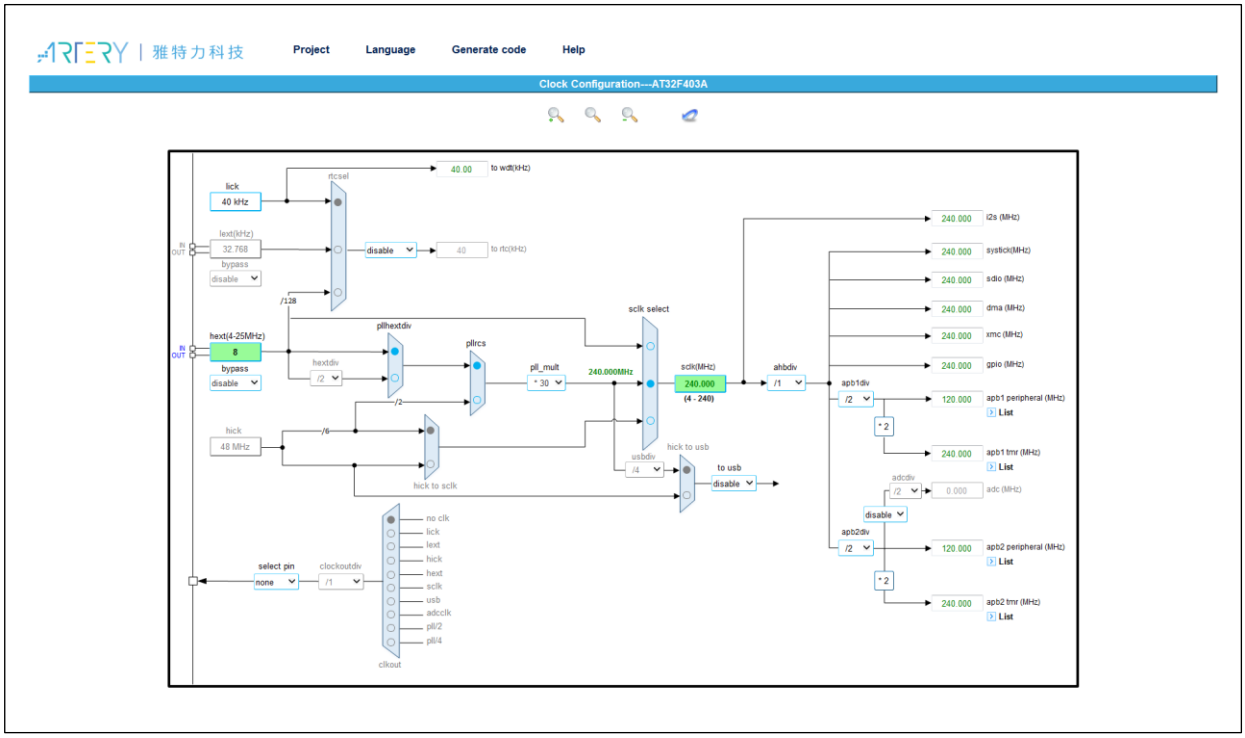


Figure 4. Configuration interface



4.4 Menu bar

The menu bar is shown below.

Figure 5. Menu bar

- **Project**
  - New — Create a new configuration project
  - Open — Open an existing configuration project
  - Save — Save the current configuration project
- **Language**
  - English — Select English as the display language
  - Chinese — Select simplified Chinese as the display language
- **Generate code**

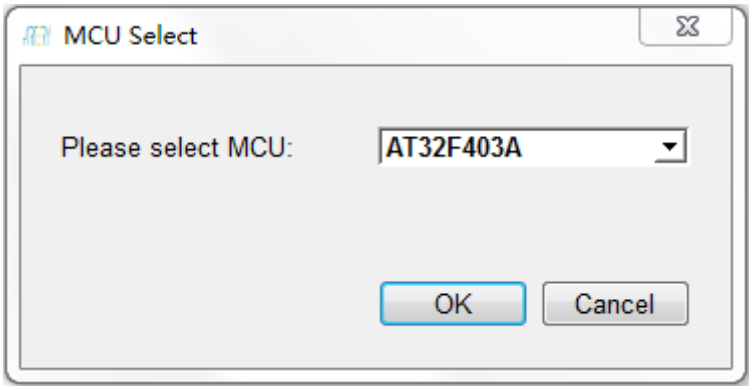
Configure the required clock path and clock frequency of the corresponding MCU series, then click “Generate code” to select the storage path and generate the corresponding source file.

- **Help**
  - New version download — Connect to network to download the latest version
  - Version — View the current version

4.5 Create configuration project

Double click to run the *AT32_New_Clock_Configuration.exe*; go to Project-->New in the startup interface to create a new configuration project, and select the corresponding MCU series, as shown in Figure 6.

Figure 6. Select MCU



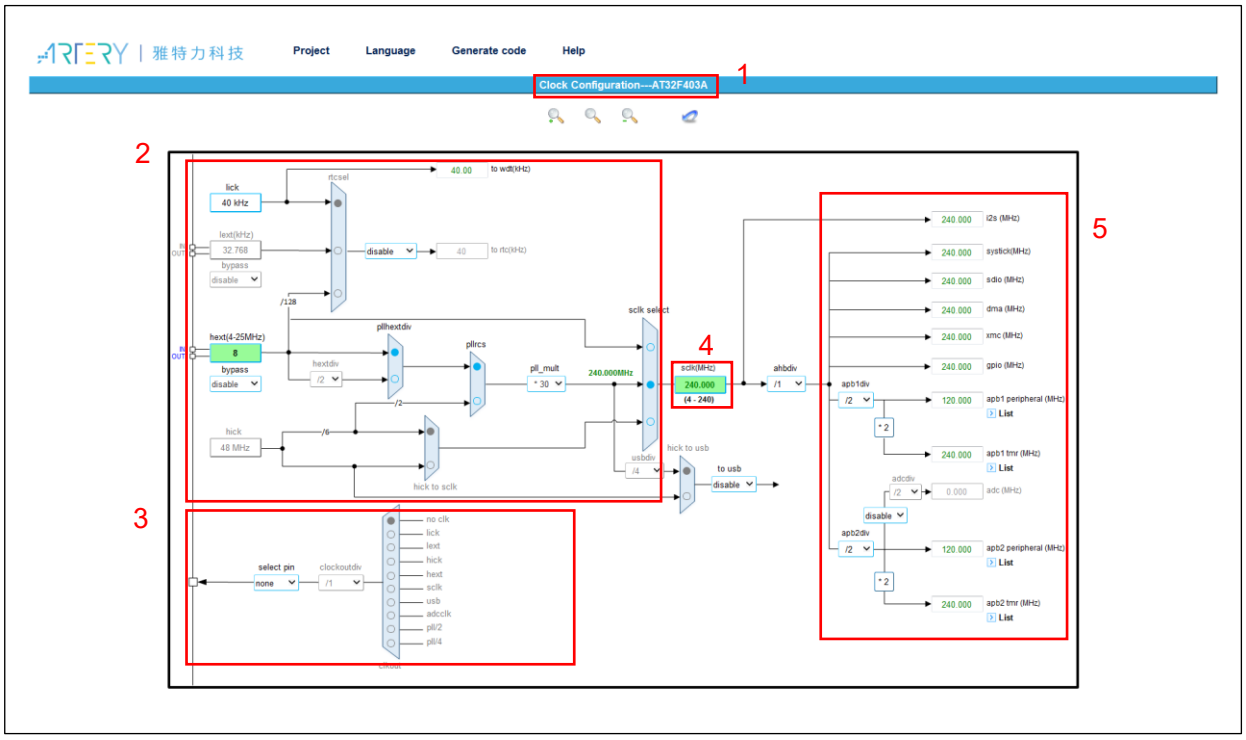
Click on the drop-down box to select the corresponding MCU series; then click on “OK” to enter the configuration interface.

4.6 Clock configuration interface

Users can configure the clock path and parameters in the clock configuration interface. Figure 7 shows the clock configuration for AT32F403A series MCU.

The clock configuration interface mainly contains four blocks, as shown below.

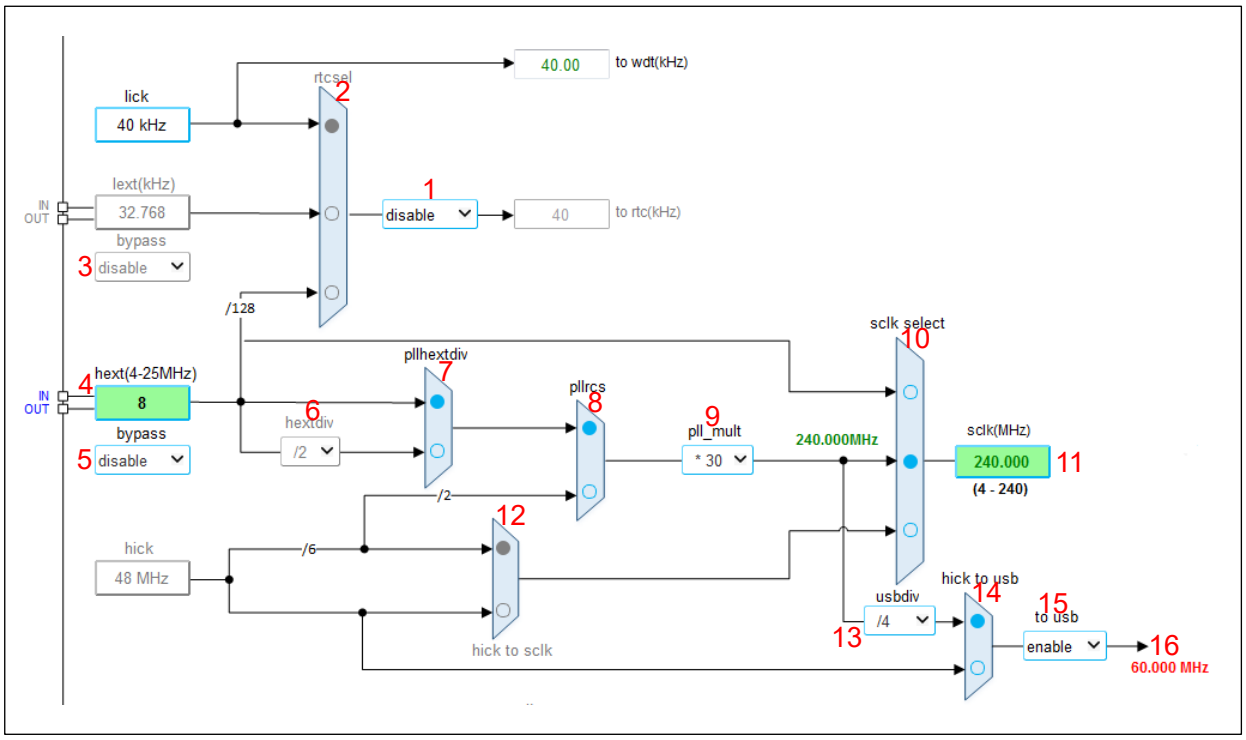
Figure 7. Clock configuration interface



1. Title: Display the corresponding MCU series of the current clock configuration project.
2. Configuration: Select and configure the clock path and parameters to meet application requirements.
3. Output: Configure the clock output (CLKOUT).
4. SCLK: When the PLL is used as system clock, the SCLK field can be used as an input box of desired system clock frequency to automatically configure the multiplication factor reversely.
5. Result: Display the clock frequency of peripherals and peripherals on the bus.

Figure 8 illustrates the clock configuration block. The configuration process corresponds to MCU clock tree, which may be slightly different for different MCU series. The clock path can be configured by clicking on each checkbox according to the configuration process, as shown below.

Figure 8. Clock configuration block



1. rtc enable: drop-down box, which is used to enable RTC clock code configuration.
2. rtcsel: checkbox for RTC clock source selection; when the rtc is enabled, click on this checkbox for configuration.
3. lext bypass: LEXT bypass enable.
4. hext: input box, which is 8 MHz by default; it can be modified according to the actual external clock source (note: if other frequency value is set, the HEXT_VALUE in *inc/at32f403a_407_conf.h* in the corresponding BSP demo should be modified, or use the *at32f403a_407_conf.h* file generated by using tools).
5. hext bypass: HEXT bypass enable.
6. hextdiv: drop-down box, which is used to configure the HEXT division factor when a divided HEXT is used as PLL clock source.
7. pllhextdiv: checkbox, which is used to configure HEXT or divided HEXT when HEXT is used as PLL clock source.
8. pllrcs: checkbox, which is used to select HEXT or HICK as the PLL clock source.
9. Multiplication factor: the PLL_MULT is used for frequency multiplication (PLLCLK = PLL entry clock * PLL_MULT). After the PLL entry clock is selected, input the desired frequency into the output “sclk” field and then press “Enter”, and a set of proper or approximate multiplication factors will be calculated automatically to meet requirements.
10. sclk select: checkbox, which is used to configure HEXT, PLL or HICK as the system clock.
11. sclk frequency: In forward configuration, it displays the configuration result of system clock frequency. When it is used as an input box, users can input the desired frequency and press “Enter”, and then a set of proper or approximate PLL multiplication factors will be calculated.
12. hick to sclk: checkbox, which is used to configure HICK/6 (8 MHz) or HICK (48 MHz) as the system clock when HICK is selected in the “sclk select” checkbox (note: if 48 MHz HICK is

selected, CLKOUT HICK frequency is also 48 MHz).

13. usbdiv: drop-down box, which is used to configure PLL division factor when PLL clock is selected as the USB clock source.
14. hick to usb: checkbox, which is used to configure USB clocked by PLL clock or HICK 48 MHz. The USB clock configuration code is controlled by the “to usb” drop-down box. The USB clock is required to be fixed 48 MHz; therefore, with the division factor in “usbdiv”, USB 48 MHz may not be implemented through PLL multiplication.
15. USB enable: drop-down box for enabling USB clock code configuration.
16. USB clock frequency display: this field calculates and displays USB clock frequency in a real-time manner. If the configured USB clock frequency is not equal to 48 MHz, the frequency value will be displayed in red; if USB clock is not used, select “disable” in the drop-down box and no message will be displayed (note: it is only used for USB clock frequency configuration; USB peripheral clocks are enabled separately as needed).

4.7 Generate code

After clock configuration is completed, click on “Generate code”, select and confirm the code storage path, and then two folders “inc” and “src” are generated to save the header file and source file, respectively. These files can be used together with the project in BSP_V2.x.x. The generated clock code file (*at32f4xx_clock.c/ at32f4xx_clock.h/ at32f4xx_conf.h*) can replace the corresponding file in the original BSP demo, and can be used by calling the *system_clock_config* in the main function.

5 Notes

5.1 Modification of HEXT

The HEXT involved in demo and configuration tools in this application note is 8 MHz. To modify the 8 MHz HEXT in actual applications, the following aspects should be noted:

◆ Code

1. Compile the code according to the actual external clock frequency and the configuration process and method as described in this application note to set the required clock configuration and clock path.
2. Modify the HEXT_VALUE in *at32f4xx_conf.h* file of the corresponding demo project according to the actual HEXT clock frequency. For example, if the 12.288 MHz external crystal oscillator or clock source is used, modify the *at32f4xx_conf.h* file as below:

```
#if !defined   HEXT_VALUE
#define HEXT_VALUE                ((uint32_t)12288000)
#endif
```

◆ Tools

1. Input the actual frequency of external clock source to the HEXT input box, and then press “Enter”.
2. Configure the required clock path and clock frequency, and then generate code. Use the generated clock code file (at32f4xx_clock.c/ at32f4xx_clock.h/ at32f4xx_conf.h) to replace the corresponding file or function content in the original BSP demo, and then call the *system_clock_config* in the main function.

5.2 Usage of tools

Notes on using the New Clock Configuration tool:

1. The clock configuration source file generated by the New Clock Configuration tool should be used together with BSP_V2.x.x provided by Artery.
2. The clock configuration source file generated for the specific MCU series should be used in the corresponding project only.
3. After modifying parameters in each input box, press “Enter” to confirm.

6 Application case 1: Switch system clock

6.1 Introduction

Switch the system clock when the system is running.

6.2 Resources

1) Hardware

AT-START BOARD of the corresponding MCU series

2) Software

project\at_start_f403a\examples\crm\sysclk_switch

6.3 Software design

1) Configuration process

- Initialize buttons;
- Configure PLL/4 for CLKOUT;
- Compile the code to select HICK to be PLL clock source and implement PLLCLK 64 MHz through frequency multiplication;
- Compile the code to select HICK/2 to be PLL clock source and implement PLLCLK 96 MHz through frequency multiplication.

2) Code

Main function code

```
int main(void)
{
  system_clock_config();                          /* Configure system clock to be 240 MHz, by
default */
  at32_board_init();                              /* Initialize buttons and LEDs */
  clkout_config();                                /* Configure PLL/4 for CLKOUT */
  while(1)
  {
    if(at32_button_press() == USER_BUTTON)        /* Check whether the button is pressed */
    {
      switch_system_clock();                      /* Switch system clock between 64 MHz and
96 MHz alternately */
      at32_led_toggle(LED4);                      /* Switch once, and LED4 toggles once */
    }
    at32_led_toggle(LED2);                        /* LED2 is the operating status indicator */
    delay_ms(100);                                /* Delay for 100 ms */
  }
}
```

Code of configuring 64 MHz PLLCLK through PLL multiplication

```
static void sclk_64m_hick_config(void)
{
  crm_reset();                                    /* CRM reset */
```

```
  crm_clock_source_enable(CRM_CLOCK_SOURCE_HICK, TRUE);      /* Enable HICK clock source */
  while(crm_flag_get(CRM_HICK_STABLE_FLAG) != SET)           /* Wait until HICK stable flag is
 set */
  {
  }
  crm_pll_config(CRM_PLL_SOURCE_HICK, CRM_PLL_MULT_16,
CRM_PLL_OUTPUT_RANGE_LE72MHZ);                               /* Configure PLL: select HICK as
the PLL clock source; set multiplication factor = 16, and PLL clock output range ≤ 72 MHz; formula: PLLCLK
= 8 / 2 * 16 = 64 MHz */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_PLL, TRUE);     /* Enable PLL */
  while(crm_flag_get(CRM_PLL_STABLE_FLAG) != SET)          /* Wait until PLL becomes stable */
  {
  }
  crm_ahb_div_set(CRM_AHB_DIV_1);                          /* SCLK/1 is used as AHB bus clock */
  crm_apb2_div_set(CRM_APB2_DIV_2);                        /* AHBCLK/2 is used as APB2 bus clock */
  crm_apb1_div_set(CRM_APB1_DIV_2);                        /* AHBCLK/2 is used as APB1 bus clock */
  //crm_auto_step_mode_enable(TRUE);                       /* Enable auto step-by-step switch
 mode; do not enable when PLLCLK 64 MHz <108 MHz */
  crm_sysclk_switch(CRM_SCLK_PLL);                         /* Switch system clock source to
PLL */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL)    /* Wait until the system clock is
switched to PLL */
  {
  }
  //crm_auto_step_mode_enable(FALSE);                      /* Disable auto step-by-step switch mode */
  system_core_clock_update();                              /* Update system core frequency */
  delay_init();                                            /* Initialize delay after system clock switch */
  clkout_config();                                         /* Configure CLKOUT after CRM reset */
}
```

Code of configuring 96 MHz PLLCLK through PLL multiplication

```
static void sclk_96m_hext_config(void)
{
  crm_reset();                                             /* CRM reset */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_HEXT, TRUE);    /* Enable HEXT clock source */
  while(crm_hext_stable_wait() == ERROR)                   /* Wait until HEXT clock becomes
stable */
  {
  }
  crm_pll_config(CRM_PLL_SOURCE_HEXT_DIV, CRM_PLL_MULT_24,
CRM_PLL_OUTPUT_RANGE_GT72MHZ);                             /* Configure PLL: select HEXT/2 as
```

```
the PLL clock source; set multiplication factor = 24, and PLL clock output range >72 MHz; formula: PLLCLK
= 8 / 2 * 24 = 96 MHz */
  crm_hext_clock_div_set(CRM_HEXT_DIV_2);                    /* HEXT is divided by 2 */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_PLL, TRUE);   /* Enable PLL */
  while(crm_flag_get(CRM_PLL_STABLE_FLAG) != SET)            /* Wait until PLL becomes stable */
  {
  }
  crm_ahb_div_set(CRM_AHB_DIV_1);                        /* SCLK/1 is used as AHB bus clock */
  crm_apb2_div_set(CRM_APB2_DIV_2);                      /* AHBCLK/2 is used as APB2 bus clock */
  crm_apb1_div_set(CRM_APB1_DIV_2);                      /* AHBCLK/2 is used as APB1 bus clock */
  //crm_auto_step_mode_enable(TRUE);                         /* Enable auto step-by-step switch mode;
do not enable when PLLCLK 96 MHz <108 MHz */
  crm_sysclk_switch(CRM_SCLK_PLL);                         /* Switch system clock source to PLL */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL)      /* Wait until the system clock is switched
to PLL */
  {
  }
  //crm_auto_step_mode_enable(FALSE);                       /* Disable auto step-by-step switch mode */
  system_core_clock_update();                              /* Update system core frequency */
  delay_init();                                            /* Initialize delay after system clock switch */
  clkout_config();                                         /* Configure CLKOUT after CRM reset */
}
```

6.4 Test result

- Power on and run, and LED2 blinks at an interval of 100ms, and CLKOUT (PA8) output is 60 MHz.
- Press USER button, and the system clock switches between 64 MHz and 96 MHz; the CLKOUT output is divided by 4, and LED4 toggles once.

7 Application case 2: Clock fail detector

7.1 Introduction

The clock fail detector is designed to respond to HEXT clock failure when the HEXT is used as the system clock, directly or indirectly. If a failure is detected on the HEXT clock, a NMI interrupt is generated so that the software can perform rescue operations.

7.2 Resources

1) Hardware

   AT-START BOARD of the corresponding MCU series

2) Software

   project\at_start_f403a\examples\crm\clock_failure_detection

7.3 Software design

1) Configuration process

   - Configure PLL/4 for CLKOUT;
   - Enable clock fail detector and complete *void NMI_Handler(void)* function;
   - Compile the code to select HICK to be PLL clock source and implement PLLCLK 240 MHz through frequency multiplication.

2) Code

   Main function code

```
int main(void)
{
  system_clock_config();                          /* Configure system clock */
  at32_board_init();                              /* Initialize LED and delay function */
  clkout_config();                                /* Configure PLL/4 for CLKOUT */
  crm_clock_failure_detection_enable(TRUE);       /* Enable clock fail detector */
  while(1)
  {
    at32_led_toggle(LED2);                        /* LED2 is the operating status indicator light */
    delay_ms(200);                                /* Delay for 200 ms */
  }
}
```

   Code of configuring 240 MHz PLLCLK through PLL multiplication

```
static void sclk_240m_hick_config(void)
{
  crm_reset();                                                /* CRM reset */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_HICK, TRUE);       /* Enable HICK clock source */
  while(crm_flag_get(CRM_HICK_STABLE_FLAG) != SET)            /* Wait until HICK stable flag is
set */
  {
  }
```

```
  crm_pll_config(CRM_PLL_SOURCE_HICK, CRM_PLL_MULT_60,
CRM_PLL_OUTPUT_RANGE_GT72MHZ);                    /* Configure PLL: select HICK as
PLL clock source; set multiplication factor = 60, and PLL clock output range > 72 MHz; formula: PLLCLK = 8
/ 2 * 60 = 240 MHz */
  crm_clock_source_enable(CRM_CLOCK_SOURCE_PLL, TRUE);     /* Enable PLL */
  while(crm_flag_get(CRM_PLL_STABLE_FLAG) != SET)          /* Wait until PLL becomes stable */
  {
  }
  crm_ahb_div_set(CRM_AHB_DIV_1);                  /* SCLK/1 is used as AHB bus clock */
  crm_apb2_div_set(CRM_APB2_DIV_2);                /* AHBCLK/2 is used as APB2 bus clock */
  crm_apb1_div_set(CRM_APB1_DIV_2);                /* AHBCLK/2 is used as APB1 bus clock */
  crm_auto_step_mode_enable(TRUE);                 /* Enable auto step-by-step switch
mode */
  crm_sysclk_switch(CRM_SCLK_PLL);                 /* Switch system clock source to
PLL */
  while(crm_sysclk_switch_status_get() != CRM_SCLK_PLL)    /* Wait until the system clock is
switched to PLL */
  {
  }
  crm_auto_step_mode_enable(FALSE);                /* Disable auto step-by-step switch mode */
  system_core_clock_update();                      /* Update system core frequency */
  delay_init();                                    /* Initialize delay after system clock switch */
  clkout_config();                                 /* Configure CLKOUT after CRM reset */
}
```

NMI interrupt implementation

```
void NMI_Handler(void)
{
  clock_failure_detection_handler();
}

void clock_failure_detection_handler(void)
{
  if(crm_flag_get(CRM_CLOCK_FAILURE_INT_FLAG) != RESET)    /* Check the clock failure flag */
  {
    crm_clock_failure_detection_enable(FALSE);             /* Disable clock fail detector */
    sclk_240m_hick_config();                               /* System clock rescue; select HICK and
implement SCLK 240 MHz through frequency multiplication */
    crm_flag_clear(CRM_CLOCK_FAILURE_INT_FLAG);      /* Clear clock failure flag */
  }
```

```
}
```

7.4 Test result

- A clock failure occurs when the crystal is disconnected or the crystal pin is grounded during operation. Generally, HEXT is more stable than HICK, which is suitable to be used to observe CLKOUT (PA8) output. A slight frequency fluctuation can be found after rescue operations when HICK is used as the system clock source.

8 Revision history

Table 1. Document revision history

Date	Version	Revision note
2021.8.18	2.0.0	Initial release.
2021.9.30	2.0.1	Updated sample code processing flow.
2021.10.21	2.0.2	Updated clock block diagram and partial function name.
2021.11.24	2.0.3	Updated the screenshot of New Clock Configuration tool.
2022.06.10	2.0.4	Updated the screenshot of New Clock Configuration tool and description of operation procedures.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

Purchasers are solely responsible for the selection and use of ARTERY's products and services; ARTERY assumes no liability for purchasers' selection or use of the products and the relevant services.

No license, express or implied, to any intellectual property right is granted by ARTERY herein regardless of the existence of any previous representation in any forms. If any part of this document involves third party's products or services, it does NOT imply that ARTERY authorizes the use of the third party's products or services, or permits any of the intellectual property, or guarantees any uses of the third party's products or services or intellectual property in any way.

Except as provided in ARTERY's terms and conditions of sale for such products, ARTERY disclaims any express or implied warranty, relating to use and/or sale of the products, including but not restricted to liability or warranties relating to merchantability, fitness for a particular purpose (based on the corresponding legal situation in any unjudicial districts), or infringement of any patent, copyright, or other intellectual property right.

ARTERY's products are not designed for the following purposes, and thus not intended for the following uses: (A) Applications that have specific requirements on safety, for example: life-support applications, active implant devices, or systems that have specific requirements on product function safety; (B) Aviation applications; (C) Aerospace applications or environment; (D) Weapons, and/or (E) Other applications that may cause injuries, deaths or property damages. Since ARTERY products are not intended for the above-mentioned purposes, if purchasers apply ARTERY products to these purposes, purchasers are solely responsible for any consequences or risks caused, even if any written notice is sent to ARTERY by purchasers; in addition, purchasers are solely responsible for the compliance with all statutory and regulatory requirements regarding these uses.

Any inconsistency of the sold ARTERY products with the statement and/or technical features specification described in this document will immediately cause the invalidity of any warranty granted by ARTERY products or services stated in this document by ARTERY, and ARTERY disclaims any responsibility in any form.