Configure the Clock Tree in STM32

Almost every digital circuit needs clocks - the periodic signals that make circuits alive. Inside an MCU, there is a complex clock distribution network that drive every component and peripheral. When a device start, it should firstly setup its clock tree.

#arm #stm32 #clock

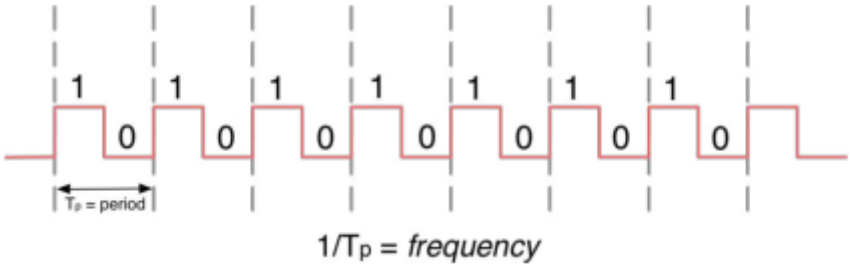
Last update: 2021-06-09 21:46:02

Table of Content

1. Clock source

A *clock* is a device that usually generates a *periodical square wave* signal, with a *50% duty cycle*. A clock signal oscillates between V_L and V_H voltage levels, which for STM32 microcontrollers are a fraction of the V_{DD} supply voltage.

The most fundamental parameter of a clock is the frequency, which indicates how many times it switches from V_L to V_H in a second. The frequency is expressed in *Hertz*.



A typical clock signal

The majority of STM32 MCUs can be clocked by two distinct clock sources alternatively:

- an internal RC oscillator *High Speed Internal (HSI)*; or
- an external dedicated crystal oscillator *High Speed External (HSE)*

There are several reasons to prefer an external crystal to the internal RC oscillator:

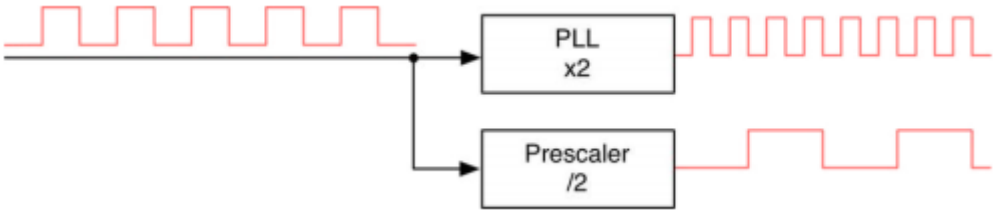
- An external crystal offers a higher precision compared to the internal RC network, which is rated of a 1% accuracy, especially when PCB operative temperatures are far from the ambient temperature of 25°C
- Some peripherals, especially high speed ones, can be clocked only by a dedicated external crystal running at a given frequency

Together with the high-speed oscillator, another clock source can be used to bias the low-speed oscillator, which in turn can be clocked by:

- an external crystal *Low Speed External (LSE)*; or
- the internal dedicated RC oscillator *Low Speed Internal (LSI)*

The low-speed oscillator is used to drive the *Real Time Clock (RTC)* and the *Independent Watchdog (IWDT)* peripheral.

Using several *Programmable Phase-Locked Loops (PLL)* and pre-scalers, it is possible to increase/decrease the source frequency at needs, depending on the requested performances, the maximum speed for a given peripheral or bus and the overall global power consumption.



PLL is used to increased/ decrease clock frequency

2. Clock tree

The clock tree configuration is performed through a dedicated peripheral named *Reset and Clock Control (RCC)*, and it is a process essentially composed by three steps:

1. The high-speed oscillator source is selected (HSI or HSE) and properly configured, if the HSE is used.
2. If need to feed the `SYSCLK` with a frequency higher than the one provided by the high-speed oscillator, then configure the main PLL (which provides the `PLLCLK` signal).
3. The *System Clock Switch (SW)* is configured to choose the system clock source from HSI, HSE, or `PLLCLK`. Then select the AHB, APB1 and APB2 (if available) pre-scaler settings to reach the wanted frequency of the High-speed clock (`HCLK` - that is the one that feeds the core, DMAs and AHB bus), and the frequencies of Advanced Peripheral Bus 1 (APB1) and APB2 (if available) buses.

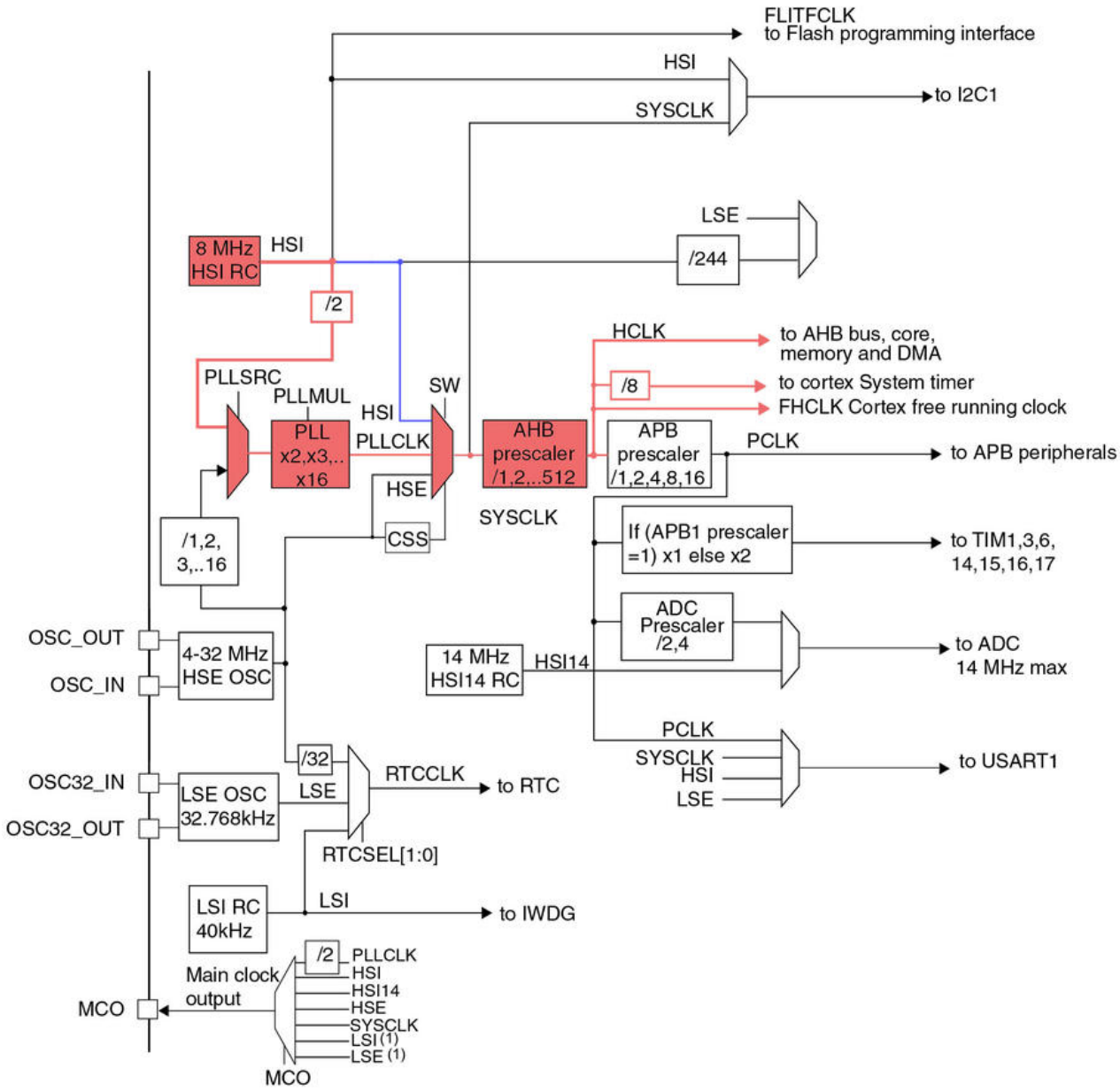
3. Master Clock Output

The RCC peripheral also allows to enable the *Master Clock Output (MCO)*, which is a pin that can be connected to a clock source. It can be used to clock another external device, allowing to save on the external crystal for this other IC. Once the MCO is enabled, it is possible to choose its clock source using the Clock Configuration view.

4. Clock Security System

The *Clock Security System (CSS)* is a feature of the RCC peripheral used to detect malfunctions of the external HSE. The CSS is an important feature in some critical applications, and the detection of a failure is noticed through the NMI exception - a Cortex-M exception that cannot be disabled.

When the failure of HSE is detected, the MCU automatically switch to the HSI clock, which is selected as source for the `SYSCLK` clock. So, if a higher core frequency is needed, it needs to perform proper initializations inside the NMI exception handler.



Example of a clock tree

5. Configure Clocks

After a reset:

- The device is running from the Internal High Speed oscillator (HSI 8 MHz) with Flash 0 wait state, Flash prefetch buffer is enabled, and all peripherals are off except internal SRAM, Flash and JTAG.
- There is no prescaler on High speed (AHB) and Low speed (APB) buses; all peripherals mapped on these buses are running at HSI speed.
- The clock for all peripherals is switched off, except the SRAM and FLASH.

- All GPIOs are in input floating state, except the JTAG pins which are assigned to be used for debug purpose.

Once start up, the user application has to:

- Configure the clock source to be used to drive the System clock (if the application needs higher frequency/ performance)
- Configure the System clock frequency and Flash settings
- Configure the AHB and APB buses prescalers
- Enable the clock for the peripheral(s) to be used
- Configure the clock source(s) for peripherals whose clocks are not derived from the System clock (RTC, ADC, I2C, USART, TIM, USB FS, etc.)

6. CubeMX Usage

Here are two examples of setting up the clock on 2 boards - one with only internal oscillator, one with external crystals. Note that when changing the clock speed, there are some limitation of the frequency of the target buses, e.g. `HCLK`, `PCK1`, `PCK2`.

For both `HSE` and `LSE` external crystals, CubeMX offers three configuration options:

- **Disable**: the external oscillator is not available/used, and the corresponding internal oscillator is used.
- **Crystal/Ceramic Resonator**: an external crystal/ceramic resonator is used and the corresponding main frequency is derived from it. This implies that `RCC_OSC_IN` and `RCC_OSC_OUT` pins are used to interface the HSE, and the corresponding signal I/Os are unavailable for other usages (if use an external low-speed crystal, then the corresponding `RCC_OSC32_IN` and `RCC_OSC32_OUT` I/Os are used too).
- **BYPASS Clock Source**: an external clock source is used. The clock source is generated by another active device. This means that the `RCC_OSC_OUT` is leaved unused, and it is possible to use it as regular GPIO.

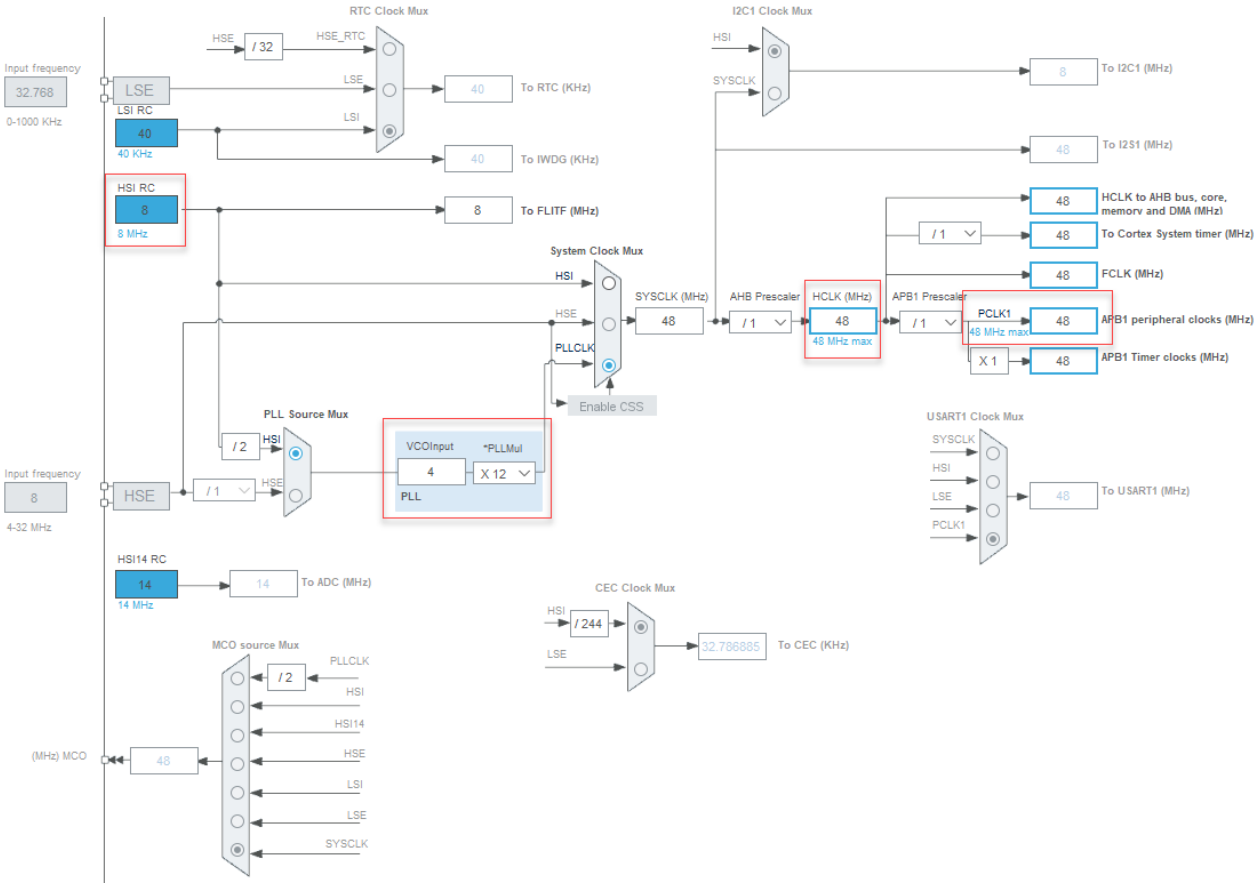
✏️ User can fill in a number in a frequency node, and press Enter to let IDE automatically calculate the multiplier's and divider's value.

6.1. Internal RC oscillators

This is an example guide on an official STM32F0 Discovery board, which only has internal RC oscillators:

1. Select PLL Source Mux is **HSI**
2. Set PLL Multiplier and Divider to get `PLLCLK`

3. Select System Clock Mux is `PLLCLK` to get `SYSCLK`
4. Set divider for AHB Bus `HCLK` and APB1 Bus `PCLK1`



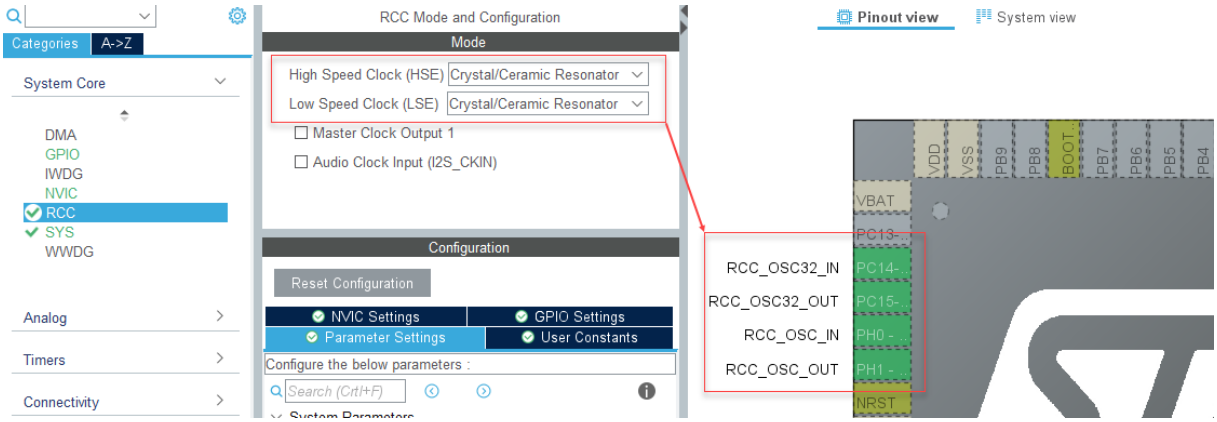
Setup clock with HSI on STM32F051R8

6.2. External crystals

This is an example guide on a custom black pill STM32F411CE board, which has external crystals:

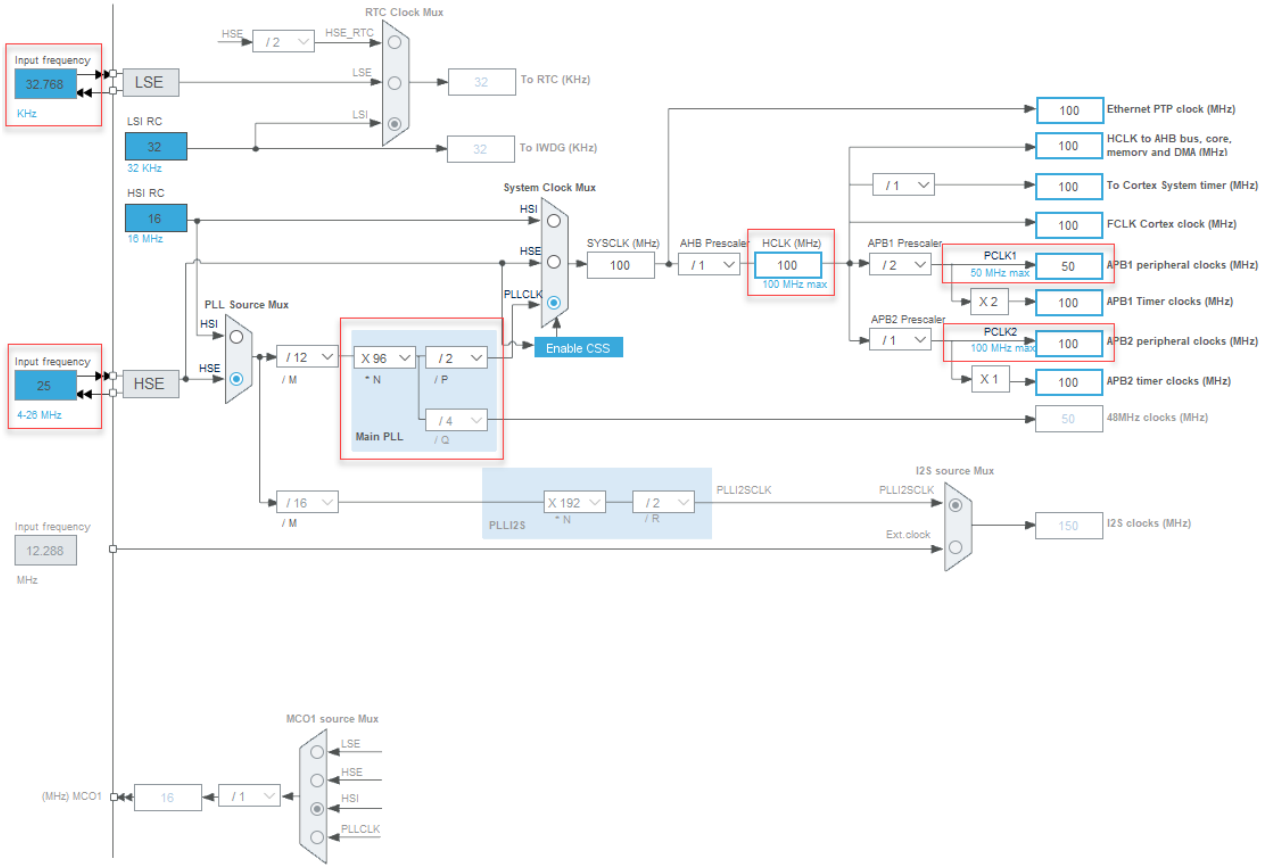
1. Enable **HSE** and **LSE**:

   In the *RCC* peripheral under the *System Core* modules, choose option *Crystal/Ceramic Resonator*. This will automatically setups 2 pins for HSE, and 2 pins for LSE.



Enable HSE and LSE

2. Select PLL Source Mux is **HSE**
3. Set PLL Multiplier and Divider to get `PLLCLK`
4. Select System Clock Mux is `PLLCLK` to get `SYSCLK`
5. Set divider for AHB Bus `HCLK`, APB1 Bus `PCLK1`, APB2 Bus `PCLK2`



Setup clock with HSE on STM32F411CE

7. STM32Cube HAL Usage

CubeMX is designed to generate the right code initialization for the clock tree of MCU. All the necessary code is packed inside the `SystemClock_Config()` routine in the file `main.c`:

- The most relevant C struct to configure the clock tree are `RCC_OscInitTypeDef` and `RCC_ClkInitTypeDef`.
- Use `HAL_RCC_OscConfig()` function to setup the Oscillator source, and then use `HAL_RCC_ClockConfig()` function to setup the System clock (CPU) and Bus clocks (AHB, APB).
- To configure the Master Clock Output, use the function `HAL_RCC_MCOConfig()` to select the clock source and its dividing factor. Note that when configuring the MCO pin as output GPIO, its speed (that is, the slew rate) affects the quality of the output clock.

Example 1 - Setting clock on F051R8, with 8 MHz HSI, to make 48 MHZ SYSCLK, and drive MCO at 48 MHz using SYSCLK:

```
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Initializes the RCC Oscillators according to the specified parameters
      in the RCC_OscInitTypeDef structure.*/
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL12;
  RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks */
  RCC_ClkInitStruct.ClockType =
    RCC_CLOCKTYPE_HCLK |RCC_CLOCKTYPE_SYSCLK | RCC_CLOCKTYPE_PCLK1;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK) {
    Error_Handler();
  }

  HAL_RCC_MCOConfig(RCC_MCO, RCC_MCO1SOURCE_SYSCLK, RCC_MCODIV_1);
}
```

Example 2 - Setting clock on F411CE, with 25 MHZ HSE and 32.765KHz LSE, to make 100 MHz SYSCLK, 50 MHz APB1 Bus, 100 MHZ APB2, with CSS Enabled:

```
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 12;
  RCC_OscInitStruct.PLL.PLLN = 96;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_3) != HAL_OK)
  {
    Error_Handler();
  }
  /** Enables the Clock Security System
  */
  HAL_RCC_EnableCSS();
}
```

Note that when CSS is enabled, `HAL_RCC_NMI_IRQHandler()` is added into the `NMI_Handler()` ISR. User should override the `HAL_RCC_CSSCallback()` function to re-configure the system clocks as the same way as it only has the internal RC oscillators.