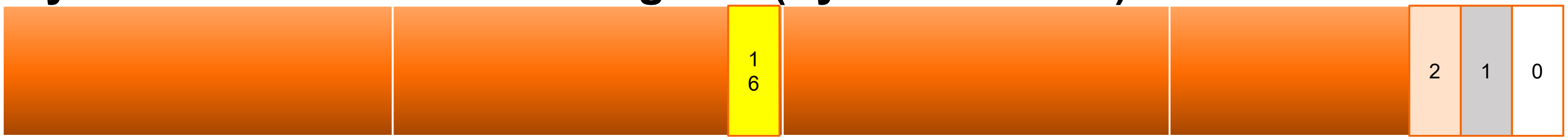
# SysTick

- 24-bit System Timer
- Counts down from the reload value to zero
- The maximum value that can be loaded to the load register of system timer is 2^(24-1)
- Generates SysTick interrupts
- System Timer can be used to execute task periodically such as periodic polling and Scheduler

*References:*
*STM32 Cortex®-M4 MCUs and MPUs programming manual*
*Book: Embedded Systems with ARM CORTEX-M Microcontrollers in Assembly Language and C (Dr. Yifeng Zhu)*

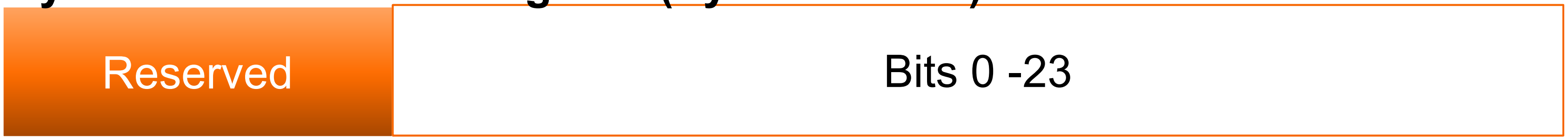
# SysTick Registers

## SysTick control and status register (SysTick->CTRL)



Bit 0 – Enable
Bit 1 – (Tick) Interrupt Enable
Bit 2 – Clock Source
Bit 16 – Count Flag

## SysTick current value register (SysTick->VAL)

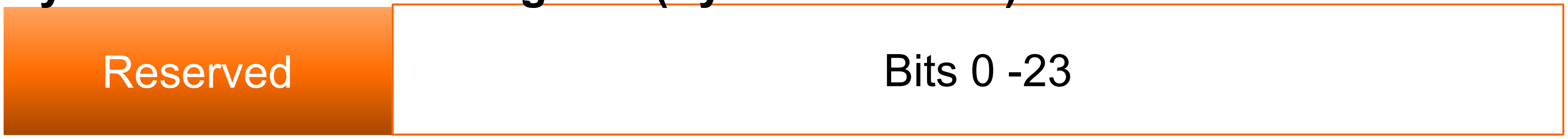


Bit 0-23 – Current Value

*References:*
*STM32 Cortex®-M4 MCUs and MPUs programming manual*

# SysTick Registers

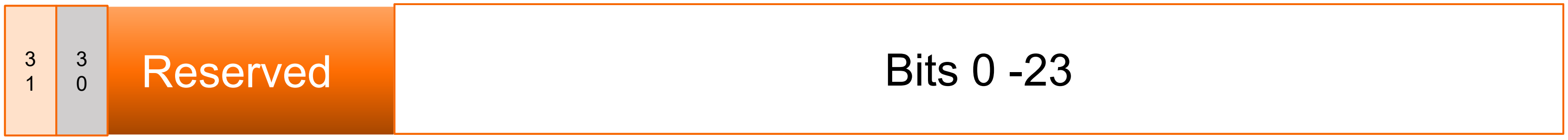
## SysTick Reload value register (SysTick->LOAD)



| Reserved | Bits 0 -23 |
|---|---|

Bit 0-23 – Reload Value

## SysTick calibration value register (SysTick> CALIB)



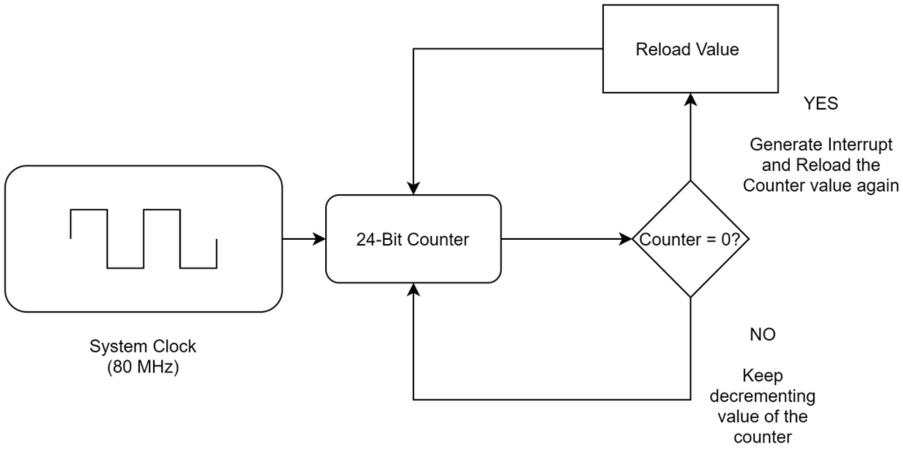
| 31 | 30 | Reserved | Bits 0 -23 |
|---|---|---|---|

Bit 0-23 – Calibration Value
Bit 30 – Skew Flag
Bit 31 – No Reference Flag

***In STM32L4 Processors, External Clock is System Internal Clock divided by 8***

*References:*
*STM32 Cortex®-M4 MCUs and MPUs programming manual*

# How does SysTick work?



Reload Value

YES

Generate Interrupt and Reload the Counter value again

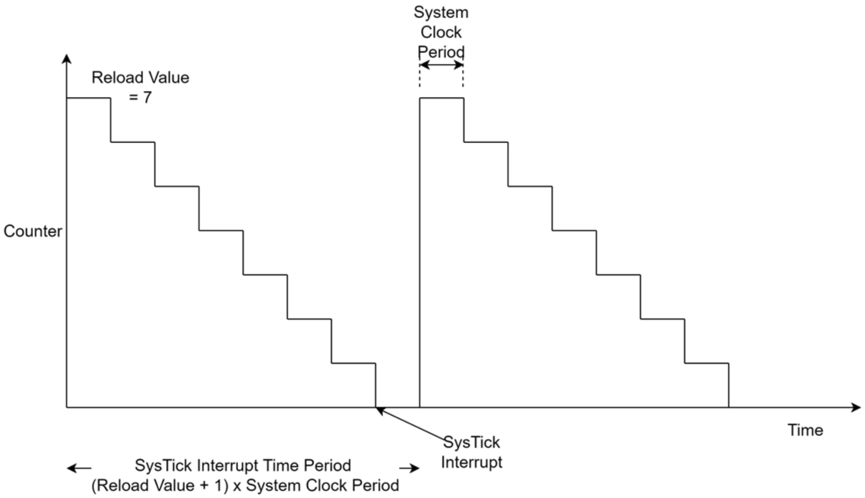
System Clock (80 MHz)

24-Bit Counter

Counter = 0?

NO

Keep decrementing value of the counter

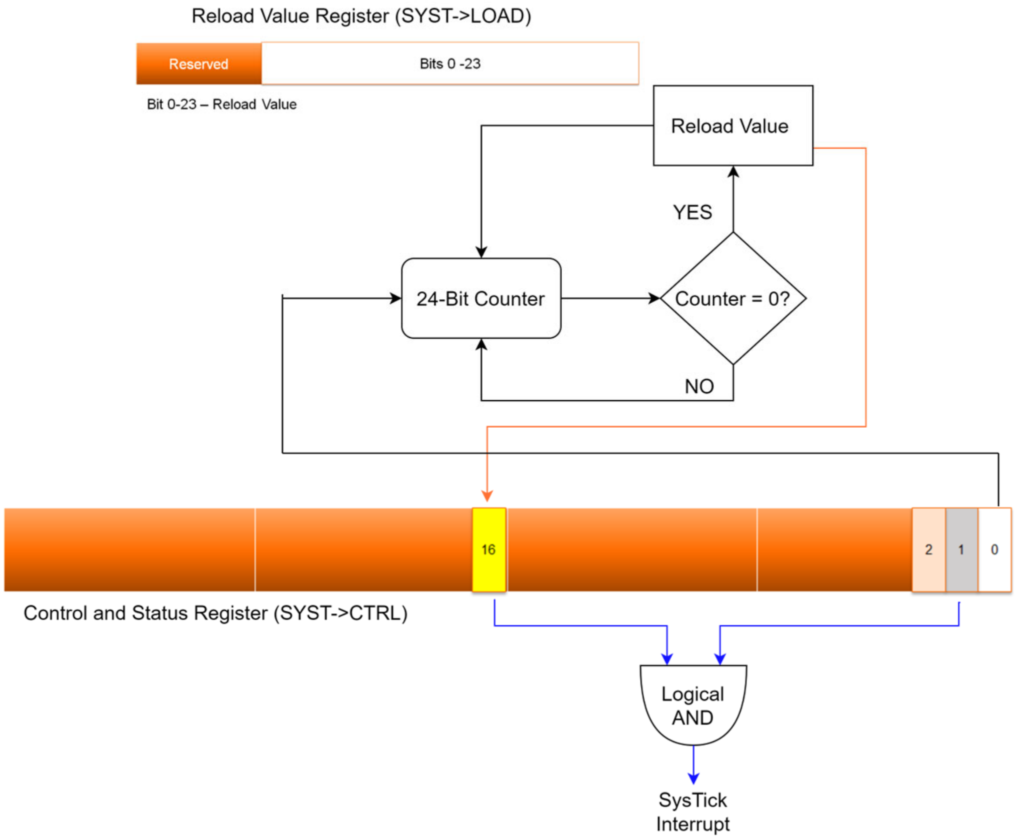


With 80 MHz System clock, if 1 Interrupt is desired at 1 second time interval then reload value must be

Reload Value = (Time Period / System Clock Period) – 1
Reload Value =(1 Second / (1 / 80 MHz) ) – 1
Reload Value = 80,000,000 – 1 = 79,999,999

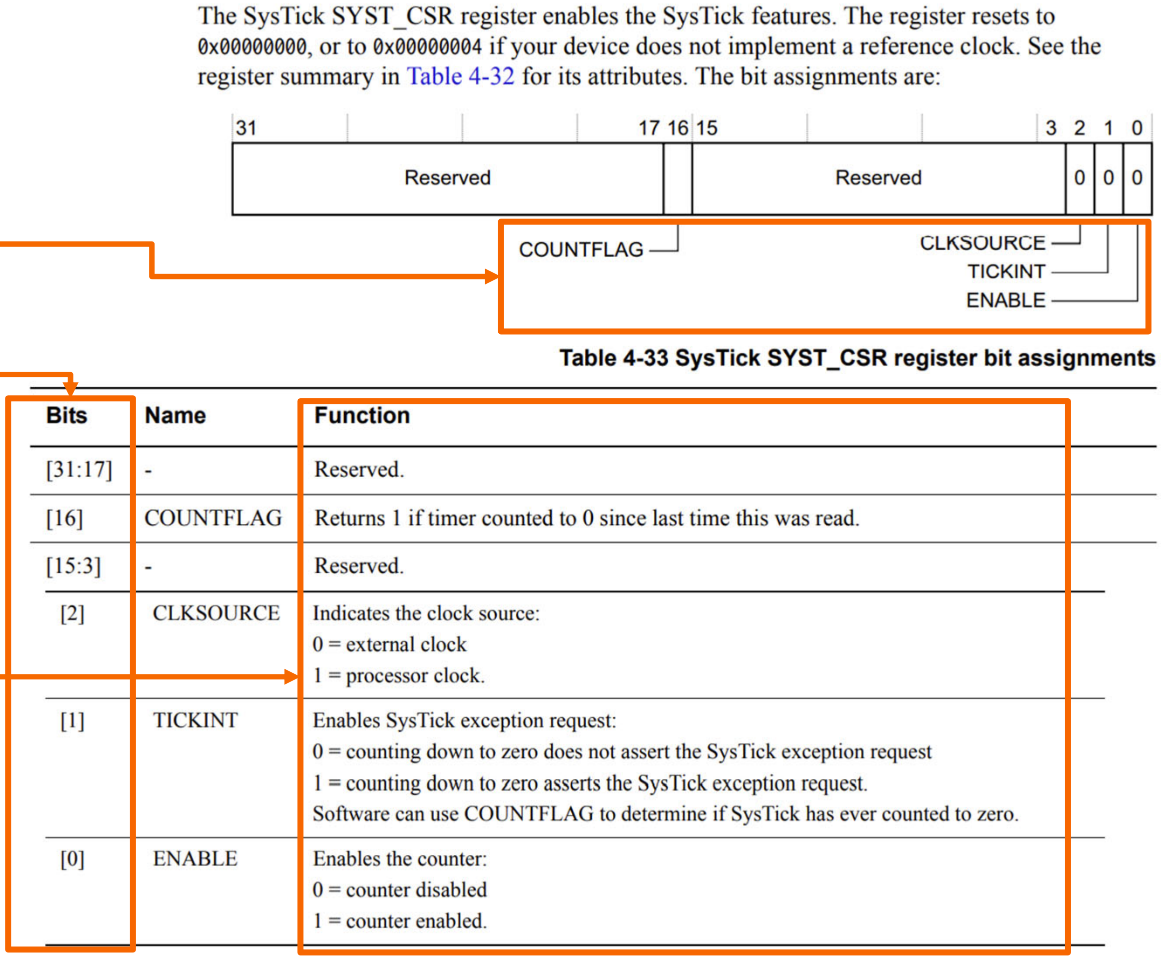
*References:*
*STM32 Cortex®-M4 MCUs and MPUs programming manual*
*Book: Embedded Systems with ARM CORTEX-M Microcontrollers in Assembly Language and C (Dr. Yifeng Zhu)*

# How does SysTick work?



*References:*
*STM32 Cortex®-M4 MCUs and MPUs programming manual*
*Book: Embedded Systems with ARM CORTEX-M Microcontrollers in Assembly Language and C (Dr. Yifeng Zhu)*

# Register Description



The SysTick SYST_CSR register enables the SysTick features. The register resets to `0x00000000`, or to `0x00000004` if your device does not implement a reference clock. See the register summary in Table 4-32 for its attributes. The bit assignments are:

| 31 … 17 | 16 | 15 … 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| Reserved | | Reserved | 0 | 0 | 0 |

COUNTFLAG

CLKSOURCE

TICKINT

ENABLE

Table 4-33 SysTick SYST_CSR register bit assignments

| Bits | Name | Function |
|---|---|---|
| [31:17] | - | Reserved. |
| [16] | COUNTFLAG | Returns 1 if timer counted to 0 since last time this was read. |
| [15:3] | - | Reserved. |
| [2] | CLKSOURCE | Indicates the clock source:<br>0 = external clock<br>1 = processor clock. |
| [1] | TICKINT | Enables SysTick exception request:<br>0 = counting down to zero does not assert the SysTick exception request<br>1 = counting down to zero asserts the SysTick exception request.<br>Software can use COUNTFLAG to determine if SysTick has ever counted to zero. |
| [0] | ENABLE | Enables the counter:<br>0 = counter disabled<br>1 = counter enabled. |

Used bits

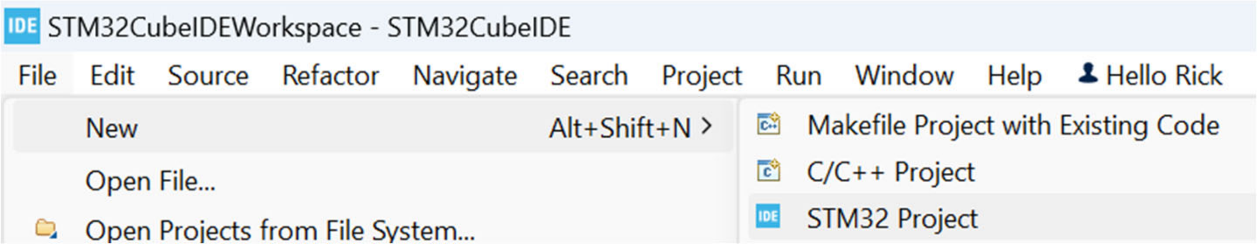
Value size (in bits)

Description of meaning of each bit in a value/function

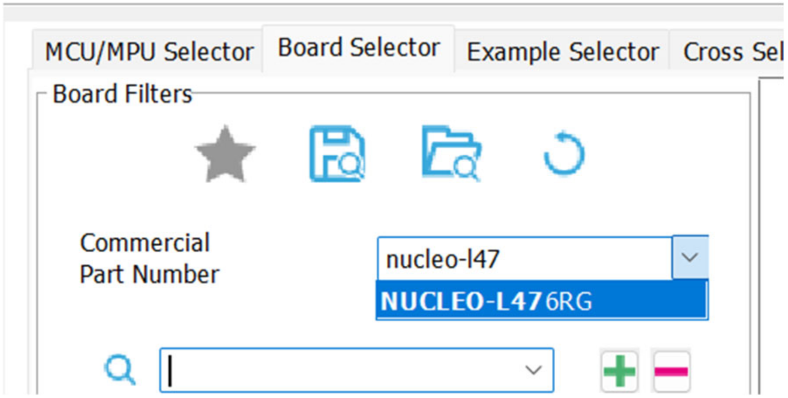
# SysTick Simulation

- **Create a project in Cube IDE**

1) File>New>STM32 Project



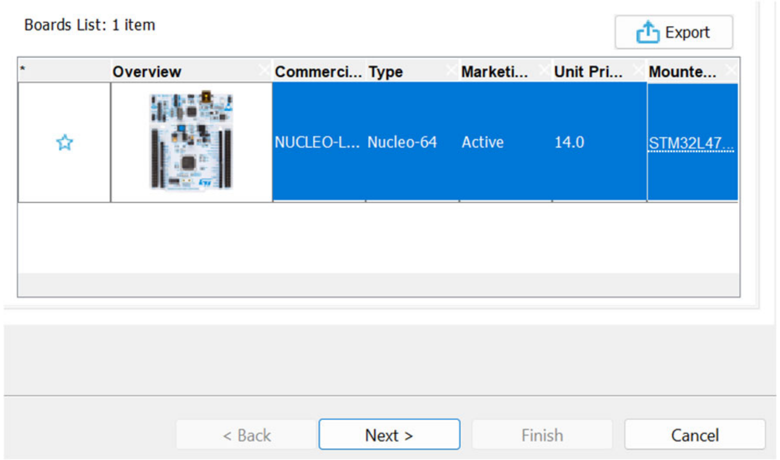
2) Board Selector>Nucleo-L476RG



Commercial Part Number: nucleo-l47

NUCLEO-L476RG

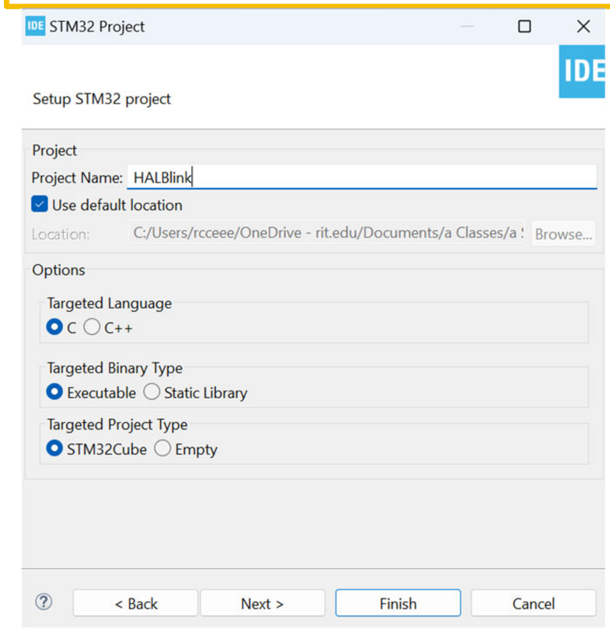
3) Select Nucleo Board>Next



Boards List: 1 item

| * | Overview | Commerci... | Type | Marketi... | Unit Pri... | Mounte... |
|---|---|---|---|---|---|---|
| ☆ | | NUCLEO-L... | Nucleo-64 | Active | 14.0 | STM32L47... |

4) Name the project SimSys and Finish



STM32 Project

Setup STM32 project

Project

Project Name: HALBlink

☑ Use default location

Location: C:/Users/rcceee/OneDrive - rit.edu/Documents/a Classes/a !

Options

Targeted Language

● C ○ C++

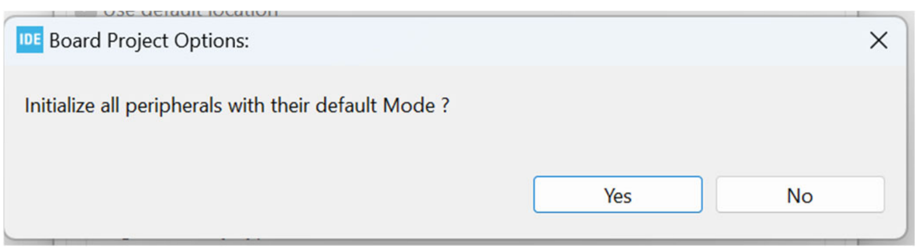
Targeted Binary Type

● Executable ○ Static Library

Targeted Project Type

● STM32Cube ○ Empty

5) YES! To defaults!

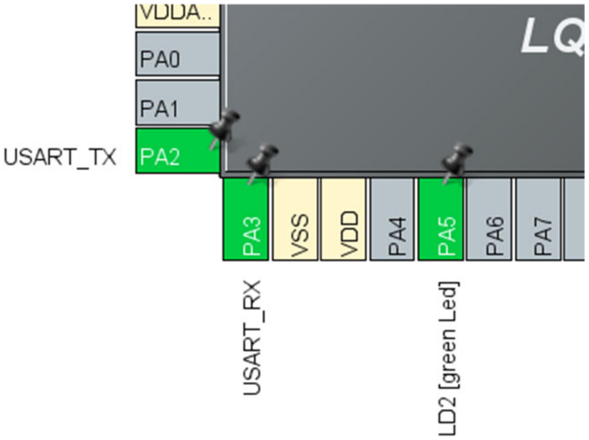


Board Project Options:

Initialize all peripherals with their default Mode ?

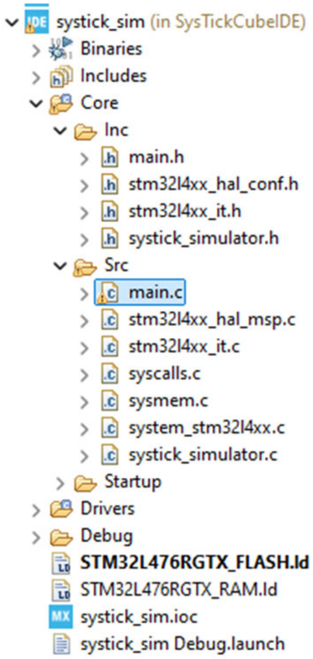
# UART setup for you

ioc – I/O Configuration File

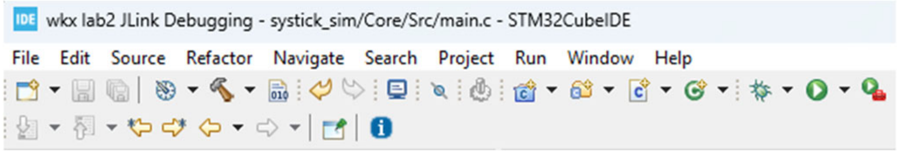


Graphical configuration settings:
1) pin assignments – On Board LED – Port A, Pin 5
2) peripheral configurations – UART – printing to screen
3) much, much more.

1) Download and systick_sim.zip
2) Drag systick_simulator.c into Src directory – make a copy
3) Repeat for main.c
4) Drag systick_simulator.h into Inc directory



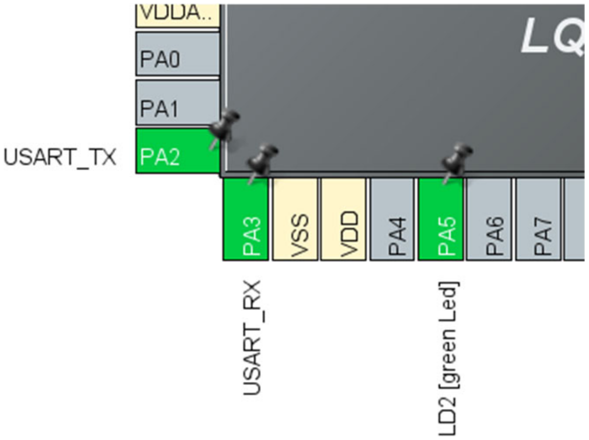
```
systick_sim (in SysTickCubeIDE)
  Binaries
  Includes
  Core
    Inc
      main.h
      stm32l4xx_hal_conf.h
      stm32l4xx_it.h
      systick_simulator.h
    Src
      main.c
      stm32l4xx_hal_msp.c
      stm32l4xx_it.c
      syscalls.c
      sysmem.c
      system_stm32l4xx.c
      systick_simulator.c
    Startup
  Drivers
  Debug
  STM32L476RGTX_FLASH.ld
  STM32L476RGTX_RAM.ld
  systick_sim.ioc
  systick_sim Debug.launch
```


wkx lab2 JLink Debugging - systick_sim/Core/Src/main.c - STM32CubeIDE
File Edit Source Refactor Navigate Search Project Run Window Help

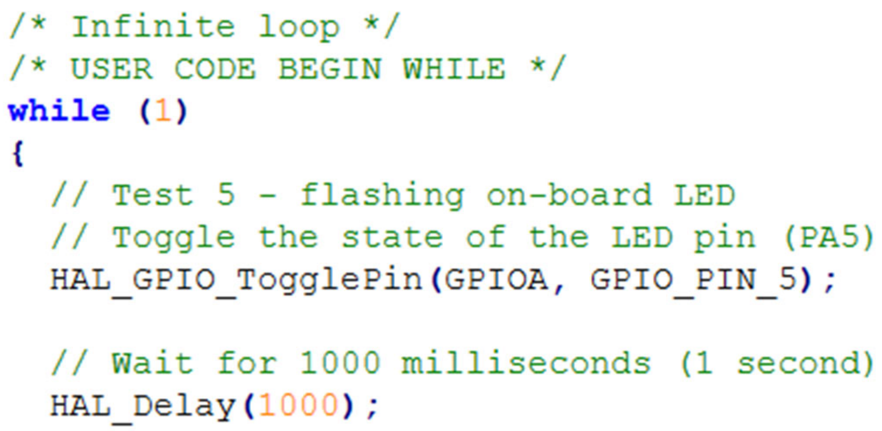
1) Compile and run your project – play button on right side.
2) Hammer just to compile and bug to debug.
3) Open a terminal program and communicate with the board. Options:
    1) Baud rate: 115200.
    2) Built into CubeIDE (Window > Show view > Terminal)
    3) Putty, TeraTerm (Windows)
    4) Screen (MacOS).
4) Compile and run again to see test results.
5) Follow instructions in main.c to simulate the systick hardware.

## ioc – I/O Configuration File

# On Board LED setup for you



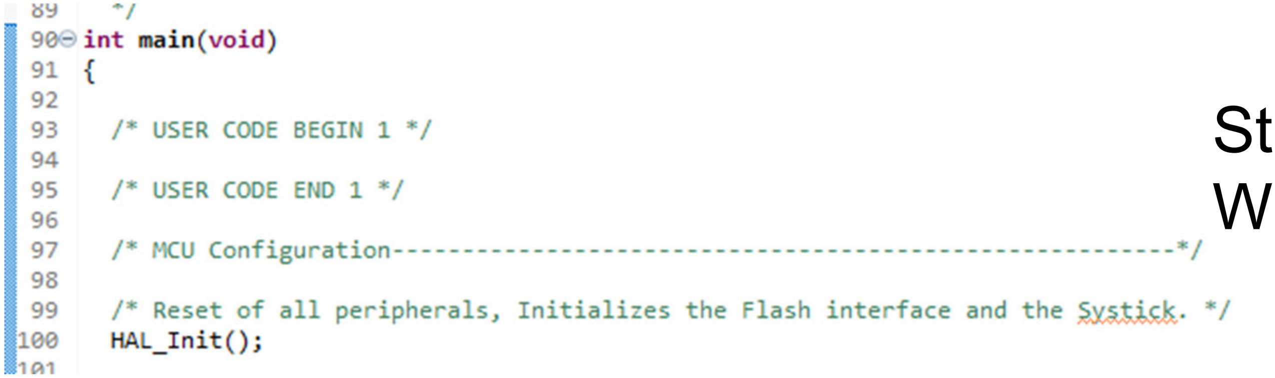
Graphical configuration settings:
1) pin assignments – On Board LED – Port A, Pin 5
2) peripheral configurations – UART – printing to screen
3) much, much more.



```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  // Test 5 - flashing on-board LED
  // Toggle the state of the LED pin (PA5)
  HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

  // Wait for 1000 milliseconds (1 second)
  HAL_Delay(1000);
```

1) Look at the code in the while(1) loop
2) Compile and run your project – play button on right side.
3) Test 5, observe LED flashing on/off every second.
4) Note – user code between comments.
5) Let’s change SysTick and see what happens to HAL_Delay(1000)

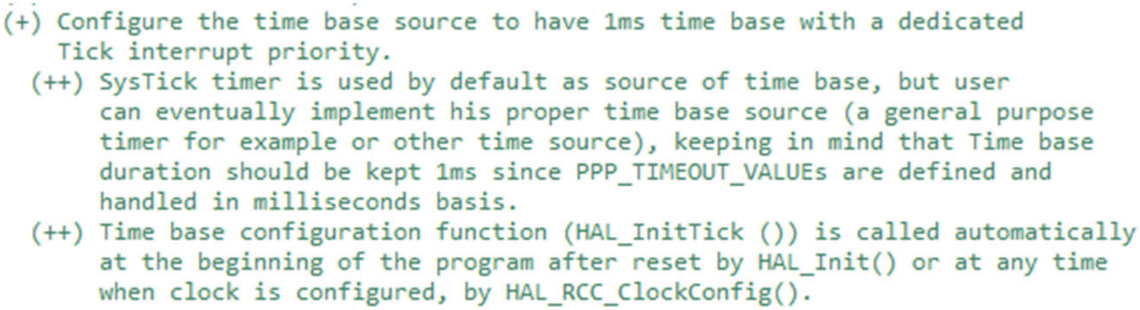
# SysTick – setting of the registers



```
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration----------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();
```

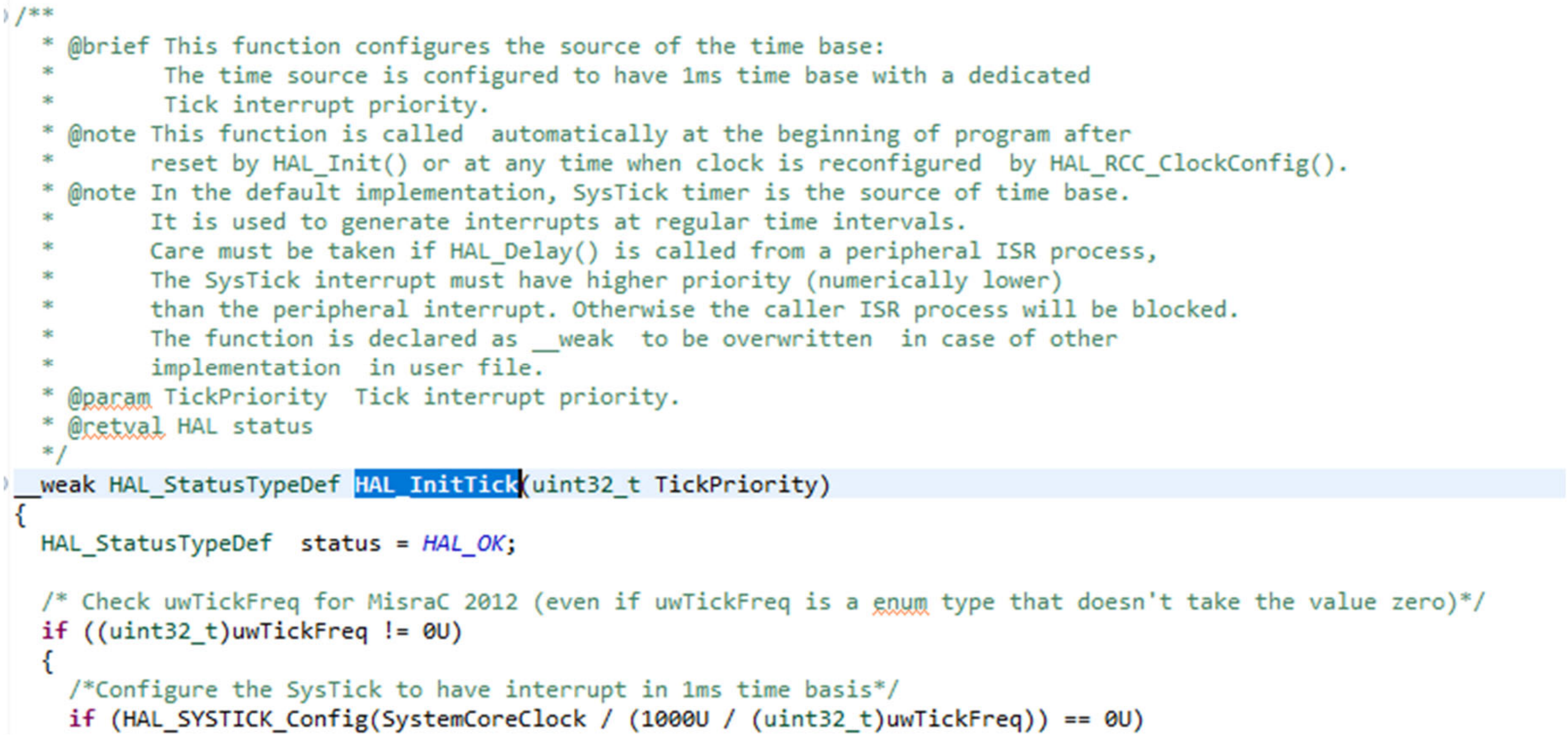
Starts in main,
With HAL_Init()

Select HAL_Init() and press F3 - stm32l4xx_hal.c



```
(+) Configure the time base source to have 1ms time base with a dedicated
    Tick interrupt priority.
  (++) SysTick timer is used by default as source of time base, but user
       can eventually implement his proper time base source (a general purpose
       timer for example or other time source), keeping in mind that Time base
       duration should be kept 1ms since PPP_TIMEOUT_VALUEs are defined and
       handled in milliseconds basis.
  (++) Time base configuration function (HAL_InitTick ()) is called automatically
       at the beginning of the program after reset by HAL_Init() or at any time
       when clock is configured, by HAL_RCC_ClockConfig().
```

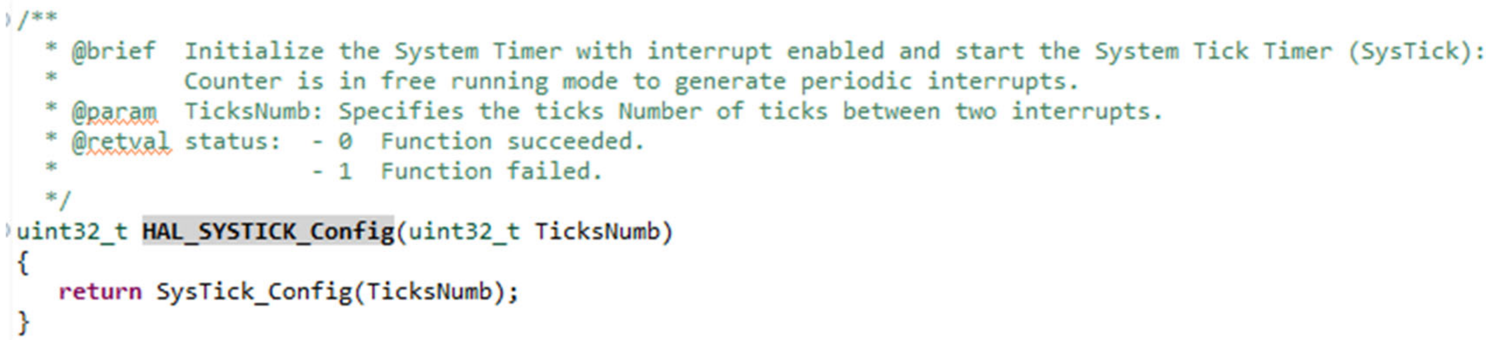
# HAL_InitTick()



```
/**
  * @brief This function configures the source of the time base:
  *        The time source is configured to have 1ms time base with a dedicated
  *        Tick interrupt priority.
  * @note This function is called  automatically at the beginning of program after
  *       reset by HAL_Init() or at any time when clock is reconfigured  by HAL_RCC_ClockConfig().
  * @note In the default implementation, SysTick timer is the source of time base.
  *       It is used to generate interrupts at regular time intervals.
  *       Care must be taken if HAL_Delay() is called from a peripheral ISR process,
  *       The SysTick interrupt must have higher priority (numerically lower)
  *       than the peripheral interrupt. Otherwise the caller ISR process will be blocked.
  *       The function is declared as __weak  to be overwritten  in case of other
  *       implementation  in user file.
  * @param TickPriority  Tick interrupt priority.
  * @retval HAL status
  */
__weak HAL_StatusTypeDef HAL_InitTick(uint32_t TickPriority)
{
  HAL_StatusTypeDef  status = HAL_OK;

  /* Check uwTickFreq for MisraC 2012 (even if uwTickFreq is a enum type that doesn't take the value zero)*/
  if ((uint32_t)uwTickFreq != 0U)
  {
    /*Configure the SysTick to have interrupt in 1ms time basis*/
    if (HAL_SYSTICK_Config(SystemCoreClock / (1000U / (uint32_t)uwTickFreq)) == 0U)
```

Next, HAL_SYSTICK_Config() – press F3 again

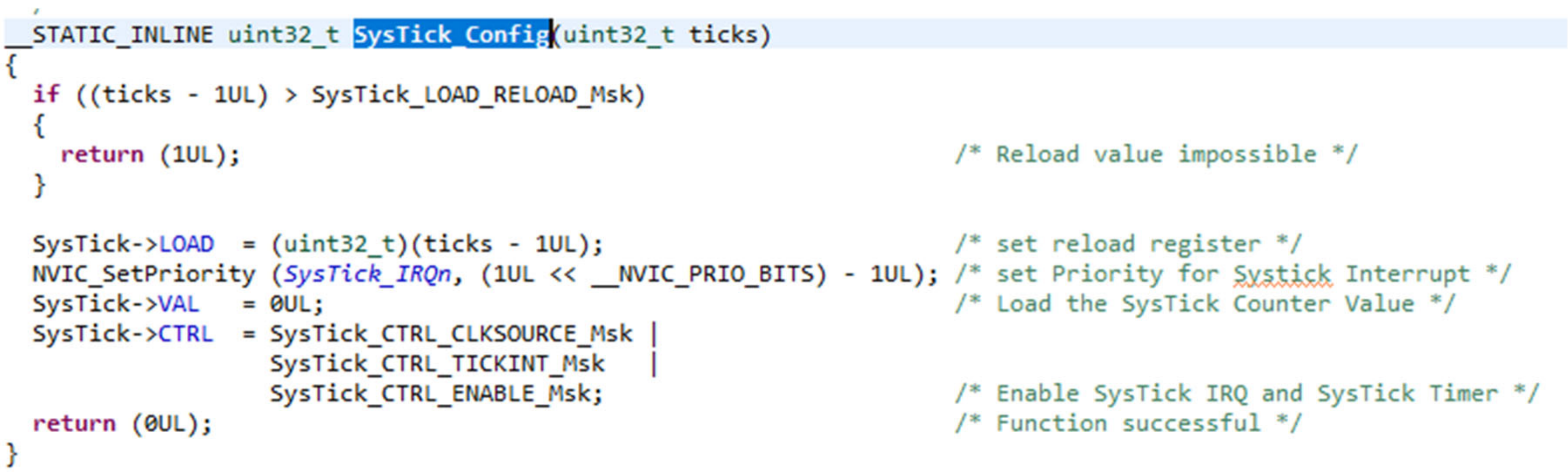
# HAL_SYSTICK_Config() in stm32l4xx_hal_cortex



```
/**
  * @brief  Initialize the System Timer with interrupt enabled and start the System Tick Timer (SysTick):
  *         Counter is in free running mode to generate periodic interrupts.
  * @param  TicksNumb: Specifies the ticks Number of ticks between two interrupts.
  * @retval status:  - 0  Function succeeded.
  *                  - 1  Function failed.
  */
uint32_t HAL_SYSTICK_Config(uint32_t TicksNumb)
{
   return SysTick_Config(TicksNumb);
}
```

Next, SysTick_Config() – press F3 again

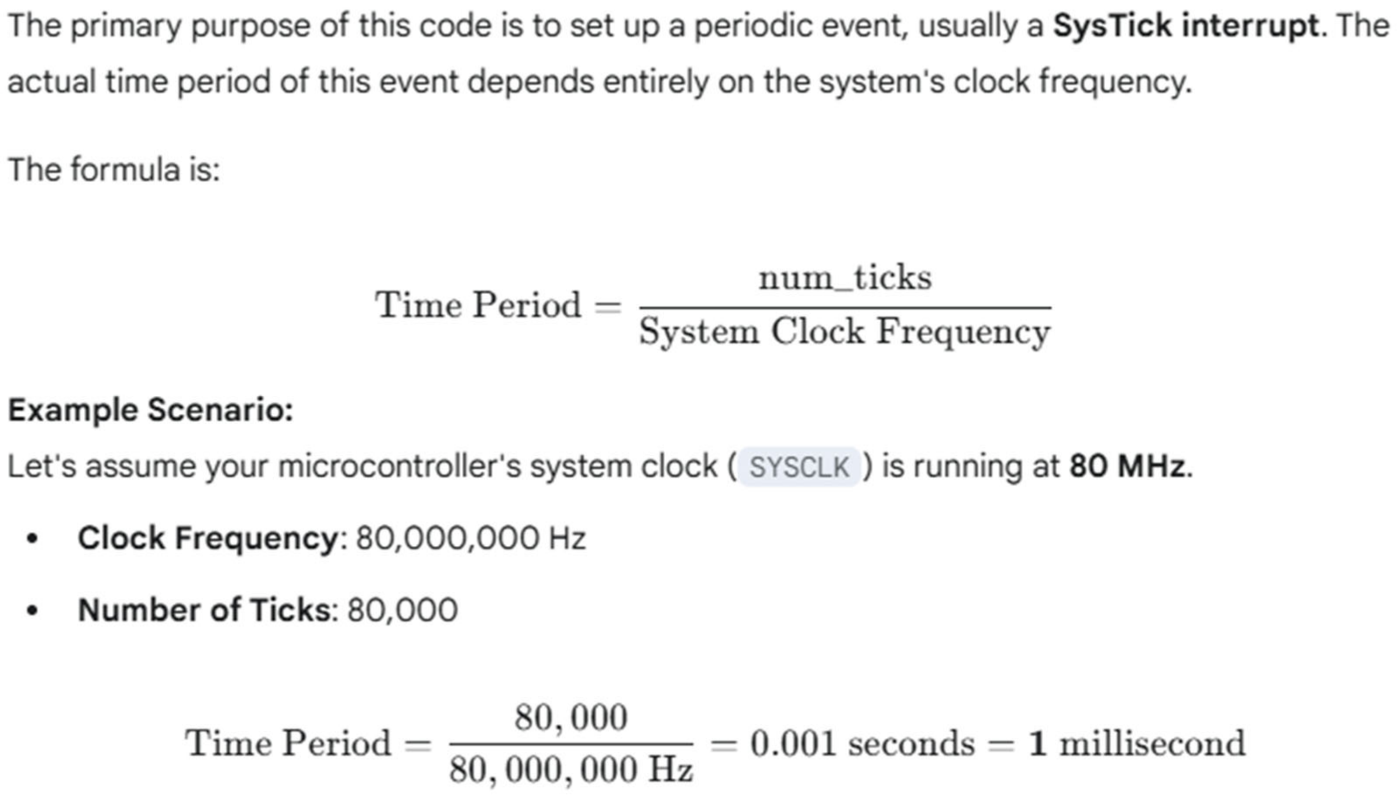
# Core_cm4.h



```
__STATIC_INLINE uint32_t SysTick_Config(uint32_t ticks)
{
  if ((ticks - 1UL) > SysTick_LOAD_RELOAD_Msk)
  {
    return (1UL);                                                   /* Reload value impossible */
  }

  SysTick->LOAD  = (uint32_t)(ticks - 1UL);                         /* set reload register */
  NVIC_SetPriority (SysTick_IRQn, (1UL << __NVIC_PRIO_BITS) - 1UL); /* set Priority for Systick Interrupt */
  SysTick->VAL   = 0UL;                                             /* Load the SysTick Counter Value */
  SysTick->CTRL  = SysTick_CTRL_CLKSOURCE_Msk |
                   SysTick_CTRL_TICKINT_Msk   |
                   SysTick_CTRL_ENABLE_Msk;                         /* Enable SysTick IRQ and SysTick Timer */
  return (0UL);                                                     /* Function successful */
}
```

Test 6 in main.c. Create a variable like ticks.
Set it equal to something other than 80000.
Use the SysTick->LOAD to set the reload register.
What happens?

# Summary



The primary purpose of this code is to set up a periodic event, usually a **SysTick interrupt**. The actual time period of this event depends entirely on the system's clock frequency.

The formula is:

$$\text{Time Period} = \frac{\text{num\_ticks}}{\text{System Clock Frequency}}$$

**Example Scenario:**

Let's assume your microcontroller's system clock ( `SYSCLK` ) is running at **80 MHz**.

- **Clock Frequency**: 80,000,000 Hz
- **Number of Ticks**: 80,000

$$\text{Time Period} = \frac{80,000}{80,000,000\ \text{Hz}} = 0.001\ \text{seconds} = \mathbf{1}\ \text{millisecond}$$