# **Lab 3:** System Timer Interrupts

**Instructor:**
Dr. Carl Latino
carl.latino@okstate.edu

**Graduate Teaching Assistant:**
Francisco E. Fernandes Jr.
feferna@okstate.edu

**School of Electrical and Computer Engineering**
**Oklahoma State University**

Fall 2019

# Schedule and Grading

- All labs are now **ONE WEEK ONLY**:
  - **You MUST demo your working LAB 3 on September 30, 2019 by the end of your lab section!**

- **Grading for lab 3:**
  - **Pre-lab quiz:** 2 points (due on Sept. 30). **Available on Canvas! READ CHAPTER 11!**
  - **Functionality and Correctness:** 8 points (No partial credits).
  - **Total: 10 points.**

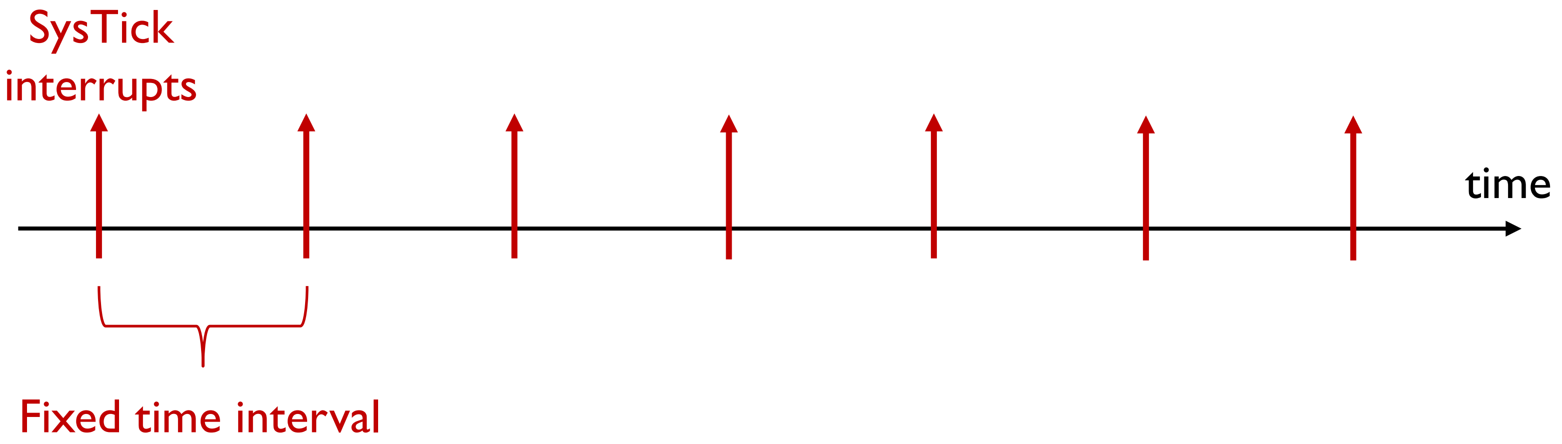
- **Grading penalization:**
  - Students who disrupt the lecture by talking and not paying attention will lose 2 points in their lab 3's grade!
  - Students who do not follow the lab safety procedures (e.g. coming to lab with shorts and flip flops) will lose 1 points in their lab 3's grade!

# Lab Assignment

- **Write an Assembly program that uses the System Timer Interrupts to toggle both the red and green LEDs in the development kit with a period of 0.5 seconds.**

# System Timer (SysTick)

- Generates **SysTick interrupts** at a fixed time interval.

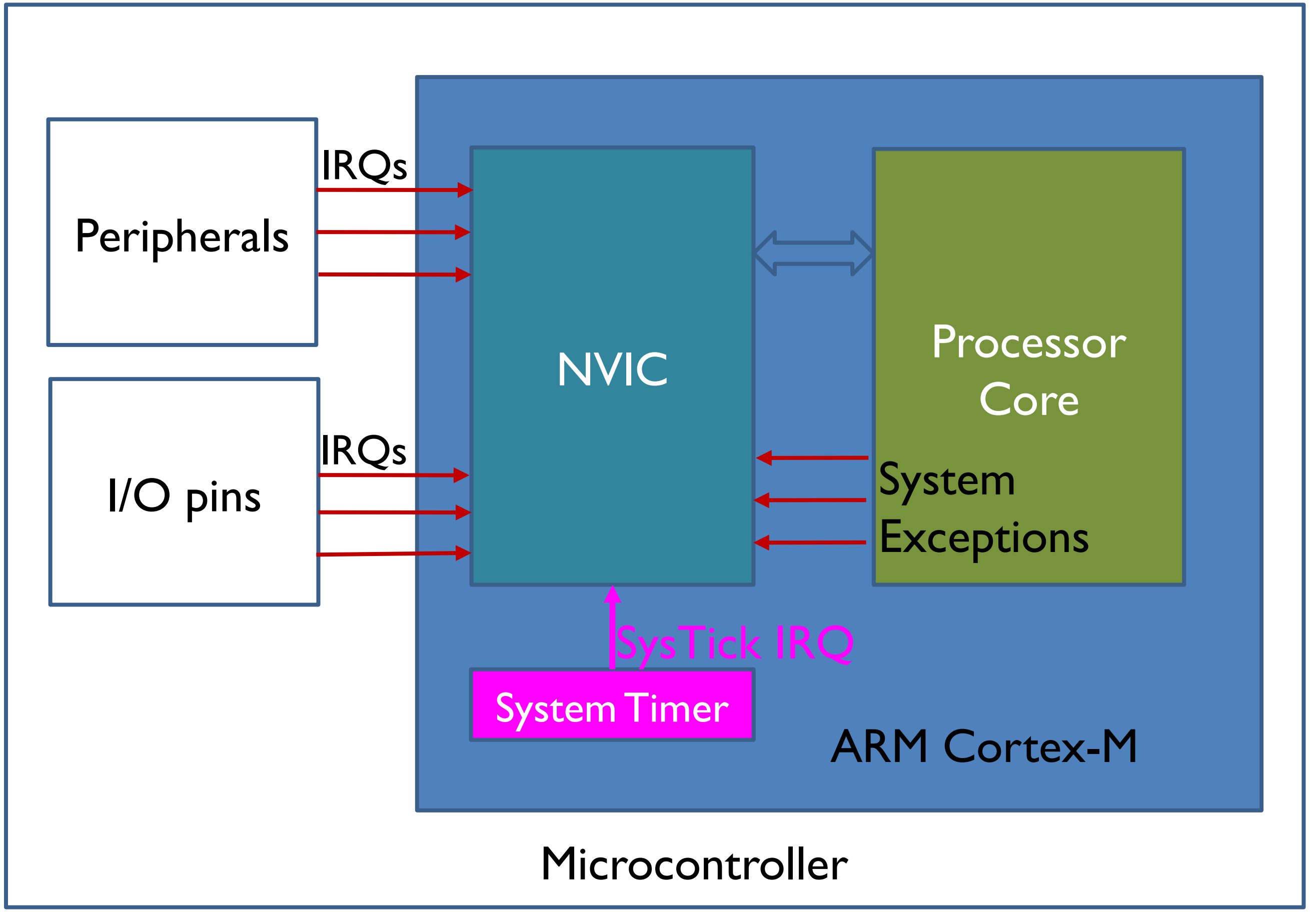


- **Example Usages:**
  - Measuring time elapsed, such as time delay function
  - Executing tasks periodically, such as periodic polling, and OS CPU scheduling

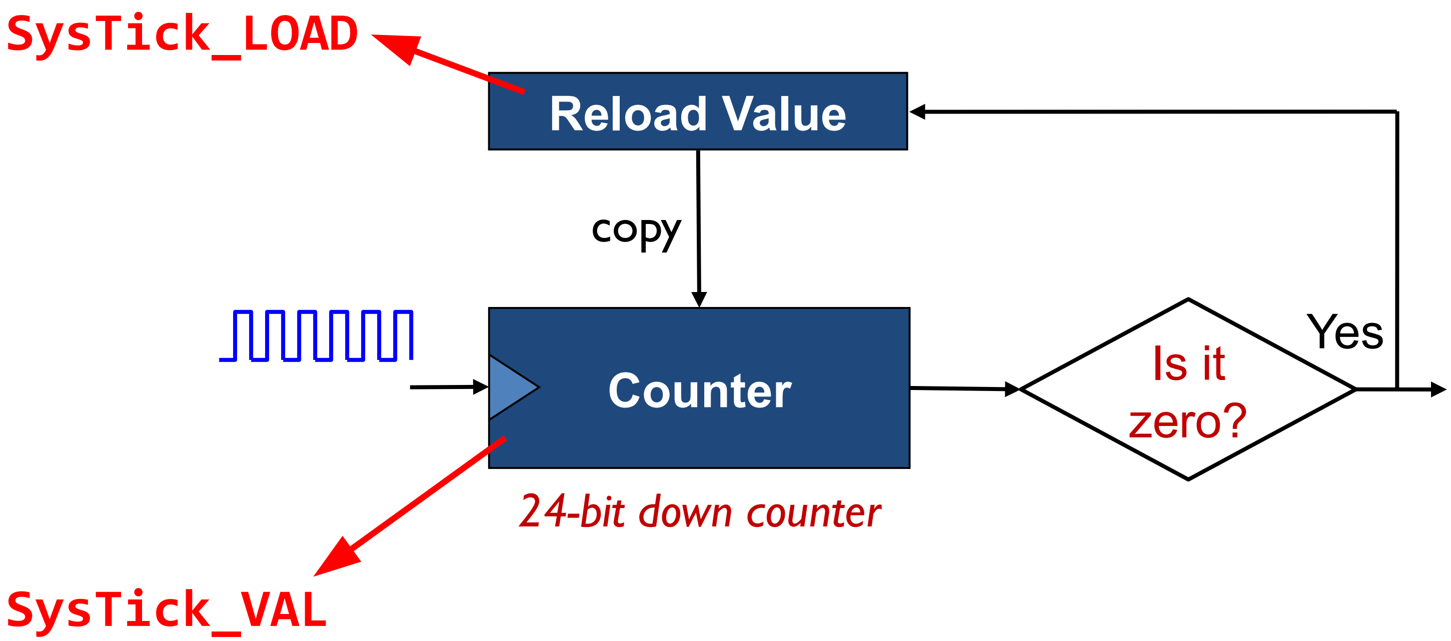
# System Timer (SysTick)

- System timer is a **standard** hardware component built into ARM Cortex-M.
- This hardware **periodically** forces the processor to execute the following **Interupt Subroutine**:

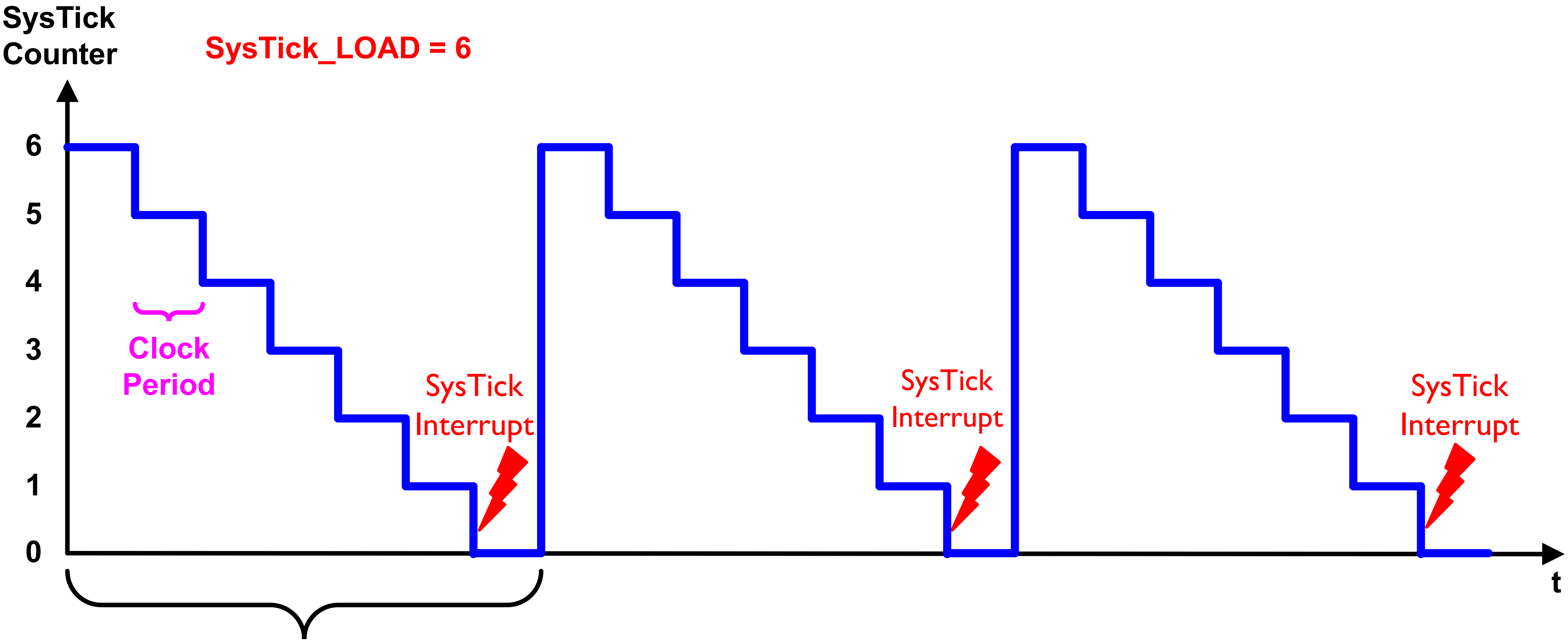
```
.type SysTick_Handler, %function
SysTick_Handler:
   // Do Something
   BX LR
```



# Diagram of System Timer (SysTick)

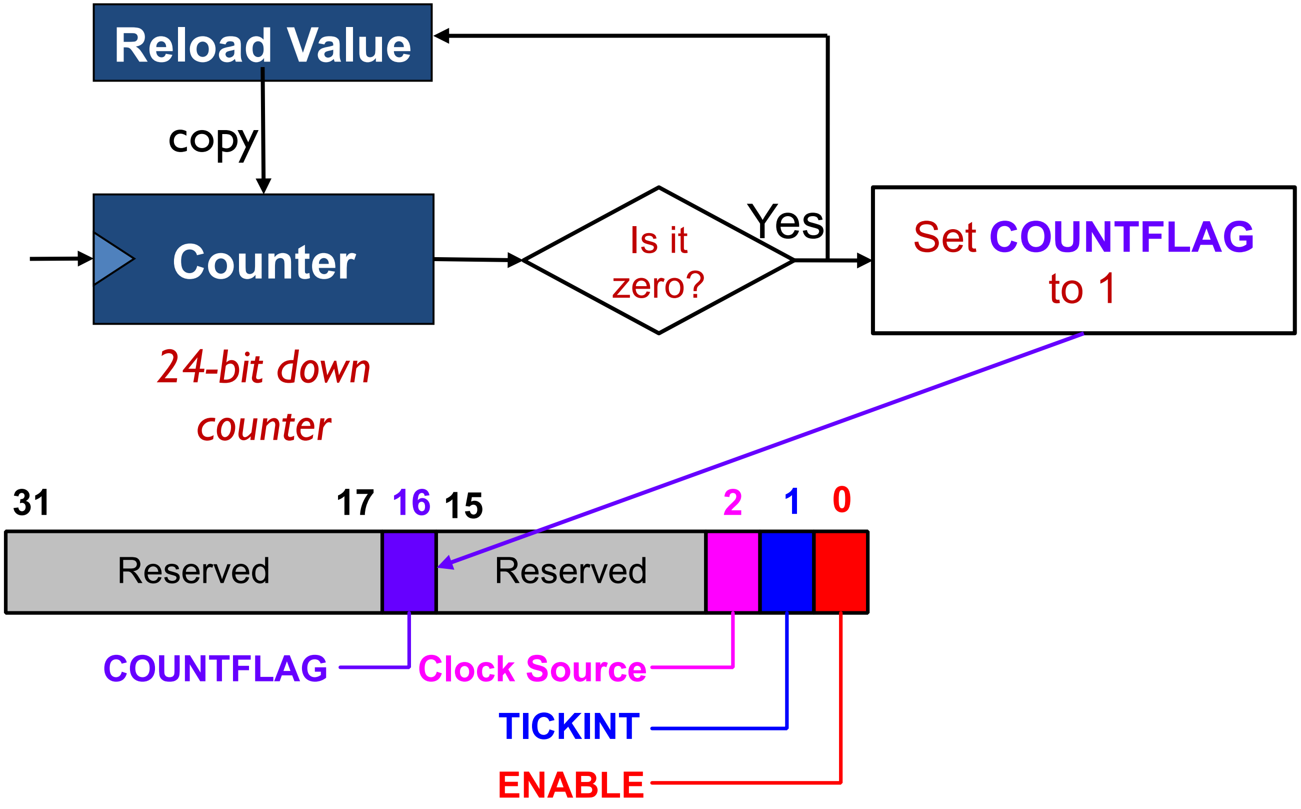


# System Timer



**SysTick Interrupt Time Period = (SysTick_LOAD + 1) × Clock Period = 7 × Clock Period**

# Diagram of System Timer (SysTick)



Reload Value

copy

Counter

*24-bit down counter*

Is it zero?

Yes

Set **COUNTFLAG** to 1

SysTick control and status register (SysTick_CTRL)

31

17

16

15

2

1

0

Reserved

Reserved

**COUNTFLAG**

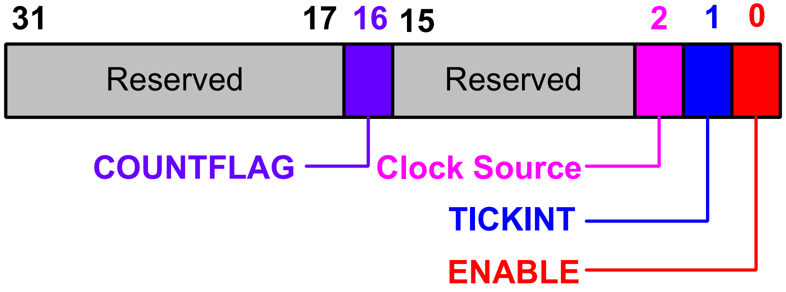
**Clock Source**

**TICKINT**

**ENABLE**

# System Timer Registers

## SysTick control and status register (SysTick_CTRL)
Memory Location 0xE000E010



31 17 16 15 2 1 0

Reserved | 16 | Reserved | 2 | 1 | 0

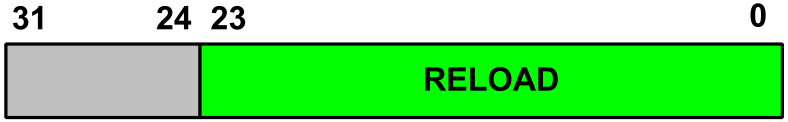
COUNTFLAG

Clock Source

TICKINT

ENABLE

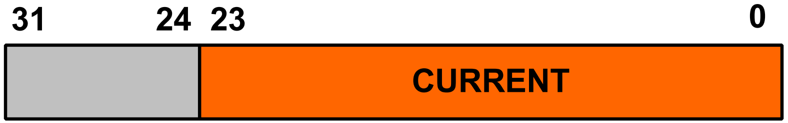
## SysTick reload value register (SysTick_LOAD)
Memory Location 0xE000E014



31 24 23 0

RELOAD

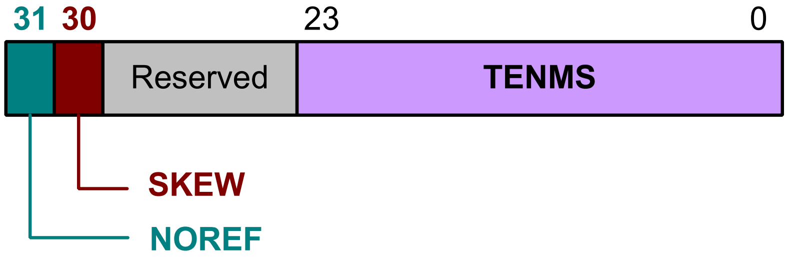
## SysTick current value register (SysTick_VAL)
Memory Location 0xE000E018



31 24 23 0

CURRENT

## SysTick calibration register (SysTick_CALIB)
Memory Location 0xE000E01C



31 30 23 0

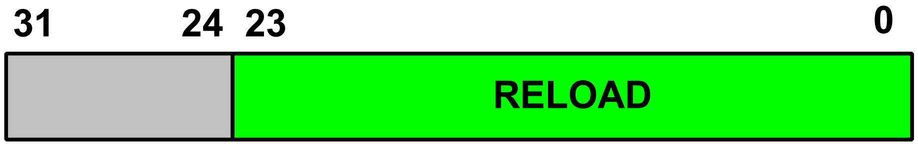
Reserved | TENMS

SKEW

NOREF

# Registers of System Timer

SysTick reload value register (SysTick_LOAD)



- 24 bits, maximum value **`0x00FFFFFF`** (16,777,215)
- Counter counts down from `RELOAD` value to 0.
- Writing `RELOAD` to 0 disables `SysTick`, independently of `TICKINT`
- Time interval between two SysTick interrupts

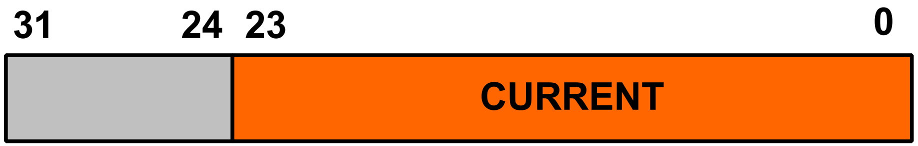
**`Interval = (RELOAD + 1) × Source_Clock_Period`**

- If 100 clock periods between two SysTick interrupts

**`RELOAD = 99`**

# Registers of System Timer

SysTick current value register (SysTick_VAL)



- Reading it returns the current value of the counter
- When it transits from 1 to 0, it generates an interrupt
- Writing to `SysTick_VAL` clears the counter and `COUNTFLAG` to zero
  - Cause the counter to reload on the next timer clock
  - But, does not trigger an `SysTick` interrupt
- It has random value on reset.
  - Always clear it before enabling the timer

# Calculating Reload Value

- Suppose clock source = `80MHz`
- **Goal:** SysTick Interval = `10ms`
- **What is RELOAD value?**

$$Reload = \frac{10\ ms}{Clock\ Period} - 1$$

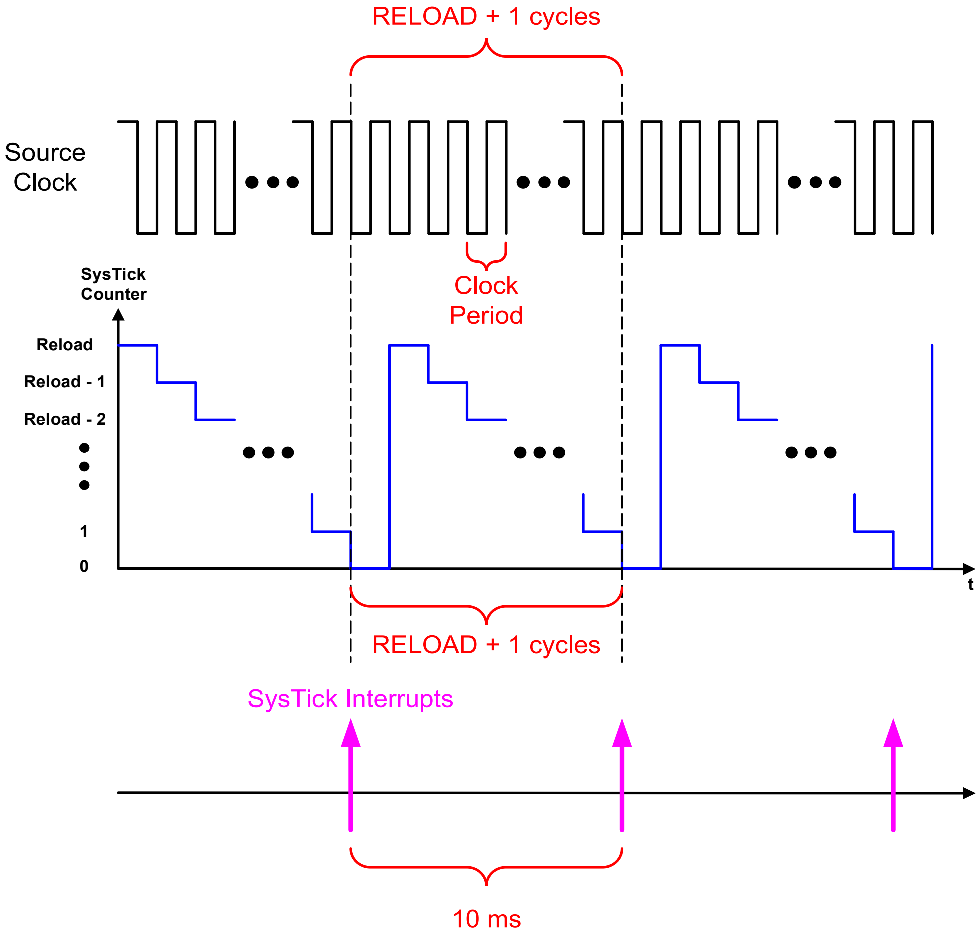
$$= 10ms \times Clock\ Frequency\ - 1$$

$$= 10ms \times 80MHz - 1$$

$$= 10 \times 10^{-3} \times 80 \times 10^{6} - 1$$

$$= 800000 - 1$$

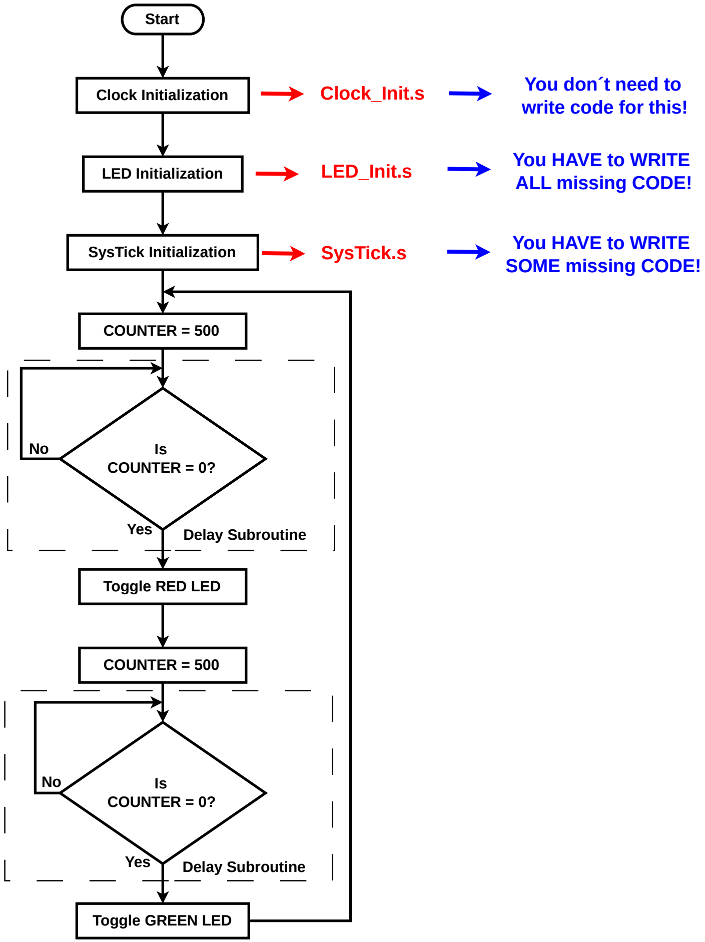
$$= 799999$$



# Lab 3: Startup Code

- A startup code in a zip-file (filename: ***Lab 3 – Startup Code.zip***) is available on ***Canvas***. It contains the following files:
  - ***main.s***
  - ***Clock_Init.s*** → You don't need to change anything in this file!
  - ***LED_Init.s***
  - ***SysTick.s***
  - ***stm32l476xx_constants.s*** → You don't need to change anything in this file!
- **Download** and **EXTRACT** the startup code.
- Create a new project from scratch using the STM32CubeIDE.
- Move **ALL** files to you project's ***src*** folder, and follow the standard steps when we create new projects.

# Lab 3: Flowchart



Start

Clock Initialization → Clock_Init.s → You don´t need to write code for this!

LED Initialization → LED_Init.s → You HAVE to WRITE ALL missing CODE!

SysTick Initialization → SysTick.s → You HAVE to WRITE SOME missing CODE!

COUNTER = 500

Is COUNTER = 0? No / Yes

Delay Subroutine

Toggle RED LED

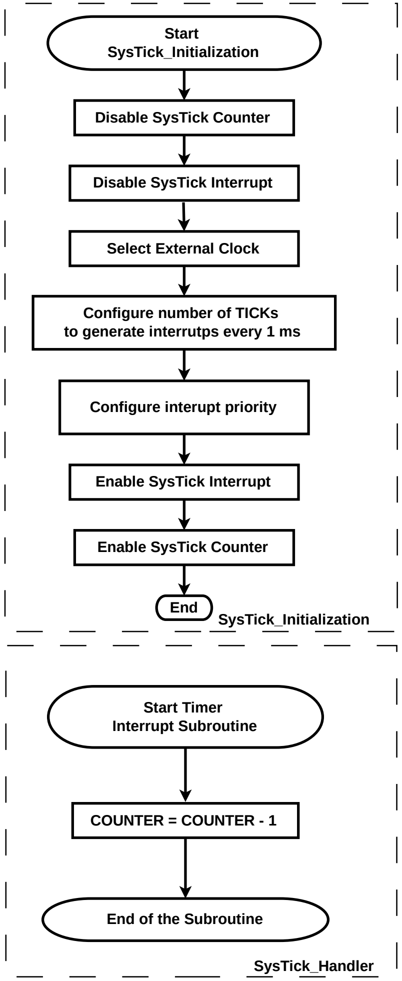
COUNTER = 500

Is COUNTER = 0? No / Yes

Delay Subroutine

Toggle GREEN LED

main.s



Start SysTick_Initialization

Disable SysTick Counter

Disable SysTick Interrupt

Select External Clock

Configure number of TICKs to generate interrutps every 1 ms

Configure interupt priority

Enable SysTick Interrupt

Enable SysTick Counter

End

SysTick_Initialization

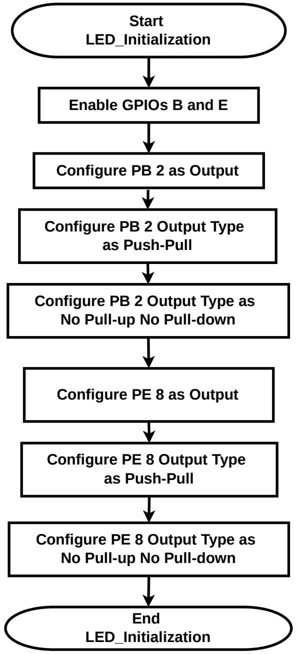
Start Timer Interrupt Subroutine

COUNTER = COUNTER - 1

End of the Subroutine

SysTick_Handler

SysTick.s



Start LED_Initialization

Enable GPIOs B and E

Configure PB 2 as Output

Configure PB 2 Output Type as Push-Pull

Configure PB 2 Output Type as No Pull-up No Pull-down

Configure PE 8 as Output

Configure PE 8 Output Type as Push-Pull

Configure PE 8 Output Type as No Pull-up No Pull-down

End LED_Initialization

LED_Init.s