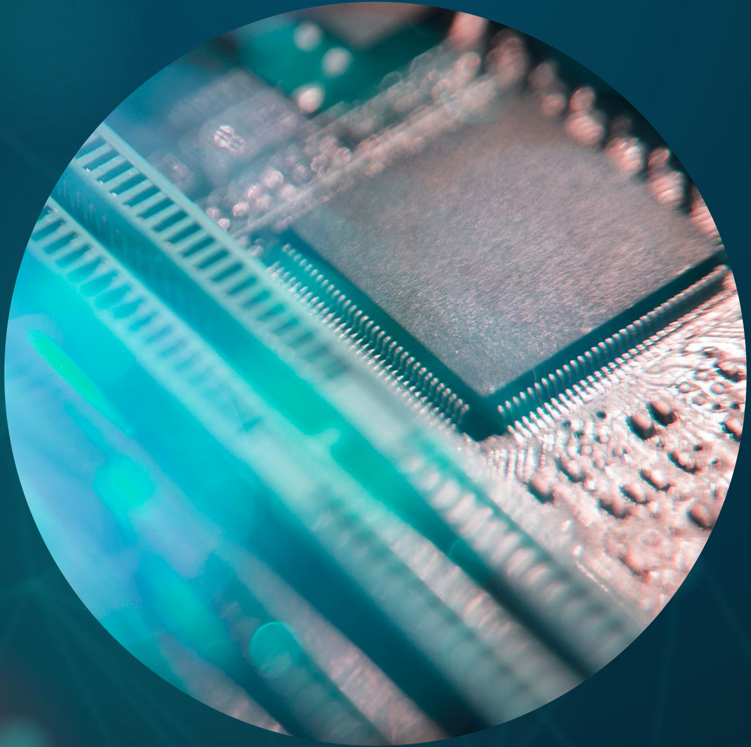
IMT SCHOOL

ARM Based Microcontroller

SYSTICK

Lecture 6



Advanced RISC Machines

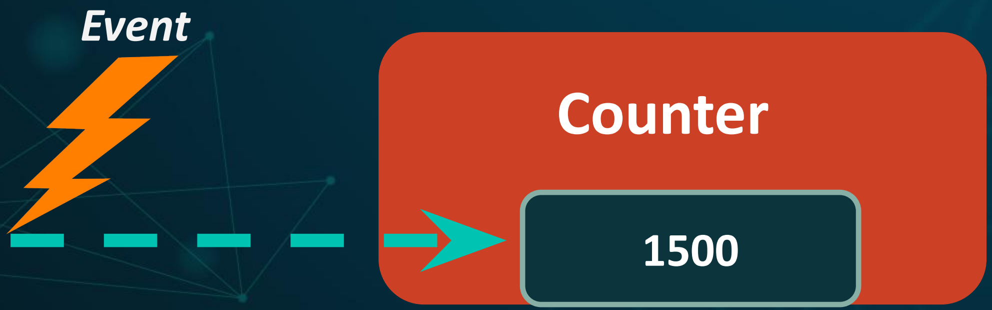
Introduction To Timers

01

System Timer

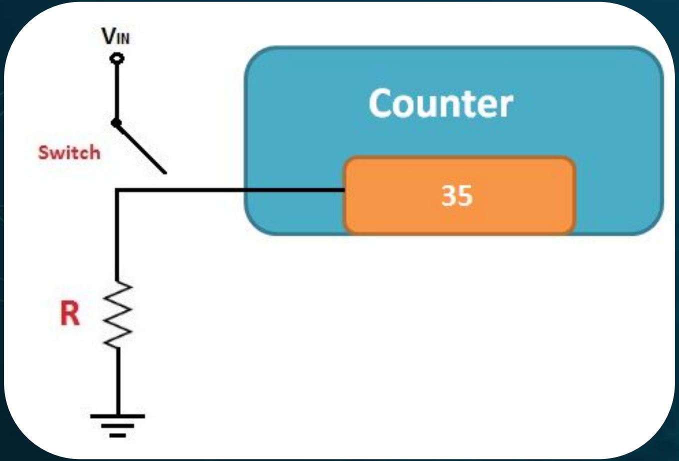
What is a Counter ?

Simply **counter** is a peripheral that counts events in a specific register. These events are electrical signals like rising edge or falling edge. If these events are periodic (Comes every defined period) then this counter is counting time, hence it is called Timer.



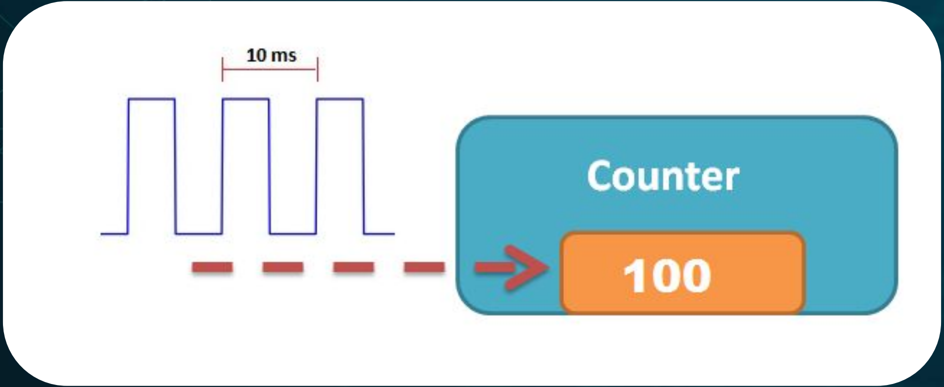
Example

Think of a peripheral that counts a switch presses, The Counter is connected to one terminal of the switch which gives a rising edge every press. The counter increments its register value by 1 count every press. Then if we read the counter register and found 35, it means that there are 35 switch presses have been done.



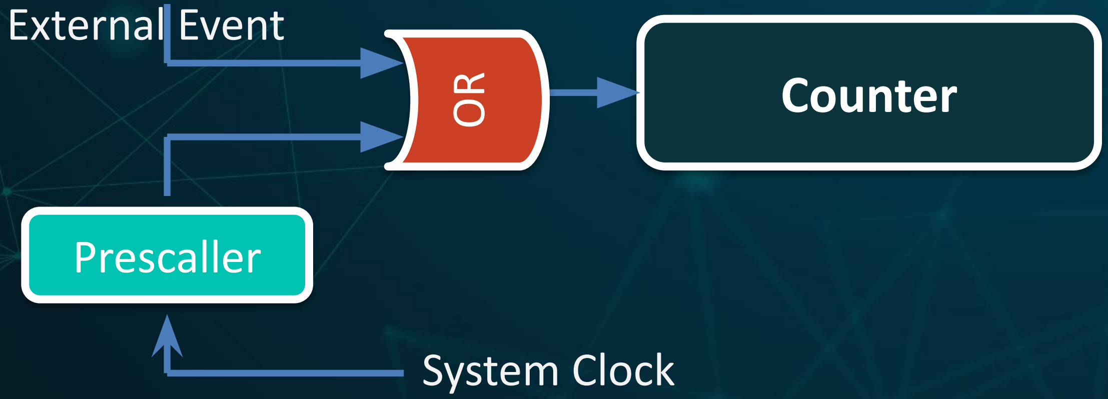
Example

On the other hand, if the counter is connected to a periodic signal with a periodic time equals to 10 ms. The counter is configured to count every **rising edge** as the previous example. If we read the counter register and found 100, it means that 100 rising edge are happened. Because the signal is periodic and the it gives rising edge every 10 ms, then w**e can conclude that 1000 ms have been passed** from the time we started the counter. This is the timer mode.

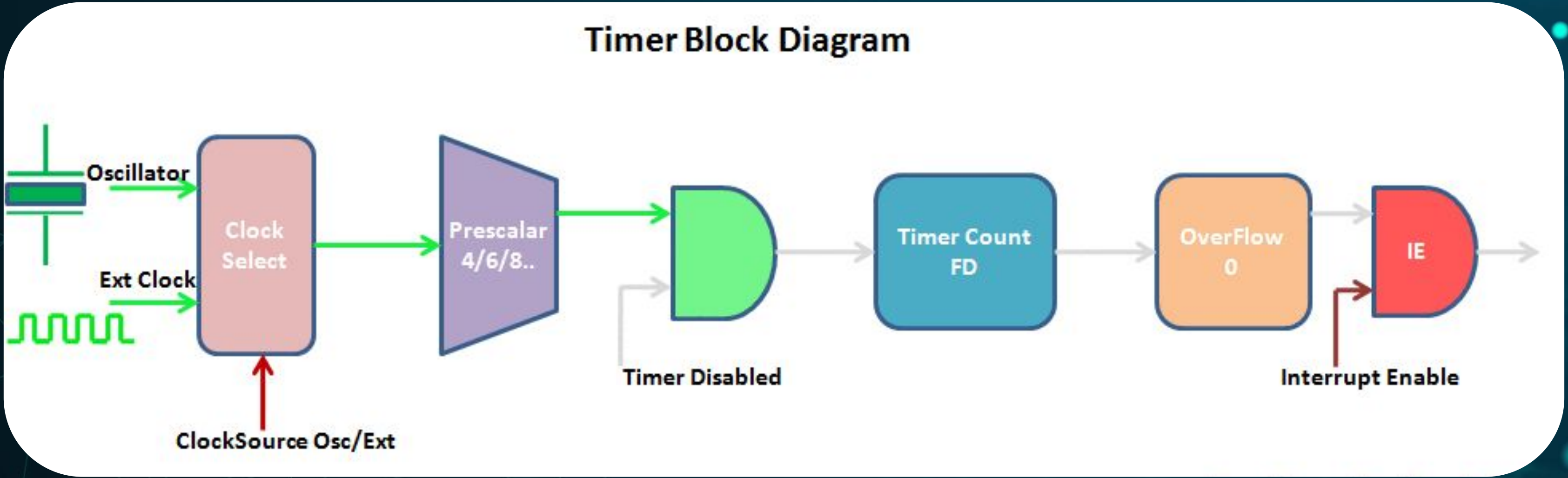


Example

From previous examples, we can say that a timer is a normal counter but counting a periodic signal. So that we can calculate the time from the value in the timer register.
In summary, the timer and the counter are the same peripheral, depending on the input we could classify if it is working as timer or as counter.



Example


Timer Block Diagram
Oscillator
Ext Clock
Clock Select
ClockSource Osc/Ext
Prescalar 4/6/8..
Timer Disabled
Timer Count FD
OverFlow 0
IE
Interrupt Enable

So..?

Timers are used everywhere. Without timers, you would end up nowhere! The range of timers vary from a few microseconds (like the ticks of a processor) to many hours (like the lecture classes)

counters are hardware mechanisms for counting some form of event like the clock pulses.

Timers If we know the time of one count ,so it will be easy to calculate the hole time by multiply the No of counts by the count time .

Timers Terminology

Resolution: Number of bits represents the timer register.

Timer Tick Time: The time between to consecutive events, i.e. time needed to increment the timer register by 1

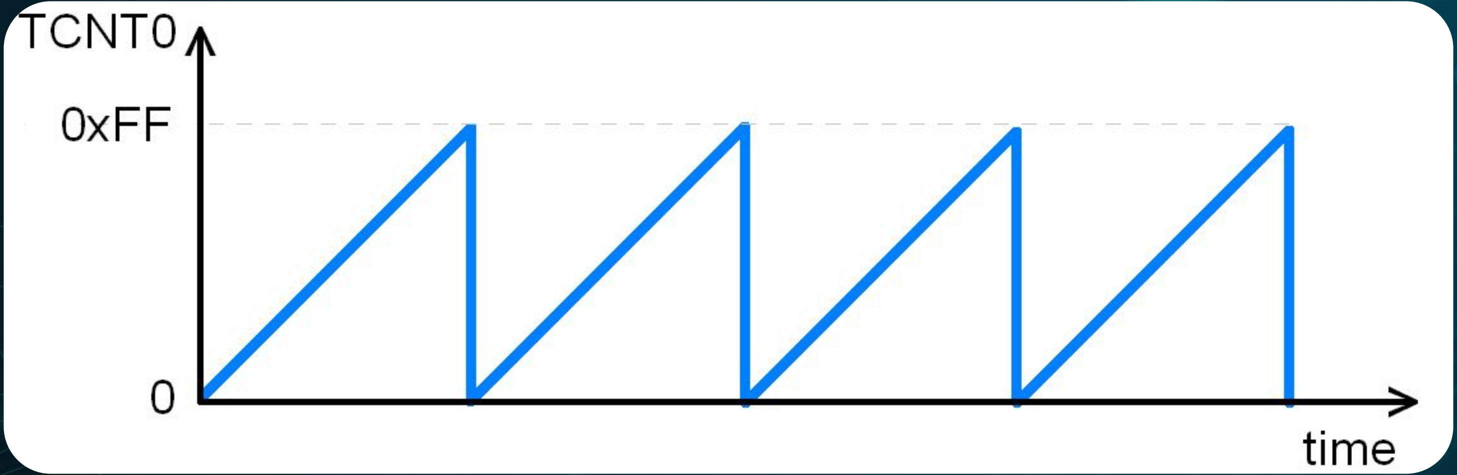
Timer Count: Number of counts needed by the timer to count from 0 till it reaches 0 again. It shall be equals to

Timer Overflow: The time needed by the timer to count from 0 until it reaches 0 again. It shall be equals to Timer Count x Timer Tick Time

Prescaler: Prescaler is a division factor for the system clock (Processor Clock) before it is applied to the timer.

Timers Overflow

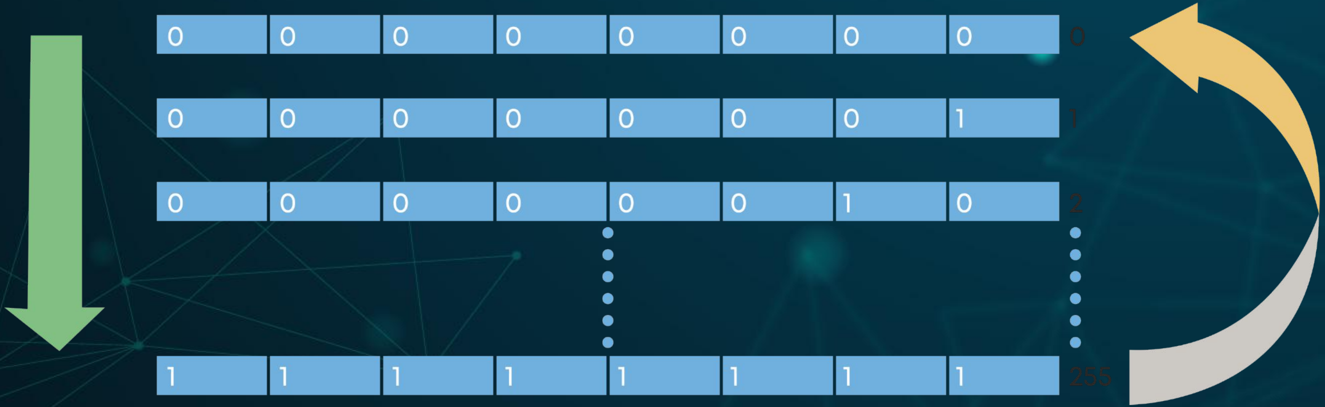
Overflow timer is fire an interrupt (flag) whenever the timer register overflows .



Timers Overflow

Overflow timer is fire an interrupt (flag) whenever the timer register overflows .

In case of 16 bit timer register , the overflow flag will be set when the value reaches 16535



0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	1	0	2
				⋮				⋮
1	1	1	1	1	1	1	1	255

In case of 16 bit timer register , the overflow flag will be set when the value reaches 16535

Timers Calculations

- **Timer Clock =** $\dfrac{\text{System Clock}}{\text{Prescaller}}$
- **Timer Tick Time =** $\dfrac{1}{\text{Timer Clock}} = \dfrac{\text{Prescaller}}{\text{System Clock}}$

Example

Assume a microcontroller working on a frequency of 4 MHz, calculate the timer tick time assuming the timer is using a prescaller of 4.

Solution

Timer Tick Time = $\dfrac{4}{4 \text{ x } 10\text{^}6}$ = 1 microsecond.

i.e. this timer would be incremented every 1 micro second.

Timers Calculations

Timer Overflow Timer = Timer Count x Timer Tick Time

$$= \boxed{2^{\text{Resolution}} \times \frac{\text{Prescaller}}{\text{System Clock}}}$$

Example

From the previous example, assuming the timer resolution is 8 bit, calculate the overflow time.

Solution

Overflow time = 1 microsecond x 2^8 = 256 microseconds.

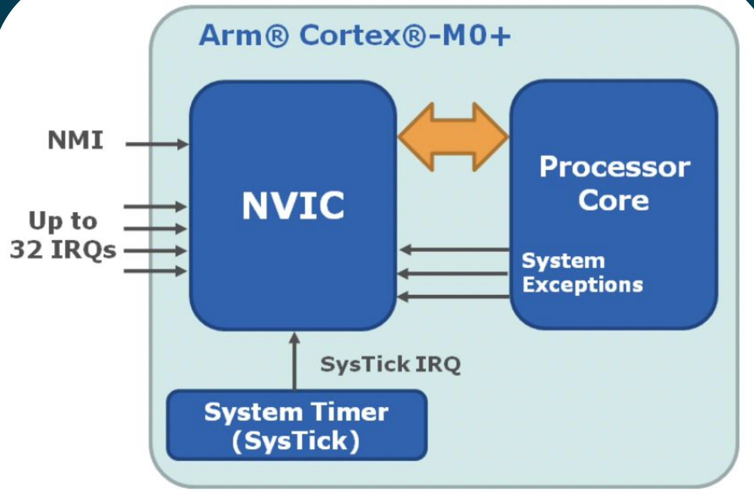
SYSTICK 02

System Timer

SYSTICK

Systick is simply a timer present inside ARM based microcontrollers. Basic purpose is to provide help generate accurate interrupts to different tasks (of RTOS).

It has multiple uses aside from that. For example, many developers use it to generate an accurate delay function. Other benefits are portability where you can easily take an RTOS task from one microcontroler to a different one, and not end up changing the scheduling time and time dependent interrupts for tasks, as there can be different clock sources being used on the new microcontroler.



ARM Cortex-M0+ processor

SYSTICK

The System Tick Timer is an integral part of the Cortex-M3. The System Tick Timer is intended to generate a fixed 10 millisecond(user configurable) interrupt for use by an operating system or other system management software.

The System Tick Timer is a 24-bit timer that counts down to zero and generates an interrupt. The intent is to provide a fixed time interval between interrupts.
In order to generate recurring interrupts at a specific interval, the STRELOAD register must be initialized with the correct value for the desired interval.

SYSTICK Features

- Systick is allocated inside the processor so it will be faster in performing any operations (Core peripheral).
- Memory Mapped, means that its registers which is allocated inside the core have addresses on the memory, which means that we can access its registers using C language.
- Access level is privileged so that's why we use Systick mainly in OS.
- SysTick is a 24-bit down counter so it means that it can count 224 ticks.
- SysTick is a down counter timer which means that it can count from (224 – 1) to zero and it can generate an interrupt when it reaches zero.
- The main advantage of counting down is when calculating the preload value, you don't need to subtract the preload value from the maximum value of the timer counter to get the actual preload value, but we will directly assign the preload value in the timer preload register to make the timer start to count from.

SYSTICK Features

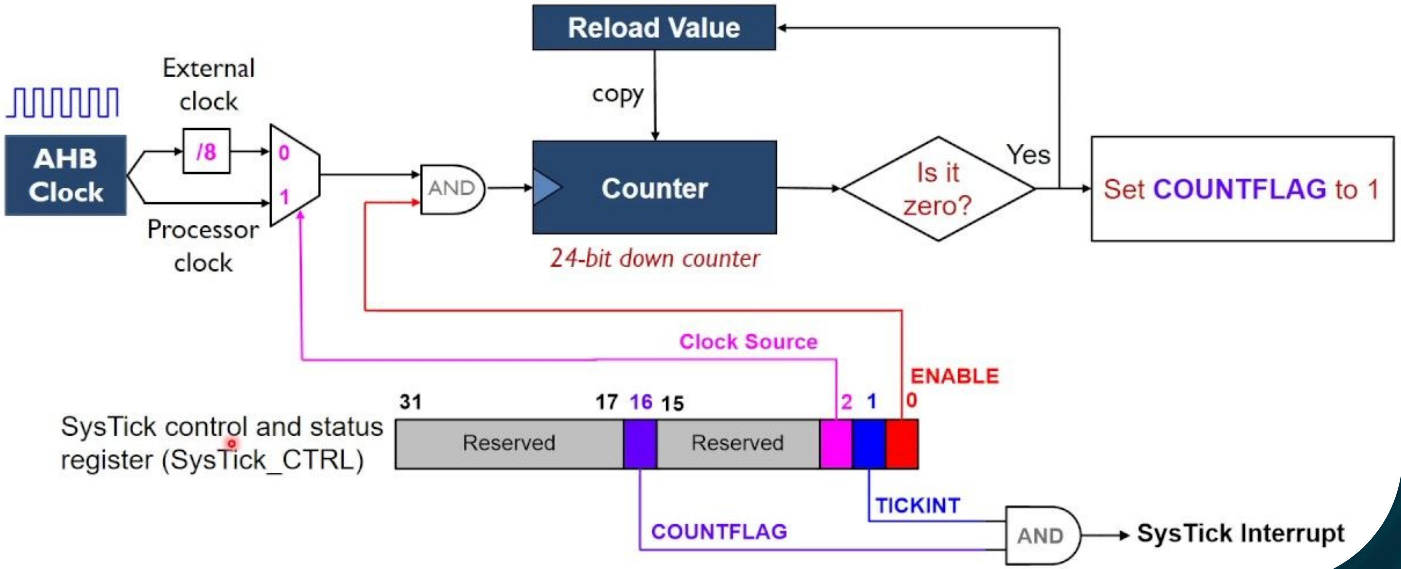
- Working clock could be AHB or AHB/8.
- SysTick timer stops counting when the processor is halted during debugging. Depending on the design of the microcontroller, the SysTick Timer could also be stopped when the processor enters certain type of sleep modes.
- SysTick timer is portable between all cortex same family (M3).
- When the SysTick timer counter reaches zero, the counter loads the reload value from the RELOAD register. It does not stop until the enable bit in the SYSTICK Control and Status register is cleared

SYSTICK How to Operate

- We need to choose the Input clock source and the options are AHB or AHB/8
- We need to enable the peripheral interrupt by setting TICKINT bit in The SysTick control and status register.
- Starting Address of SysTick is at 0xE000 E010
- We mainly have two registers used to set the beginning of the counter.
- Load Register : This register used to set the preload value.
- Value Register : This register is the counter which is decremented by hardware.
- When the Value Register reaches zero, it will load the Load Register and then start counting down.
- If the Value Register is accessed using Software, it will be reset without setting the flag.
- If the Load Register is loaded with a value, the SysTick will start counting from this value, but at the next underflow.
- If we want to Set the value written in the Load Register to be loaded immediately to the Value Register, then we need to write the preload value into the Load Register and then put any value into the Value Register.
- The flag will return 1 when the timer is counted to zero.

SYSTICK Block Diagram

Diagram of System Timer (SysTick)



Delays 03

System Timer

Delays

There are two methods to make a **delay** or to perform an action after some time.
For example, we want to toggle a LED every two seconds.
The two methods which we could perform this delay using them are called

- **Busy wait delay.**
- **Interval delay.**

Busy wait delay

- The delay can be performed by using for loop or while loop.
- The processor will be halted inside this loop until this loop is terminated.
- Busy wait delay method is the simplest way to perform a delay, however it is not the best method to perform a delay because it causes the processor to be halted inside this loop until this loop is terminated.
- The processor will not execute any instruction and it will only execute this loop only whatever it takes.

Delays

Interval wait delay

- It is the second method of performing a delay.
- It is the most preferred method to perform a delay because this method is based on a timer interrupt.
- In this method we are going to use a timer peripheral to perform this delay.
- In this method the timer interrupt is configured to take place whenever the application needs to take an action.
- As the previous example where we need to toggle a LED every one second, so we will configure the interrupt timer to be fired when the timer counter counts a one second.
- Whenever the timer interrupt fires, the processor will execute the timer handler.
- In this method the processor could execute some other instructions without being halted executing inside the loop waiting this loop to be terminated
- There are three possibilities of using a timer interrupt after a desired time.

Delays

Interval timer

downflows. The timer sets a flag and fires an interrupt if enabled.

Interval timer

How to calculate a certain time based on an interval timer?

- Step 1: Calculate the overflow time of the used timer
- Step 2: Compare the desired time with the overflow time, and follow one of the following possibilities.

Delays

Possibility 2

Desired Time equals to overflow time

This is the happy scenario possibility, which normally doesn’t happen. If you are lucky and the desired time is equal to overflow time, then just enable the interrupt and take the desired action inside the ISR.

Delays

Possibility 2

Desired Time than overflow time

- Step 1: Calculate the number of counts needed

Number of counts needed = (Desired Time / Overflow Time) ∗ Overflow count

- Step 2: Preload the timer register with the following value: In case of SysTick timer which counts in a decrementing way so we will directly load the Number of counts needed into the timer register.

Preload Value = Number of counts needed

- Step 3: Enable the timer interrupt and take the desired action

Delays

Example on Possibility 2

Assuming 24 bit timer using a Clock of AHB/8 and the clock on AHB is 8MHz. An action is needed to be taken after 64 microseconds. Do the needed calculations and write the code.

Solution

- Overflow time = (2^24) * 1 Microsecond = 16777216 Microseconds
- The desired time is 64 microseconds which is less than the overflow time. Then we calculate the needed number of counts
- Number of needed counts = (64 / 16777216) x 16777216 = 64 count
- Preload value = 64
- If the timer register is initialized with the preload value (64), then it needs only 64 count to overflow, i.e. the overflow interrupt would fire after only 64 count. Which is the desired time.

Delays

Possibility 3 (Case 1)

Desired Time more than Overflow time

- Step 1: Calculate the number of overflows needed
Number of overflows = Desired Time / Overflow Time

- Assume that the number of the overflows needed is a decimal value, then define a variable and increments it inside the overflow interrupt till we reach the desired number of overflows, then take the desired action.

Delays

Possibility 3 (Case 2)

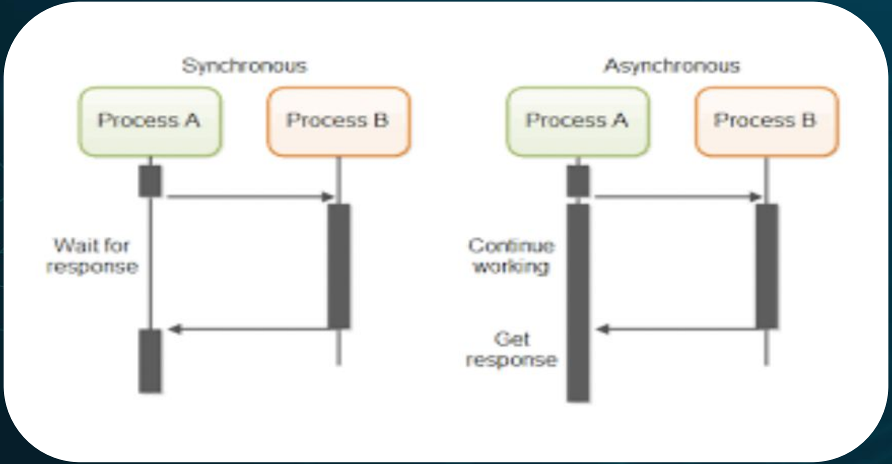
Desired Time more than Overflow time

Step 1: Calculate the number of overflows needed
Number of overflows = Desired Time / Overflow Time

- Assume that the number of the overflows needed is a floating value, then define a variable and increments it inside the overflow interrupt till we reach the desired number of overflows, then take the desired action.

Asynchronous vs Synchronous Design

Synchronous basically means that you can only execute one thing at a time. Asynchronous means that you can execute multiple things at a time and you don't have to finish executing the current thing in order to move on to the next one.



Asynchronous vs Synchronous Design

Synchronous Design

In the case of synchronous design, the expectation is that there will be an immediate return of data. The application requests data and waits for it until a value is returned.

So in the synchronous design, function returns after all the required processing is finished, it may do some sort of busy wait for a specific operation like I/O.

Asynchronous Design

In the case of Asynchronous design, the function returns immediately after submitting an operation request and the operation makes some sort of callback to your code or an Interrupt when it completes to give its completion status.

Callback Function

04

System Timer

Callback Function

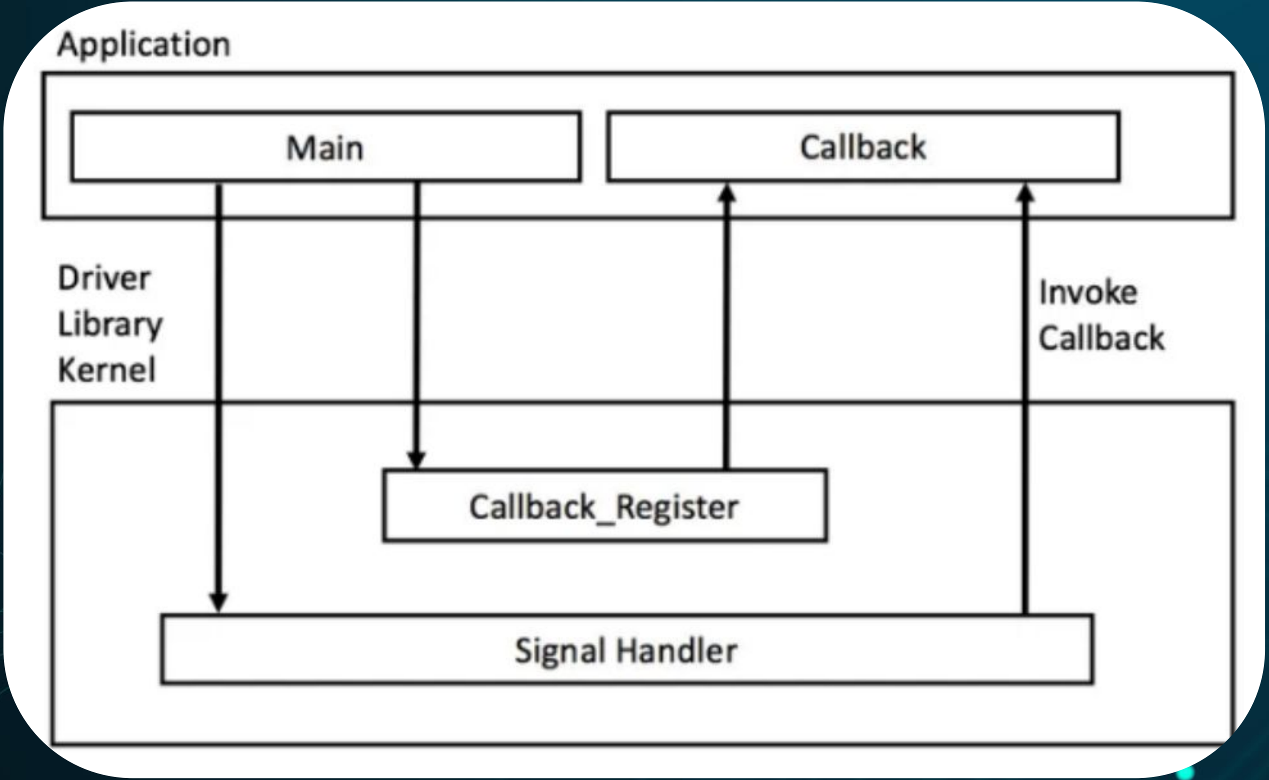
Callback functions are an essential and often critical concept that developers need to create drivers or custom libraries.

A callback function is a reference to executable code that is passed as an argument to other code that allows a lower-level software layer to call a function defined in a higher-level layer. A callback allows a driver or library developer to specify a behavior at a lower layer but leave the implementation definition to the application layer.

A callback function is just a function pointer that is passed to another function as a parameter. In most instances, a callback will contain three pieces:

- The callback function
- A callback registration
- Callback execution

Callback Function



IMT SCHOOL

STM32
Is AWESOME

Session LAb

Create **SYSTICK** Driver

Time To Code

THANKS!

Do you have any question?

www.imtschool.com

www.facebook.com/imaketechnologyschool/

This material is developed by IMTSchool for educational use only