Microprocessors

Fall 2020

8. Timers

Tentative Weekly Schedule

- Week x1 - Introduction to Course
- Week x2 - Architecture
- Week x3 - Assembly Language Introduction
- Week x4 - Assembly Language Usage, Memory and Faults
- Week x5 - Embedded C and Toolchain
- Week x6 - Exceptions and Interrupts
- Week x7 - GPIO, External Interrupts and Timers
- **Week x8 - Timers**
- Week x9 - Serial Communications I
- Week xA - Serial Communications II
- Week xB - Analog Interfacing
- Week xC - DMA
- Week xD - RTOS
- Week xE - Wireless Communications

Review: Timers in STM32G0

There are different timer related modules in G0

Inside the ARM core

- SysTick Timer

As peripherals

- General-Purpose Timers
- Advanced-control Timers - various additional functionalities such as input capture, output compare and PWM
- Watchdog Timers
  - Independent WDG
  - Window WDG

Review: The SysTick Timer

- In order to allow an Operating System to carry out periodical *context switching* to support multi-tasking, **the program execution must be interrupted** by a hardware device like a timer.
- When the timer interrupt is triggered, an exception handler that handles **OS task scheduling** is executed
  - The handler might also carry out other OS maintenance tasks.
- For Cortex-M processors, a simple timer called **SysTick** is included **inside the processor** to perform the function of generating this periodic interrupt request.
- For non-OS uses, this acts as a generic timer.

Review: Timers in STM32G0

Timer type	Timer	Counter resolution	Counter type	Maximum operating frequency	Prescaler factor	DMA request generation	Capture/ compare channels	Comple-mentary outputs
Advanced-control	TIM1	16-bit	Up, down, up/down	128 MHz	Integer from 1 to 2^{16}	Yes	4	3
General-purpose	TIM2	32-bit	Up, down, up/down	64 MHz	Integer from 1 to 2^{16}	Yes	4	-
	TIM3	16-bit	Up, down, up/down	64 MHz	Integer from 1 to 2^{16}	Yes	4	-
	TIM14	16-bit	Up	64 MHz	Integer from 1 to 2^{16}	No	1	-
	TIM16 TIM17	16-bit	Up	64 MHz	Integer from 1 to 2^{16}	Yes	1	1
Low-power	LPTIM1 LPTIM2	16-bit	Up	64 MHz	2^n where n=0 to 7	No	N/A	-

Review: Timers in STM32G0

- These timers can be used for
  - standard up/down counter mode
  - measuring pulse lengths of input signals (input capture)
  - generating output waveforms (output compare)
  - pulse-width modulation generation
  - interrupt / DMA generation
- They can be 16 / 32 bits, so be careful
- Include synchronization circuit to control the timer with external signals and to connect several timers together

Basic Timer Usage

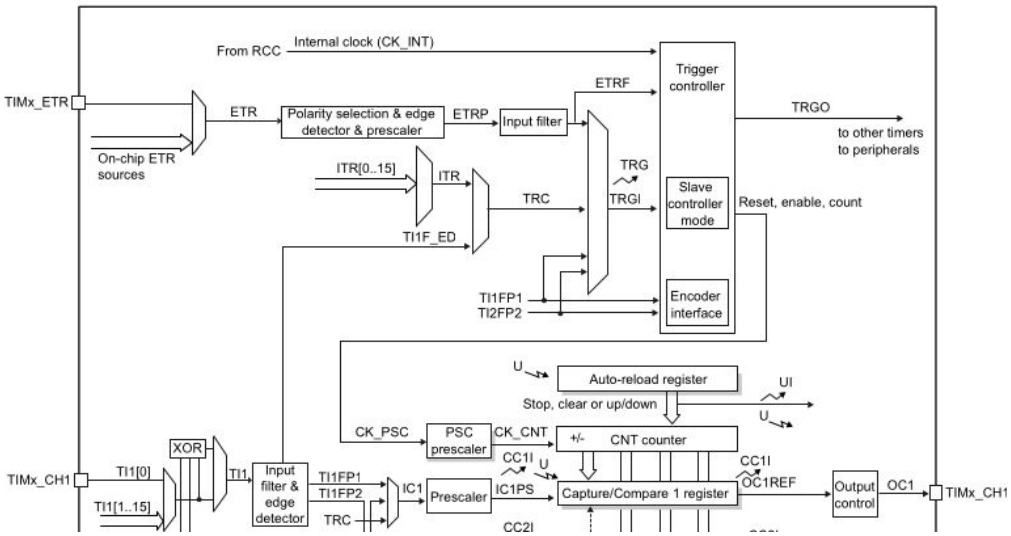
For basic usage, we need 6 registers

- **CR1** for configuration
- **CNT** for current counter value
- **PSC** for prescaler
- **ARR** for auto reload
- **DIER** for update interrupt enable
- **SR** for status register (check and clear pending)

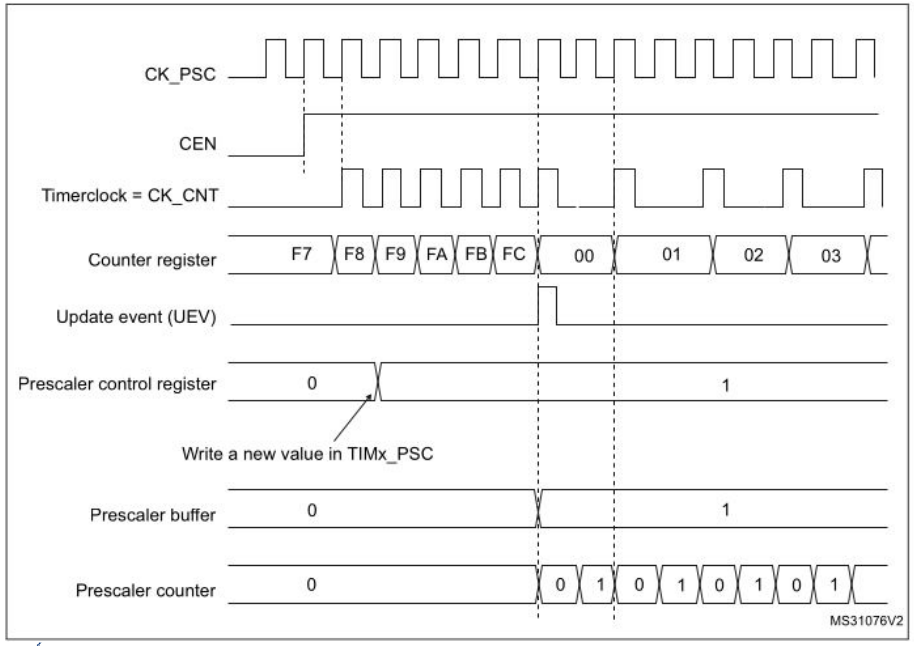
Everytime an overflow/underflow happens an update event is generated.

Note: Make sure to enable TIMx module clock from RCC in the beginning of your code. This is the **golden rule** for any peripheral you want to use.

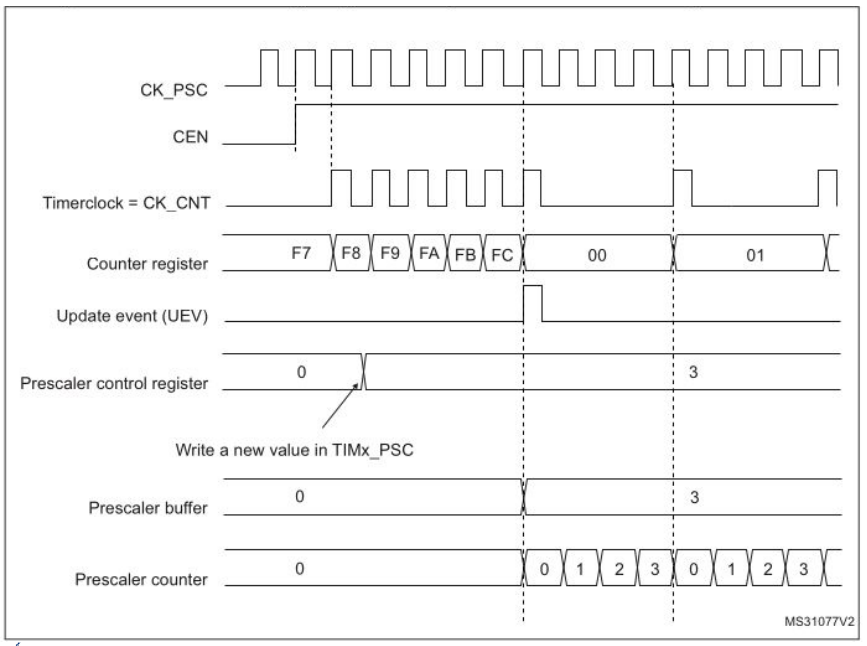
STM32G0 - General-purpose timer block



Basic counter with prescaler 1 -> 2

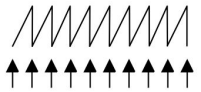


Basic counter with prescaler 1 -> 4



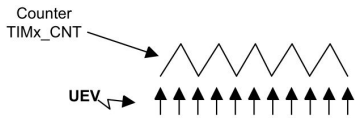
Counter modes

- **Up counting mode** - the counter counts from 0 to the auto-reload value, then restarts from 0 and generates a counter overflow event.

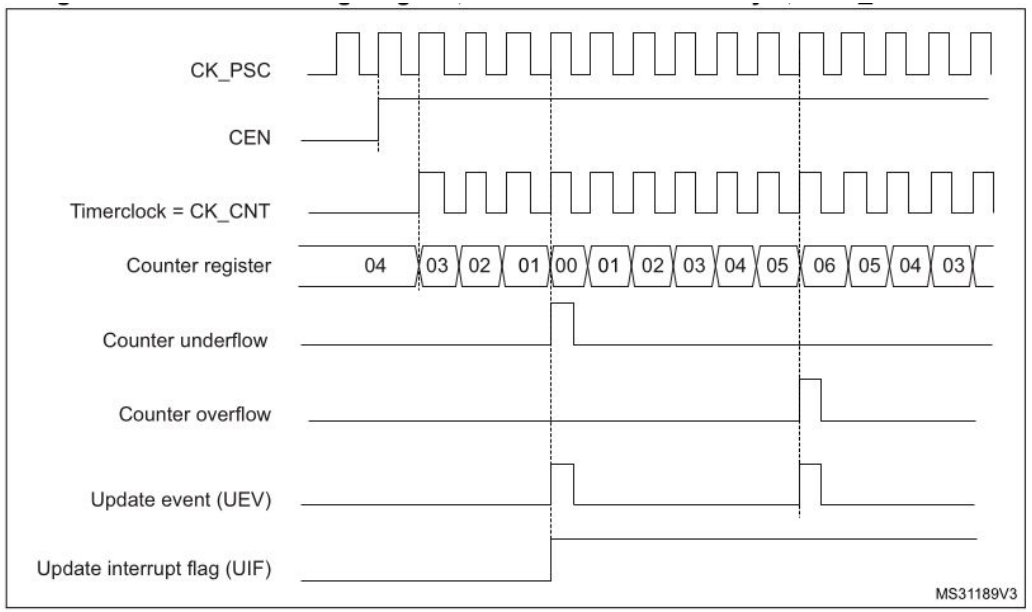


- **Down counting mode** - the counter counts from auto-reload value down to 0, then restarts from the auto-reload value and generates a counter underflow event.

- **Center-aligned mode (up/down counting)** - counts from 0 to auto-reload value - 1, generates a counter overflow event, then counts from the auto-reload value down to 1 and generates a counter underflow event, then restarts from 0.



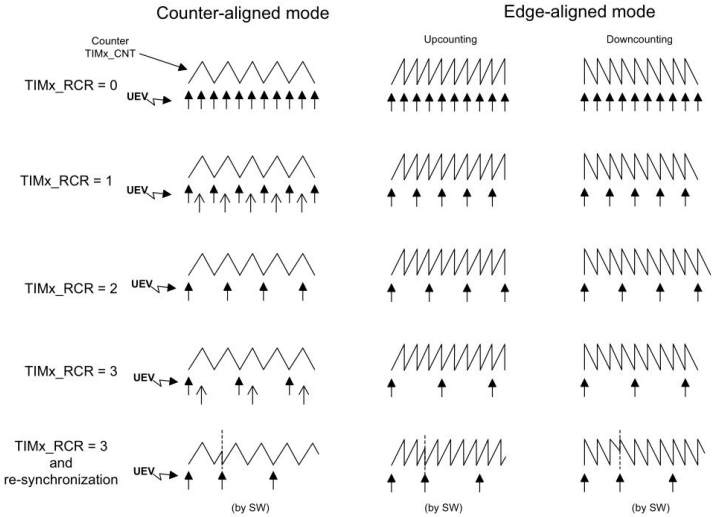
Center-aligned mode (TIMx_ARR = 0x6)



Repetition counter

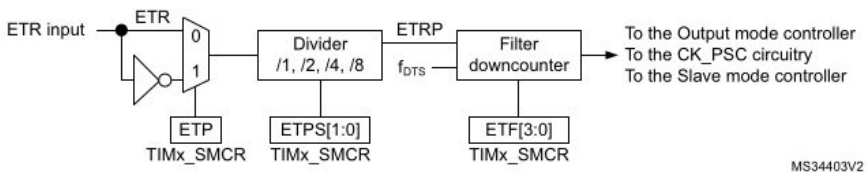
- **Repetition counter** is **actually** used to generate an update event every time this counter reaches to 0
- By default, this counter is at 0, so update is generated every time overflow/underflow happens
- It is decremented
  - at each counter **overflow** in **upcounting mode**
  - at each counter **underflow** in **downcounting mode**
  - at each counter **overflow / underflow** in **center-alined mode**
- Useful for when generating PWM signals

Repetition counter



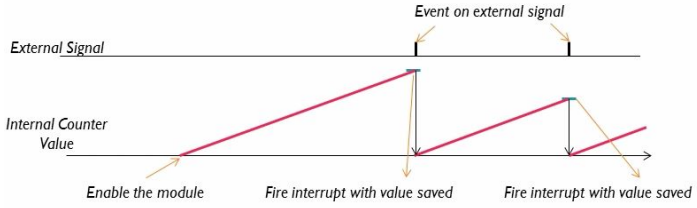
External trigger input

- Some timers can feature an **external trigger** (ETR) input which can be used as:
  - External clock
  - Trigger for the slave mode
  - PWM reset input for cycle-by-cycle current regulation



Input Capture Mode

- **Input capture** is used to capture the value of the counter after a transition is detected by the corresponding pin
- I/O pin is the external input signal (ICx signal)
- When valid edge detected on pin
  - store current counter value
  - generate interrupt request



Example: Anemometer (Wind Speed Indicator)

- Rotational speed (and pulse frequency) is proportional to wind velocity
- Two measurement options
  - Frequency (best for high speeds)
  - Pulse Width (best for low speeds)

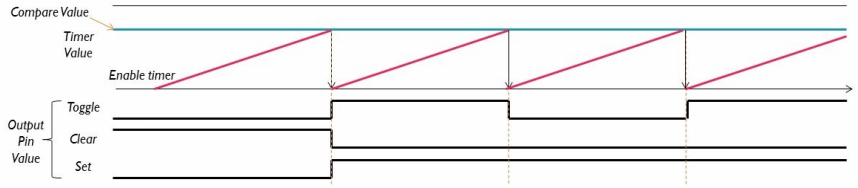
- How can we measure the rotation speed assuming it generates a pulse every rotation?

- Use a timer in input capture mode to measure the period of the input signal. Then make the necessary calculation

Output Compare Mode

This function is used to **control a digital output waveform**. If a **match** happens with the counter value, a desired **output signal level is generated**.

- Modify output signal when timer reaches specified value. (set, clear, toggle, pulse)
- Generate a pulse with desired width
- Generate a pulse after a defined delay



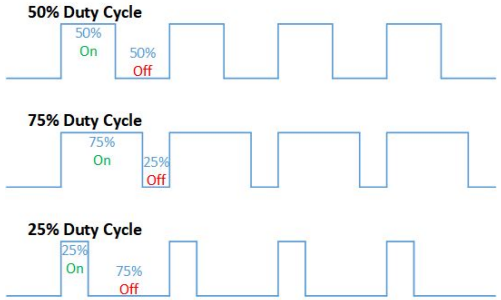
Pulse Width Modulation

- **Pulse Width Modulation** is used to **change the average voltage** by changing the **width** of the **pulse** at **given time intervals**.
  - Digital power amplifiers are **more efficient and less expensive** than analog power amplifiers
    - Motor speed control, light dimmer, switch-mode power conversion
    - Load responds slowly, averaging PWM signals
  - Digital communication is l**ess sensitive to noise** than analog method
    - PWM provides a digital encoding of an analog value
    - Much less vulnerable to noise

PWM concepts

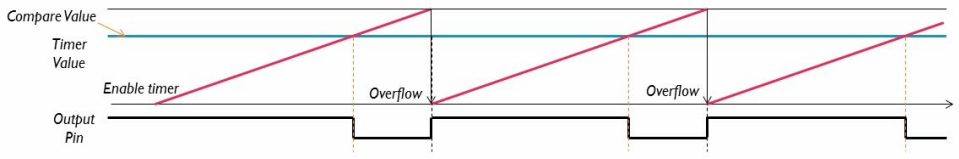
Modulation frequency - how many pulses occur per second. (fixed)
Period - 1 / (modulation frequency)
On time - amount of time that each pulse is on
Duty cycle - on time / period * 100 %

By adjusting on time (or duty cycle) we can represent analog voltage



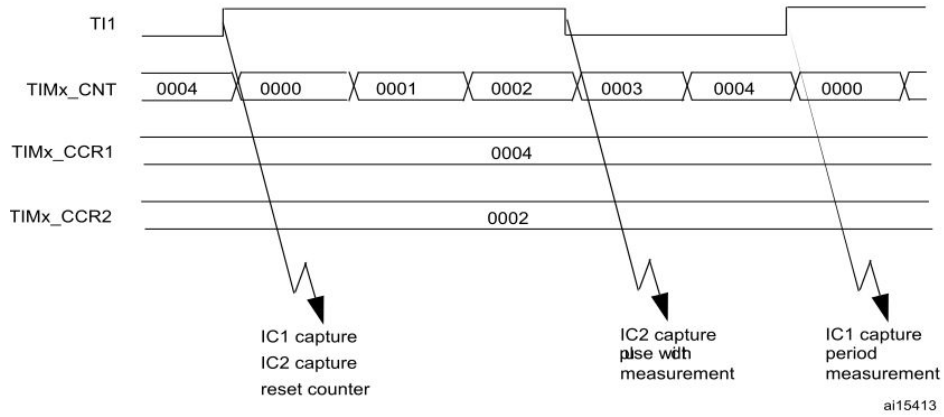
PWM Mode

- PWM duty cycle proportional to compare value
- **Period** equals to max timer value
- **Pulse width** equals to compare value
- **Polarity** can usually be selected.

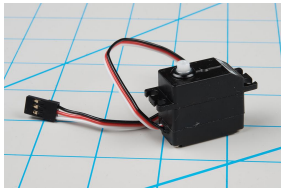


PWM Input Mode

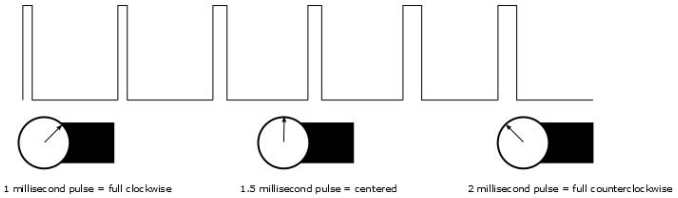
- A **PWM input** mode exists that maps two ICx signals to the same TIx input to measure the period and duty cycle of the applied PWM.



PWM to Drive Servo Motor



- A **DC motor** has two wires, and simply turns continuously when power is applied.
- **Reverse polarity** power will spin it in the **opposite direction**
- You need a way to measure if you want to know how are it has turned.
- **Servo motors** turn using **carefully-timed pulses** with an additional wire to carry these pulses
- Usually **20 ms pulses** with **1-2 ms duty cycle**
- Common ones are **90°** with **continues** ones exist



Question

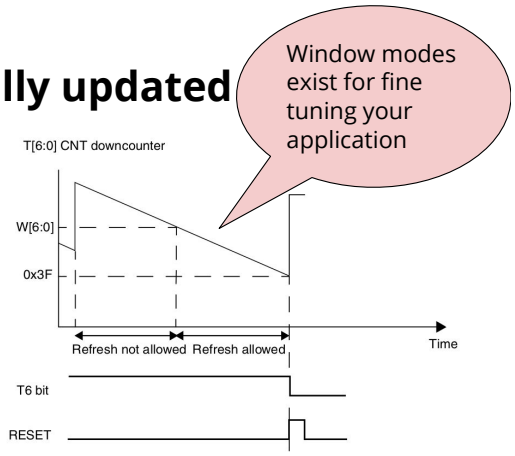
- Assume something happened with SysTick peripheral and COUNTFLAG (bit 16 in CTRL register) never gets updated
- you are stuck in that while loop.

```
void delay(volatile uint32_t s) {
    for (; s>0; s--){
        while( !(SYSTICK->CTRL & (1 << 16)));
    }
}
```

Q. How do you recover from this?

Watchdog Timer

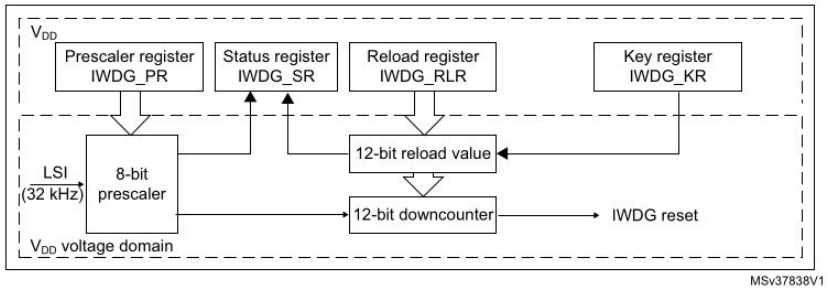
- To increase reliability, and prevent any lockup situations, vendors include a **Watchdog Timer** module that will **generate a reset** in case of **a software failure**.
- It needs to be **periodically updated**
  - or ***fed***, as in ***feeding the dog***
- In case it doesn't get updated in a given period of time, it will **generate a reset**



Independent Watchdog Timer

- **Independent watchdog** peripheral acts like a regular watchdog timer, but is **clocked** by its own dedicated **low-speed clock (LSI)** and thus stays **active** even if the **main clock fails.**
- Best suited for applications that require the watchdog to run as a totally independent process, but have lower timing accuracy constraints.
- Free-running down counter
- Can operate in Standby and Stop modes
- Once running, **IWDG** cannot be stopped

Independent Watchdog Timer



- Write `0x0000CCCC` to **KR** register to enable IWDG
- Write `0x00005555` to **KR** register to enable configuring peripheral
- Choose prescaler from **PR** (`/4 ... /256, 0..7`)
- Write `0x0000AAAA` to **KR** register to periodically feed the dog.

Window Watchdog Timer

- **Window watchdog timer** peripheral is similar to independent watchdog timer, but runs on the main clock.
- Has a update window to update.
- 7-bit down-counter, and if bit6 becomes 0, it will reset.
- Prescaled from the APB clock
- Best suited for applications that require the watchdog to react **within an accurate timing window**.

This week

- Read Sections 20,21,27,28 from RM0444
- Lab4 is due 9th of December
- HW4 is due 14th of December
- Project 2 is due on 21st December