# Lecture 11: STM32 Advanced Timer/Counters

Seyed-Hosein Attarzadeh-Niaki

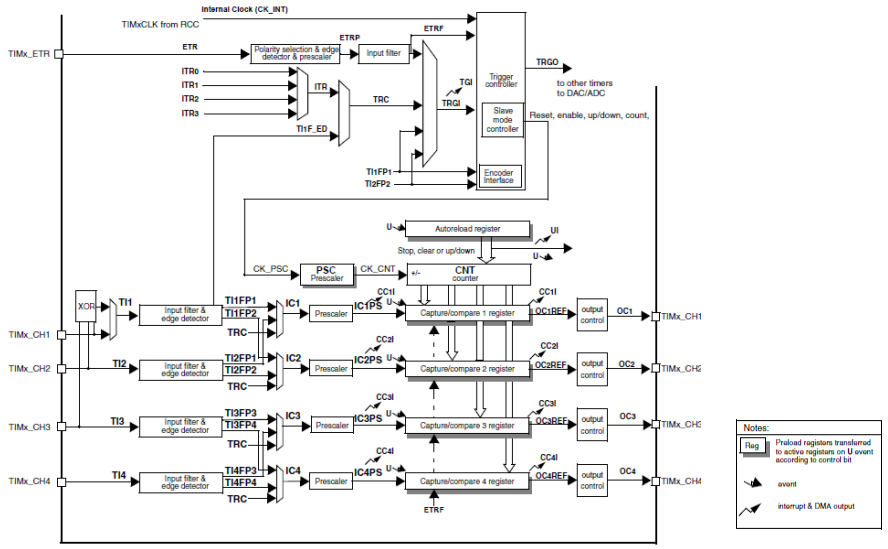
Based on Slides by ARM



## Review

- STM32 timer peripherals
- Timer registers
- Clock source and prescaling
- Upcounting mode and periodic interrupt generation
- Cortex-M4 SysTick timer

# General Purpose Timer Block Diagram






# Outline

- Overview of timer modes
- Capture/Compare channels
  - Input capture
  - Output compare
- Pulse-width modulation

## STM32F4x Timer Counting Modes

**Upcounting**

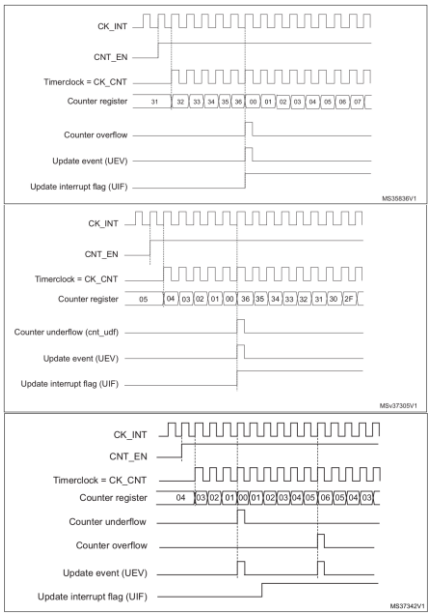
- Counts from 0 to the auto-reload value (TIMx_ARR)
- Then restarts from 0 and generates a counter overflow event
- An Update event can be generated at each counter overflow

**Downcounting**

- Counts from the auto-reload value (TIMx_ARR) down to 0
- Then restarts from the auto-reload value and generates a counter underflow event
- An Update event can be generate at each counter underflow

**Center-aligned**

- Counts from 0 to the auto-reload value (TIMx_ARR)-1 and generates a counter overflow event
- Then counts from the auto-reload value down to 1 and generates a counter underflow event.
- Then it restarts counting from 0.
- The update event can be generated at each counter overflow and at each counter underflow






# STM32F4x Timer Operating Modes

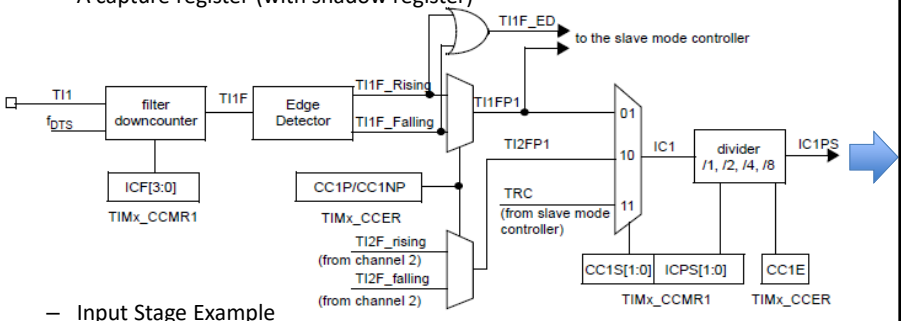
| | |
|---|---|
| Input capture mode | • Capture/Compare Registers (TIMx_CCRx) are used to latch the value of the counter after a transition detected by the corresponding ICx signal. |
| PWM input mode | • Particular case of input capture mode<br>• Can be used to measure freq and duty cycle of the PWM applied on TI1 |
| Forced output mode | • Each output compare signal can be forced to active or inactive level directly by software |
| Output compare mode | • Used to control an output waveform or indicating when a period of time has elapsed |
| PWM mode | • Used to generate a signal with desired frequency and duty cycle |
| One-pulse mode | • allows the counter to be started in response to a stimulus and to generate a pulse with a programmable length after a programmable delay |
| Encoder interface mode | • used to interface to an incremental encoder |

# CAPTURE/COMPARE MODE



# Capture/Compare Channels

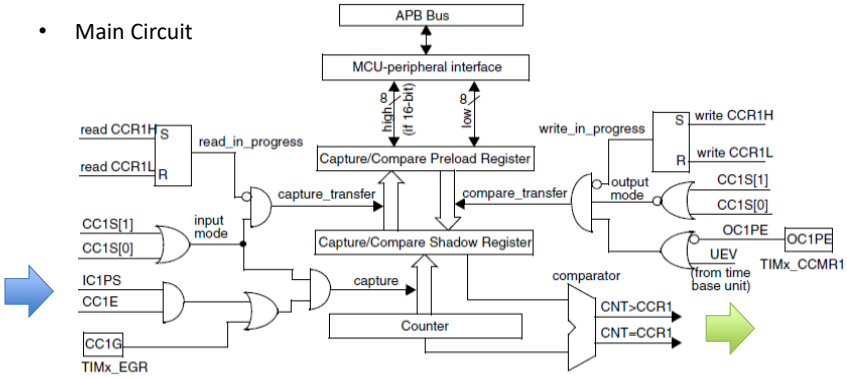
- Different channels but same blocks
  - Capture mode can be used to measure the pulse width or frequency
  - Input stage includes digital filter, multiplexing and prescaler
  - Output stage includes comparator and output control
  - A capture register (with shadow register)




  - Input Stage Example
    - Input signal->filter->edge detector->slave mode controller or capture command

# Capture/Compare Channels

- Main Circuit

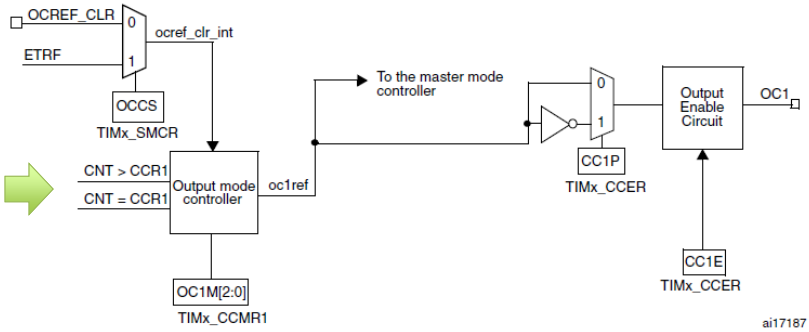



- The block is made of one preload register and a shadow register.
  - In **capture** mode, captures are done in shadow register then copied into preload register
  - In **compare** mode, the content of the preload register is copied into the shadow register which is compared to the counter



# Capture/Compare Channels

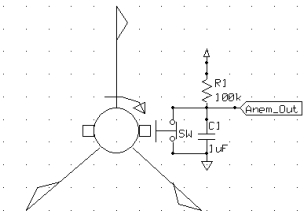
- Output stage

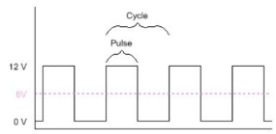



- Generates an intermediate waveform which is then used for reference.

# Wind Speed Indicator (Anemometer)

- Rotational speed (and pulse frequency) is proportional to wind velocity
- Two pulse measurement options:
  - Frequency (best for high speeds)
  - Width (best for low speeds)
- Can solve for wind velocity v
- How can we use the Timer for this?
  - Use Input Capture Mode to measure period of input signal






# Input Capture Mode for Anemometer

- Operation: Repeat
  - First capture - on rising edge
    - Reconfigure channel for input capture on falling edge
    - Clear counter, start new counting
  - Second Capture - on falling edge
    - Read capture value, save for later use in wind speed calculation
    - Reconfigure channel for input capture on rising edge
    - Clear counter, start new counting
- Solve the wind speed
  - $V_{wind} = K \div (C_{falling} - C_{rising}) \times Freq$

# Registers for Anemometer

- **TIMx_ARR** (Refer to the periodic interrupt)
- **TIMx_PSC** (Refer to the periodic interrupt )
- **TIMx_CCMR1** Capture/compare mode register 1 (Channel 1 and 2)
- **TIMx_CCMR2** (for channel 3 and 4)
- **TIMx_CCER** Capture/compare enable register
- **TIMx_DIER** DMA/interrupt enable register
- **TIMx_CR1** Control register
- **TIMx_CCRx** Capture compare register
- **TIMx_SR** Status register



# Capture/Compare Mode Register 1

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| OC2CE | OC2M[2:0] | | | OC2PE | OC2FE | CC2S[1:0] | | OC1CE | OC1M[2:0] | | | OC1PE | OC1FE | CC1S[1:0] | |
| IC2F[3:0] | | | | IC2PSC[1:0] | | | | IC1F[3:0] | | | | IC1PSC[1:0] | | | |
| rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw | rw |

- [0:7] bits are for channel 1, [8:15] bits are for channel 2
- Set CC1S (Capture/compare 1 selection) to 01, configure channel 1 as input and map IC1 to TI1 (Only writable when channel is OFF)
- IC1PSC (Input capture 1 prescaler)
- IC1F (Input capture 1 filter) decide sampling frequency and N events needed to validate a transition on the output

| | |
|---|---|
| 0000: No filter, sampling is done at $f_{DTS}$ | 1000: $f_{SAMPLING}=f_{DTS}/8$, N=6 |
| 0001: $f_{SAMPLING}=f_{CK\_INT}$, N=2 | 1001: $f_{SAMPLING}=f_{DTS}/8$, N=8 |
| 0010: $f_{SAMPLING}=f_{CK\_INT}$, N=4 | 1010: $f_{SAMPLING}=f_{DTS}/16$, N=5 |
| 0011: $f_{SAMPLING}=f_{CK\_INT}$, N=8 | 1011: $f_{SAMPLING}=f_{DTS}/16$, N=6 |
| 0100: $f_{SAMPLING}=f_{DTS}/2$, N=6 | 1100: $f_{SAMPLING}=f_{DTS}/16$, N=8 |
| 0101: $f_{SAMPLING}=f_{DTS}/2$, N=8 | 1101: $f_{SAMPLING}=f_{DTS}/32$, N=5 |
| 0110: $f_{SAMPLING}=f_{DTS}/4$, N=6 | 1110: $f_{SAMPLING}=f_{DTS}/32$, N=6 |
| 0111: $f_{SAMPLING}=f_{DTS}/4$, N=8 | 1111: $f_{SAMPLING}=f_{DTS}/32$, N=8 |

  - For example, if set to 0001, and set to capture rising edge, when rising edge detected, sample the channel twice with the FCK_INT ,if they are both high then the capture is validated.

# Capture/Compare Enable Register

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CC4NP | Res. | CC4P | CC4E | CC3NP | Res. | CC3P | CC3E | CC2NP | Res. | CC2P | CC2E | CC1NP | Res. | CC1P | CC1E |
| rw | | rw | rw | rw | | rw | rw | rw | | rw | rw | rw | | rw | rw |

- 3 Bits for each channel and 4 bits in total are reserved
- CC1NP/CC1P: 00 to be sensitive to rising edge, 01 to be sensitive to falling edge, different meaning in other modes
- CC1E, set to 1 to enable the capture

- Also needs to enable the interrupt on capture
  - TIMx_DIER: set CC1IE
- Finally, enable the counting

- Also remember to clear the pending bit (write 0 to TIM_SR_CC1IF) in the ISR

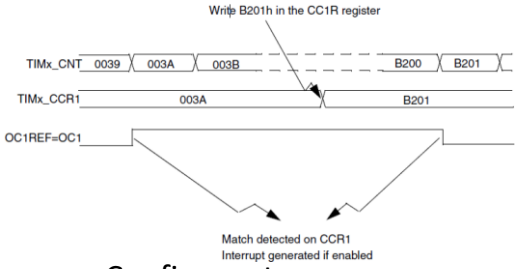


# Configure the GPIO - AF

- Refer to the user manual to make sure which pin is able to connect to the TIM
- Then configure the GPIO as AF mode, be careful with the pull up or down setting since it should match the setting of edge detection
- Configure the GPIO AF register

# Output Compare Mode

- Control an output waveform or indicating when a period of time has elapsed
- When a Match occur (CCRx=CNT)
  - Generate specific output on corresponding pin
  - Set the CCxIF (Interrupt status) bit in the SR
  - Generate Interrupt if configured
  - Generate DMA request if configured




- Configure steps
  - Select the counter clock
  - Write the desired data in ARR and CCR registers
  - Enable Interrupt or DMA request if needed
  - Select the output mode
  - Enable the counter



# Example

1. Toggle Green LED using TIM2 Compare mode
   - Prescaler divides by 1600 and counter wraps around at 10000
     - The 16 MHz system clock is divided by 1600 then divided by 10000 to become 1Hz
   - Channel 1 is configured for compare mode to toggle the output pin every time the timer counter matches the CCR1 register
   - The output pin of TIM2 CH1 is PA5 and the alternate function of PA5 should be set to AF1
2. TIM3 Measuring Period and Frequency
   - TIM3 CH1 is set to do Input Capture from PA6
   - Waits for capture flag (CC1IF) to set then calculates the period and frequency of the input signal
   - A jumper should be used to connect PA5 to PA6

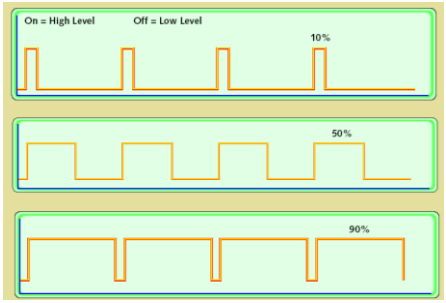
```c
#include "stm32f4xx.h"
int period;
float frequency;
int main(void) {
    int last = 0;
    int current;
    // configure PA5 as output of TIM2 CH1
    RCC->AHB1ENR  |=  1;              /* enable GPIOA clock */
    GPIOA->MODER  &= ~0x00000C00;     /* clear pin mode */
    GPIOA->MODER  |=  0x00000800;     /* set pin to alternate function */
    GPIOA->AFR[0] &= ~0x00F00000;     /* clear pin AF bits */
    GPIOA->AFR[0] |=  0x00100000;     /* set pin to AF1 for TIM2 CH1 */
    // configure TIM2 to wrap around at 1 Hz
    // and toggle CH1 output when the counter value is 0
    RCC->APB1ENR |= 1;                /* enable TIM2 clock */
    TIM2->PSC     = 1600 - 1;         /* divided by 1600 */
    TIM2->ARR     = 3000 - 1;         /* divided by 3000 */
    TIM2->CCMR1   = 0x30;             /* set output to toggle on match */
    TIM2->CCR1    = 0;                /* set match value */
    TIM2->CCER   |= 1;                /* enable ch 1 compare mode */
    TIM2->CNT     = 0;                /* clear counter */
    TIM2->CR1     = 1;                /* enable TIM2 */
    // configure PA6 as input of TIM3 CH1
    RCC->AHB1ENR  |=  1;              /* enable GPIOA clock */
    GPIOA->MODER  &= ~0x00003000;     /* clear pin mode */
    GPIOA->MODER  |=  0x00002000;     /* set pin to alternate function */
    GPIOA->AFR[0] &= ~0x0F000000;     /* clear pin AF bits */
    GPIOA->AFR[0] |=  0x02000000;     /* set pin to AF2 for TIM3 CH1 */
    // configure TIM3 to do input capture with prescaler ...
    RCC->APB1ENR |= 2;                /* enable TIM3 clock */
    TIM3->PSC     = 16000 - 1;        /* divided by 16000 */
    TIM3->CCMR1   = 0x41;             /* set CH1 to capture at every edge */
    TIM3->CCER    = 0x0B;             /* enable CH 1 capture both edges */
    TIM3->CR1     = 1;                /* enable TIM3 */
    while (1) {
        while (!(TIM3->SR & 2)) {}    /* wait until input edge is captured */
        current   = TIM3->CCR1;       /* read captured counter value */
        period    = current - last;   /* calculate the period */
        last      = current;
        frequency = 1000.0f / period;
        last      = current;
    }
}
```



# PWM MODE

# Pulse-Width Modulation

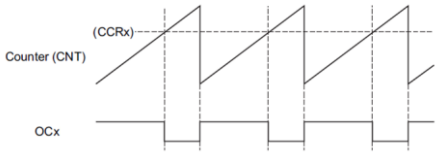
- Uses of PWM
  - Digital power amplifiers are more efficient and less expensive than analog power amplifiers
    - Applications: motor speed control, light dimmer, switch-mode power conversion
    - Load (motor, light, etc.) responds slowly, averages PWM signal
  - Digital communication is less sensitive to noise than analog methods
    - PWM provides a digital encoding of an analog value
    - Much less vulnerable to noise
- PWM signal characteristics
  - *Modulation frequency*: how many pulses occur per second (fixed)
  - *Period*: 1/(modulation frequency)
  - *On-time*: amount of time that each pulse is on (asserted)
  - *Duty-cycle*: on-time/period
  - Adjust on-time (hence duty cycle) to represent the analog value





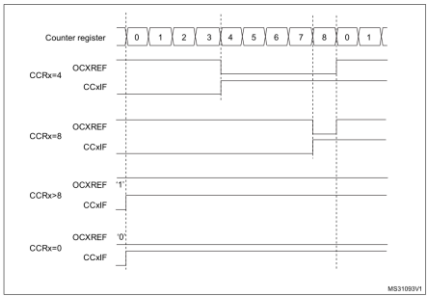

# Pulse-Width Modulation

- Timer also provides the PWM mode
  - generates a signal with frequency determined by the value of **ARR** and a duty cycle determined by the **CCR**.
- In **TIMx_CCMRx** register, set **OCxM** bits to 110 (PWM mode 1) or 111 (PWM Mode 2) (differs in polarity)
- Enable the corresponding preload register (**OCxPE** in **TIMx_CCMRx**)
- Enable the auto reload by setting ARPE bit in **TIMxCR1**
- Enable the output **OCx** in **TIMx_CCER CCxE** bit
- PWM done by comparing **TIMx_CCR** and **TIMx_CNT**, i.e., if **TIMx_CCRx≤TIMx_CNT** or **TIMx_CNT≤TIMx_CCRx** then output high or low.



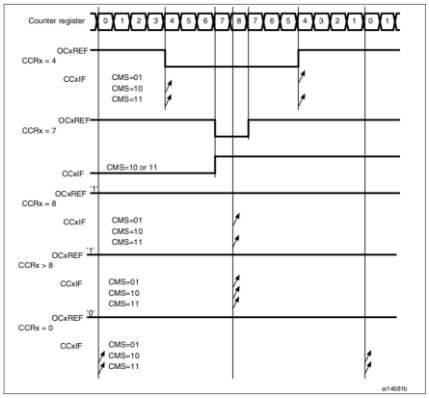
# PWM Modes

**edge-aligned**




- CMS bits in TIMx_CR1 are 00
- As long as TIMx_CNT<TIMx_CCRx then the OCXREF is high
- Select down-counting or up-counting using DIR bit in TIMx_CR1

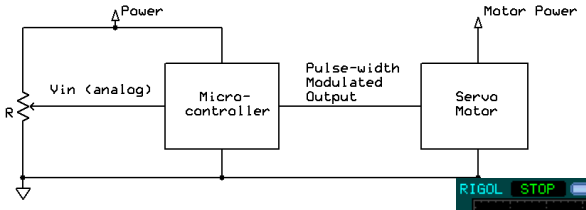
**center-aligned**

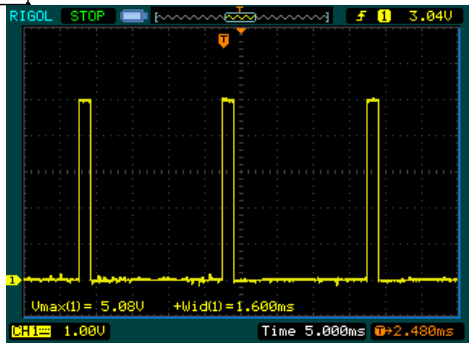



- CMS bits in TIMx_CR1 are not 00: many possible wave forms



# PWM to Drive Servo Motor




- Servo PWM signal
  - 20 ms period
  - 1 to 2 ms pulse width

# Example

- Configure TIM2 with prescaler dividing by 10 so that the timer counter is counting at 1.6MHz
- ARR register is loaded with 26666 and CCR1 is loaded with 8888
  - Timer reloads every 1sec
- Channel 1 is configured as PWM mode
- Output of Ch1 is turned on when the counter starts counting from 0
- When counter matches content of CCR1, Ch1 output is turned off
- When the counter matches ARR
  - the counter is cleared to 0
  - the output is turned on and
  - the counter starts counting up again
- The LED will be on for 8889/26667 = 30% of the time
- output pin of TIM2 Ch1 is PA5 and the alternate function of PA5 should be set to AF1



```
#include "stm32f4xx.h"

int main(void) {
   RCC->AHB1ENR |= 1;              /* enable GPIOA clock */

   GPIOA->AFR[0] |= 0x00100000; /* PA5 pin for tim2 */
   GPIOA->MODER &= ~0x00000C00;
   GPIOA->MODER |=  0x00000800;

   /* setup TIM2 */
   RCC->APB1ENR |= 1;              /* enable TIM2 clock */
   TIM2->PSC = 10 - 1;             /* divided by 10 */
   TIM2->ARR = 26667 - 1;          /* divided by 26667 */
   TIM2->CNT = 0;
   TIM2->CCMR1 = 0x0060;           /* PWM mode */
   TIM2->CCER = 1;                 /* enable PWM Ch1 */
   TIM2->CCR1 = 8889 - 1;          /* pulse width 1/3 of the period */
   TIM2->CR1 = 1;                  /* enable timer */

   while(1) {
   }
}
```