

Additional Timer Functionalities

Capture, Compare and PWM

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory

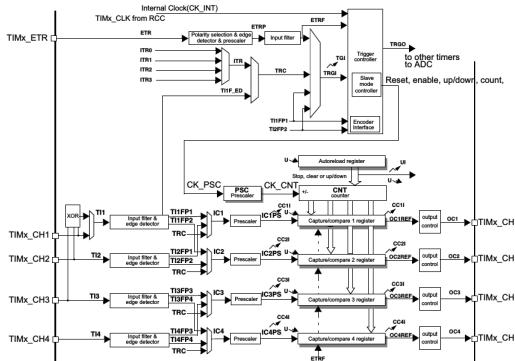
Dipartimento di Matematica e Informatica - Università di Catania, Italy

santoro@dmf.unict.it

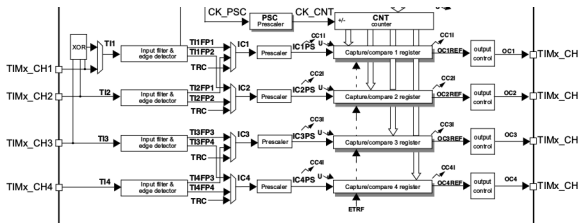


L.A.P. 1 Course

Timer Slave Circuits



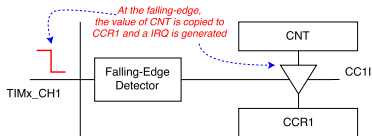
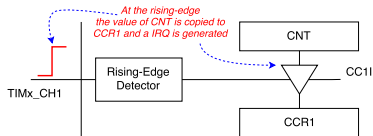
- Timers has **slave circuits** that can be used
 - to generate particular periodic signals
 - to measure the period or pulse of input signals



- Each STM32 timer has **four slave channels**
- Each channel can be **connected to a GPIO line** and can be programmed for:
 - **input capture**
 - **PWM input measurement**
 - **output compare**
 - **PWM output generation**
 - Each channel as an additional register **CCRx** (y is the channel number) that has the same size of the counter register

Input Capture

Input Capture



- When a channel **y** is configured as **input capture** the relevant GPIO line must be configured as **input**
- The capture circuit is able to **detect an edge (rising or falling)** in the input signal, according to configuration
- Then the **edge occurs** the value of the **CNT** is copied to **CCRx**
- A **capture-event interrupt** is generated

stm32_unict_lib Functions for Input Capture

- **Configure the channel:**

```
void TIM_config_IC(TIM_TypeDef * tim,  
                  int channel, ICMode mode)
```

- **tim**, the Timer
- **channel**, the slave channel, from 1 to 4
- **mode**, the capture mode, **IC_RISING_EDGE**, **IC_FALLING_EDGE**, **IC_BOTH_EDGES**

- **Check a capture event:**

```
int TIM_cc_check(TIM_TypeDef * tim, int channel)
```

- **tim**, the Timer
- **channel**, the slave channel, from 1 to 4
- returns "true" or "false"

- **Clear a capture event:**

```
void TIM_cc_clear(TIM_TypeDef * tim, int channel)
```

- **tim**, the Timer
- **channel**, the slave channel, from 1 to 4

Case-Study: a “Reaction Test” Game

- We want to measure the **reaction time** of a person
- We lit a LED and wait for the person to press a button
- We measure the **time** between LED lighting and button press
- After 5 secs, we notify a **timeout**

Case-Study: a “Reaction Test” Game

- We want to measure the **reaction time** of a person
 - We use a timer with a “capture” function
- We lit a LED
 - We set to “1” the GPIO and reset the timer
- and wait for the person to press a button
 - We wait for capture interrupt
- We measure the **time** between LED lighting and button press
 - We read the capture-compare register
- After 5 secs, we notify a **timeout**
 - We use the “period match” (ARR)

Case-Study: a “Reaction Test” Game (1)

```
#include "stm32_unict_lib.h"

int main()
{
    DISPLAY_init();
    GPIO_init(GPIOB);
    GPIO_config_output(GPIOB, 0); // the LED
    GPIO_config_alternate(GPIOB, 4, 2); // button "Y" as AF2 --> TIM3,CH1

    TIM_init(TIM3);
    TIM_config_timebase(TIM3, 8400, 50000); // prescaler at 100us timeout of 5 secs
    TIM_config_IC(TIM3, 1, IC_FALLING_EDGE); // capture on falling-edge (pushbutton)
    TIM_set(TIM3, 0);
    TIM_on(TIM3);
    TIM_enable_irq(TIM3, IRQ_UPDATE | IRQ_CC1);

    ...
}
```

Case-Study: a “Reaction Test” Game (2)

```
int main()
{
    ...
    for (;;) {
        delay_ms(3000); // wait 3 seconds to start the game
        TIM_set(TIM3, 0); // clear timer and flags
        measure_done = 0;
        timeout = 0;
        GPIO_write(GPIOB, 0, 1); // turn on LED
        DISPLAY_puts(0, "    ");
        while (!measure_done && !timeout) ; // wait for capture or timeout
        if (measure_done) {
            // capture OK!
            int ms = TIM3->CCR1 / 10;
            char s[4];
            sprintf(s, "%4d", ms);
            DISPLAY_puts(0, s);
        }
        else {
            // timeout
            DISPLAY_puts(0, "tout");
        }
    }
}

...
```

Case-Study: a “Reaction Test” Game (3)

```
void TIM3_IRQHandler(void)
{
    if (TIM_update_check(TIM3)) {
        // period match, timeout
        GPIO_write(GPIOB, 0, 0);
        timeout = 1;
        TIM_update_clear(TIM3);
    }
    if (TIM_cc_check(TIM3,1)) {
        // capture
        GPIO_write(GPIOB, 0, 0);
        measure_done = 1;
        TIM_cc_clear(TIM3,1);
    }
}
```

PWM Output

PWM Output

- A channel **y** can be configured to generate a **PWM Signal** with specific **frequency/period and duty cycle**
- the **period** is given in terms of **count units** and the value must be assigned to **ARR** register
- the duty cycle is specified as the duration of **positive part**, it is given in terms of **count units** and the value must be assigned to **CCRy** register

- To configure a channel in PWM mode the following function is provided:

```
void TIM_config_PWM(TIM_TypeDef * tim, int channel)
```

- **tim**, the Timer
 - **channel**, the slave channel, from 1 to 4
- The **period** is set as the “autoreload” parameter in the **TIM_config_timebase** function
 - The **duty cycle** is set by assigning the **CCRx** register

Case-Study: LED intensity control using PWM

```
#include <stdio.h>
#include "stm32_unict_lib.h"

#define PERIOD_PWM    1000

int main(void)
{
    DISPLAY_init();
    GPIO_init(GPIOB);
    GPIO_config_alternate(GPIOB, 0, 2); // AF2 --> TIM3_CH3

    ADC_init(ADC1, ADC_RES_8, ADC_ALIGN_RIGHT);
    ADC_channel_config(ADC1, GPIOC, 0, 10);
    ADC_on(ADC1);
    ADC_sample_channel(ADC1, 10);

    TIM_init(TIM3);
    TIM_config_timebase(TIM3, 420, PERIOD_PWM);
    TIM_set(TIM3, 0);
    TIM_config_PWM(TIM3, 3); // channel 3
    TIM_on(TIM3);

    for (;;) {
        ADC_start(ADC1);
        while (!ADC_completed(ADC1)) {}
        int t_on = (ADC_read(ADC1) * 1000)/255;
        TIM3->CCR3 = t_on;
        char s[4];
        sprintf(s, "%4d", t_on);
        DISPLAY_puts(0,s);
    }
    return 0;
}
```

Additional Timer Functionalities

Capture, Compare and PWM

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory

Dipartimento di Matematica e Informatica - Università di Catania, Italy

santoro@dmf.unict.it



L.A.P. 1 Course