LAB: Input Capture - Ultrasonic

Date: 2023-10-07

Author: EunChan Kim 21801017

Demo Video: [Youtube link]

Introduction

In this lab, you are required to create a simple program that uses input capture mode to measure the distance using an ultrasonic distance sensor. The sensor also needs trigger pulses that can be generated by using the timer output.

Requirement

Hardware

- MCU
  - NUCLEO-F411RE
- Actuator/Sensor/Others:
  - HC-SR04
  - breadboard

Software

- Keil uVision, CMSIS, EC_HAL library

Problem 1: Create HAL library

Create HAL library

ecTIM.c

According to the STM32 reference manual, the Input Capture function has been completed. The code has been structured to ensure proper operation based on the timer and channel parameters provided.

```
/* -------- Timer Input Capture -------- */

void ICAP_init(PinName_t pinName){
// 0. Match Input Capture Port and Pin for TIMx
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int TIn;
```

```
    ICAP_pinmap(pinName, &TIMx, &TIn);
    int ICn = TIn;                                                  // (default) 
TIx=ICx

// GPIO configuration -----------------------------------------------------------
----------
// 1. Initialize GPIO port and pin as AF
    GPIO_init(port, pin, EC_AF);                             // AF=2
    GPIO_ospeed(port, pin, EC_HIGH);                         // speed VHIGH=3

// 2. Configure GPIO AFR by Pin num.
    if(TIMx == TIM1 || TIMx == TIM2)                 
port->AFR[pin >> 3] |= 0x01 << (4*(pin % 8)); // TIM1~2
    else if(TIMx == TIM3 || TIMx == TIM4 || TIMx == TIM5)  port->AFR[pin >> 3] 
|= 0x02 << (4*(pin % 8)); // TIM3~5
    else if(TIMx == TIM9 || TIMx == TIM10|| TIMx == TIM11) port->AFR[pin >> 3] 
|= 0x03 << (4*(pin % 8)); // TIM9~11


// TIMER configuration -----------------------------------------------------------
-----------
// 1. Initialize Timer Interrpt
    TIM_UI_init(TIMx, 1);                            // TIMx Interrupt initialize

// 2. Modify ARR Maxium for 1MHz
    TIMx->PSC = 84-1;                                              // Timer counter 
clock: 1MHz(1us)  for PLL
    TIMx->ARR = 0xFFFF;                                            // Set auto 
reload register to maximum (count up to 65535)

// 3. Disable Counter during configuration
    TIMx->CR1 &= ~TIM_CR1_CEN;                               // Disable Counter 
during configuration



// Input Capture configuration -----------------------------------------------------
-------------------
// 1. Select Timer channel(TIx) for Input Capture channel(ICx)
    // Default Setting
    TIMx->CCMR1 &=  ~TIM_CCMR1_CC1S;
    TIMx->CCMR1 &=  ~TIM_CCMR1_CC2S;
    TIMx->CCMR2 &=  ~TIM_CCMR2_CC3S;
    TIMx->CCMR2 &=  ~TIM_CCMR2_CC4S;
    TIMx->CCMR1 |=  TIM_CCMR1_CC1S_0;       //01<<0   CC1S    TI1=IC1
    TIMx->CCMR1 |=  TIM_CCMR1_CC2S_0;           //01<<8   CC2s    TI2=IC2
    TIMx->CCMR2 |=  TIM_CCMR2_CC3S_0;       //01<<0   CC3s    TI3=IC3
    TIMx->CCMR2 |=  TIM_CCMR2_CC4S_0;           //01<<8   CC4s    TI4=IC4


// 2. Filter Duration (use default)

// 3. IC Prescaler (use default)

// 4. Activation Edge: CCyNP/CCyP
    TIMx->CCER &= ~(0b1010 << 4*(ICn -1));                  // CCy(Rising) for 
ICn
```

```c
// 5.   Enable CCy Capture, Capture/Compare interrupt
    TIMx->CCER |= 1 << (ICn -1);                    // CCn(ICn) Capture Enable

// 6.   Enable Interrupt of CC(CCyIE), Update (UIE)
    TIMx->DIER |= 1 << 0;                // Capture/Compare Interrupt Enable for
ICn
    TIMx->DIER |= TIM_DIER_UIE;                         // Update Interrupt
enable

// 7.   Enable Counter
    TIMx->CR1    |= TIM_CR1_CEN;                            // Counter enable
}


// Configure Selecting TIx-ICy and Edge Type
void ICAP_setup(PinName_t pinName, int ICn, int edge_type){
// 0. Match Input Capture Port and Pin for TIMx
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int CHn;
    ICAP_pinmap(pinName, &TIMx, &CHn);

// 1. Disable  CC. Disable CCInterrupt for ICn.
    TIMx->CCER &= ~(1 << (4*(ICn - 1)));
// Capture Enable
    TIMx->DIER &= ~(1 << ICn);
// CCn Interrupt enabled


// 2. Configure  IC number(user selected) with given IC pin(TIMx_CHn)
    switch(ICn){
            case 1:
                    TIMx->CCMR1 &= ~TIM_CCMR1_CC1S;
//reset   CC1S
                    if (ICn==CHn) TIMx->CCMR1 |=    TIM_CCMR1_CC1S_0;
//01<<0   CC1S    Tx_Ch1=IC1
                    else TIMx->CCMR1 |=     TIM_CCMR1_CC1S_1;
//10<<0   CC1S    Tx_Ch2=IC1
                    break;
            case 2:
                    TIMx->CCMR1 &= ~TIM_CCMR1_CC2S;
//reset   CC2S
                    if (ICn==CHn) TIMx->CCMR1 |=    TIM_CCMR1_CC2S_0;
//01<<0   CC2S    Tx_Ch2=IC2
                    else TIMx->CCMR1 |=     TIM_CCMR1_CC2S_1;
//10<<0   CC2S    Tx_Ch1=IC2
                    break;
            case 3:
                    TIMx->CCMR2 &= ~TIM_CCMR2_CC3S;
//reset   CC3S
                    if (ICn==CHn) TIMx->CCMR2 |=    TIM_CCMR2_CC3S_0;
 //01<<0   CC3S    Tx_Ch3=IC3
                    else TIMx->CCMR2 |=     TIM_CCMR2_CC3S_1;
//10<<0   CC3S    Tx_Ch4=IC3
                    break;
```

```
            case 4:
                    TIMx->CCMR2 &= ~TIM_CCMR2_CC4S;
//reset   CC4S
                    if (ICn==CHn) TIMx->CCMR2 |=    TIM_CCMR2_CC4S_0;
//01<<8   CC4S    Tx_Ch4=IC4
                    else TIMx->CCMR2 |=     TIM_CCMR2_CC4S_1;
//10<<8   CC4S    Tx_Ch3=IC4
                    break;
            default: break;
        }


// 3. Configure Activation Edge direction
    TIMx->CCER  &= ~(0b1010 << 4*(ICn - 1));                    // Clear
CCnNP/CCnP bits
    switch(edge_type){
        case IC_RISE: TIMx->CCER &= ~(0b1010 << 4*(ICn - 1)); break; //rising:
00
        case IC_FALL: TIMx->CCER |= (0b0010 << 4*(ICn - 1));     break;
//falling: 01
        case IC_BOTH: TIMx->CCER |= (0b1010 << 4*(ICn - 1));     break; //both:
   11
    }

// 4. Enable CC. Enable CC Interrupt.
    TIMx->CCER |= 1 << (4*(ICn - 1));                                       //
Capture Enable
    TIMx->DIER |= 1 << ICn;
// CCn Interrupt enabled
}

// Time span for one counter step
void ICAP_counter_us(PinName_t pinName, int usec){
// 0. Match Input Capture Port and Pin for TIMx
    GPIO_TypeDef *port;
    unsigned int pin;
    ecPinmap(pinName, &port, &pin);
    TIM_TypeDef *TIMx;
    int CHn;
    ICAP_pinmap(pinName, &TIMx, &CHn);

// 1. Configuration Timer Prescaler and ARR
    TIMx->PSC = 84*usec-1;                        // Timer counter clock: 1us *
usec
    TIMx->ARR = 0xFFFF;                                 // Set auto reload
register to maximum (count up to 65535)
}

uint32_t is_CCIF(TIM_TypeDef *TIMx, uint32_t ccNum){
    return (TIMx->SR & (0x1UL << ccNum)) != 0;
}

void clear_CCIF(TIM_TypeDef *TIMx, uint32_t ccNum){
    TIMx->SR &= ~(1 << ccNum);
}

uint32_t ICAP_capture(TIM_TypeDef* TIMx, uint32_t ICn){
    uint32_t capture_Value;
```

```c
    if (ICn == 1)
        capture_Value = TIMx->CCR1;
    else if (ICn == 2)
        capture_Value = TIMx->CCR2;
    else if (ICn == 2)
        capture_Value = TIMx->CCR3;
    else
        capture_Value = TIMx->CCR4;

    return capture_Value;
}

//DO NOT MODIFY THIS
void ICAP_pinmap(PinName_t pinName, TIM_TypeDef **TIMx, int *chN){
     GPIO_TypeDef *port;
     unsigned int pin;
     ecPinmap(pinName, &port, &pin);

   if(port == GPIOA) {
      switch(pin){
         case 0 : *TIMx = TIM2; *chN = 1; break;
         case 1 : *TIMx = TIM2; *chN = 2; break;
         case 5 : *TIMx = TIM2; *chN = 1; break;
         case 6 : *TIMx = TIM3; *chN = 1; break;
         //case 7: *TIMx = TIM1; *chN = 1N; break;
         case 8 : *TIMx = TIM1; *chN = 1; break;
         case 9 : *TIMx = TIM1; *chN = 2; break;
         case 10: *TIMx = TIM1; *chN = 3; break;
         case 15: *TIMx = TIM2; *chN = 1; break;
         default: break;
      }
   }
   else if(port == GPIOB) {
      switch(pin){
         //case 0: *TIMx = TIM1; *chN = 2N; break;
         //case 1: *TIMx = TIM1; *chN = 3N; break;
         case 3 : *TIMx = TIM2; *chN = 2; break;
         case 4 : *TIMx = TIM3; *chN = 1; break;
         case 5 : *TIMx = TIM3; *chN = 2; break;
         case 6 : *TIMx = TIM4; *chN = 1; break;
         case 7 : *TIMx = TIM4; *chN = 2; break;
         case 8 : *TIMx = TIM4; *chN = 3; break;
         case 9 : *TIMx = TIM4; *chN = 3; break;
         case 10: *TIMx = TIM2; *chN = 3; break;

         default: break;
      }
   }
   else if(port == GPIOC) {
      switch(pin){
         case 6 : *TIMx = TIM3; *chN = 1; break;
         case 7 : *TIMx = TIM3; *chN = 2; break;
         case 8 : *TIMx = TIM3; *chN = 3; break;
         case 9 : *TIMx = TIM3; *chN = 4; break;

         default: break;
      }
```

```
    }
}
```

Problem 2: Ultrasonic Distance Sensor (HC-SR04)

The HC-SR04 Ultrasonic Range Sensor Features:

- Input Voltage: 5V
- Current Draw: 20mA (Max)
- Digital Output: 5V
- Digital Output: 0V (Low)
- Sensing Angle: 30° Cone
- Angle of Effect: 15° Cone
- Ultrasonic Frequency: 40kHz
- Range: 2cm - 400cm

Procedure

- Connect the ultrasonic sensor to the board and configure the trigger and echo pins.
- Set up a timer to measure the distance based on the time it takes for the ultrasonic sensor to detect an object's presence.
- Convert the time measured by the timer into distance and display the result.

Measurement of Distance

- Generate a trigger pulse as PWM to the sensor.
- Receive echo pulses from the ultrasonic sensor
- Measure the distance by calculating pulse-width of the echo pulse.
- Display measured distance in [cm] on serial monitor of Tera-Term for

  (a) 10mm (b) 50mm (c) 100mm

Configuration

System Clock	PWM	Input Capture
PLL (84MHz)	PA6 (TIM3_CH1)	PB6 (TIM4_CH1)
	AF, Push-Pull, No Pull-up Pull-down, Fast	AF, No Pull-up Pull-down
	PWM period: 50msec pulse width: 10usec	Counter Clock : 0.1MHz (10us) TI4 -> IC1 (rising edge) TI4 -> IC2 (falling edge)

Circuit Diagram

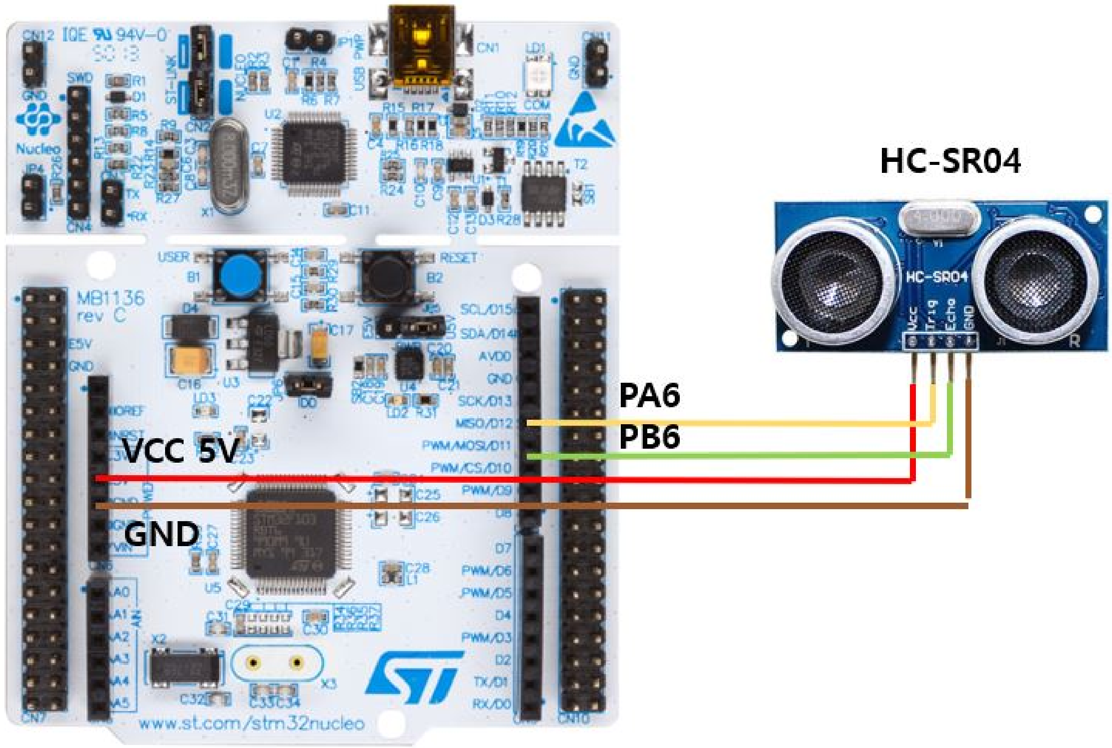


Figure1. Ultrasonic Distance Sensor Circuit

Discussion

1. **There can be an over-capture case, when a new capture interrupt occurs before reading the CCR value. When does it occur and how can you calculate the time span accurately between two captures?**

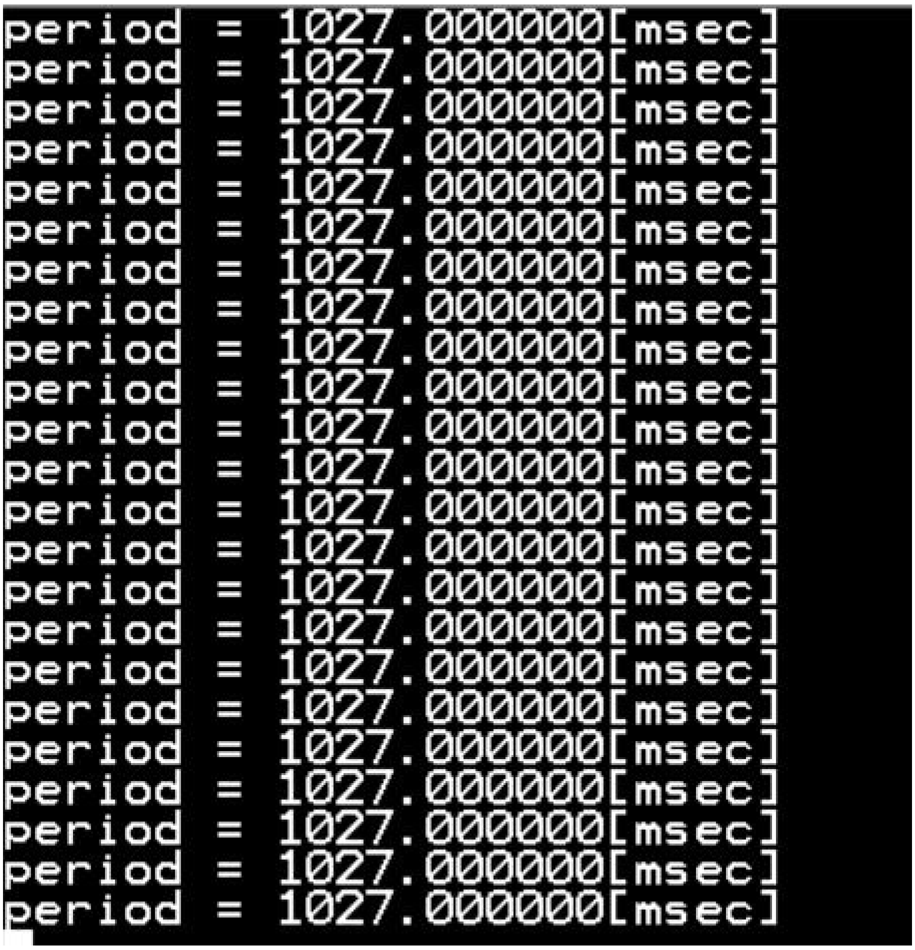
Over-capture is referred to a case where a new capture interrupt occurs before the CCR value is read. This happens when the interrupt processing is not fast enough and the next capture event occurs before the previous capture value is read. To address this problem, most timer hardware is designed to detect over-capture and set a flag, allowing the software to verify it.

In order to accurately calculate the time between two captures, the timestamp captured from the CCR must be preserved first. When the next capture event occurs, the new timestamp is subtracted from the previous value to calculate the time between the two captures.

In the code, the time between two captures is calculated by implementing it as `timeInterval = 10 * ((ovf_cnt * (TIM4->ARR+1)) + (time2 - time1));` `ovf_cnt` represents the number of timer overflows, which increases when an over-capture occurs. It is designed to calculate the time between two captures considering the over-capture.

2. **In the tutorial, what is the accuracy when measuring the period of 1Hz square wave? Show your result.**

When a 1Hz signal was applied using the Function Generator, the period was measured as 1027ms. Therefore, an error of 2.7% occurred. It is presumed that the error occurred due to the processing delay present in all interrupts. This delay refers to the time between when the interrupt occurs and when the ISR actually runs. This delay can be caused by various factors and can result in errors in the measured period.



```
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
period = 1027.000000[msec]
```

Figure2. the period of 1Hz square wave

Code

LAB_Timer_InputCaputre_Ultrasonic.c

In the `TIM4_IRQHandler(void)`, the time interval is measured using timer 4's Interrupt Capture channels 1 and 2. `time1` is used as a variable to measure the start time of the ultrasonic waveform, which is the time captured at TIM4 channel 1 (IC1). `time2` is used as a variable to measure the end time of the ultrasonic waveform, which is the time captured at TIM4 channel 2 (IC2). `ovf_cnt` acts as a timer overflow counter, which increases its count each time a timer overflow occurs due to the prolonged measurement of the ultrasonic waveform. `TIM4->ARR+1` represents Timer 4's Auto Reload Register. This value determines the period at which the timer overflows. Using these variables, the time interval of the ultrasonic waveform is measured as `timeInterval`.

In the `setup()`, pins, period, and pulse width were set according to the configuration. And also In the `main()`, the measured distance is converted from [mm] to [cm] and is set to be output at 0.5 second intervals in Tera Term.

0.034 represents the speed of ultrasound through air, expressed in mm/μs. This value is approximately 34000mm per second, representing the speed at which ultrasound travels through air. We use 0.034 to convert this value into μs. The division by 2 accounts for the roundtrip time. The measured time interval, `timeInterval`, represents the total time from the generation of the ultrasound to its reflection and return. However, the distance we need is from the sensor to the obstacle, so we divide the roundtrip time by 2.

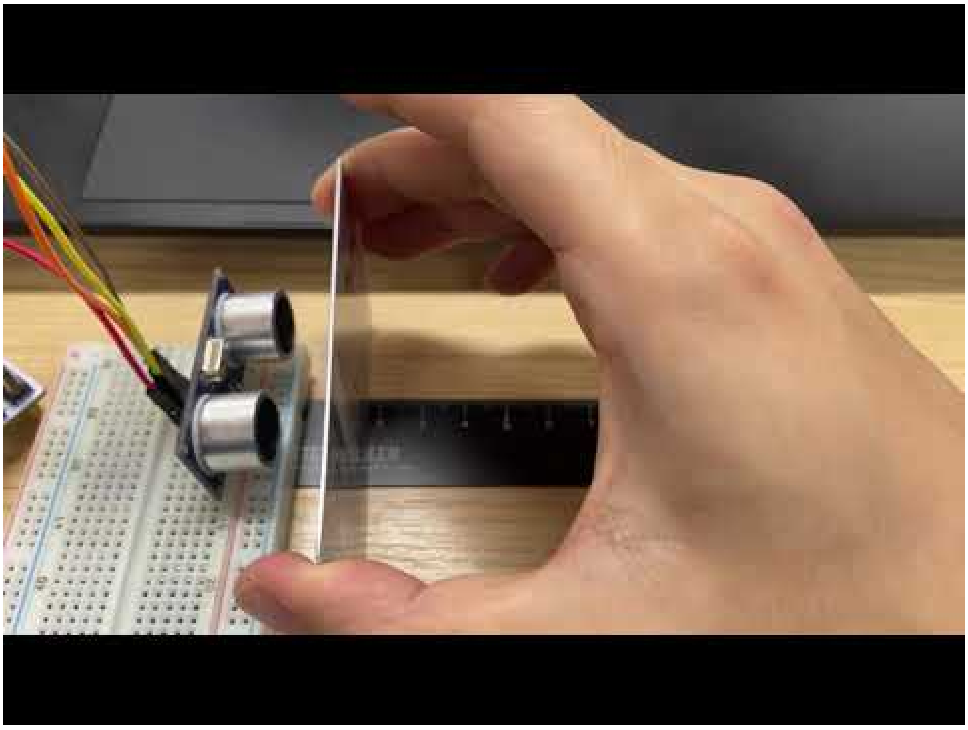
```
uint32_t ovf_cnt = 0;
float distance = 0;
float timeInterval = 0;
float time1 = 0;
float time2 = 0;

#define TRIG PA_6
#define ECHO PB_6

void setup(void);

int main(void){

    setup();

    while(1){
        distance = (float) timeInterval * 0.034 / 2;    // [mm] -> [cm]
        printf("%f cm\r\n", distance);
        delay_ms(500);
    }
}

void TIM4_IRQHandler(void){
    if(is_UIF(TIM4)){                          // Update interrupt
        ovf_cnt++;                                                 // overflow
count
        clear_UIF(TIM4);                                 // clear update
interrupt flag
    }
    if(is_CCIF(TIM4, 1)){                            // TIM4_Ch1 (IC1)
Capture Flag. Rising Edge Detect
        time1 = ICAP_capture(TIM4, IC_1);                              //
Capture TimeStart
        clear_CCIF(TIM4, 1);                 // clear capture/compare interrupt
flag
    }
    else if(is_CCIF(TIM4, 2)){                               // TIM4_Ch2
(IC2) Capture Flag. Falling Edge Detect
        time2 = ICAP_capture(TIM4, IC_2);                              //
Capture TimeEnd
        timeInterval = 10 * ((ovf_cnt * (TIM4->ARR+1)) + (time2 - time1));  //
(10us * counter pulse -> [msec] unit) Total time of echo pulse
        ovf_cnt = 0;                         // overflow reset
        clear_CCIF(TIM4,2);                              // clear
capture/compare interrupt flag
    }
}

void setup(){
```

```
    RCC_PLL_init();
    SysTick_init();
    UART2_init();

// PWM configuration -------------------------------------------------------------
---------
    PWM_init(TRIG);         // PA_6: Ultrasonic trig pulse
    PWM_period_us(TRIG, 50000);     // PWM of 50ms period. Use period_us()
    PWM_pulsewidth_us(TRIG, 10);    // PWM pulse width of 10us


// Input Capture configuration ---------------------------------------------------
---------------------
    ICAP_init(ECHO);        // PB_6 as input caputre
    ICAP_counter_us(ECHO, 10);      // ICAP counter step time as 10us
    ICAP_setup(ECHO, 1, IC_RISE);   // TIM4_CH1 as IC1 , rising edge detect
    ICAP_setup(ECHO, 2, IC_FALL);   // TIM4_CH2 as IC2 , falling edge detect

}
```

Results

As can be seen in the video, it can be confirmed that the measurements are accurate at distances of 50 and 100[mm], except for 10[mm]. The errors occurring at 50 and 100 [mm] are attributed to the measurement error of 2.7%. In the case of 10[mm], inaccurate measurements are shown because the minimum measuring distance of the ultrasonic sensor used is 20[mm].

Demo video↓



Reference

- STM32 Cortex®-M4 MCUs and MPUs programming manual [Download Link]
- STM32 Cortex®-M4 MCUs and MPUs reference manual [Download Link]