LAB REPORT: EMBEDDED COMPUTING

LAB III: ULTRASONIC SENSOR

Tailei WANG, Hao DENG

Electric Vehicle Propulsion and Control (E-PiCo)

ÉCOLE CENTRALE DE NANTES

January 2022

1 Principle

This lab focuses on the interrupts, but still uses GPIOs and timers. The objective is to write the driver for an ultrasonic sensor, without any polling implementation.

The sensor embedds an integrated circuit for an easy management by the MCU. The interface consists in only 2 logic signals (Trigger and echo), as in figure 1. The sensor works as follow(figure 2):

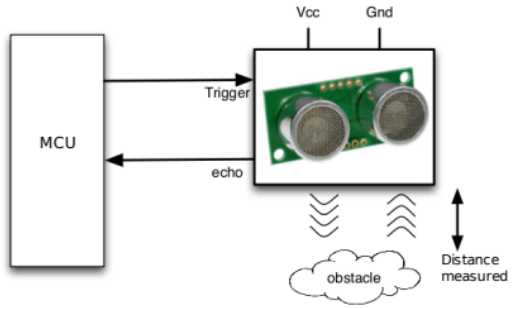

Figure 1: Connection between the ultrasonic sensor and the MCU.

- a pulse to the Trigger signal is a request to perform a measure.
- the integrated circuit generates a sequence of 40KHz ultrasonic burst.
- an analog circuit detects the echo from an obstacle.
- The distance to the obstacle is deduced from the time elapsed between the trigger ultrasonic burst and the incoming echo, knowing the sound speed. A pulse is transmitted back on the echo signal, on which the pulse width depends on the distance to the obstacle: 58 s/cm.

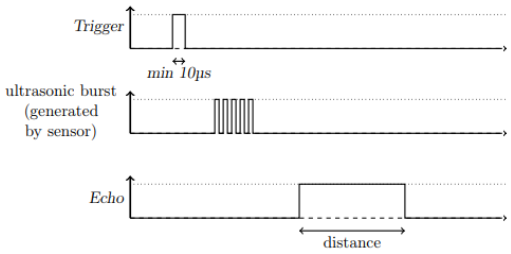

Figure 2: Time sequence of the ultrasonic sensor.

The echo pulse width depends on the distance to the obstacle(58us/cm) In our configuration, signals Trigger and Echo share the same pin (PA10), we will have to change its configuration dynamically (output/input).

2 Trigger Signal

The Trigger signal is just a pulse that should not be too fast, so that the integrated electronic in the sensor can detect it. The signal should be in high state for at least 10µs.

We will insert a simple waiting loop to respect this constraint:

```
// set PA10
for( volatile int i =0; i <20; i ++);
// reset PA10
```

We will send Trigger request at 10Hz, using an interrupt and timer TIM6. To be sure that the Trigger signal is detected by the sensor, we just wait a little:

For an easy debug, we will toggle the green led PB0 in the interrupt handler.

Question 1 *Write the trigger part of the application:*

- *configuration of the sensor pin PA10 as an output.)*
- *interrupt @10Hz using TIM6.*
- *generation of the Trigger signal.*

There are 3 functions defined in our code. The trigger function configure PA10 as an output and setup function configure a period at 0.1s(10Hz) for TIM6 by setting the prescaler register and ARR register. The ConfigIT function generate an interrupt by using TIM6.

```
void trigger(){
    pinMode(GPIOA,10, OUTPUT);
        digitalWrite(GPIOA,10,1);
        // Use a for loop to wait a while
    wait();
        digitalWrite(GPIOA,10,0);
        pinMode(GPIOA,10, INPUT);
}
void setup(){
        // Timer6 Configuration
        //input clock = 64MHz.
    RCC->APB1ENR |= RCC_APB1ENR_TIM6EN;
    __asm("nop");
    //reset peripheral (mandatory!)
    RCC->APB1RSTR |= RCC_APB1RSTR_TIM6RST;
        RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM6RST;
        __asm("nop");
        TIM6->PSC = 64000-1;
        TIM6->CNT = 0;
        TIM6->ARR = 100-1;
        TIM6->CR1 |= TIM_CR1_CEN;
    //prescaler  : tick@1ms
    //auto-reload: counts 100 ticks
        //config reg : enable
    RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM6RST;
        __asm("nop");
    while(! (TIM6->SR & TIM_SR_UIF)); //wait...
    //I/O signal, pinMode setting
    pinMode(GPIOB,0, OUTPUT);
}
void configIT() {
    //local validation
    TIM6->DIER |= TIM_DIER_UIE;
        //NVIC validation
        NVIC_EnableIRQ(TIM6_DAC1_IRQn);
    //Interrupt prority setting
        NVIC_SetPriority (TIM6_DAC1_IRQn,1) ;
}
```

NOTE: pin PA10 shares both Trigger (output) and Echo (input) signals. As a consequence, the pin should be in the output mode as little time as possible.

3 Echo Signal

The distance to the obstacle is given by the duration of the Echo pulse. We use the EXTI peripheral to detect rising/falling edges, and a timer (TIM7) as a stopwatch to get the pulse width. The measure is stored in a global variable (figure 3).

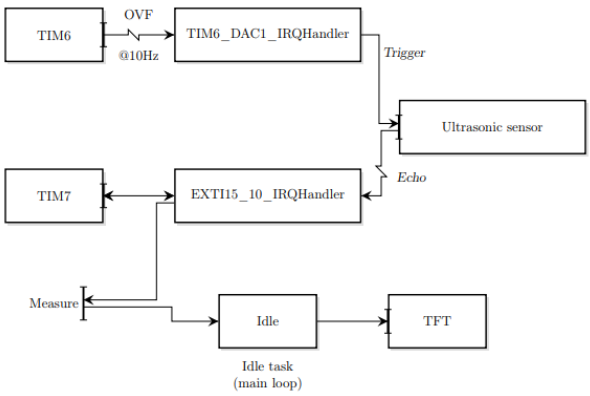

Figure 3: Electronic control interface.

The principle uses TIM7 as a chronometer (resolution 1µs) and an interrupt in Echo:

- if there is a rising edge on Echo, TIM7 count value is reset)
- if there is a falling edge on Echo, the value of TIM7 is stored to the measure value.

To determine if an interrupt is a consequence of a rising or falling edge, you just have to read the logical level of the pin.

Important note: The Trigger signal should be updated, because it will generate an interrupt (modifying the output state generates an interrupt). So the EXTI interrupt should be deconfigured during the trigger generation, with IMR (Interrupt Mask Register):

```
EXTI -> IMR &= ~EXTI_IMR_MR10 ;
// generate trigger signal ..
EXTI -> IMR |= EXTI_IMR_MR10 ;
```

Question 2 *Program the whole application, that refresh the value of measure periodically. check the correct value using the debugger.*

To implement this, the Timer7 and an external interrupt are configured. The function setupT7 is used to configure Timer7 and EXTIConfig is used to configure the External interrupt.The related interrupt handler of the EXTI is also configured by the function EXTI15_10_IRQHandler. Correspondingly, the trigger function is updated and the new trigger function is shown in the code below.

```
1 void setupT7(){
2 //Configure Timer7
3     RCC->APB1ENR |= RCC_APB1ENR_TIM7EN;
4     __asm("nop");
5     RCC->APB1RSTR |= RCC_APB1RSTR_TIM7RST;
6         RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM7RST;
```

```
        __asm("nop");
        TIM7->PSC = 64-1;
        TIM7->CR1 |= TIM_CR1_CEN;
    //Configuration of GPIOA
        //clock for GPIOA
        RCC->AHBENR |= RCC_AHBENR_GPIOAEN_Msk;
        //wait until GPIOB clock is Ok.
    __asm("nop");
}
void EXTIConfig(){}
        //Configuration of EXTI
    //attachInterrupt();
    EXTI->IMR |= EXTI_IMR_MR10;
        EXTI->FTSR |= EXTI_FTSR_TR10;
        //EXTI> FTSR |= EXTI_FTSR_TR10;
    SYSCFG->EXTICR[2]|= SYSCFG_EXTICR3_EXTI10_PA;
        NVIC_SetPriority (EXTI15_10_IRQn,3);
        NVIC_EnableIRQ(EXTI15_10_IRQn);
}
void trigger(){
        EXTI->IMR &= ~EXTI_IMR_MR10;
    pinMode(GPIOA,10, OUTPUT);
        digitalWrite(GPIOA,10,1);

        // Use a for loop to wait a while
    wait();
        digitalWrite(GPIOA,10,0);
        pinMode(GPIOA, 10, INPUT);
    EXTI->IMR |= EXTI_IMR_MR10;
}
extern "C" {
void EXTI15_10_IRQHandler(){
if (digitalRead(GPIOA,10) == 1){
        TIM7->CNT = 0;
        }
else{
        measure = TIM7->CNT;
        }
EXTI->PR |= EXTI_PR_PR10;
}
}
```

Question 3 *Add a continuous display of the sensor value on the TFT (in the main loop),in mm.*

The code below give setup of the printing configurations. The main function is also updated to implement the display function.

```
void setup(){
        Tft.setup();
        Tft.setTextCursor(4, 1); //col,line
        Tft.print("Coro! ");
        Tft.print("test");

        ADCInit(); //potentiometer
```

```
}
int main(void) {
        static unsigned int prevPot = 0;

     setup();
     configIT();
    while(1)
        {
                //potentiometer
                // int pot = ADCRead(); //12 bits -> [0,4095] -> needs 4 digits in decimal (prin
                //update only when value changes significantly
                if(prevPot != measure)
                {
                        //set cursor centered on line 4.
                        Tft.setTextCursor(5, 3);
                        Tft.eraseText(5);        //remove previous value (4 digits)
                        Tft.print(measure*10/58);
                        prevPot = measure;
                }
        }
 }
```

4 Extension

The echo signal should occurs less than 50ms after the trigger.

Question 4 *Add a timeout function that informs the user that the sensor is not available if there is no response after 50ms (using an interrupt of course - TIM7 for instance).*

```
 void setup ( ) {
 //reload for timeout(@50ms)
 TIM7->ARR = 50000-1; TIM7->CR1 |= TIM_CR1_CEN ;
 NVIC_SetPriority (TIM7_IRQn, 2 ) ;
 NVIC_EnableIRQ (TIM7_IRQ) }

 //Reset the current value
 void trigger(){
 TIM7->CNT = 0;
 }
 // Printing display
 serialPutString("sensor is not available");
 TIM->SR &= ~TIM_SR_UIF;
```