Centrale Nantes

MAC : 5^{th} Lab's Report
Interrupts — Ultrasonic sensor

1^{st} year Embedded Systems Engineering

By

EL KHAYDER Zakaria

SAOUTI Rayan

Professor

BRIDAY Mikael

December 24, 2023

Session

2023-2024

Made with LaTeX

Contents

Listings

1 Trigger Signal

1.1 Timer

We should set up one of the internal `STM32` timers as an internal interrupt. We chose to use `TIM6` for that.

```
void setup()
{
    // input clock = 64MHz.
    RCC->APB1ENR |= RCC_APB1ENR_TIM6EN | RCC_APB1ENR_TIM7EN;
    // reset peripheral (mandatory!)
    RCC->APB1RSTR |= RCC_APB1RSTR_TIM6RST | RCC_APB1RSTR_TIM7RST;
    RCC->APB1RSTR &= ~(RCC_APB1RSTR_TIM6RST | RCC_APB1RSTR_TIM7RST);

    // Configure timer
    TIM6->CNT = 0; // Reset counter
    TIM6->SR = 0; // Reset status register
    TIM6->PSC = 64000 - 1; // Prescaler: 64K => F = 1K => T = 1ms
    TIM6->ARR = 100 - 1; // 100ms
    TIM6->CR1 |= TIM_CR1_CEN; // Tiemr enable
}
```

Listing 1: TIM6 Configuration

1.2 Interrupt Configuration

```
void setup() {
    ...

    // enable interrupt
    TIM6->DIER |= TIM_DIER_UIE;
    NVIC_EnableIRQ(TIM6_DAC1_IRQn);
}
```

Listing 2: TIM6 Interrupt enable

1.3 Interrupt Handler: Trigger signal generation

Whenever we get a `TIM6` interrupt (each 100ms), we should send a 10μs HIGH signal in the `TRIG_ECHO` pin. Since the `TRIG_ECHO` serves both as an input and an output for our ultrasonic sensor, we should set the pin to output at the beginning of our interrupt handler, and then flip it back to output so we can read the echo signal from the sensor.

To handle our `TIM6` interrupt, we should declare a function with exactly `TIM6_DAC1_IRQHandler` as a

name, but given we are using C++ as our programming language and not C, we have to be careful of name mangling, hence declaring our interrupt handlers as external C functions using `extern "C"`

```
extern "C" void TIM6_DAC1_IRQHandler(void)
{
    pinMode(TRIG_ECHO, OUTPUT); // Set pin as OUTPUT

    digitalWrite(TRIG_ECHO, 1); // Set HIGH
    for (volatile int i = 0; i < 50; i++); //  A short delay (we measured around 12us)
    digitalWrite(TRIG_ECHO, 0); // Set LOW

    pinMode(TRIG_ECHO, INPUT); // Set the pin back to input

    TIM6->SR &= ~TIM_SR_UIF; // acknowledge
}
```

Listing 3: TIM6 Interrupt handler: Trig signal generation

2 Echo signal

2.1 Theory

Once the ultrasonic sensor has emitted the ultrasonic burst, it will set the `Echo` signal to HIGH until it receives back the signal it generated, and then it sets `Echo` signal back to LOW. The measured time between the Rising and the Falling edges is how long the ultrasonic burst took to travel back and forth ($2d$).

Given the ultrasonic burst travels at the speed of sound $s \approx 343 m/s$ (at 20 °C), we can pretty accurately estimate the distance between the object and the sensor up to ± 2cm.

We will be using `TIM7` with a resolution of 1 μs

$$
\begin{aligned}
s_{(m/s)} = \frac{2d_{(m)}}{t_{(s)}} \Rightarrow d_{(m)} &= \frac{s_{(m)} \cdot t_{(s)}}{2} \\
&= \frac{343_{(m/s)}}{2} \cdot t \\
&= 171.5_{(m/s)} \cdot t_{(s)} \\
\Rightarrow d_{(cm)} &= 171.5_{(m/s)} \cdot \frac{10^2_{(cm/m)}}{10^6_{(\mu s/s)}} \cdot t_{(\mu s)} \\
&= 0.01715_{(cm/\mu s)} \cdot t_{(\mu s)} \\
&\approx \frac{t_{(\mu s)}}{58_{(\mu s/cm)}}
\end{aligned}
$$

2.2 Setup

2.2.1 Timer

```
void setup() {
    ...

    // Configure timer
    TIM7->CNT = 0; // Reset Counter
    TIM7->SR = 0; // Reset status register
    TIM7->PSC = 64 - 1; // Prescaler: 64 => F = 1Mhz => T = 1us
    TIM7->ARR = 50000 - 1; // 50ms (will come in handy later)
    TIM7->CR1 |= TIM_CR1_CEN;
}
```

Listing 4: TIM7 Config

2.2.2 External Interrupt

Thanks to the provided function helpers, configuring external interrupts is easy.

We attach an external interrupt to `TRIG_ECHO` pin on each `CHANGE` (rising and falling edges)

```
void setup() {
    ...

    EXTI->IMR |= EXTI_IMR_MR10;
    attachInterrupt(TRIG_ECHO, CHANGE);
}
```

Listing 5: Echo interrupt setup

Since we are using the same pin for both `Trig` and `Echo`, and we are listening to changes on this pin now, this means that whenever we manipulate the pin for the `Trig` signal, we will get an interrupt because we change its state. We need to update the previous `Trig` signal generation code to disable the External interrupts on this pin before sending the signal, and then re-enabling it.

```
extern "C" void TIM6_DAC1_IRQHandler(void)
{
    EXTI->IMR &= ~EXTI_IMR_MR10; // Disable pin 10 external interrupt

    ...

    EXTI->IMR |= EXTI_IMR_MR10; // Re-enable pin 10 external interrupts

    TIM6->SR &= ~TIM_SR_UIF; // acknowledge
}
```

Listing 6: TIM6 Interrupt handler — Updated

2.3 Interrupt handler

Whenever we get an external interrupt from `Echo` pin, we have to determine whether it is a rising or falling edge:

- Rising: We reset the TIM7 counter
- Falling: We save the current TIM7 counter value in a global variable that we can access later on to calculate the distance using the previous equation.

```
volatile uint32_t dt_us = 0; // Let's not forget the volatile keyword so that the compiler
    does not do anything funny

extern "C" void EXTI15_10_IRQHandler(void)
{
    if (digitalRead(TRIG_ECHO) == 1) // Rizing edge
        TIM7->CNT = 0; // Reset
    else // Falling edge
        dt_us = TIM7->CNT; // Save

    EXTI->PR |= EXTI_PR_PR15; // acknowledge
}
```

Listing 7: Echo interrupt handler

3 Interface

Now that we have all the parts that we need, maybe we should display it to the user, you know... the purpose of this device?

We can use the functions already provided by `tft.h` to manipulate the display and draw the bar graph.

```
int main(void)
{
    setup();
    Tft.setup();

    while(1) {
        Tft.erase();

        uint32_t distance = dt_us / 58; // cm

        const int16_t MAX_HEIGHT = Tft.height() - 20,
                      MIN_HEIGHT = 4,
                      MAX_DISTANCE = 100,
                      MIN_DISTANCE = 2;
```

```
15
16          float per = (float)distance / (float)(MAX_DISTANCE - MIN_DISTANCE);
17
18          if (per > 1)
19              per = 1;
20
21          int16_t height = MIN_HEIGHT + (float)(MAX_HEIGHT - MIN_HEIGHT) * per;
22
23          Tft.fillRect(
24              Tft.width() / 2 - 25,
25              MAX_HEIGHT - height,
26              50,
27              height,
28              ST7735_BLUE * (1 - per) + ST7735_RED * per);
29
30          Tft.setTextCursor(3, 80);
31          Tft.print("Distance: ");
32          Tft.print(distance);
33          Tft.println(" cm");
34
35          for (volatile int i = 0; i < 2000000; i++); // Delay
36      }
37  }
```

Listing 8: Echo interrupt handler

4 Robustness

The last part is adding a mechanism to detect if the sensor is present or not. For this, we will start a 50ms timer whenever we send a `Trig` signal. If the duration elapses without receiving any response, we assume that the sensor is unavailable (disconnected or broken).

We can use the `TIM7` that we already use to get the travel time. If you can remember, we have already configured its `ARR` register to 50ms, the only thing left now is to enable an interrupt on it and handle it.

Whenever we send a `Trig` signal, we should enable the timer, and disable it on the first `Echo` pin state change. The code will be something like this.

```
1  volatile bool unavailable = false;
2
3  extern "C" void TIM6_DAC1_IRQHandler(void) {
4      ...
5
6      // Enable 7 interrupt
7      TIM7->DIER |= TIM_DIER_UIE;
8      NVIC_EnableIRQ(TIM7_DAC2_IRQn);
9      TIM7->CNT = 0;
```

```
}

extern "C" void EXTI15_10_IRQHandler(void)
{
    TIM7->DIER &= ~TIM_DIER_UIE;
    NVIC_DisableIRQ(TIM7_DAC2_IRQn);
    unavailable = false;

    ...
}

extern "C" void TIM7_DAC2_IRQHandler(void)
{
    unavailable = true;
    TIM7->CNT = 0;

    // acknowledge
    TIM7->SR &= ~TIM_SR_UIF;
}

int main(void) {
    ...

    while(1) {
        Tft.erase();

        if (unavailable) {
            Tft.setTextCursor(1, 1);
            Tft.println("Sensor unavailable");
        } else {
            ...
        }
    }
}
```

Listing 9: Sensor availability checker

Resources

The code files, and this report's source code, are available on this GitHub repository: elkhayder/sec1-tp-mac