Centrale Nantes

MAC : 6^{th} Lab's Report
Interrupts — Ultrasonic sensor

1^{st} year Embedded Systems Engineering

By

EL KHAYDER Zakaria

SAOUTI Rayan

Professor

BRIDAY Mikael

February 09, 2023

Session

2023-2024

Made with LaTeX

Contents

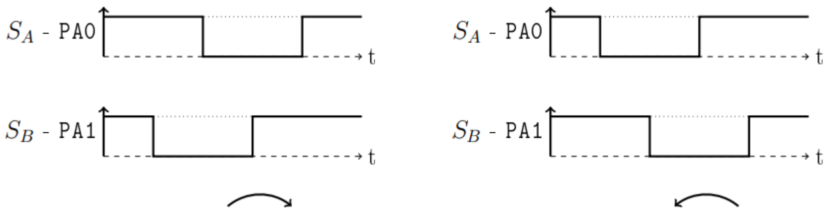
Listings

1 Encoder

Unlike the actual process we followed in this lab, we will start this report by implementing the `Rotating encoder` code given it is relatively the simplest in this project.

1.1 Logic

We should implement an external interrupt (falling edge) on one of the two encoder inputs, and read the state of the other one. With these two states, we can determine the rotation direction based on the following figure.



1.2 Implementation

1.2.1 Setup

We chose to link the external interrupt on S_A (PA0). Luckily this is easy thanks to the already provided `pinMode` and `attachInterrupt`.

This logic will be available in the `encoderInit` function to simplify the API usage for the user.

```
void encoderInit(void)
{
    pinMode(GPIOA, 0, INPUT_PULLUP);
    pinMode(GPIOA, 1, INPUT_PULLUP);

    attachInterrupt(GPIOA, 0, FALLING);
}
```

Listing 1: encoderInit Implementation

1.2.2 Interrupt handler

The interrupt is linked to `EXTI0`.
Once the interrupt is triggered, we should read the state of the other pin (S_B), and based on its state, we can deduct the rotation direction.

```
extern "C" void EXTI0_IRQHandler(void)
{
    if (digitalRead(GPIOA, 1) == 0) { // SA == 0 & SB == 0 => Clockwise => Increment
        if (_encoderValue < 45) // < Max
            _encoderValue++;
    } else { // SA == 0 & SB == 1 => Counter Clockwise => Decrement
        if (_encoderValue > 0) // > Min
            _encoderValue--;
    }
    EXTI->PR |= EXTI_PR_PR0; // acknowledge
}
```

Listing 2: EXTI0_IRQHandler Implementation

1.2.3 State

The `__encoderValue` is a private (local) integer variable in the current file (encoder.cpp). We can get the value of the said variable using a simple getter function.

```
int encoderValue(void) { return _encoderValue; }
```

Listing 3: _encoderValue getter

We are doing things this way to forbid (or at least discourage) the user from changing the value of the `_encoderValue` variable directly.

2 ServoMotor

The `ServoMotor` rotation degree depends on the input PWM signal ON time (f = 50Hz). The output rotation degree goes linearly from 0deg at PWM_{on} = 1ms to 90deg at PWM_{on} = 2ms.

2.1 Init

PWM signal generation is kin of complicated to config on STM32, but it can be divided into 3 steps:

- Output pin config with alt mode
- Timer config
- Pin/Timer and channel linking

```
#define PB0 GPIOB, 0

```

```c
void servoInit()
{
    // 1. Output config
    pinMode(PB0, OUTPUT);
    pinAlt(PB0, 2);

    // Input clock = 64MHz.
    RCC->APB1ENR |= RCC_APB1ENR_TIM3EN;
    // Reset peripheral (mandatory!)
    RCC->APB1RSTR |= RCC_APB1RSTR_TIM3RST;
    RCC->APB1RSTR &= ~RCC_APB1RSTR_TIM3RST;

    // 2. Configure timer
    TIM3->CNT = 0;          // Reset
    TIM3->SR = 0;           // Reset
    TIM3->PSC = 64 - 1;     // 1MHz
    TIM3->ARR = 20000 - 1; // 20ms

    // 3. PWM configuration
    TIM3->CCMR2 &= ~TIM_CCMR2_CC3S_Msk;     // channel 3 as output
    TIM3->CCMR2 |= 6 << TIM_CCMR2_OC3M_Pos; // output PWM mode 1
    TIM3->CCMR2 |= TIM_CCMR2_OC3PE;         // Pre-load register TIM3_CCR3
    TIM3->CR1 &= ~TIM_CR1_CMS_Msk;          // mode 1 // edge-aligned mode
    TIM3->CR1 |= TIM_CR1_CEN;               // enable
    TIM3->CCER |= TIM_CCER_CC3E;            // enable
    TIM3->CCR3 = 1000 - 1;                  // Start at 1ms (0 deg)
}
```

Listing 4: servoInit

2.2 Rotate

```c
void servoSet(int setPoint)
{
    if (setPoint < 0) // MIN
        setPoint = 0;
    else if (setPoint > 1000) // MAX
        setPoint = 1000;

    TIM3->CCR3 = 1000 + setPoint - 1;
}
```

Listing 5: servoSet

The `servoSet` function takes a value between 0 and 1000, but our `Rotary Encoder` API returns a value between 0 and 90 instead, we can add a helper function to do the translation between the two parts (declared in encoder.cpp)

```c
int encoderValueInSteps(void)
```

```
{
    return (_encoderValue * 1000) / 90;
}
```

Listing 6: encoderValueInSteps

3 Application

Now that we have implemented the cores of our project, we can move into actually using them to build something of value.

3.1 Manual Mode

The manual mode is straightforward, we should just copy the value of the `Rotary Encoder` to the PWM output, and thanks to our already implemented function, it is as easy as the following line

```
    servoSet(encoderValueInSteps());
```

Listing 7: Manual Mode one-liner

3.2 Scan Mode

For the scan mode, we have to add a variable to keep track of our current rotation angle (we used a steps tracker instead for higher precision). We will also need a function to be called periodically (using a timer interrupt) to increase or decrease the current rotation angle (Value between 0deg and 45deg).

3.2.1 Timer

We decided to go with `TIM6`, with a period of 100ms (rotation speed was chosen pseudo-randomly).

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

    // Configure timer
    TIM6->CNT = 0;
    TIM6->SR = 0;
    TIM6->PSC = 64000 - 1; // 1ms
    TIM6->ARR = 10 - 1;    // 100ms
    TIM6->CR1 = 1;
```

```

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

Listing 8: TIM6 Config

3.2.2 Handler

```
    // Verify bounds
    if (_steps <= 0)
        _dir = Direction::Clockwise;
    else if (_steps >= 500)
        _dir = Direction::CounterClockwise;

    // Rotate
    if (_dir == Direction::Clockwise)
        _steps++;
    else // CounterClockwise
        _steps--;
```

Listing 9: Scan Mode handler

3.3 Transition mode

To switch between the two previous modes (Scan and Manual) we will need an extra mode adequately named `Transition Mode`.

If we switch directly from `Scan` to `Manual`, and the current `Rotary Encoder` value is different from the `Scan Mode` steps, the motor will suddenly jump from one position to another in a blink, which may damage the motor.

To avoid this, the `Transition Mode` will act as an intermediate step between the two modes, only in one way (Scan to Manual) as the other way around is safe to switch directly.

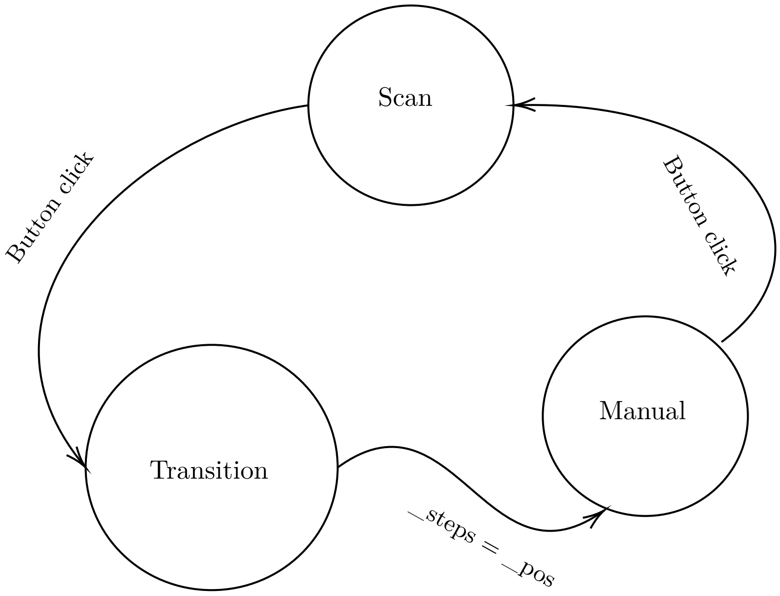


Figure 1: Modes switching

```
extern "C" void EXTI1_IRQHandler(void)
{
    switch (_mode)
    {
    case Mode::Scan:
    case Mode::Transition:
        _mode = Mode::Transition;
        break;

    case Mode::Manual:
        _steps = encoderValueInSteps();
        _mode = Mode::Scan;
        break;
    }
    EXTI->PR |= EXTI_PR_PR1; // acknowledge
}
```

Listing 10: Mode Switching handler

```
extern "C" void TIM6_DAC1_IRQHandler(void)
{
    if (_mode == Mode::Scan || _mode == Mode::Transition)
    {
        if (_steps <= 0)
            _dir = Direction::Clockwise;
        else if (_steps >= 500)
            _dir = Direction::CounterClockwise;

        if (_dir == Direction::Clockwise)
            _steps++;
        else // CounterClockwise
            _steps--;
```

```

        if (_mode == Mode::Transition && _steps == encoderValueInSteps())
        {
            _mode = Mode::Manual;
        }
    }

    if (_mode == Mode::Manual)
        servoSet(encoderValueInSteps());
    else
        servoSet(_steps);

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

Listing 11: Updated Rotation TIM6 handler

3.4 Display

Given the verbose nature of the Display code, it will not be included in this report (about 60 lines of code).

The only thing that is somewhat different from the previous display code in the previous project, is the specific screen clearing instead of erasing the whole screen as we were used to doing previously. The usual method was causing an annoying flickering event due to the slow display speeds and the lack of a double display buffer.

Resources

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