**ES2C4 Design Assignment 2 Report**

Student ID/Assigned computer – \_\_A206-7\_\_\_\_

Configuration-\_\_\_odd\_\_\_\_

# Task i

The whole program is used to remind the drivers go through or stop. First the program should turn on the red light, close the barriers to make divers stop. In this program, the register operations: timer, interrupt, GPIO, PWM (servo motor) was used. Also, state machine contains to response to external signals or timing.

## Design description

* **Main program：**

Initialization configuration:

Open the timer6

RCC->APB1ENR1 |= RCC\_APB1ENR1\_TIM6EN;

Open the GPIO clock

RCC->AHB2ENR |= RCC\_AHB2ENR\_GPIOAEN;

RCC->AHB2ENR |= RCC\_AHB2ENR\_GPIOBEN;

Make interrupt:

Because all the time interrupt is the multiple of 500ms, so make the 500ms as the prescaler. To get 500ms, we use 4MHz divide 4000 and divide 2.

    TIM6->PSC = 4000 - 1;    // Prescaler

    TIM6->ARR = 500 - 1;      // Auto-Reload

    TIM6->CNT = 0;            // counter

    TIM6->DIER |= 0X1UL << 0; // Update interrupt enabled

    TIM6->CR1 |= 0X1UL << 0;  // Enabled

Set Interrupt priority

NVIC\_SetPriorityGrouping(0x03);                 // Set Group

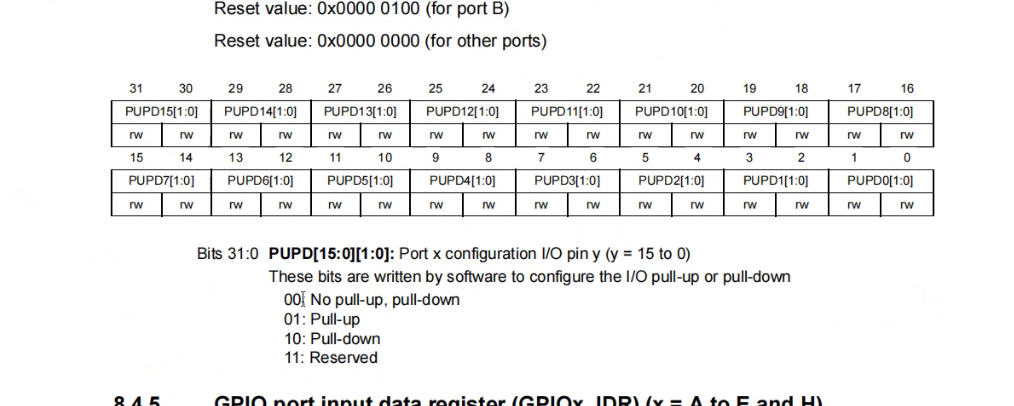
    CP = NVIC\_EnCP(0x03, 1, 2);

    NVIC\_SetPriority(TIM6\_IRQn, CP);

    NVIC\_EnableIRQ(TIM6\_IRQn); // Enable

Then to configure four LEDs:

Make Green LED as an example, the green LED use the GPIOA PA\_8.



According to the data, two bits control one pin, so make the value move 16bits. To make the pin to 0 we use ‘&’ to change 1to 0, and use ‘|’to change 0 to 1.

 GPIOA->MODER &= ~(0x3UL << 16);

 GPIOA->MODER |= 0x1UL << 16; // PA\_8 - Green LED

And here I choose pull-down cause grounding ensures stable output.

GPIOA->PUPDR &= ~(0x3UL << 16); // clear

GPIOA->PUPDR |= 0x2UL << 16;    // PA\_8 - Green LED

The configure of red, yellow, orange LEDs is same as the green one.

Servo-PWM:

Because we need a 1/20 duty cycle, so we need a 50Hz PWM wave frequency, 4MHz divide 4000 and divide 20 to get 50Hz.

TIM1->PSC = 4000 - 1;

TIM1->ARR = 20 - 1;

TIM1->CNT = 0;

By using the data below,

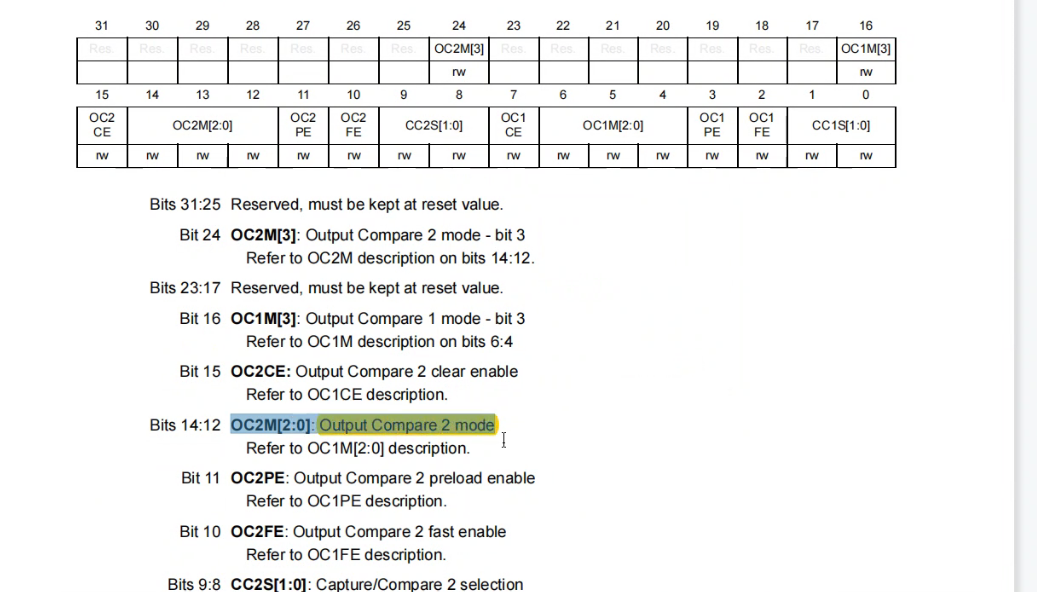
TIM1->CCMR1 |= (0X6UL << 12); // PWM 0110

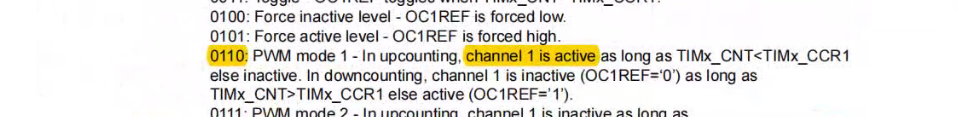
TIM1->BDTR |= 1UL << 15;      //  output Enable

TIM1->CCR2 = 1;               // Compare register

TIM1->CCER |= 0X1UL << 4;     // compare enable register TIM1\_CC2E

TIM1->CR1 |= 0X1UL;           // Counter enable





* **Control program:**

Step 1: Timer timing

int STATE\_A = 8  // 1 0 0 0

int STATE\_B = 12 // 1 1 0 0

int STATE\_C = 3  // 0 0 1 1

int STATE\_D = 4  // 0 1 0 0

int State = 0; // all the state

Step 2: trigger interrupt

Use interrupt with 500ms period.

void TIM6\_IRQHandler(void)   // Timer6 Interrupt

{

    if (TIM6->SR & 0X0001) // interrupt flag

    {

Step 3: counter processing (reset cycle)

When counter>7 set to 0, reset to form a cycle.

if (INT\_counter > 7 )

            INT\_counter = 0;

        }

Step 4: state judgment of state machine

Determine the state according to the time interval represented by the counter.

 if ( INT\_counter == 0 )

            State = STATE\_A;

        else if (INT\_counter >= 1 & INT\_counter <=2 )

            State = STATE\_B;

        else if (INT\_counter >= 3 & INT\_counter <=6 )

            State = STATE\_C;

        else

            State = STATE\_D;

Step 5: output according to the obtained state

By taking the remainder, get whether one is zero or not. 对二取余余数只能是0或1，

Extract the 1 or 0 of each bit, which is used to represent the state of the lamp, and carry out GPIO operation.

      TIM1->CCR2 = State % 2 + 1;

Use green LED as an example, calculate the bit is 1 or 0, turn on the LED when it is 1, and turn off when it is 0.

 if ((State >> 3) % 2)

        {

            GPIOA->ODR |= 0x1UL << 8;

        }

        else

        {

            GPIOA->ODR &= ~(0x1UL << 8);

        }

Step 6: counter counting

The counter plus 1 every 500ms.

INT\_counter = INT\_counter + 1;

Step 7: interrupt ending and enable the next interrupt.

    TIM6->SR &= ~(1 << 0);

## Finite State Machine (if implemented)

Taking three switches and one motor's switch state as variables, a binary state table is formed.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| No. | RED | YELLOW | GREEN | SERVO | STATE | TIME | COUNTER |
| A | 1 | 0 | 0 | 0 | 8 | 0-1000ms | 0 1 |
| B | 1 | 1 | 0 | 0 | 12 | 1000-1500ms | 2 |
| C | 0 | 0 | 1 | 1 | 3 | 1500-3500ms | 3 4 5 6 |
| D | 0 | 1 | 0 | 0 | 4 | 3500-4000ms | 7 |

Counter=1

It has four state A, B, C, D. while A, only transit to B when counter=2 when counter =2, it remains A. Now in the new state, counter=3 to move on to C. And when counter=2 it remains B. Move to D when counter=7. While counter=0, return to idle.

Counter=0

Counter=2

Counter=7

Counter=4/5/6

Counter=3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Current state(output) | | | |  | Input(Switching conditions) | | |  | Next state | | | |  |
| No. | S0 red | S1 yellow | S2 green | S3 servo | state | Counter\_2 | Counter\_1 | Counter\_0 | No. | S0’ | S1’ | S2’ | S3’ | state |
| A | 1 | 0 | 0 | 0 | 8 | 0 | 0 | 1 | B | 1 | 1 | 0 | 0 | 12 |
| B | 1 | 1 | 0 | 0 | 12 | 0 | 1 | 1 | C | 0 | 0 | 1 | 1 | 3 |
| C | 0 | 0 | 1 | 1 | 3 | 1 | 1 | 0 | D | 0 | 1 | 0 | 0 | 4 |
| D | 0 | 1 | 0 | 0 | 4 | 1 | 1 | 1 | A | 1 | 0 | 0 | 0 | 8 |

## Reflection

In the program, the timer was used, It doesn't need CPU's participation and is accurate. The timer is used for internal clock of the system. According to the clock, all kinds of output that need to adjust the frequency or perform the advantage of timing task can be used. After the system clock is divided into frequency, it will be used as the clock source to save external resources. And for the interrupt, It can improve the efficiency of CPU, CPU does not have to spend a lot of time waiting and querying peripheral work. It has the function of real-time processing, which can make a rapid response to various parameters and states in the real-time control system and deal with them in time. It can realize time division operation and control multiple peripherals to work at the same time.

# Task ii

## Design description

**Main program:**

Initialization configuration:

Open the timer6

Open the GPIO clock

RCC->APB1ENR1 |= RCC\_APB1ENR1\_TIM6EN;

RCC->AHB2ENR |= RCC\_AHB2ENR\_GPIOAEN;

RCC->AHB2ENR |= RCC\_AHB2ENR\_GPIOBEN;

Make timer interrupt:

Use 200ms timer so 4MHz divided 1kHz and divided 5Hz to get a 200ms timer

TIM6->PSC = 4000 - 1;    // Prescaler

TIM6->ARR = 200 - 1;       // Auto-Reload

TIM6->CNT = 0;            // counter

TIM6->DIER |= 0X1UL << 0; // Update interrupt enabled

TIM6->CR1 |= 0X1UL << 0;  // Enabled

Set Interrupt priority

 NVIC\_SetPriorityGrouping(0x03);                 // Set Group

    CP = NVIC\_EnCP(0x03, 1, 2); // Set Priority

    NVIC\_SetPriority(TIM6\_IRQn, CP);

    NVIC\_EnableIRQ(TIM6\_IRQn); // Enable

Then to configure four LEDs:

Make Green LED as an example, the green LED use the GPIOA PA\_8.

According to the data, two bits control one pin, so make the value move 16bits. To make the pin to 0 we use ‘&’ to change 1to 0, and use ‘|’to change 0 to 1.

 GPIOA->MODER &= ~(0x3UL << 16);

 GPIOA->MODER |= 0x1UL << 16;

Also choose pull-down cause grounding ensures stable output.

 GPIOA->PUPDR &= ~(0x3UL << 16);

 GPIOA->PUPDR |= 0x2UL << 16;

**Control program:**

Step 1: Timer timing

int Limit = 16

int State\_counter = 0;

int CP = 0;

Step 2: trigger interrupt

Timer 6 interrupt 5Hz

void TIM6\_IRQHandler(void)  // Timer6 interrupt 5Hz

{

    if (TIM6->SR & 0X0001)

    {

Step 3: counter processing (reset cycle)

When counter>limit set to 0, reset 15 to 0 and form a cycle.

 if (State\_counter >= Limit) // reset 15 to 0

        {

            State\_counter = 0;

        }

Step 5: output according to the obtained state

Extraction of each bit by shifting and extracting remainder.

Extract the 1 or 0 of each bit, which is used to represent the state of the lamp, and carry out GPIO operation. Use green LED as an example, calculate the bit is 1 or 0, turn on the LED when it is 1, and turn off when it is 0.

 if ((State\_counter >> 1) % 2)

        {

            GPIOA->ODR |= 0x1UL << 8; // Turn On

        }

        else

        {

            GPIOA->ODR &= ~(0x1UL << 8); // Turn Off

        }

Step 6: counter counting

The counter plus 1 every 200ms.

         State\_counter = State\_counter + 1;

Step 7: interrupt ending and enable the next interrupt.

 TIM6->SR &= ~(1 << 0); // Clear SR

## At what rate does the red LED blink and why?

According to the state machine below, the state changes every 0.2s. for the orange LED, ‘1’ happened 8 times of whole process, so the rate of orange is 1/0.2/2=2.5Hz. For the green LED, ‘1’occur4 times of the process. So the rate of green is 2.5/2=1.25Hz. For yellow, 2 times, and for red only one time. So rate of yellow is 1.25/2=0.625. rate of red is 0.625/2=0.3125.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| No. | Red | Yellow | Green | Orange | State | Time | counter |
| 1 | 0 | 0 | 0 | 0 | 0 | 0-200ms | 0 |
| 2 | 0 | 0 | 0 | 1 | 1 | 200-400ms | 1 |
| 3 | 0 | 0 | 1 | 0 | 2 | 400-600ms | 2 |
| 4 | 0 | 0 | 1 | 1 | 3 | 600-800ms | 3 |
| 5 | 0 | 1 | 0 | 0 | 4 | 800-1000ms | 4 |
| 6 | 0 | 1 | 0 | 1 | 5 | 1000-1200 | 5 |
| 7 | 0 | 1 | 1 | 0 | 6 | 1200-1400 | 6 |
| 8 | 0 | 1 | 1 | 1 | 7 | 1400-1600 | 7 |
| 9 | 1 | 0 | 0 | 0 | 8 | 1600-1800 | 8 |
| 10 | 1 | 0 | 0 | 1 | 9 | 1800-2000 | 9 |
| 11 | 1 | 0 | 1 | 0 | 10 | 2000-2200 | 10 |
| 12 | 1 | 0 | 1 | 1 | 11 | 2200-2400 | 11 |
| 13 | 1 | 1 | 0 | 0 | 12 | 2400-2600 | 12 |
| 14 | 1 | 1 | 0 | 1 | 13 | 2600-2800 | 13 |
| 15 | 1 | 1 | 1 | 0 | 14 | 2800-3000 | 14 |
| 16 | 1 | 1 | 1 | 1 | 15 | 3000-3200ms | 15 |

# Code

## Code for Task i

## Code for Task ii