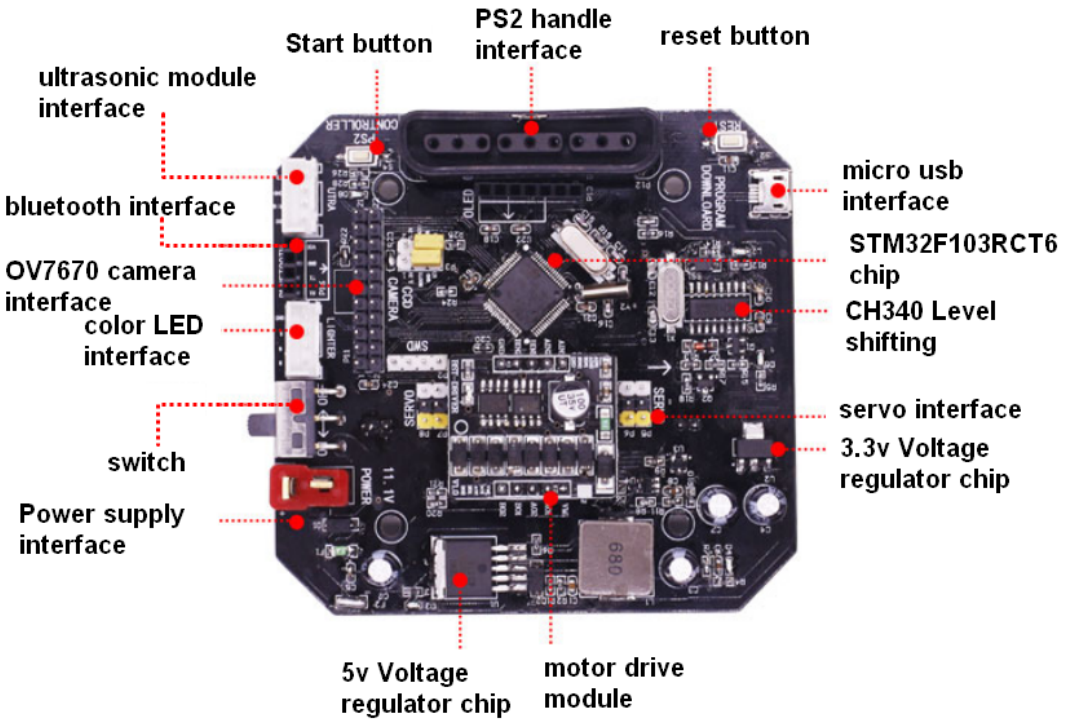
3.STM32 platform-------Servo control

1) Preparation



1-1 STM32 expansion board

1-2 STM32 smart car

2)Purpose of Experimental

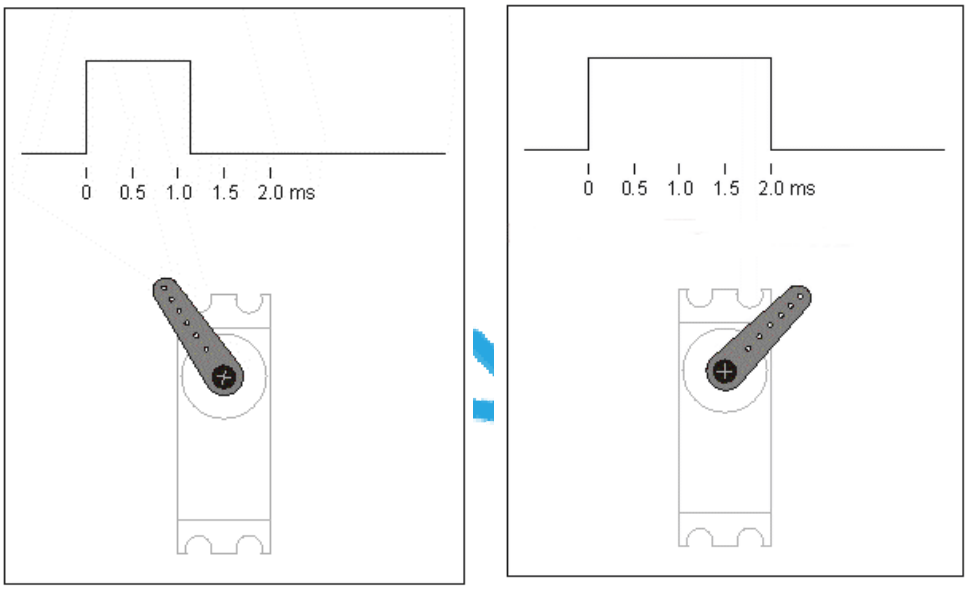
This experiment is mainly to learn to control the rotation of the servo and the PWM timer. When the car is powered on, press the start button next to the PS2 mark, the car advances 1s and then goes back 1s to the left 1s to the right 1s and then stops 1s for this cycle.

3)Principle of experimental

The working principle of the servo: the control signal enters the signal modulation chip from the channel of the receiver to obtain the bias voltage of the DC. It has a reference circuit inside, which generates a reference signal with a period of 20ms and a width of 1.5ms, and compares the obtained DC bias voltage with the voltage of the potentiometer to obtain a voltage difference output. The positive and negative outputs of the final voltage difference to the motor drive chip determine the forward and reverse of the motor. When the motor speed is constant, the potentiometer is rotated by the cascade reduction gear, so that the voltage difference is 0, and the motor stops rotating.

Servo control: Generally, a time base pulse of about 20ms is required, and the high level portion of the pulse is generally an angle control pulse portion in the range of 0.5ms-2.5ms. The servo used in this experiment is a 180 degree servo. The control relationship is as follows:

0.5ms----------0 degrees
1.0ms----------45 degrees
1.5ms----------90 degrees
2.0ms----------135 degrees
2.5ms----------180 degrees



Note: Which switch is used to open the switch in bsp_server.h.
As follows:

```
   #define USE_SERVO_J1          need to open the comment
//#define USE_SERVO_J2    no need to open the comment
//#define USE_SERVO_J3    no need to open the comment
//#define USE_SERVO_J4    no need to open the comment
//#define USE_SERVO_J5    no need to open the comment
//#define USE_SERVO_J6    no need to open the comment
```

The method of using the timer to control the servo.
We use timer 1 to drive the servo.

First, we first turn on the TIM1 clock as follows:
RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE); //Enable colck
Second, initialize Timer 1 as follows:
TIM_TimeBaseStructure.TIM_Period = arr; //Set the value of the auto-reload register cycle of the next update event load activity
TIM_TimeBaseStructure.TIM_Prescaler = (psc-1); //Set the prescaler value used as the TIMx clock frequency divisor
TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1; //Set the clock split: TDTS = Tck_tim //36Mhz
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up; //TIM count up mode
TIM_TimeBaseStructure.TIM_RepetitionCounter = 0; // repeat count off
TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure); //Initialize the time base unit of TIMx according to the specified parameters
Third, define the timer update interrupt:
TIM_ITConfig(TIM1, TIM_IT_Update, ENABLE); //Enable the specified TIM1 interrupt, allowing update interrupts
/ / Interrupt priority NVIC settings
NVIC_InitStructure.NVIC_IRQChannel = TIM1_UP_IRQn; //TIM1 interrupt
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; //Preemptive priority level 0
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3; //From priority level 3
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //IRQ channel is enabled
NVIC_Init(&NVIC_InitStructure); //Initialize the NVIC register
Finally, enable timer 1:
TIM_Cmd(TIM1, ENABLE); //Enable TIMx

void TIM1_Int_Init(u16 arr,u16 psc)
The 2 parameters of this function are used to set the overflow time of TIM1. In the clock system part, we explained that when the system is initialized, the clock of the APB1 is initialized by 2 in the default system initialization function SystemInit function, so the clock of APB1 is 36M, and the internal clock tree diagram of STM32 is: When the clock division number of APB1 is 1, the clock of TIM2~7 is the clock of APB1, and if the clock division number of APB1 is not 1, the clock frequency of TIM2~7 will be twice that of APB1 clock. Therefore, the clock of TIM3 is 72M, and based on the values of arr and psc we designed, we can calculate the interruption time. Calculated as follows:
Tout= ((arr+1)*(psc+1))/Tclk；
Tclk: Input clock frequency of TIM1 (in Mhz).
Tout: TIM1 overflow time (in us).
TIM1_Int_Init(9, 72);
Here we set the overflow time as: (9+1)*(71+1)/72M = 10us
void TIM1_Int_Init(u16 arr,u16 psc)
The 2 parameters of this function are used to set the overflow time of TIM1. In the clock system part, we explained that when the system is initialized, the clock of the APB1 is initialized by 2 in the default system initialization function SystemInit function, so the clock of APB1 is 36M, and the internal clock tree diagram of STM32 is: When the clock division

number of APB1 is 1, the clock of TIM2~7 is the clock of APB1, and if the clock division number of APB1 is not 1, the clock frequency of TIM2~7 will be twice that of APB1 clock. Therefore, the clock of TIM3 is 72M, and based on the values of arr and psc we designed, we can calculate the interruption time. Calculated as follows:

Tout= ((arr+1)*(psc+1))/Tclk；

Tclk: Input clock frequency of TIM1 (in Mhz).

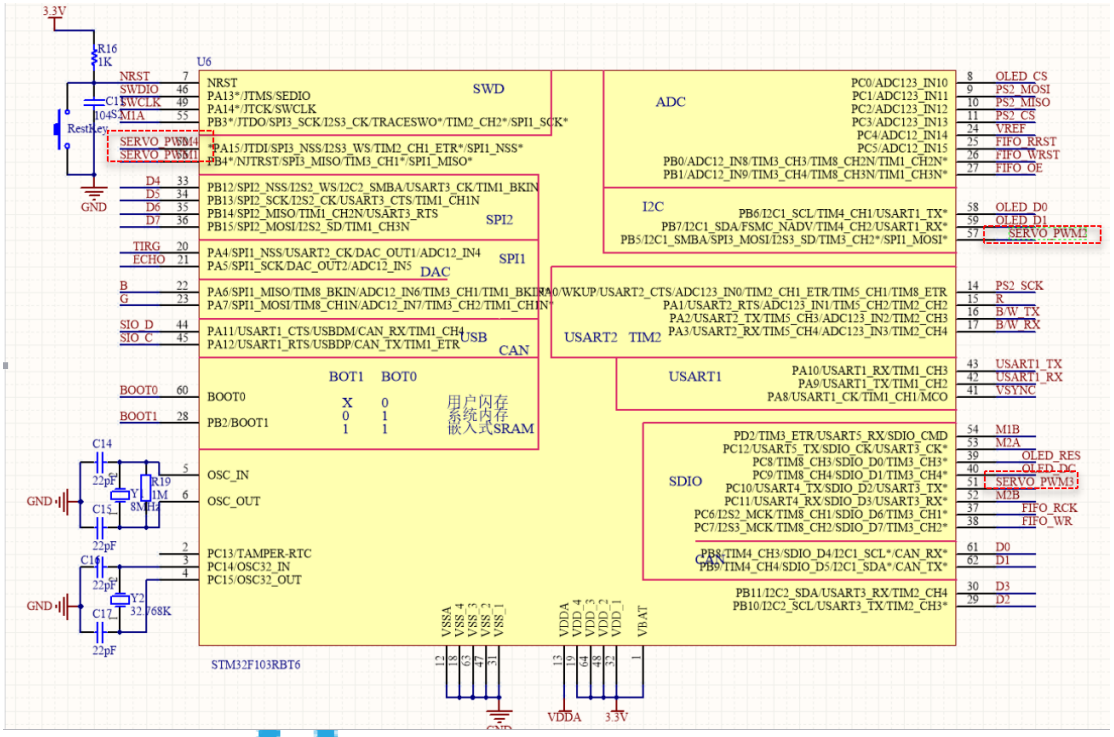
Tout: TIM1 overflow time (in us).

TIM1_Int_Init(9, 72);

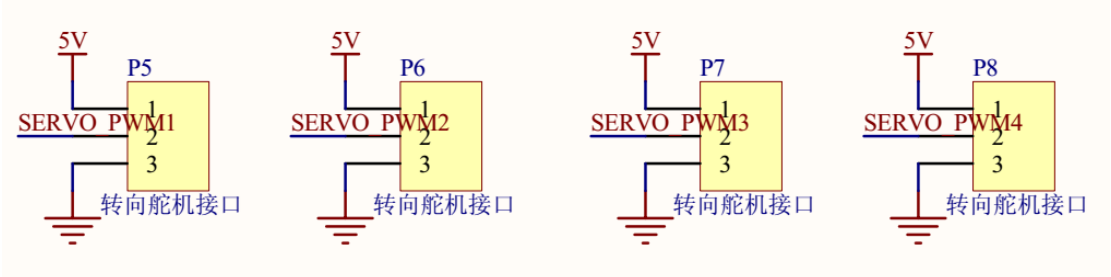
Here we set the overflow time as: (9+1)*(71+1)/72M = 10us

2) Experimental Steps

4-1 About the schematic



4-1 STM32 main control board circuit diagram



4-2 servo interface

4-2 According to the circuit schematic:

In this experiment, only the P5 be used, other three interfaces can be added by yourself.

SERVO PWM1--------PB4

4-3 About the code

Please see the folder named Servo control in the code folder.