Lesson 3 Buzzer Sound

1. Project Purpose

Control the buzzer on controller to make sound through the timer.

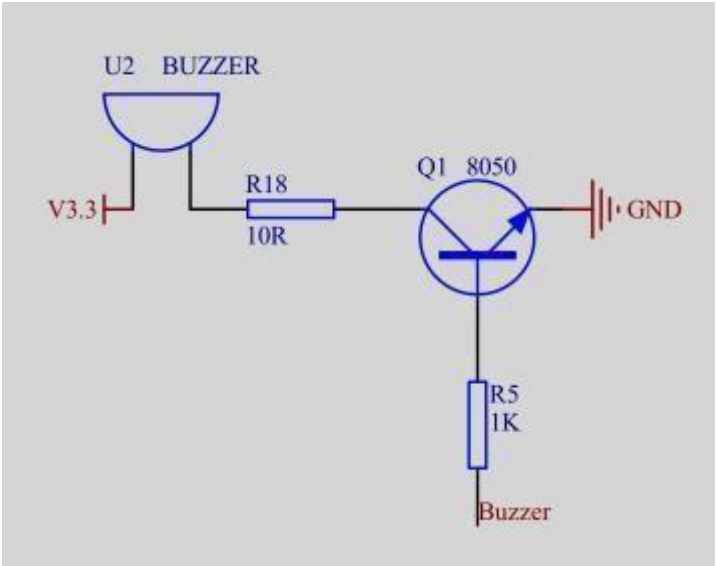
2. Project Principle

The buzzer is a electronic sound device with an integrated structure.It is widely used in computers,printers and other electronic products.

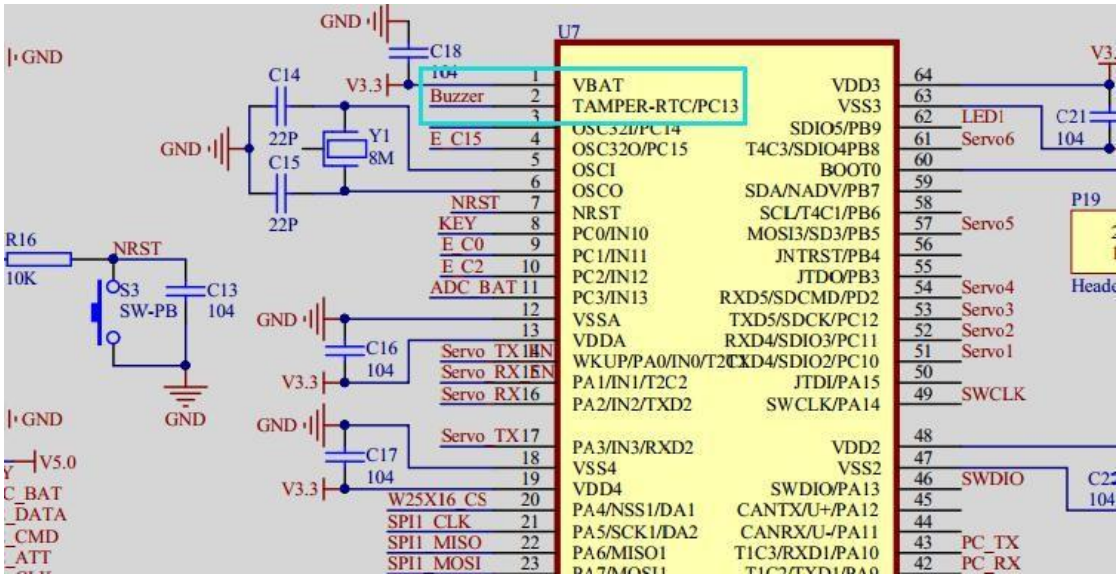
Its sounding principle is when the current passes through the oscillator, it produces a magnetic field to drive the vibration diaphragm to make sound. Therefore, a certain amount of current is required for the buzzer to make sound.

By changing the signal frequency, the pitch of the buzzer can be adjusted. The higher the frequency, the higher the pitch. In addition, the volume of the buzzer cab be controlled by changing the duty ratio of the high or low levels of the driving signal.

Because the current required by buzzer is relatively large, we use a triode to drive buzzer. The triode is an 8050 NPN triode, which is turned on at high level and cut off at low level.



The base electrode of triode is connected to the PC13I/O port. We need to generate a signal on PC13 to drive the buzzer



3. Program Analyst

1) We use a timer to flip the high and low levels of PC13 periodically to produce signal. At the same time, we have configured the timer 2 to be interrupted every 100us. Other than that, we need to configure the working parameters of timer 2, the parameters for interrupting the controller and the start of timer 2.

```
void InitTimer2(void)       //100us
{
    NVIC_InitTypeDef NVIC_InitStructure;

    TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE); //clock enable

    TIM_TimeBaseStructure.TIM_Period = (10 - 1); //Set the value of the auto-reload register period to be loaded into the activity at the next update event
    TIM_TimeBaseStructure.TIM_Prescaler =(720-1); //Set the prescaler value used as the divisor of the TIMx clock frequency
    TIM_TimeBaseStructure.TIM_ClockDivision = 0; //Set clock division:TDTS = Tck_tim
    TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;  //TIM Up counting mode
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure); //Initialize the time base unit of TIMx according to the parameters specified in TIM_TimeBaseIn itStruct

    TIM_ITConfig(  //Enable or disable the specified TIM interrupt
        TIM2, //TIM2
        TIM_IT_Update  |  //TIM interrupt source
        TIM_IT_Trigger,   //TIM trigger interrupt source
        ENABLE  //enable
    );

    TIM_Cmd(TIM2, ENABLE);  //Enable TIMx peripherals

    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;  //TIM2 interrupt
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 2;  //Preemption priority level 0
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 3;  //SubPriority 3
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //Enable IRQ channel
    NVIC_Init(&NVIC_InitStructure);  //Initial peripherals NVIC register according the  parameters specified in NVIC_InitStruct.
}
```

2) After the timer 2 and the interrupting are properly configured, the interrupting function will be executed every 100us. We can find that the Buzzer function is called in timer 2 from observing the interrupting function. This function is used to process the buzzer control as shown in the figure below:

```
void TIM2_IRQHandler(void)   //TIM2 interrupt
{//Timer2 interrupt 100us
    static uint32 time = 0;
    static uint16 timeBattery = 0;
    if(TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET)  //Check whether specified TIM interrupt occurs: TIM interrupt source
    {
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);    //Clear the interrupt pending bit of TIMx: TIM interrupt source


        Buzzer();
        if(++time >= 10)
        {
            time = 0;
            gSystemTickCount++;
            Ps2TimeCount++;
            if(GetBatteryVoltage() < 5500)//Alarm when less than 5.5V
            {
                timeBattery++;
                if(timeBattery > 5000)//last 5 seconds
                    BuzzerState = 1;
                }
            }
            else
            {
                timeBattery = 0;
                if(manual == FALSE)
                {
                    BuzzerState = 0;
                }
            }
        }
    }
}
```

3) In the code shown in the figure below, the red frame is the part for generating PWM and the blue frame is the part for controlling the time of the sound. When fBuzzer is true, I/O will be flipped once every 200us, otherwise it will output low-level.

The code in blue box is to control the state of fBuzzer. When BuzzerState is false, fBuzzer is set to false and stop sound output.

When Buzzerstate is true, the state of fBuzzer can be changed once every 500ms so as to realize the effect of beeping every 500ms. The final effect is that the buzzer will beep every 0.5s.