**实验9 STM32单片机开发入门**

1. **实验目的**
2. 了解STM32单片机的基本结构、特点
3. 掌握STM32cubeIDE的基本使用
4. 掌握STM32常用模块的基本控制方法
5. **实验任务**

参照下面给出的实验步骤，完成任务：

[练习0：STM32CubeIDE的使用入门](#_练习0：进一步熟悉STM32cubeIDE上机过程。)

[练习1: 掌握基本输入输出控制](#_练习1：_基本输入/输出控制)

[练习2：掌握外中断的编程方法](#_练习2：中断方式响应按键)

[练习3：掌握查询方式实现异步串行通信](#_练习3：_查询方式实现串行通信)

[练习4: 掌握定时器PWM输出方法](#_练习4:_通用定时器的PWM输出)

[**(提高)**练习5：掌握定时器定时功能实现方法](#_(提高)练习5:_通用定时器的定时功能)

[**(提高)**练习6：中断方式实现串行接收](#_(提高)练习6：_中断方式实现串行接收)

[**(提高)**综合练习A：模拟遥控小车运动](#_(提高)综合练习A：模拟遥控小车运动显示)

[**(提高)**综合练习B：简易电子琴基本功能](#_(提高)_综合练习B_简易电子琴基本功能)

1. **实验步骤**

# **练习0：进一步熟悉STM32cubeIDE上机过程**。

任务：参看stm32单片机实验平台搭建中的“STM32Cube使用入门.doc”中的项目TestGPIO，新建一个LabSTM32，完成与TestGPIO项目同样的操作和功能。现象按下实验板上按键B1，发光二极管LD2亮；释放按键B1，发光二极管LD2灭。

# **练习1： 基本输入/输出控制**

任务：编程按下按键，让LED闪烁三下。

**实验操作步骤：**

1. 保持LabSTM32项目配置不变，也可以用使用入门创建的项目TestGPIO；
2. 打开main.c文件，在main()函数内，将while(1)主循环/\* USER CODE BEGIN 3 \*/后的代码改为：

**if** ( HAL\_GPIO\_ReadPin(GPIOC, GPIO\_PIN\_13) ) // 判断按键是否按下

{ for (i=0; i<6; i++) //LED闪

{ HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5); //对引脚求反

HAL\_Delay(500); //延时500ms

}

}

3) 在**main() 函数外**的/\* USER CODE BEGIN PV \*/中，加入变量i的定义：

/\* USER CODE BEGIN PV \*/

**unsigned int** i;

/\* USER CODE END PV \*/

4) 重新编译连接，在DEBUG下运行程序。实验现象是：每按下一次按键，LED闪三下。

5) **(提高)学习CubeIDE下GPIO基于寄存器级编程方法**：参看stm32单片机开发入门实验课件“三、srm32头文件和句柄定义”，了解头文件stm32g431xx.h中对stm32单片机内容各模块寄存器的定义方法。注释或去除步骤2)中主循环添加的代码，改为在主循环中添加下面代码，完成对引脚的读取和控制，可以看到用步骤2)中函数调用相同的实验现象。

**if** ( ( GPIOC->IDR & GPIO\_PIN\_13 )!=0 ) // 判断按键是否按下

{

**for** (i=0;i<6;i++)

{ **GPIOA->ODR**^=GPIO\_PIN\_5; //对PA5引脚取反

HAL\_Delay(500); //延时500毫秒

};

}

# **练习2：中断方式响应按键**

任务：以中断方式响应按键，每按下一次按键，发光二极管状态求反一次。

**实验操作步骤：**

1. 点击Project Explorer 下的TestGPIO>Debug>TestGPIO.IOC文件，进入到项目配置中；
2. 保持当前项目的RCC、SYS、Clock树的设置不变；
3. 配置PC13为外中断引脚功能：如图37，点击Pinout&Configuration，在Pinout View窗口，点击PC13引脚，配置PC13为外中断引脚；
4. 配置GPIO中PC13引脚外中断设置：如图38，点击Pinout & Configurations>Categories >Systerm Core >GPIO , 点击GPIO Mode and Configuration 中的PC13，出现与引脚PC13有关的配置。点击GPIO Mode 中PC13对应的下拉图标，选择External Interrupt Mode With Falling edge detection，即用下降沿作为中断源信号，其他使用默认配置即可。

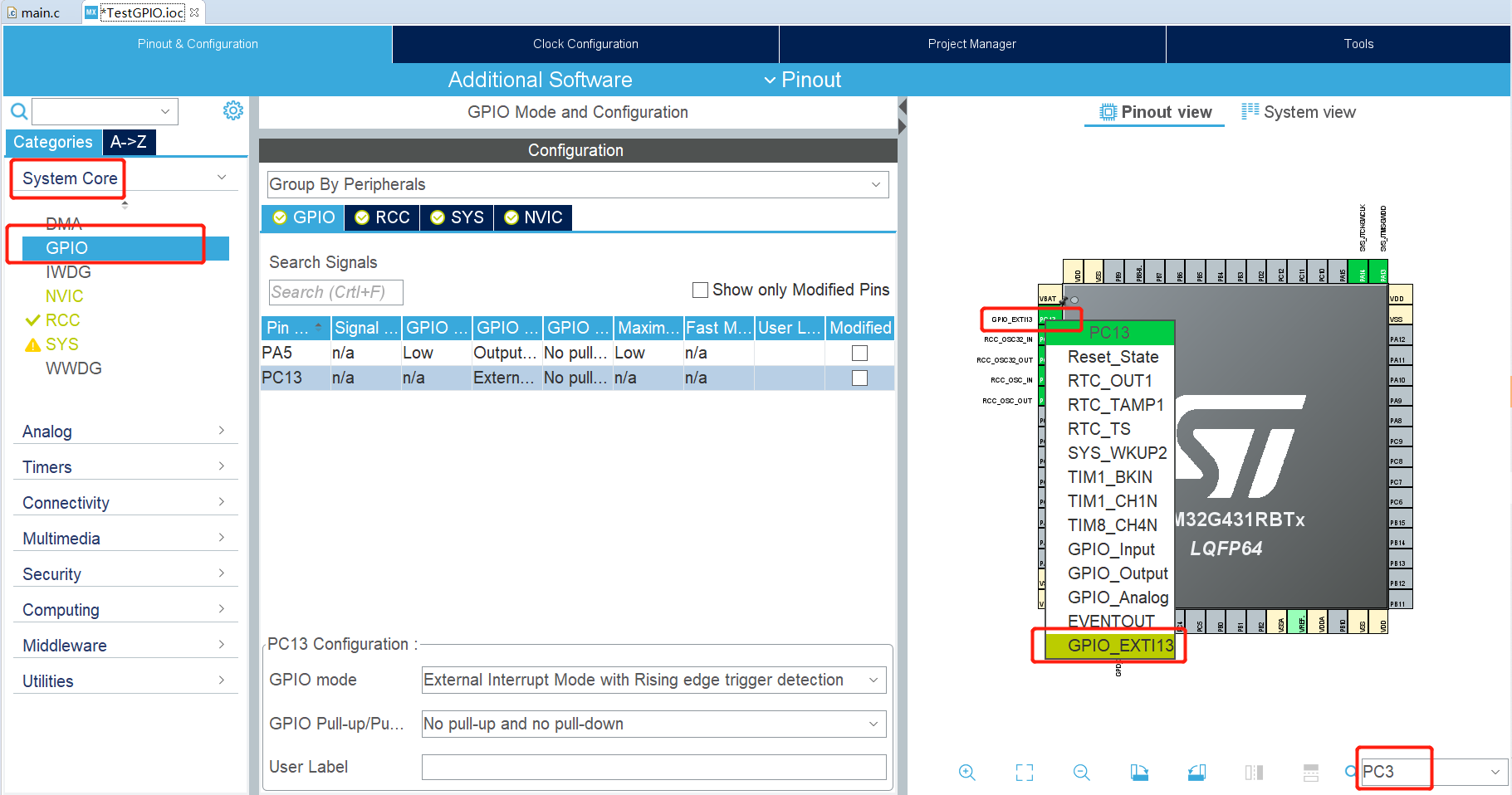


图37 配置为外中断引脚

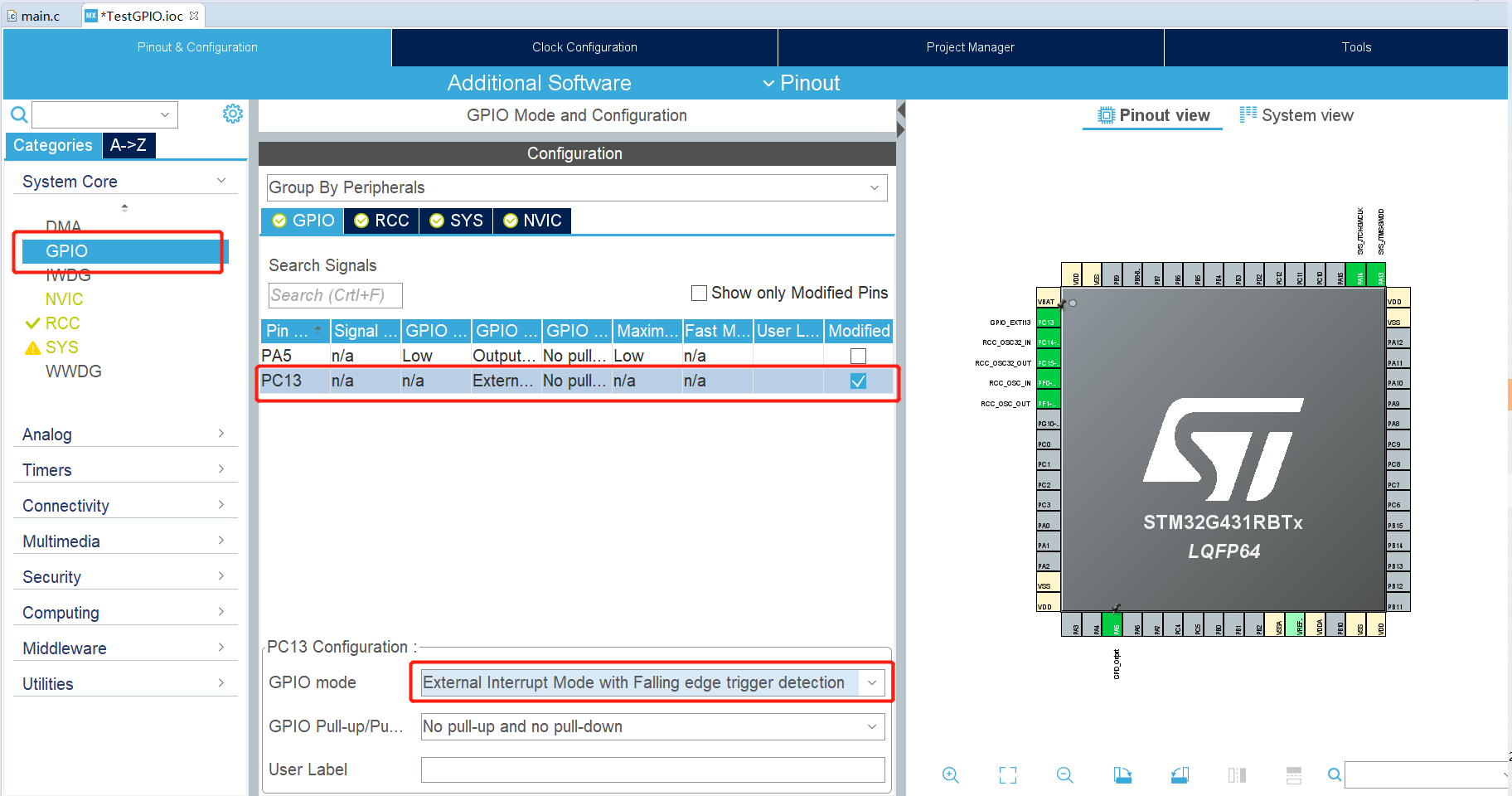


图38 配置为外中断引脚

1. 设置NVIC：如图39，点击左侧Pinout & Configurations>Categories >Systerm Core >NVIC , 勾选NVIC Mode and Configuration 中EXTI Line[15:10] interrupt的Enalbled（即使能该中断），并设置Preemption Priority 为1（抢占优先级别为1），其他设置保持默认。

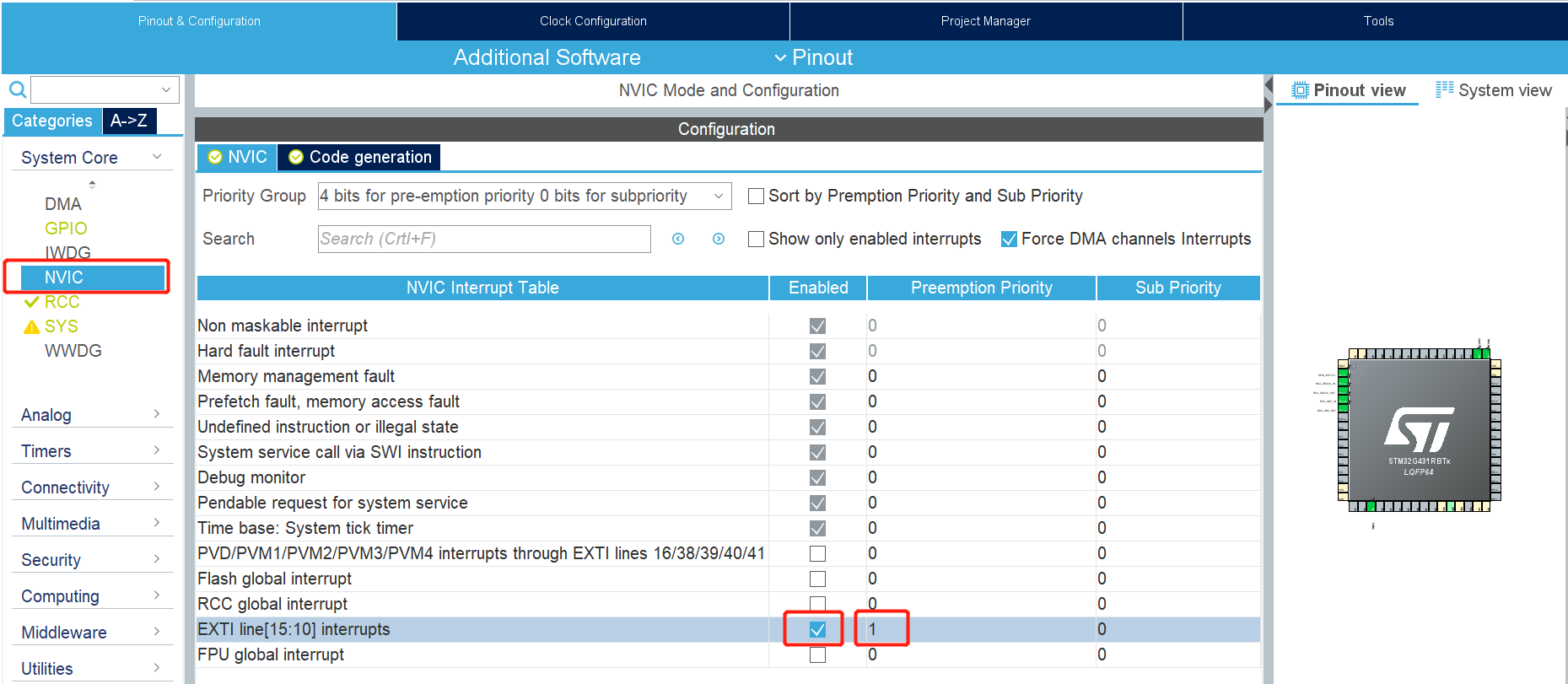


图39 配置NVIC

5）点击保存，生成程序代码；

6）打开TestGPIO项目的main.c文件，在**main()函数外**的 /\* USER CODE BEGIN 4 \*/后，添加代码：

/\* USER CODE BEGIN 4 \*/

**void** HAL\_GPIO\_EXTI\_Callback(uint16\_t GPIO\_Pin)

{

**if** (GPIO\_Pin==GPIO\_PIN\_13 )

{

HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

}

}

/\* USER CODE END 4 \*/

7）将main()函数内练习1while(1)循环中/\* USER CODE BEGIN 3 \*/后的代码注释掉。可如图40，选中要注释的代码，右键鼠标，在出现的窗口中，选择Source>Toggle Commet，用“//”注释；或选择Source>Add block Commet，用“/\*..\*/”块注释。也可同时按下控制键ctl和7这两个按键，注释掉选中的代码，或取消对它的注释。

/\* USER CODE BEGIN 3 \*/

**/\* if** ( HAL\_GPIO\_ReadPin(GPIOC, GPIO\_PIN\_13) ) // 判断按键是否按下

{ for (i=0; i<6; i++) //LED闪

{ HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

HAL\_Delay(500); //延时500ms

}

}

\*/

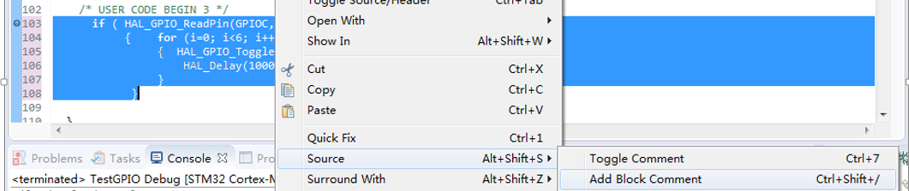


图40 注释程序段

8) 编译连接程序（Build Project），进入DEBUG下，并运行。每操作一次操作按键，将看到LED求反一次。

9) 调试练习：如图41，在中断回调函数入口处，双击行首左侧，即设置一个断点，然后运行程序，每操作一次按键，将在中断回调函数中的断点处，停下来，与CCS下设置断点的功能相似。

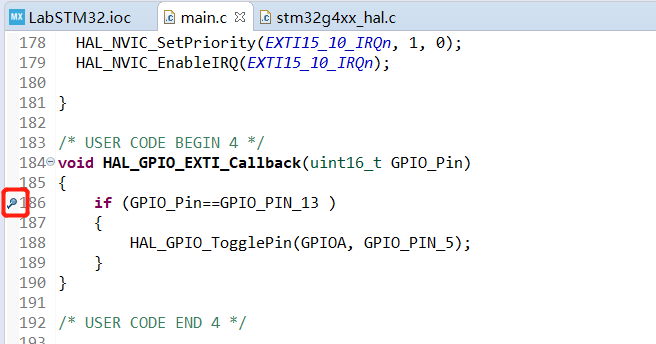


图41 在中断函数入口出加断点

**说明：中断函数的调用关系**

1. **查看函数：**

点击Project Exploer>TestGPIO>Core>Src>STM32g4xx\_it.c, 双击打开该文件，在文件的最后，可查看函数EXIT15\_10\_IRQHandler( )，该函数内调用了HAL\_GPIO\_EXTI\_IRQHandler( ); 按下ctrl键，用鼠标点击该函数名，即可打开存放在STM32g4xx\_hal\_gpio.c中的HAL\_GPIO\_EXTI\_IRQHandler( )函数。可以看到在该函数中，清除了对应的中断标志，并调用了HAL\_GPIO\_EXTI\_Callback( )函数。

三个函数EXIT15\_10\_IRQHandler( )、HAL\_GPIO\_EXTI\_IRQHandler( )、HAL\_GPIO\_EXTI\_Callback()，前两个有HAL库函数提供，只有HAL\_GPIO\_EXTI\_Callback()由用户编写。中断相关的函数执行关系参见图42。

1. **函数执行关系：**

中断产生时，触发CPU以中断方式执行函数EXIT15\_10\_IRQHandler( ), 这个函数的入口地址，已存放在中断向量表中中断源EXTI15:10对应的单元中，所以外中断源EXTI15：10源发出中断申请，CPU响应该中断的话，必然执行该函数。该函数调用HAL\_GPIO\_EXTI\_IRQHandler( )将对应的中断标志位清除（函数\_\_HAL\_GPIO\_EXTI\_CLEAR\_IT(GPIO\_Pin)），(**注意：函数清标志的位置，想想对处理按键抖动时的影响)**。然后调用由用户编写的HAL\_GPIO\_EXTI\_Callback(GPIO\_Pin)。所有有关中断向量的设置和中断标志的清除，有HAL库函数完成，用户只用写出中断申请发出后，要完成的程序功能。注意中断回调函数的名字不能写错。

**(3)下面是库中三个函数的定义**

**void** **EXTI15\_10\_IRQHandler**(**void**)

{

/\* USER CODE BEGIN EXTI15\_10\_IRQn 0 \*/

/\* USER CODE END EXTI15\_10\_IRQn 0 \*/

HAL\_GPIO\_EXTI\_IRQHandler(GPIO\_PIN\_13);

/\* USER CODE BEGIN EXTI15\_10\_IRQn 1 \*/

/\* USER CODE END EXTI15\_10\_IRQn 1 \*/

}

**void** **HAL\_GPIO\_EXTI\_IRQHandler**(uint16\_t GPIO\_Pin)

{

/\* EXTI line interrupt detected \*/

**if** (\_\_HAL\_GPIO\_EXTI\_GET\_IT(GPIO\_Pin) != 0x00u)

{

\_\_HAL\_GPIO\_EXTI\_CLEAR\_IT(GPIO\_Pin);

HAL\_GPIO\_EXTI\_Callback(GPIO\_Pin);

}

}

**void** **HAL\_GPIO\_EXTI\_Callback**(uint16\_t GPIO\_Pin)

{

**if** (GPIO\_Pin==GPIO\_PIN\_13 )

{

HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

}

}

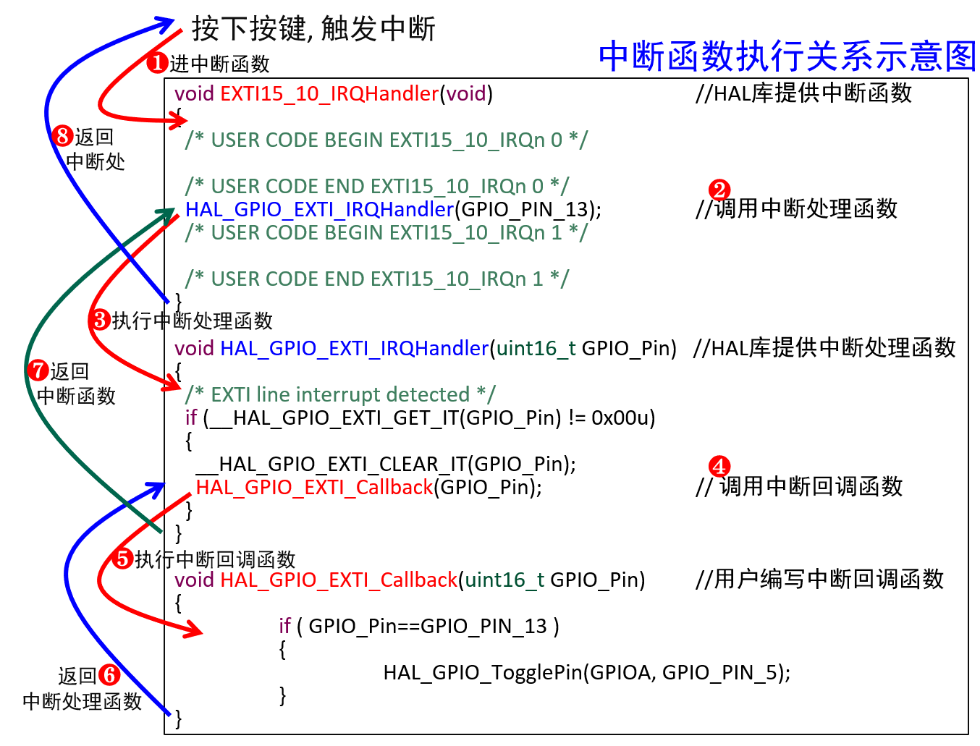


图42 中断相关的函数执行关系

# **练习3： 查询方式实现串行通信**

任务：利用串行通信接口，实现在msp430g2553上完成的串行通信测试程序TestSCI.c功能。

**实验操作步骤：**

1. 在已建好的LabSTM32或testGPIO项目基础上，点击项目下的.ioc文件; 或再新建一个STM32g431项目TestUSART，同练习0配置好RCC、SYS、时钟，即（1）配置RCC(用上外部两个晶振)（参看使用入门练习0图9）；（2）配置SYS（serial Ware）（参看练习0图10）；（3）配置时钟树（选用外部晶振24MHz, HCLK=170MHz）（参看练习0图15）；
2. 如图43，在Categories中，点击Connectivity>LPUSART1；在出现的LPUSART1 Mode and Configuration中，Mode 选择 Asynchronous ,即异步串行方式，修改波特率为115200，其他选用默认设置(8位数据，1位停止位，无校验)；此时PA2、PA3自动配置为LPUART1\_TX、LPUART1\_RX，如果没有，请手动完成这两个引脚的LPUART1收、发功能的配置。注意，一定是PA2、PA3作为发送和接收的引脚，因为这两个引脚和实验板上调试接口的USB转串口部分的收发引脚已在实验板上对接好。方便单片机利用连个引脚与计算机进行异步串行通信。

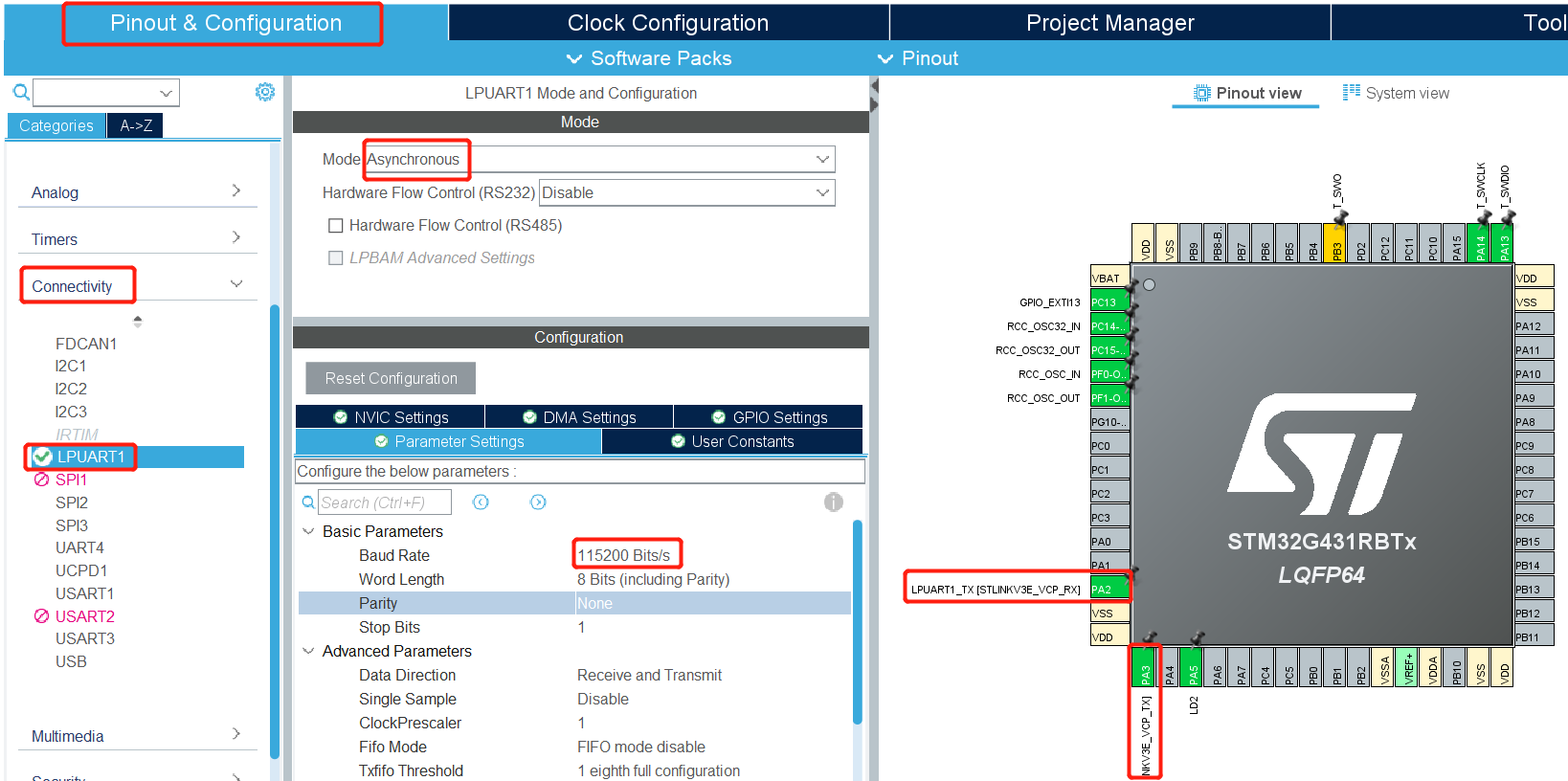


图43 配置LPUART1

1. 点击保存，生成程序代码；

4) 打开main.c文件，将while(1)主循环里之前添加的代码去掉，或注释掉，然后/\* USER CODE BEGIN 3 \*/后添加代码：

/\* USER CODE BEGIN 3 \*/

i=0;

**while**(string[i]!='\0') //发送提示输入信息

{ HAL\_UART\_Transmit(&hlpuart1, &string[i], 1, 0xffff);

i++;}

**for** (i=0;i<7;i++) //接收字符到缓冲区

{ HAL\_UART\_Receive(&hlpuart1, &buffer[i], 1, 0xffff);};

**for** (i=0;i<7;i++) //发送缓冲区字符

{ HAL\_UART\_Transmit(&hlpuart1, &buffer[i], 1, 0xffff); }

5) 在main() 函数外的/\* USER CODE BEGIN PV \*/中，加入变量i和字符串的定义：

/\* USER CODE BEGIN PV \*/

uint8\_t buffer[20], string[]="Please input 7 Characters:\r\n\0";

uint16\_t i=0;

/\* USER CODE END PV \*/

1. 编译连接程序（Build Project），进入DEBUG下。
2. 打开电脑的设备管理器，查看端口（COM和LPT）下实验板(STMCroelectronics STLink Virtual COM Port)对应的串口号，在PC侧打开串口助手，打开该串口，并设置波特率为115200；
3. 在STM32cubeIDE下运行单片机程序，实验现象与TestSCI同，即在PC机的串口助手上输入7个字符,单片机接收后，又将字符发送给PC。

注意：该程序有超时强行退出查询方式发送和接收，继续执行发送和接收执行下面的语句。不会一直查询是否可以发送或接收。

9) 将USER CODE BEGIN 3 中的代码改为下面方式，重新编译、连接，运行程序。亦可实现4)中代码的功能。

两段代码的不同是：4）中调用的串口函数，每次是发送或接收1个字符；而9）中的方法，调用串口函数每次发送或接收指定个数的字符，但本质上串口函数内仍然是查询方式发送和接收每个字符。

/\* USER CODE BEGIN 3 \*/

HAL\_UART\_Transmit(&hlpuart1, string,28, 0xffff); //发送提示输入信息

HAL\_UART\_Receive(&hlpuart1, buffer, 7, 0xffff); //接收7个字符到缓冲区

HAL\_UART\_Transmit(&hlpuart1, buffer, 7, 0xffff); //发送7个缓冲区字符

10) **(提高)学习CubeIDE下串行通信接口基于寄存器级编程方法**：注释或去除步骤9)中主循环添加的代码，

改为在主循环中添加下面代码，完成收发控制，可以看到相同的实验现象。

i=0;

while(string[i]!='\0') //输出提示信息

{ while((LPUART1->ISR&USART\_ISR\_TXE)==0); //检测发送缓冲是否空

LPUART1->TDR=string[i]; //取一个数据发送

i++;

};

for (i=0; i<7; i++) //接收字符串

{ while((LPUART1->ISR&USART\_ISR\_RXNE)==0); //检测接收缓冲器是否满

buffer[i]=LPUART1->RDR; //接收一个数据并保存

};

for (i=0;i<7;i++) //发送字符串

{ while((LPUART1->ISR&USART\_ISR\_TXE)==0); //检测发送缓冲是否空

LPUART1->TDR=buffer[i]; //取一个数据发送

};

1. **(提高)** 用STM32单片机完成控制PC机上自由钢琴弹奏。

采用波特率115200，编程让单片机循环向计算机发送3字节的MIDI信息，控制计算机上自由钢琴的各琴键反复从低音向高音顺序弹奏。

# **练习4: 通用定时器的PWM输出**

任务：利用通用定时器，实现PWM波形输出。

**实验操作步骤：**

1) 新建立一个STM32g431项目TestPWM，或者也可以在上面TestGPIO项目练习基础上，点击.ioc文件，完成或确认（1）配置RCC(用上外部两个晶振)（参看使用入门练习0图9）；（2）配置SYS（serial Ware）（参看练习0图10）；（3）配置时钟树（选用外部晶振24MHz, HCLK=170MHz）（参看练习0图15）；（5）配置PC13为外中断（参看上面练习2图37），并打开NVVIC中的EXTI15~10的中断允许（参看练习2图39）。

2) 如图44, 在当前默认设置基础上，配置定时器Timers>TIM3：

* Mode> Clock
  + Clock Source> **Internal Clock** (选择内部HCLK时钟作为计数时钟)
  + Channel1 > **PWM Generantion CH1** (设置通道1输出PWM)

此时将看到有**引脚PA6自动配置为TIM3\_CH1**, 即TIM3通道1的PWM输出引脚。

如配置在其他引脚上，可以手动选中PA6，右键选上**TIM3\_CH1功能。**

* Configuration>Parameter Settings
  + Counter Settings

Prescaler(PSC-16 bits value>**16999**  (计数时钟为HCLK的(PSC+1)分频)

Counter mode>Up (增计数方式)

Counter Peirod(AutoReload Register – 16 bit value)>**9999**

(设置影响PWM周期的ARR值，相当于msp430g2553中TA的CCR0值)

auto-reload preload > **Enable**

(当更新事件发生时，由硬件自动将新设置值一起加载到相关的设置寄存器中，使定时器的几个设置寄存器的值同时更新，避免因顺序执行程序带来的不同时)

* PWM Generation Channel 1
  + Mode >  **PWM mode 1** (设置为输出模式1)

Pulse(16 bit value) > **5000** (设置影响PWM占空比的CCR1值)

* + Output compare preload > Enable
  + Fast Mode > Disable
  + Ch Polarity > High (计数从0到CCR1时，输出高电平)

**说明：计算输出PWM波形的频率、占空比**

1. 从图45可以看到，当前设置的系统时钟HCLK=170MHz，该时钟是内部提供定时器计数的时钟源;
2. 输出PWM的频率=HCLK时钟频率/(预分频值PSC+1)/(计数周期值ARR+1);
3. 上面设置的结果是:

PWM频率=170MHz/(16999+1)/(9999+1)=1Hz;

占空比=CCR1/(ARR+1)=5000/(9999+1)=50%;

1. 如果设置Pulse(16 bit value) >9000，则占空比= 9000/(9999+1)=90%

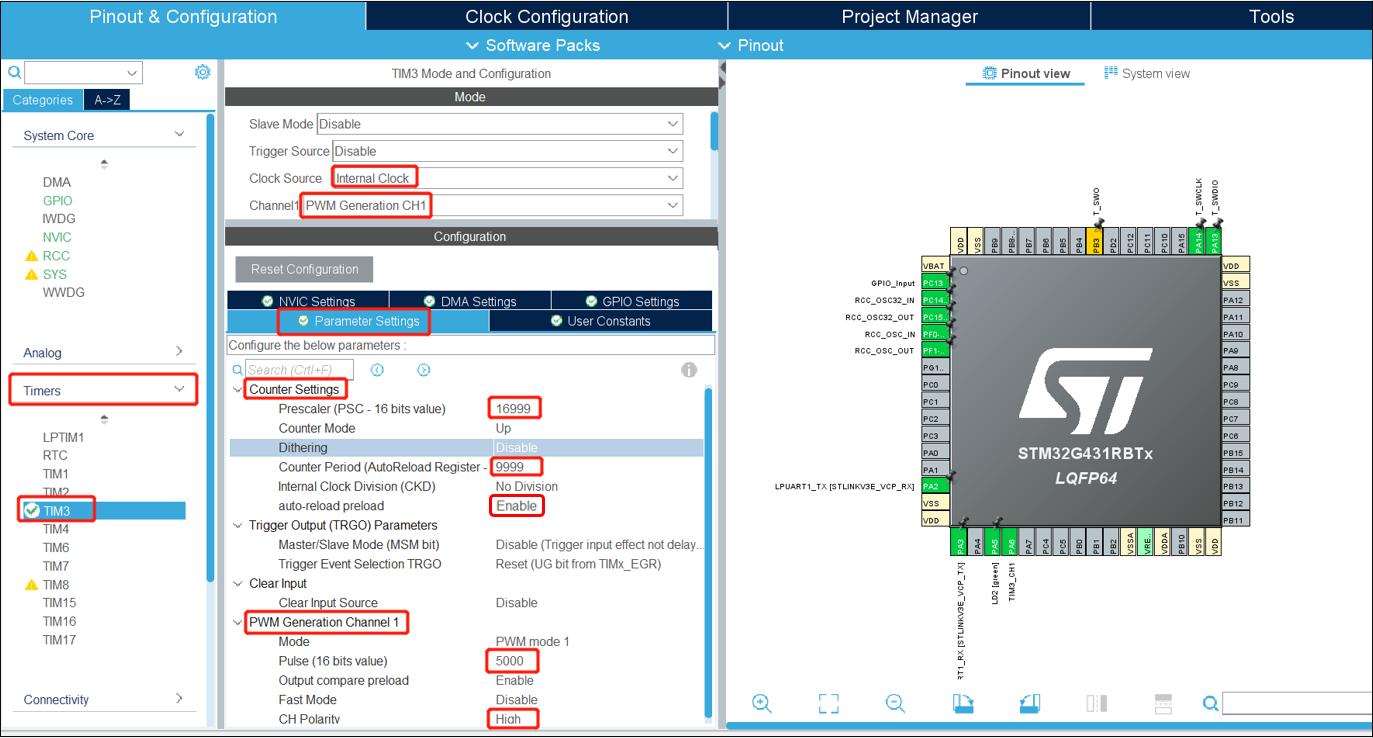


图44 配置TIM3

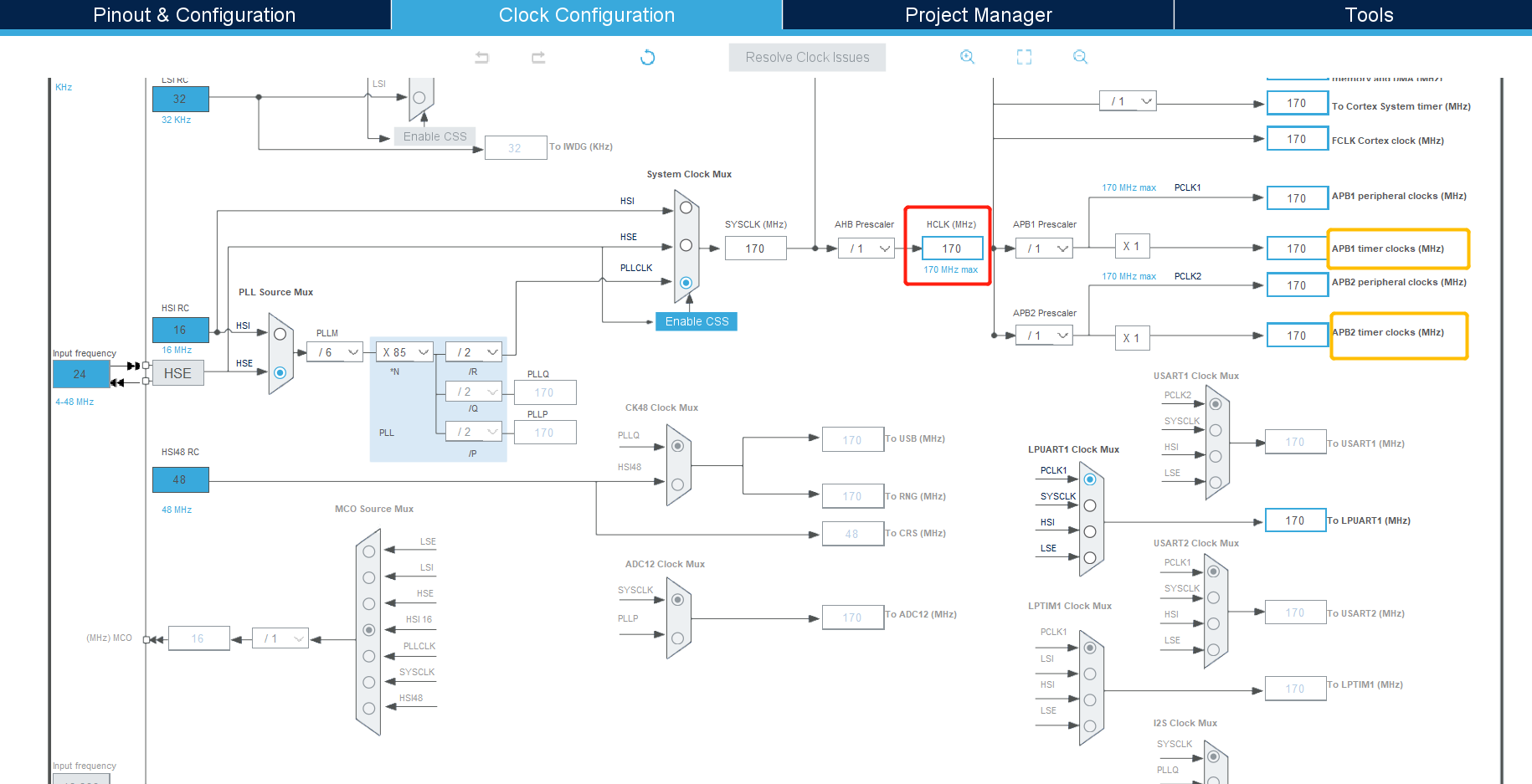


图45 时钟HCLK作为定时器的内部计数时钟

3) 点击保存，生成程序初始化代码；

4) 打开main.c文件，在while(1)主循环前的/\* USER CODE BEGIN 2 \*/后添加代码：

/\* USER CODE BEGIN 2 \*/

HAL\_TIM\_PWM\_Start(&htim3, TIM\_CHANNEL\_1); //启动定时器TIM3通道1的PWM输出

/\* USER CODE END 2 \*/

5) 如图46，用导线将STM32单片机板上的3.3v、GND，分别与外部电路扩展板上的VCC、GND连接，

PA6信号引出与一个LED灯连接。

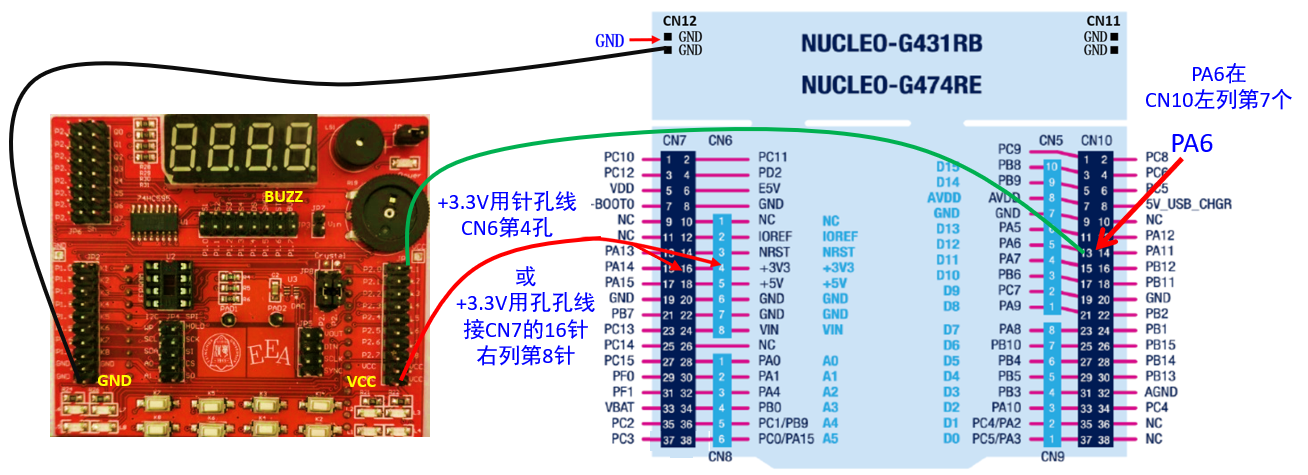


图46 用PWM控制LED闪烁速度接线图

1. 编译连接程序（Build Project），进入DEBUG下，然后运行程序，将看到PA6连接LED以1秒频率闪烁，半秒亮，半秒亮。注意扩展板上的发光二极管是低电平亮，高电平灭。
2. 返回.ioc配置文件，将ARR改为19999(**修改数值后，注意按回车**)，其他部分不变，如下：

* Configuration>Parameter Settings

Counter Peirod(AutoReload Register – 16 bit value)>1**9999** (设置影响PWM周期的ARR值)

则：PWM频率=170MHz/(16999+1)/(19999+1)=0.5Hz;

占空比=CCR1/(ARR+1)=5000/(19999+1)=**25%**;

1. 重新生成代码、编译连接（build projiect），下载和运行程序，将看到PA6连接的LED以2秒(即0.5Hz)频率闪烁，其中亮1.5秒，灭0.5秒，即占空比是25%。
2. 返回.ioc配置文件，在当前情况下，再将Pulse改为10000(**修改数值后，注意按回车**)，其他部分不变，如下：

* PWM Generation Channel 1
  + Mode >  **PWM mode 1 (**设置为输出模式1)

Pulse(16 bit value) > **10000** (设置影响PWM占空比的CCR1值)

则：PWM频率=170MHz/(16999+1)/(19999+1)=0.5Hz;

占空比=CCR1/(ARR+1)=10000/(19999+1)=50%;

1. 再重新生成代码、编译连接（build projiect），下载和运行程序，将看到PA6连接的LED以2秒(即0.5Hz)频率闪烁，其中亮1秒，灭1秒，即占空比是50%。
2. 如果图形化界面中的Ch Polarity从Hign改为Low, 如下：

* PWM Generation Channel 1
  + Ch Polarity >Low

则：输出PWM的占空比=1-CCR1/(ARR+1)=1-Pulse/(ARR+1)

如果Pulse=5000，Ch Polarity 设为Low，占空比=1-Pusle/(APP+1)=1-5000/(19999+1)=75%

此时PA6连接的扩展板上发光二极管亮0.5秒，灭1.5秒。

1. **(提高)学习CubeIDE下基于TIMER寄存器级编程方法**：在上面的项目文件中，增加PC13的按键中断，在中断函数中用修改寄存器值方式，改变PA6输出的PWM波形频率和占空比。
2. 如图37和38，配置PC13为外中断引脚功能，并打开其中断允许；
3. 生成初始化代码；
4. main.c文件，在main()函数外的 /\* USER CODE BEGIN 4 \*/后，添加代码，代码见下页；
5. 编译连接，下载后运行，操作实验板上连接在PC13上的蓝色按键，将看到PA6上连接的LED以相应的频率和占空比进行闪烁。据此方法可以在程序中方便修改PWM输出的频率和占空比，实现对蜂鸣器发出不同音调的控制。调试时，可在外中断函数每个if语句内加断点，以便观察每按一次按键，程序的执行、以及因PWM输出的频率和占空比变化发光二极管的闪烁发生变化。
6. **(提高)** 用stm32单片机完成3个8度的音程播放

任务： 利用STM32单片机实验板和扩展板，用PWM输出控制蜂鸣器发出3个8度音程各音

实验步骤：

1. 如图47，将STM32实验板的3.3V、GND分别与扩展板的Vcc、GND连接；
2. 将STM32实验板输出的定时器PWM波形PA6引脚与扩展板的蜂鸣器Buzz引脚连接；
3. 编程控制蜂鸣器播放3个8度各音。

**步骤12)的(3)需要添加的代码:：**

/\* USER CODE BEGIN 4 \*/

**void** **HAL\_GPIO\_EXTI\_Callback**(uint16\_t GPIO\_Pin)

{

**if** (GPIO\_Pin==GPIO\_PIN\_13 )

{ i++;

**if**(i==1) //在初始图形界面下配置了HCLK=170MHz;PSC=16999，可通过TIM3->PSC进行修改;PWM Mode 1;Polarity为High

{ TIM3->ARR=19999; //PWM频率=HCLK/(PSC+1)/(ARR+1)=170MHz/17000/20000=0.5Hz，周期2s

TIM3->CCR1=10000; //PWM占空比=CCR1/(ARR+1)=10000/20000=50%, 1秒高，1秒低

}

**else** **if**(i==2)

{ TIM3->ARR=39999; //PWM频率=HCLK/(PSC+1)/(ARR+1)=170MHz/17000/40000=0.25Hz，周期4s

TIM3->CCR1=10000; //PWM占空比=CCR1/(ARR+1)=10000/40000=25%，1秒高，3秒低

}

**else** **if**(i==3)

{ TIM3->ARR=9999; //PWM频率=HCLK/(PSC+1)/(ARR+1)=170MHz/17000/10000=1Hz，周期1s

TIM3->CCR1=5000; //PWM占空比=CCR1/(ARR+1)=5000/10000=50%，0.5秒高，0.5秒低

}

**else** **if**(i==4)

{

HAL\_TIM\_PWM\_Stop(&htim3, TIM\_CHANNEL\_1); //停止定时器TIM3通道1的PWM输出

}

**else** **if**(i==5)

{

HAL\_TIM\_PWM\_Start(&htim3, TIM\_CHANNEL\_1); //启动定时器TIM3通道1的PWM输出

i=0; //重新回到按键值为0初值状态

}

}

}}/\* USER CODE END 4 \*/

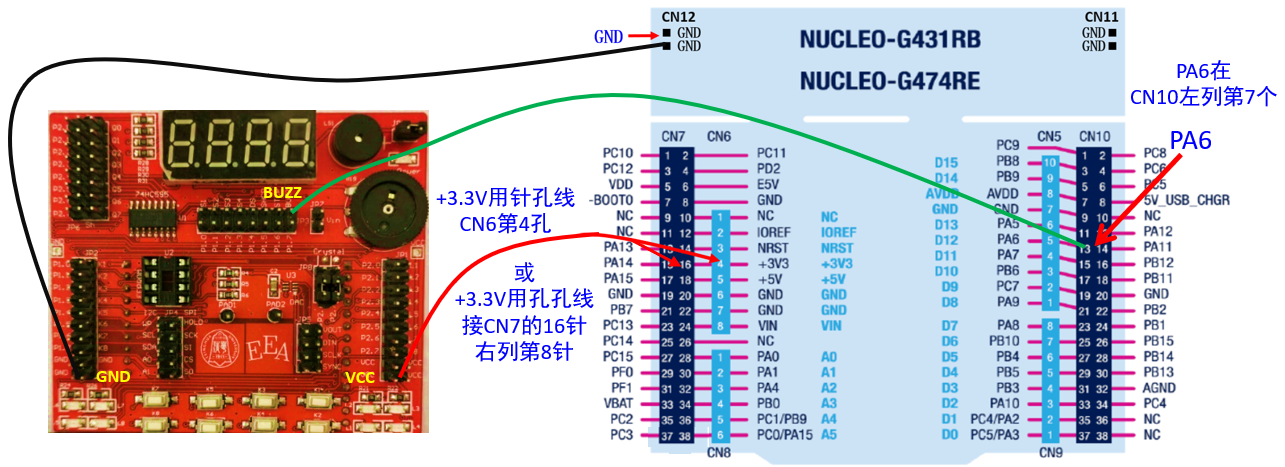


图47 定时器PWM输出控制蜂鸣器发声接线参考图

# **(提高)练习5: 通用定时器的定时功能**

任务：利用通用定时器，实现定时功能，以定时中断方式，1秒频率闪烁，即半秒亮，半秒灭，定时中断的响应周期为0.5秒。学习编写定时中断的方法。

**实验操作步骤：**

1. 新建立一个STM32g431项目TestTIM，如练习0上配置RCC(用上外部两个晶振)、SYS（serial Ware）、时钟树的设置（选用外部晶振24MHz, HCLK=170MHz）、引脚PA5为基本输出GPIO\_Output；也可以在之前建立的项目基础上(如TestGPIO项目)上，点击项目.ioc文件，完成下面的配置。

2) 如图48, 在当前TIM3默认设置基础上，配置定时器Timers>TIM3，

不用PWM输出，则可不设置与PWM输出有关的设置。

* Mode> Clock
  + Clock Source> **Internal Clock** (选择内部HCLK时钟作为计数时钟)
* Configuration>Parameter Settings
  + Counter Settings

Prescaler(PSC-16 bits value>**16999**  (计数时钟为HCLK的(PSC+1)分频)

Counter mode>Up (增计数方式)

Counter Peirod(AutoReload Register – 16 bit value)>**4999** (设置影响PWM周期的ARR值)

此时定时频率=HCLK/(PSR+1)/(ARR+1)

=170MHz/17000/5000=2Hz

定时周期=1/定时频率=0.5s

1. 如图49, 在NVIC中打开定时器TIM3的中断，并把优先级设为1。

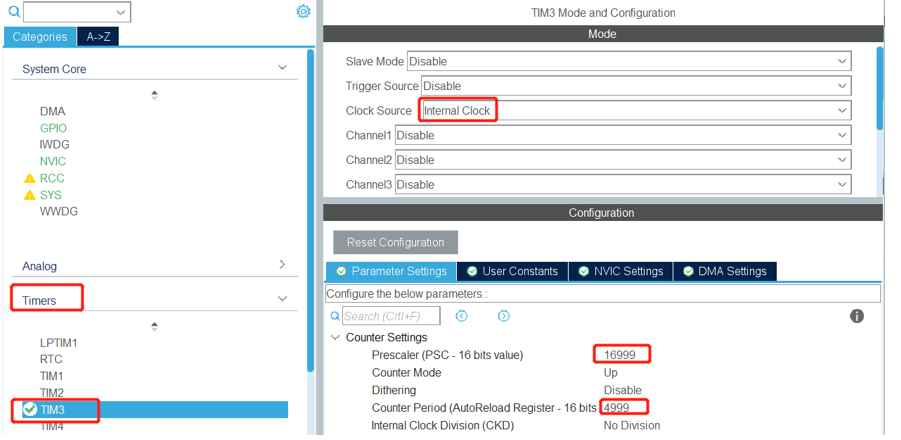


图48 设置TIM3的参数

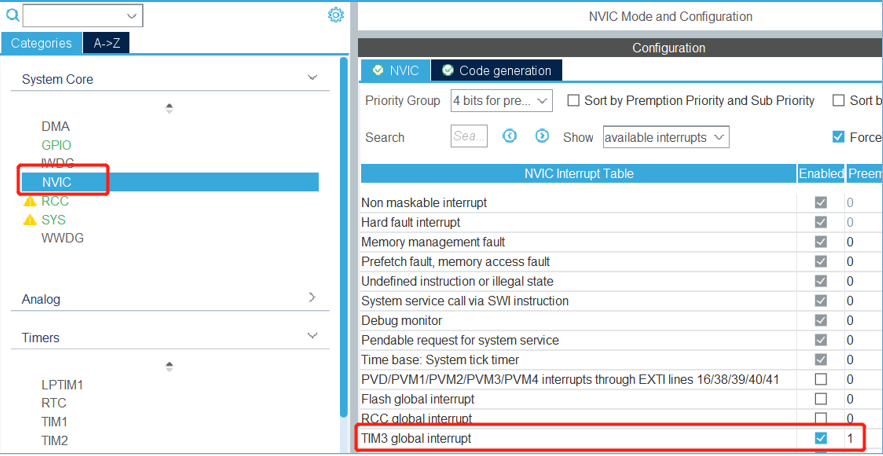


图49 打开TIM3的分中断

1. 生成初始化代码；
2. 在main.c的CODE BEGIN 2和CODE BEGIN 4两处，分别添加下面代码：

/\* USER CODE BEGIN 2 \*/

HAL\_TIM\_Base\_Start\_IT(&htim3); //定时器初始化，打开定时中断

/\* USER CODE END 2 \*/

/\* USER CODE BEGIN 4 \*/

**void** HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim) //定时器的定时回调函数

{

if (htim->Instance==TIM3) //确定是TIM3引起的中断

{

HAL\_GPIO\_TogglePin(GPIOA, GPIO\_PIN\_5);

};

}

/\* USER CODE END 4 \*/

1. 编译连接，下载到单片机中运行，将实现每半秒进一次定时器回调函数，将实验板上PA5连接的发光二极管LD2状态求反一次，出现亮半秒、灭半秒，即1秒频率闪烁的现象。

**说明：**

* 可用HAL\_TIM\_Base\_Start\_IT(&htim3)函数启动定时器TIM3的定时中断。
* 与外中断相似，定时器TIME3的定时中断函数固定为void TIM3\_IRQHandler(void)，中断源触发CPU执行中断函数TIM3\_IRQHandler。
* 中断函数TIM3\_IRQHandler调用了void HAL\_TIM\_IRQHandler(TIM\_HandlerTypeDef \*htim)，并通过HAL\_TIM\_IRQHandler调用了回调函数void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)。
* 用户只需在回调函数中编写处理中断到来后需完成的功能，跟中断有关的操作，比如中断向量的设置、中断标志的清除等，均由系统提供的TIM3\_IRQHandler()和HAL\_TIM\_IRQHandler()处理了。因各定时器的定时中断函数调用的回调函数是同一个，所以，用到多个定时器的定时中断中，可在回调函数中通过判读句柄中的实例，来确认是哪个定时器以及编写各自需要完成的功能。
* 可用HAL\_TIM\_Base\_Stop\_IT(&htim3)函数停止定时器TIM3的定时中断。

# **(提高)练习6： 中断方式实现串行接收**

任务：利用串行通信接收中断，实现在msp430g2553上完成的串行通信测试程序L5\_TestSCI.c的接收。

实验操作步骤：

1. 新建立一个STM32g431项目TestUSARTint，参看开发入门练习3的异步串行通信练习，配置RCC(用上外部两个晶振)、SYS（serial Ware）、时钟树的设置（选用外部晶振24MHz, HCLK=170MHz）；注意设置串口LPUSART1的mode(Asynchronous)、波特率(115200)，以及PA2、PA3为发送和接收引脚；
2. 如图49, 打开LPUSART1的接收中断允许，并设置中断优先级别为1

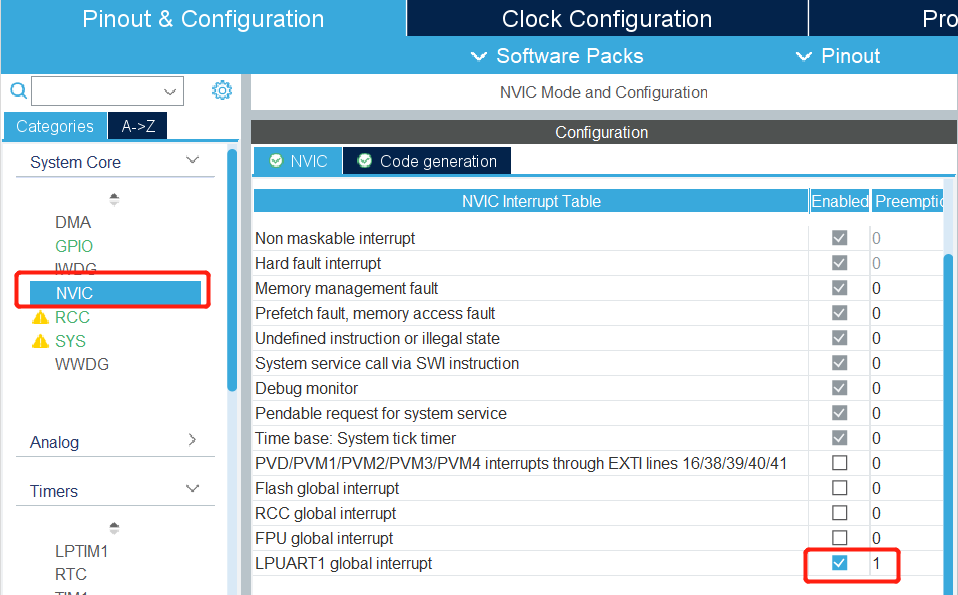


图49 配置LPUART1的中断允许

3) 点击保存，生成程序代码；

4) 打开main.c文件，在main函数前的/\* USER CODE BEGIN PV \*/中，加入接收缓冲区和提示字符串的定义：

/\* USER CODE BEGIN PV \*/

uint8\_t buffer[20], string[]="Please input 7 Characters:\r\n\0";

/\* USER CODE END PV \*/

1. 在main.c的CODE BEGIN 2和CODE BEGIN 4两处，分别添加下面代码：

/\* USER CODE BEGIN 2 \*/

HAL\_UART\_Transmit(&hlpuart1, string,28, 0xffff); //查询方式发送提示输入信息

HAL\_UART\_Receive**\_IT**(&hlpuart1, buffer, 7); //启动中断方式接收指定数目的字符串

/\* USER CODE END 2 \*/

/\* USER CODE BEGIN 4 \*/

**void** HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart**)** //j接收中断回调函数

{

if (huart->Instance==LPUART1) //确认是LPUART1的接收中断

{

HAL\_UART\_Transmit(&hlpuart1, buffer,7,0xffff ); //查询方式发送收到字符串

HAL\_UART\_Transmit(&hlpuart1, string,28, 0xffff); //查询方式发送提示输入信息

HAL\_UART\_Receive\_IT(&hlpuart1, buffer, 7); //启动中断方式接收指定数目的字符串

}

}

/\* USER CODE END 4 \*/

1. 编译连接程序（Build Project），进入DEBUG下。
2. 打开电脑的设备管理器，查看端口（COM和LPT）下实验板(STMCroelectronics STLink Virtual COM Port)对应的串口号，在PC侧打开串口助手，打开该串口，并设置波特率为115200；
3. 在STM32cubeIDE下运行单片机程序，实验现象与练习3查询方式接收相同: 先在计算机的串口助手上出现提示信息“Please input 7 Characters:”，之后在串口助手上输入7个字符，单片机以中断方式接收到由HAL\_UART\_Receive**\_IT**(&hlpuart1, buffer, 7)中指定的字符数(例子是7个)，并存放在指定的缓冲区（例子是buffer数组），然后执行串行中断接收完成回调函数HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart)，并关闭串口的中断。要再接收的话，需用函数HAL\_UART\_Receive**\_IT()**重新启动。本例在串行中断接收完成回调函数中，将收到固定长度的字符串（本例是在buffer数组中）以串行方式发送给计算机，之后再次发送了提示输入信息，并再启动串行接收中断。如此反复，实现和练习3同样的现象。
4. 也可在接收中断回调函数中计入一个接收完成标志，然后在主循环中利用该标志来处理接收到的字符串行信息。在上面步骤1)~7)的基础上：
5. 在/\* USER CODE BEGIN PV \*/中增加一个变量定义：uint16\_t RxCplt=0;
6. 将main.c的 /\*USER CODE BEGIN 4\*/处，改写接收中断回调：

/\* USER CODE BEGIN 4 \*/

**void** HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart**)** //j接收中断回调函数

{

if (huart->Instance==LPUART1) //确认是LPUART1的接收中断

{

RxCplt=1; //设置接收完成标志

}

}

/\* USER CODE END 4 \*/

1. 在主函数main()的主循环CODE BEGIN 3 后，添加下面代码：
2. 编译连接和下载运行，将看到步骤8)相同的现象。

/\* USER CODE BEGIN 3 \*/

if (RxCplt==1)

{ HAL\_UART\_Transmit(&hlpuart1, buffer,7,0xffff ); //查询方式发送收到字符串

HAL\_UART\_Transmit(&hlpuart1, string,28, 0xffff); //查询方式发送提示输入信息

HAL\_UART\_Receive**\_IT**(&hlpuart1, buffer, 7); //启动中断方式接收指定数目的字符串

RxCplt=0; //清除接收完成标志

}

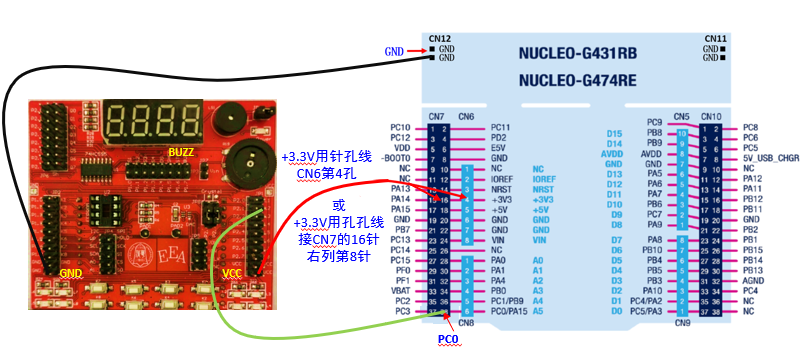
1. 串行收发函数需要指定接收和发送的长度，如果要接收和发送不固定长度的信息，可以根据要发送的内容或接收到的内容来进行控制

# **(提高)综合练习A：模拟遥控小车运动显示**

结合上面的练习，实现串行通信实验8任务2中2)的LED控制功能，即模拟遥控小车运动显示。利用STM32实验板上B1按键和LD2发光二极管，并增加一个引脚PC0做输出，控制扩展板上的发光二极管L1。通过两个LED的显示，反应接收到PC机的不同命令。

如图50，用3根杜邦线完成下面接线：

1. 将STM32实验板的地与扩展板的地相连；
2. 将STM32实验板的3.3V与扩展板的VCC相连；
3. 将STM32实验板的PC0与扩展板的L2相连。

图50 综合练习A的参考接线图

# **(提高) 综合练习B 简易电子琴基本功能**

任务： 利用STM32单片机实验板和扩展板，完成电子琴的播放和弹奏功能。

**实验步骤：**

1. 在图47的基础上，选择STM32实验板的9个基本I/O引脚(如PB0~PB8)与扩展板的8个按键和独立按键K连接，如图51 简易电子琴基本功能参考接线示意图。注意：配置与扩展板上按键连接的GPIO引脚时，因扩展板上按键电路没有上拉电阻，需如图51-2设置上拉电阻；独立按键模块K电路有上拉电阻，不用另外设置单片机的上拉电阻；
2. 在练习4的13)提高任务基础上实现一首歌曲的播放；
3. 编程实现电子琴3个8度的弹奏功能。

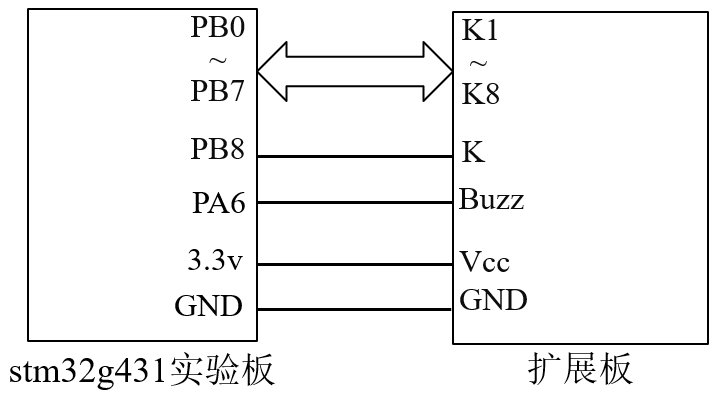


图51-1 简易电子琴基本功能参考接线示意图

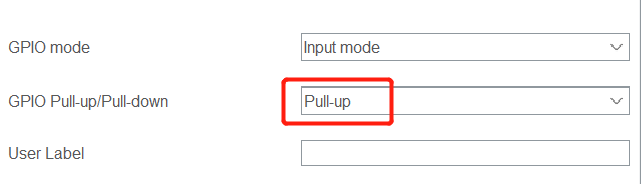


图51-2 设置GPIO输入引脚的上拉电阻

Nucleo-G431RB开发板、引脚关系对应、按键B1和发光二极管LD2原理图见下页。

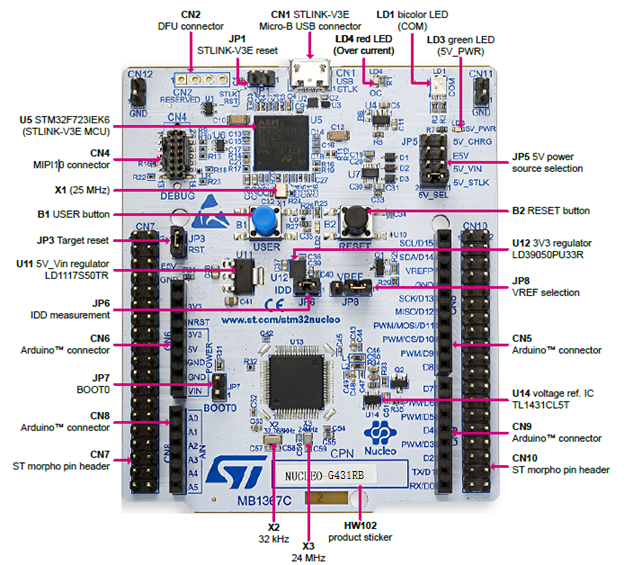


图52 Nucleo-G431RB开发板

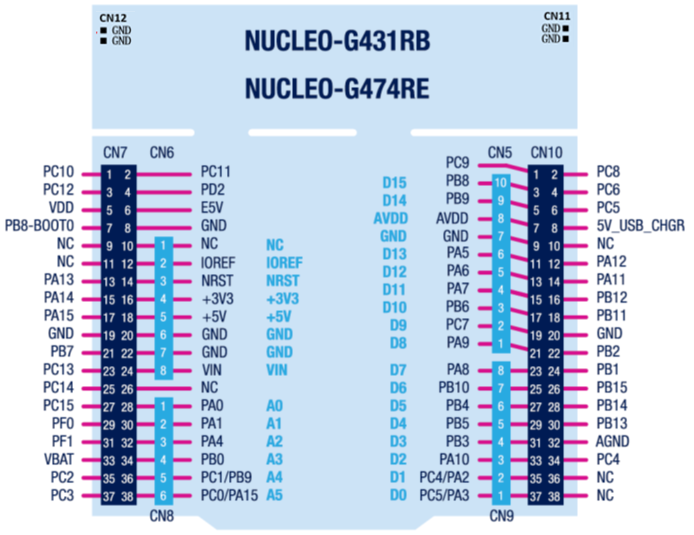


图53 Stm32G431RB单片机板引脚图

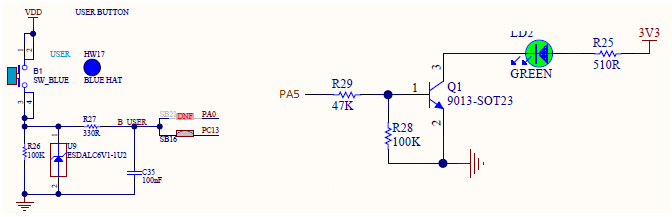


图54 Nucleo-G431RB开发板上按键和发光二极管LD2原理图