# 写在前面：

之前版本的红外传输，每次只能发送一个0-9的数字，范围小，还不连续。在此基础上，改进如下.

# 发送一个数字串（将一个大的数据，拆开成字符发送）

while (1)

{

printf("\r\n 请输入需要发送的数据: ");

scanf("%d",&INPUT1);

sprintf(tstr,"%d",INPUT1);

HAL\_Delay(100);

for(int i=0;i<500;i++)

{

INPUT=tstr[i]-48; //ASC码偏移量0的ASC码就是48

if(INPUT==0||tstr[i]==0) //数据为0或者为空

break;

else if(INPUT>0&&INPUT<10)

{

HAL\_TIM\_PWM\_Start(&htim16,TIM\_CHANNEL\_1);

HAL\_Delay(INPUT\*5);

HAL\_TIM\_PWM\_Stop(&htim16,TIM\_CHANNEL\_1);

HAL\_Delay(200-INPUT\*5);

tstr[i]=0; //发送之后得清除

}

HAL\_Delay(1000);

}

}

# 可以重复发送字符（接收字符之后串口必须初始化才能再用一次getchar）：

while (1)

{

printf("\r\n 请输入需要发送的数据: ");

test\_char=getchar();

HAL\_Delay(100);

INPUT=test\_char-96; //a的ASC码是97偏移量置为1发送小写字母 a对1

if(INPUT>0&&INPUT<10)

{

HAL\_TIM\_PWM\_Start(&htim16,TIM\_CHANNEL\_1);

HAL\_Delay(INPUT\*5);

HAL\_TIM\_PWM\_Stop(&htim16,TIM\_CHANNEL\_1);

HAL\_Delay(200-INPUT\*5);

}

MX\_LPUART1\_UART\_Init();

}

# 获取字符串：

while (1)

{

printf("\r\n 请输入需要发送的数据: ");

gets(charin);

MX\_LPUART1\_UART\_Init();

}

字符串会接受回车！！！回车的ASC码为13

# 小总结：

由以上三个小实验，并进行合理推测，我们可以得出以下结论：

理论上我们可以输入一大段字符串（实验三证明可行），字符转ASC码并进行进一步编码（实验二证明可行），最后将编码分割（+1避开0抗干扰）为一个一个的数字发送（实验一证明可行）。

最后修改就收逻辑。

通过ASC码表我们知道，我们常用的字符在“32号空格”到“126号~”之间。前31个不用，如果将31号定为新的原点，即32号空格为01则126号为126-31=95，两位新的ASC码即可表示所有常用字符。

发送的时候，将新的ASC码拆开，每个数字加一构成编码（1-10）避开0。解码的时候再减一组合恢复新的ASC码。利用新的ASC码构造解码矩阵。

# 信号的调制与解调

STM32内部的IRTIM可实现调制功能，其结构如图7-5所示。

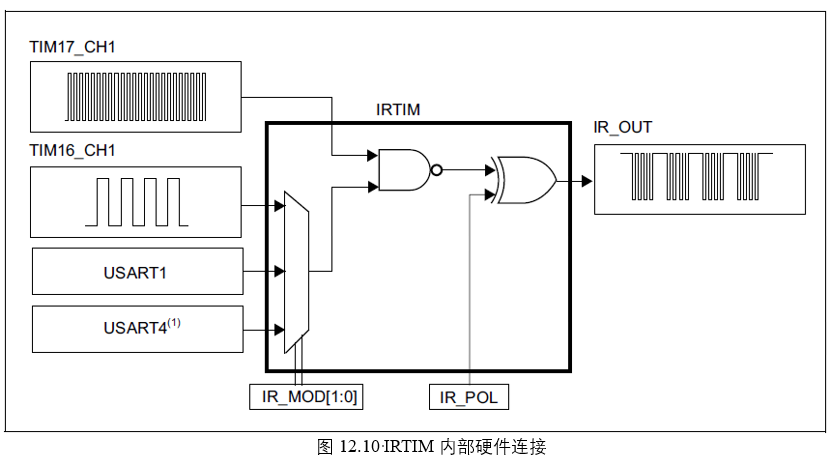


图7-5 IRTIM内部结构

在其输入端用定时器17通道1来生成载波，用定时器16通道1、串口1或串口4生成包络。此实验中我们选用定时器16通道1，两路信号经过与非门后调制成功，后边的异或门可以将生成的信号反相输出，但此实验中不用反相。上述四选一数据选择器与反相输出的设置分别在SYSCFG\_CFGR1的IR\_MOD[1:0]位与IR\_POL位。

与非门电路分析：包络（定时器16）为高电平，其调制输出端口信号IR\_OUT为载波信号反相；包络为低电平，其调制输出端口信号IR\_OUT为高电平。如图7-6所示，第一路金黄色信号为定时器16输出信号（包络），第二路天蓝色信号为IRTIM调制载之后的信号。



图7-6 定时器16输出信号与IRTIM调制信号关系图

通过查询资料不难发现，一般家用红外遥控器都是38Khz调制的红外信号。此实验中将系统时钟设置为最高的64Mhz，用于产生载波的定时器17工作在“通道1生成PWM”的模式下，预分频系数128，计数周期14，反转7（周期的一半）。理论上载波信号频率为64Mhz/128/14=35.71Khz，实际测量如图7-7通道2频率35.7KHz与理论值无误，并且接近38KHz，占空比为50%，可应用于红外发射管。

产生信号包络的定时器16，同样工作在“通道1生成PWM”的模式下，预分频系数设置为128，计数周期250，反转125（周期的一半）。包络信号频率为64Mhz/128/250=2Khz，实际测量如图7-7通道1频率2KHz与理论值无误，占空比为50%，便于接收端检测。每当定时器16工作1ms（2个周期），其产生2个脉冲信号，故发射端向外发射2个脉冲组。本实验中将定时器16工作时间量化为5ms的倍数，并且引入偏置量5ms抵抗低频干扰，并利用互补延时使得发送一个字符的时间为固定值。调制算法如下：定时器工作时间=要发送的数\*5ms+5ms（偏置）。

红外接收管（内置38KHz滤波器）与电容电阻搭建的红外接收模块可实现包络检波，输出解调信号。

通过示波器测量如图7-7所示：



图7-7 红外通信系统重要节点信号关系

图7-7中信号依次为：定时器16产生的包络信号（金黄色），定时器17产生的载波信号（天蓝色），红外发射管发射信号（紫色），红外接模块输出的信号（深蓝色）。其周期、频率非常接近理论值。红外发射管发射信号经过IRTIM与发送设备的三极管两次反向，所以其包络即定时器16产生的信号，并且相位相同。

# 信号的编码与解码

1、编码

通过控制定时器打开关闭的时间间隔，来控制发送的脉冲个数。本实验中对ASC码为32的“空格”到ASC码为126的“~”进行了编码，包含所有常用字符，并且94个字符利用两位十进制数据即可传输。

先将待传输字符串导入缓存数组Char\_In[500]（单次最长发送500字符），然后用循环遍历数组。获取其第i个字符ASC码值，若为“13”即回车，则清除Char\_In[i]与检测出的ASC码值，并且立即结束循环（时序最快）。若不是“13”，则获取ASC码偏移量ASC\_New=ASC-31，进而根据ASC码偏移量计算出偏移量的十位与个位（并且加一，避免发送“0”与不发送相同）得到发送数据。

最后按照调制算法依次对这两个数据进行调制并发送。打开定时器16，延时“编码后数据\*5ms+5ms（偏置）”，关闭定时器16，延时“55-编码后数据\*5ms”，最后再延时40ms。即发送一个数据，需要100ms，其中定时器最长工作时间10\*5ms+5ms=55ms。例如：发送数据1，定时器16工作10ms，即20个周期（定时器16产生20个脉冲）。以此类推，发送数据2时产生30个脉冲……

2、解码

解码部分系统时钟为64Mhz，解码时用到定时器2产生接收复位信号，其工作在“通道1生成PWM”的模式下，预分频系数1024，计数周期3750，反转1875（周期的一半）。接收复位信号频率为64Mhz/1024/3750=16.6hz，周期为60ms，占空比为50%。当收到发送端发出信号的那一刻，打开定时器2开始计时,并记录收到的下降沿个数（也即解调高电平个数）。定时器2开始工作60ms后产生一个下降沿触发中断，先根据记录的高电平数解码出数据。

解码算法：减五除十后赋值给整型数data（自动取整）。例如：上述编码部分提到发送数据1，定时器16工作10ms，产生20个脉冲，（20-5）/10=1.5取整为1，成功解码。

每接收两次数据（一次为ASC偏移码的十位，一次为ASC偏移码的个位），利用这两个数据再10\*10的二位译码矩阵查找对应字符输出，并及时清除各个辅助变量进行下一次接收。

# 传输速度与抗干扰性能分析

1、传输速度分析

由图7-8所示，发射端发送一个字符的时序：定时器16最大工作时间10\*量化时间（5ms）+量化时间（5ms）=55ms，互补延时5ms，等待40ms。发送一个数据总计需要100ms，发送一个字符需要2个数据（ASC偏移码的十位、个位）即200ms。接收端时序与发送端时序紧密配合，接收一个字符也需要200ms。综上所述，每秒传输5个字符。

定时器16打开

定时器16关闭（互补延时）

延时等待

信号检测开启

译码（在串口输出）

发射端

接收端

时间线

0ms

60ms

100ms

图7-8 红外通信系统时序图

拓展研究：改变定时器16工作的量化时间可改变其抗噪与传输速度性能，量化时间值越小传输速度越高，量化时间越高抗干扰性能越强。上述量化时间为5ms，经测试无误码传输的最小量化时间为3ms，即传输一个字符最小需要（12\*3ms+40ms）\*2=152ms，1s可传输6.58个字符。

2、抗干扰性能分析

发送设备串口通信与发送端红外信号时序：接收到需要发送的字符串时，每当对定时器16进行操作（此过程持续600ms）之后，均延时40ms再进行下一次输出操作做，以保证每个数据顺利发送。所有字符发送完毕后（检测到回车结束或溢出500字符）提示用户可以发送下一个字符串，避免了上一个字符串还没成功发送完毕，就又更新了串口数据内容。

红外发射管接收管工作时序：发送一个数据红外管最大工作时间为55ms，接收管复位时间为60ms大于发射管。确保了红外发射管发出的所有脉冲动能被接收管收到。

发射端采用偏置，解码时不是一一对应而是采用减五除十的方式，极大程度上降低了误码率：例如收到15到24个脉冲，都可以解码为1，即便发送端没有准确地发出20个脉冲而是小范围波动依旧可以正确解码，其次此解码算法确保了每次发送数据的标准脉冲个数都在该接收区间的正中央（例如10在105-114的正中央……）。还避免了红外发射管的误触发造成的影响，收到1-14个脉冲时，减五除十取整后为0即没接收到任何数据。

抗低频干扰（包络）分析如下：由（15-5）/10取整为1，所以接收端译码为1最少需要15个脉冲，接收端每次计数间隔60ms。即60ms内15周期，频率为15/0.06s=250Hz。综上所述，此系统抵抗包络为250Hz以下的低频红外信号干扰。

抗高频干扰（包络）分析如下：由（114-5）/10取整为10，接收端译码为10最多需要114个脉冲，接收端每次计数间隔60ms。即60ms内114周期，频率为114/0.06s=1900Hz。综上所述，此系统抵抗包络为1900Hz以上的高频红外信号干扰。

# 发射端工程创建

①在时钟配置界面下，首先将系统时钟复用更改为锁相环时钟PLLCLK，系统时钟设置为64Mhz，AHB总线与APB总线分频系数均为1，此时可以看到送给定时器的时钟就是系统时钟64Mhz，LPUART1时钟复用选用PCLK其工作频率也是64Mhz。

②定时器设置：定时器16通道1工作模式为“PWM Generation CH1”，预分频系数127，计数周期250，脉冲125。定时器17通道1工作模式为“PWM Generation CH1”，预分频系数127，计数周期14，脉冲7。

③IRTIM输出不反相，信号来源选择为定时器16，调制信号输出IR\_OUT映射在PB9引脚。

④串口LPUART1工作在异步模式，波特率115200，字长8位，无奇偶校验，停止位1位，将LPUART1\_Tx映射在PA2引脚，将LPUART1\_Rx映射在PA3引脚。

⑤发射端硬件连接：串口引脚不需要单独接线，红外发射模块的输入信号线连接PB9引脚，并连接VCC（5V供电）与GND。

|  |
| --- |
| #include "main.h"  #include "stdio.h"  #include "string.h"  //MODE大抗干扰强，MODE小传输速率高 要求大于等于3（否则出现误码）  //当前配置下，发射端MODE值必须是接收端MODE值的二分之一  #define MODE 5  #define unit8\_t unsigned char  #define PUTCHAR\_PROTOTYPE int fputc(int ch,FILE \*f)  #define GETCHAR\_PROTOTYPE int fgetc(FILE \*f)  #define BACKSPACE\_PROTOTYPE int \_backspace(FILE \*f)  unsigned char Char\_In[500]; //定义500字符长度的输入字符 初始都是0  volatile int ASC; //ASC码值  volatile int ASC\_New; //新的ASC码值  volatile int Transfer\_Ten; //发送十位  volatile int Transfer\_One; //发送十位  UART\_HandleTypeDef hlpuart1;  TIM\_HandleTypeDef htim16;  TIM\_HandleTypeDef htim17;  void SystemClock\_Config(void);  static void MX\_GPIO\_Init(void);  static void MX\_IRTIM\_Init(void);  static void MX\_LPUART1\_UART\_Init(void);  static void MX\_TIM16\_Init(void);  static void MX\_TIM17\_Init(void);  int main(void)  {  HAL\_Init();  SystemClock\_Config();  MX\_GPIO\_Init();  MX\_IRTIM\_Init();  MX\_LPUART1\_UART\_Init();  MX\_TIM16\_Init();  MX\_TIM17\_Init();  HAL\_TIM\_PWM\_Start(&htim17,TIM\_CHANNEL\_1);  while(1)  {  printf("准备发送: \r\n"); //提示信息  gets(Char\_In); //输入超长字符串至字符串数组，回车结尾13  for(int i=0;i<500;i++)  {  ASC=Char\_In[i];  if(ASC==13)  {  Char\_In[i]=0;  ASC=0;  break; //检测到回车立即结束，跳出大循环  }  else  {  ASC\_New=ASC-31;  Transfer\_Ten=ASC\_New/10+1; //计算：传输十位  Transfer\_One=ASC\_New%10+1; //计算：传输个位  HAL\_TIM\_PWM\_Start(&htim16,TIM\_CHANNEL\_1);  HAL\_Delay(Transfer\_Ten\*MODE+MODE);  HAL\_TIM\_PWM\_Stop(&htim16,TIM\_CHANNEL\_1);  HAL\_Delay(MODE\*11-Transfer\_Ten\*MODE);  HAL\_Delay(40);  HAL\_TIM\_PWM\_Start(&htim16,TIM\_CHANNEL\_1);  HAL\_Delay(Transfer\_One\*MODE+MODE);  HAL\_TIM\_PWM\_Stop(&htim16,TIM\_CHANNEL\_1);  HAL\_Delay(MODE\*11-Transfer\_One\*MODE);  HAL\_Delay(40);  Char\_In[i]=0;  ASC=0;  ASC\_New=0;  Transfer\_Ten=0;  Transfer\_One=0;  }  }  }  }  void SystemClock\_Config(void)  {  RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};  RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};  RCC\_PeriphCLKInitTypeDef PeriphClkInit = {0};  HAL\_PWREx\_ControlVoltageScaling(PWR\_REGULATOR\_VOLTAGE\_SCALE1);  RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;  RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;  RCC\_OscInitStruct.HSIDiv = RCC\_HSI\_DIV1;  RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;  RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;  RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI;  RCC\_OscInitStruct.PLL.PLLM = RCC\_PLLM\_DIV1;  RCC\_OscInitStruct.PLL.PLLN = 8;  RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;  RCC\_OscInitStruct.PLL.PLLQ = RCC\_PLLQ\_DIV2;  RCC\_OscInitStruct.PLL.PLLR = RCC\_PLLR\_DIV2;  if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)  Error\_Handler();  RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK  |RCC\_CLOCKTYPE\_PCLK1;  RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;  RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;  RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;  if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != HAL\_OK)  Error\_Handler();  PeriphClkInit.PeriphClockSelection = RCC\_PERIPHCLK\_LPUART1;  PeriphClkInit.Lpuart1ClockSelection = RCC\_LPUART1CLKSOURCE\_PCLK1;  if (HAL\_RCCEx\_PeriphCLKConfig(&PeriphClkInit) != HAL\_OK)  Error\_Handler();  }  static void MX\_IRTIM\_Init(void)  {  \_\_HAL\_SYSCFG\_IRDA\_ENV\_SELECTION(HAL\_SYSCFG\_IRDA\_ENV\_SEL\_TIM16);  \_\_HAL\_SYSCFG\_IRDA\_OUT\_POLARITY\_SELECTION(HAL\_SYSCFG\_IRDA\_POLARITY\_NOT\_INVERTED);  }  static void MX\_LPUART1\_UART\_Init(void)  {  hlpuart1.Instance = LPUART1;  hlpuart1.Init.BaudRate = 115200;  hlpuart1.Init.WordLength = UART\_WORDLENGTH\_8B;  hlpuart1.Init.StopBits = UART\_STOPBITS\_1;  hlpuart1.Init.Parity = UART\_PARITY\_NONE;  hlpuart1.Init.Mode = UART\_MODE\_TX\_RX;  hlpuart1.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;  hlpuart1.Init.OneBitSampling = UART\_ONE\_BIT\_SAMPLE\_DISABLE;  hlpuart1.Init.ClockPrescaler = UART\_PRESCALER\_DIV1;  hlpuart1.AdvancedInit.AdvFeatureInit = UART\_ADVFEATURE\_NO\_INIT;  hlpuart1.FifoMode = UART\_FIFOMODE\_DISABLE;  if (HAL\_UART\_Init(&hlpuart1) != HAL\_OK)  Error\_Handler();  if (HAL\_UARTEx\_SetTxFifoThreshold(&hlpuart1, UART\_TXFIFO\_THRESHOLD\_1\_8) != HAL\_OK)  Error\_Handler();  if (HAL\_UARTEx\_SetRxFifoThreshold(&hlpuart1, UART\_RXFIFO\_THRESHOLD\_1\_8) != HAL\_OK)  Error\_Handler();  if (HAL\_UARTEx\_DisableFifoMode(&hlpuart1) != HAL\_OK)  Error\_Handler();  }  static void MX\_TIM16\_Init(void)  {  TIM\_OC\_InitTypeDef sConfigOC = {0};  TIM\_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};  htim16.Instance = TIM16;  htim16.Init.Prescaler = 127;  htim16.Init.CounterMode = TIM\_COUNTERMODE\_UP;  htim16.Init.Period = 250;//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*320;  htim16.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;  htim16.Init.RepetitionCounter = 0;  htim16.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;  if (HAL\_TIM\_Base\_Init(&htim16) != HAL\_OK)  Error\_Handler();  if (HAL\_TIM\_PWM\_Init(&htim16) != HAL\_OK)  Error\_Handler();  sConfigOC.OCMode = TIM\_OCMODE\_PWM1;  sConfigOC.Pulse = 125;//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*160;  sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;  sConfigOC.OCNPolarity = TIM\_OCNPOLARITY\_HIGH;  sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;  sConfigOC.OCIdleState = TIM\_OCIDLESTATE\_RESET;  sConfigOC.OCNIdleState = TIM\_OCNIDLESTATE\_RESET;  if (HAL\_TIM\_PWM\_ConfigChannel(&htim16, &sConfigOC, TIM\_CHANNEL\_1) != HAL\_OK)  Error\_Handler();  sBreakDeadTimeConfig.OffStateRunMode = TIM\_OSSR\_DISABLE;  sBreakDeadTimeConfig.OffStateIDLEMode = TIM\_OSSI\_DISABLE;  sBreakDeadTimeConfig.LockLevel = TIM\_LOCKLEVEL\_OFF;  sBreakDeadTimeConfig.DeadTime = 0;  sBreakDeadTimeConfig.BreakState = TIM\_BREAK\_DISABLE;  sBreakDeadTimeConfig.BreakPolarity = TIM\_BREAKPOLARITY\_HIGH;  sBreakDeadTimeConfig.BreakFilter = 0;  sBreakDeadTimeConfig.AutomaticOutput = TIM\_AUTOMATICOUTPUT\_DISABLE;  if (HAL\_TIMEx\_ConfigBreakDeadTime(&htim16, &sBreakDeadTimeConfig) != HAL\_OK)  Error\_Handler();  HAL\_TIM\_MspPostInit(&htim16);  }  static void MX\_TIM17\_Init(void)  {  TIM\_OC\_InitTypeDef sConfigOC = {0};  TIM\_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0}; htim17.Instance = TIM17;  htim17.Init.Prescaler = 127;  htim17.Init.CounterMode = TIM\_COUNTERMODE\_UP;  htim17.Init.Period = 14;//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*16;  htim17.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;  htim17.Init.RepetitionCounter = 0;  htim17.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;  if (HAL\_TIM\_Base\_Init(&htim17) != HAL\_OK)  Error\_Handler();  if (HAL\_TIM\_PWM\_Init(&htim17) != HAL\_OK)  Error\_Handler();  sConfigOC.OCMode = TIM\_OCMODE\_PWM1;  sConfigOC.Pulse = 7;//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8;  sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;  sConfigOC.OCNPolarity = TIM\_OCNPOLARITY\_HIGH;  sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;  sConfigOC.OCIdleState = TIM\_OCIDLESTATE\_RESET;  sConfigOC.OCNIdleState = TIM\_OCNIDLESTATE\_RESET;  if (HAL\_TIM\_PWM\_ConfigChannel(&htim17, &sConfigOC, TIM\_CHANNEL\_1) != HAL\_OK)  Error\_Handler();  sBreakDeadTimeConfig.OffStateRunMode = TIM\_OSSR\_DISABLE;  sBreakDeadTimeConfig.OffStateIDLEMode = TIM\_OSSI\_DISABLE;  sBreakDeadTimeConfig.LockLevel = TIM\_LOCKLEVEL\_OFF;  sBreakDeadTimeConfig.DeadTime = 0;  sBreakDeadTimeConfig.BreakState = TIM\_BREAK\_DISABLE;  sBreakDeadTimeConfig.BreakPolarity = TIM\_BREAKPOLARITY\_HIGH;  sBreakDeadTimeConfig.BreakFilter = 0;  sBreakDeadTimeConfig.AutomaticOutput = TIM\_AUTOMATICOUTPUT\_DISABLE;  if (HAL\_TIMEx\_ConfigBreakDeadTime(&htim17, &sBreakDeadTimeConfig) != HAL\_OK)  Error\_Handler();  HAL\_TIM\_MspPostInit(&htim17);  }  static void MX\_GPIO\_Init(void)  {  GPIO\_InitTypeDef GPIO\_InitStruct = {0};  \_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();  \_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();  GPIO\_InitStruct.Pin = GPIO\_PIN\_9;  GPIO\_InitStruct.Mode = GPIO\_MODE\_AF\_PP;  GPIO\_InitStruct.Pull = GPIO\_NOPULL;  GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;  GPIO\_InitStruct.Alternate = GPIO\_AF0\_IR;  HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStruct);  }  PUTCHAR\_PROTOTYPE //重定向 fputc()函数  {  HAL\_UART\_Transmit(&hlpuart1,(unit8\_t\*) &ch,1,0xFFFF); //调用串口发送函数  return ch; //返回发送的字符  }  GETCHAR\_PROTOTYPE //重定向 fgetc()函数  {  uint8\_t value; //定义无符号字符型变量 value  while((LPUART1->ISR & 0x00000020)==0){} //判断串口是否接收到字符  value=(uint8\_t)LPUART1->RDR; //读取串口接收到的字符  HAL\_UART\_Transmit(&hlpuart1,(uint8\_t \*)&value,1,0x1000); //回显接收到的字符  return value; //返回接收到的值 value  }  BACKSPACE\_PROTOTYPE //重定向\_\_backspace 函数  { return 0; }  void Error\_Handler(void)  {  \_\_disable\_irq();  while (1)  {}  }  #ifdef USE\_FULL\_ASSERT  void assert\_failed(uint8\_t \*file, uint32\_t line)  { }  #endif |

# 接收端工程创建

1. 接收端时钟配置与发射端时钟配置相同均为64Mhz。

②定时器2通道1工作在“PWM Generation CH1”模式下，预分频系数1023，计数周期3750，脉冲1875。

③串口LPUART1设置与输入端串口设置相同。

④将PC4、PC5引脚均设置为下降沿触发的外部中断，并在NVIC界面下使能4到15号外部中断。

⑤接收端硬件连接：定时器输出引脚PA0连接外部中断引脚PC4，红外接收模块反馈信号线接外部中断引脚PC5，并连接VCC（5V供电）与GND。

|  |
| --- |
| #include "main.h"  #include "stdio.h"  #include "string.h"  //MODE大抗干扰强，MODE小传输速率高 要求大于等于6（否则出现误码）  //当前配置下，发射端MODE值必须是接收端MODE值的二分之一  #define MODE 10  #define unit8\_t unsigned char  #define PUTCHAR\_PROTOTYPE int fputc(int ch,FILE \*f)  #define GETCHAR\_PROTOTYPE int fgetc(FILE \*f)  #define BACKSPACE\_PROTOTYPE int \_backspace(FILE \*f)  unsigned char Char\_Out[10][10]=  {  ' ', ' ', '!', '"', '#', '$', '%', '&', ' ', '(',  ')', '\*', '+', ',', '-', '.', '/', '0', '1', '2',  '3', '4', '5', '6', '7', '8', '9', ':', ';', '<',  '=', '>', '?', '@', 'A', 'B', 'C', 'D', 'E', 'F',  'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P',  'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',  '[', '\ ', ']', '^', ' ', '、', 'a', 'b', 'c', 'd', //92反斜留空隙 96顿号  'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n',  'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x',  'y', 'z', '{', '|', '}', '~', ' ', ' ', ' ', ' '  };  volatile int High\_number=0; //脉冲数量  volatile int Transfer\_Data=0; //传输数据  volatile int Output\_Y=0; //解码矩阵坐标  volatile int Output\_X=0; //  int Flag\_TIM2=0; //定时器2通道1标志位 1打开0关闭 增加计数的准确性  int Flag\_Transfer=1; //传输标志，判断传输十位（1）传输个位（0）  UART\_HandleTypeDef hlpuart1;  TIM\_HandleTypeDef htim2;  void SystemClock\_Config(void);  static void MX\_GPIO\_Init(void);  static void MX\_LPUART1\_UART\_Init(void);  static void MX\_TIM2\_Init(void);  int main(void)  {  HAL\_Init();  SystemClock\_Config();  MX\_GPIO\_Init();  MX\_LPUART1\_UART\_Init();  MX\_TIM2\_Init();  printf("开始接收 \r\n");  while (1)  HAL\_Delay(10);  }  void SystemClock\_Config(void)  {  RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};  RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};  RCC\_PeriphCLKInitTypeDef PeriphClkInit = {0};  HAL\_PWREx\_ControlVoltageScaling(PWR\_REGULATOR\_VOLTAGE\_SCALE1);  RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;  RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;  RCC\_OscInitStruct.HSIDiv = RCC\_HSI\_DIV1;  RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;  RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;  RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI;  RCC\_OscInitStruct.PLL.PLLM = RCC\_PLLM\_DIV1;  RCC\_OscInitStruct.PLL.PLLN = 8;  RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;  RCC\_OscInitStruct.PLL.PLLQ = RCC\_PLLQ\_DIV2;  RCC\_OscInitStruct.PLL.PLLR = RCC\_PLLR\_DIV2;  if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)  Error\_Handler();  RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK  |RCC\_CLOCKTYPE\_PCLK1;  RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;  RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;  RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV1;  if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != HAL\_OK)  Error\_Handler();  PeriphClkInit.PeriphClockSelection = RCC\_PERIPHCLK\_LPUART1;  PeriphClkInit.Lpuart1ClockSelection = RCC\_LPUART1CLKSOURCE\_PCLK1;  if (HAL\_RCCEx\_PeriphCLKConfig(&PeriphClkInit) != HAL\_OK)  Error\_Handler();  }  static void MX\_LPUART1\_UART\_Init(void)  {  hlpuart1.Instance = LPUART1;  hlpuart1.Init.BaudRate = 112500;  hlpuart1.Init.WordLength = UART\_WORDLENGTH\_8B;  hlpuart1.Init.StopBits = UART\_STOPBITS\_1;  hlpuart1.Init.Parity = UART\_PARITY\_NONE;  hlpuart1.Init.Mode = UART\_MODE\_TX\_RX;  hlpuart1.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;  hlpuart1.Init.OneBitSampling = UART\_ONE\_BIT\_SAMPLE\_DISABLE;  hlpuart1.Init.ClockPrescaler = UART\_PRESCALER\_DIV1;  hlpuart1.AdvancedInit.AdvFeatureInit = UART\_ADVFEATURE\_NO\_INIT;  hlpuart1.FifoMode = UART\_FIFOMODE\_DISABLE;  if (HAL\_UART\_Init(&hlpuart1) != HAL\_OK)  Error\_Handler();  if (HAL\_UARTEx\_SetTxFifoThreshold(&hlpuart1, UART\_TXFIFO\_THRESHOLD\_1\_8) != HAL\_OK)  Error\_Handler();  if (HAL\_UARTEx\_SetRxFifoThreshold(&hlpuart1, UART\_RXFIFO\_THRESHOLD\_1\_8) != HAL\_OK)  Error\_Handler();  if (HAL\_UARTEx\_DisableFifoMode(&hlpuart1) != HAL\_OK)  Error\_Handler();  }  }  static void MX\_TIM2\_Init(void)  {  TIM\_MasterConfigTypeDef sMasterConfig = {0};  TIM\_OC\_InitTypeDef sConfigOC = {0};  htim2.Instance = TIM2;  htim2.Init.Prescaler = 1023;  htim2.Init.CounterMode = TIM\_COUNTERMODE\_UP;  htim2.Init.Period = 375\*MODE;//3750;  htim2.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;  htim2.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;  if (HAL\_TIM\_PWM\_Init(&htim2) != HAL\_OK)  Error\_Handler();  sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;  sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;  if (HAL\_TIMEx\_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL\_OK)  Error\_Handler();  sConfigOC.OCMode = TIM\_OCMODE\_PWM1;  sConfigOC.Pulse = 375\*MODE/2;//1875;  sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;  sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;  if (HAL\_TIM\_PWM\_ConfigChannel(&htim2, &sConfigOC, TIM\_CHANNEL\_1) != HAL\_OK)  Error\_Handler();  HAL\_TIM\_MspPostInit(&htim2);  }  static void MX\_GPIO\_Init(void)  {  GPIO\_InitTypeDef GPIO\_InitStruct = {0};  \_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();  \_\_HAL\_RCC\_GPIOC\_CLK\_ENABLE();  GPIO\_InitStruct.Pin = GPIO\_PIN\_4|GPIO\_PIN\_5;  GPIO\_InitStruct.Mode = GPIO\_MODE\_IT\_FALLING;  GPIO\_InitStruct.Pull = GPIO\_NOPULL;  HAL\_GPIO\_Init(GPIOC, &GPIO\_InitStruct);  HAL\_NVIC\_SetPriority(EXTI4\_15\_IRQn, 0, 0);  HAL\_NVIC\_EnableIRQ(EXTI4\_15\_IRQn);  }  PUTCHAR\_PROTOTYPE //重定向 fputc()函数  {  HAL\_UART\_Transmit(&hlpuart1,(unit8\_t\*) &ch,1,0xFFFF); //调用串口发送函数  return ch; //返回发送的字符  }  GETCHAR\_PROTOTYPE //重定向 fgetc()函数  {  uint8\_t value; //定义无符号字符型变量 value  while((LPUART1->ISR & 0x00000020)==0){} //判断串口是否接收到字符  value=(uint8\_t)LPUART1->RDR; //读取串口接收到的字符  HAL\_UART\_Transmit(&hlpuart1,(uint8\_t \*)&value,1,0x1000); //回显接收到的字符  return value; //返回接收到的值 value  }  BACKSPACE\_PROTOTYPE //重定向\_\_backspace 函数  { return 0; }  void HAL\_GPIO\_EXTI\_Falling\_Callback(uint16\_t GPIO\_Pin)  {  if(GPIO\_Pin==0x0010) //PC4 定时器-----输出+清零  {  //计时结束关闭定时器2通道1  HAL\_TIM\_PWM\_Stop(&htim2,TIM\_CHANNEL\_1);  Flag\_TIM2=0;  Transfer\_Data=(High\_number-MODE/2)/MODE; //解码公式  if(Transfer\_Data!=0) //传输数据是不可能等于0的（防止误触发）  {  if(Flag\_Transfer==1) //传输的是十位  {  Output\_Y=Transfer\_Data-1; //接收数据1-10返回0-9  Flag\_Transfer=0; //转变传输标志位准备接收个位  }  else //传输的是个位  {  Output\_X=Transfer\_Data-1; //个位也传输完毕  printf("%c",Char\_Out[Output\_Y][Output\_X]);//输出对应字符  Output\_Y=0;Output\_X=0; //清除输出寻址  Flag\_Transfer=1; //转变传输标志位准备接收下一个十位  }  Transfer\_Data=0; //数据用完了及时清零——防止误触发  }  High\_number=0; //清除脉冲计数  }  else if(GPIO\_Pin==0x0020) //PC5 中断来临 如果定时器关闭则将其打开 并计数  {  if(Flag\_TIM2==0)  {  HAL\_TIM\_PWM\_Start(&htim2,TIM\_CHANNEL\_1);  Flag\_TIM2=1;  }  High\_number++;  }  else ;  }  void Error\_Handler(void)  {  \_\_disable\_irq();  while (1)  {}  }  #ifdef USE\_FULL\_ASSERT  void assert\_failed(uint8\_t \*file, uint32\_t line)  {}  #endif |