Lab6 串口交互

实验目的

- 熟练掌握在STM32F103上编写交互程序的方法。

实验器材

硬件

- STM32F103核心板1块;
- ST-Link 1个;
- 杜邦线(孔-孔）4根;
- 杜邦面包线（孔-针）3根;
- 面包线（针-针）若干;
- 按钮1个;
- 面包板1块。

软件

- STM32CubeIDE;
- Putty
- Serial Port Utility(友善串口调试助手)

实验原理

USB-UART CP2102

- 主芯片为CP2102，安装驱动后生成虚拟串口;
- USB取电，引出接口包括3.3V（<40mA），5V，GND，TX，RX，信号脚电平为3.3V，正逻辑;
- 板载状态指示灯、收发指示灯，正确安装驱动后状态指示灯会常亮，收发指示灯在通信的时候会闪烁，波特率越高亮度越低;
- 支持从300bps~1Mbps间的波特率;
- 通信格式支持：
    - 5,6,7,8位数据位;
    - 支持1,1.5,2停止位;
    - odd,even,mark,space,none校验。
- 支持操作系统：windows vista/xp/server 2003/200,Mac OS-X/OS-9,Linux;
- USB头为公头，可直接连接电脑USB口。

连接电路图

- CP2102连接

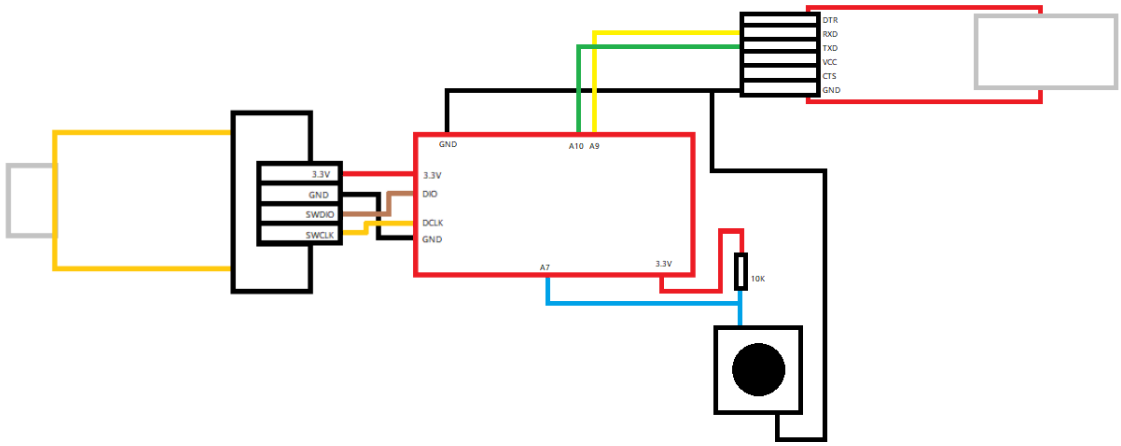
103	CP2102	颜色	意义
A9	RXD	黄色	103发送数据给PC
A10	TXD	绿色	PC发送数据给103
GND	GND	黑色	接地

实验步骤

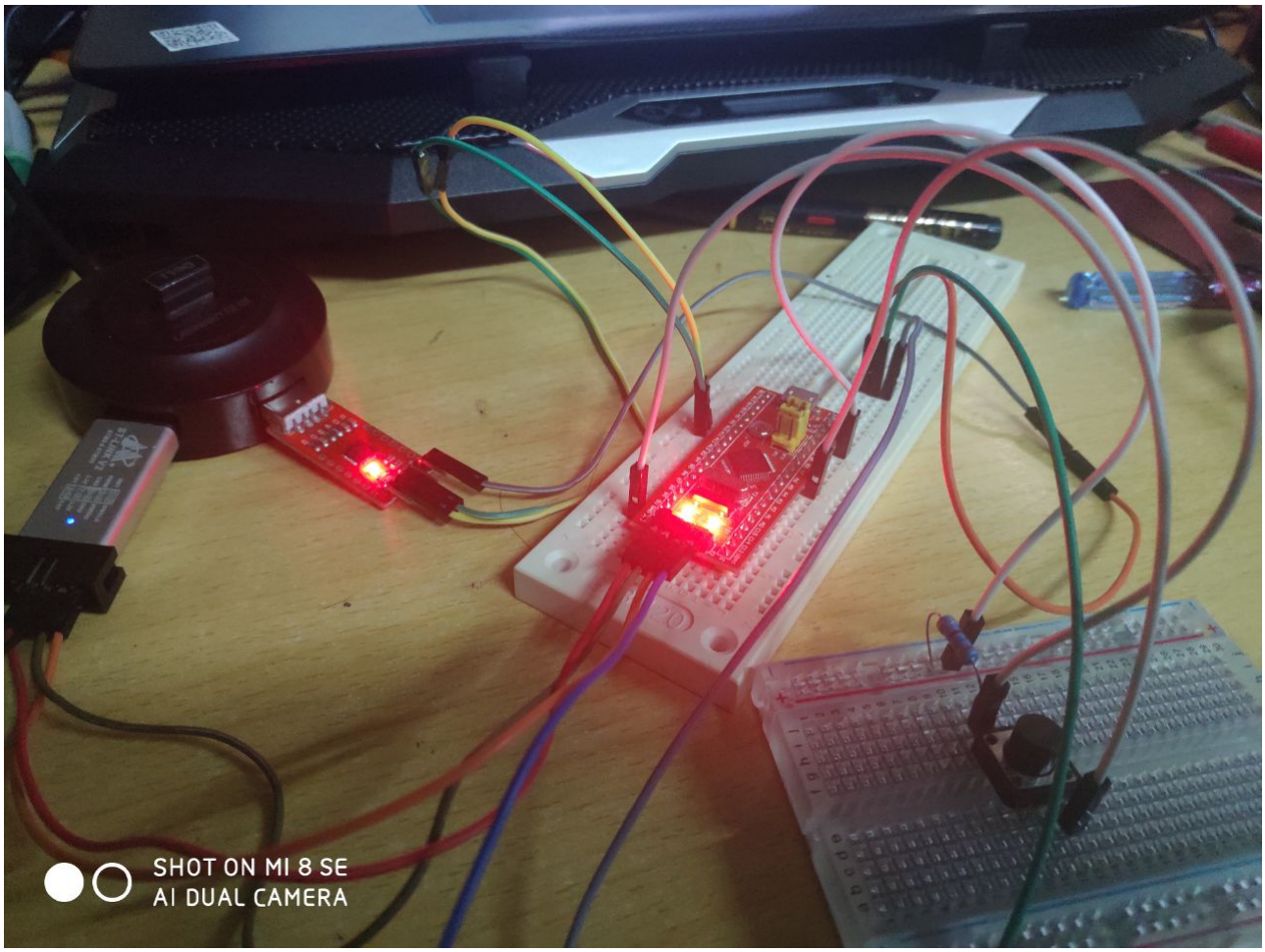
工程配置

连线

- 连线示意图



- 连线实物图



编写一个交互对话程序，能通过串口收到PC的指令，并分离出命令字和参数，将分离的结果返回;

- 代码

```
/* stm32f1xx_it.c中定义的外部变量 */
#define BUFFER_SIZE 64
extern uint8_t Receive_Buffer[64];
```

```
extern volatile uint8_t rx_length;
extern volatile uint8_t recv_end_flag;

/* stm32f1xx_it.c中定义的串口中断处理函数，用来接收由PC通过串口发送给MCU的任意长度的字符串 */
void USART1_IRQHandler(void)
{
  /* USER CODE BEGIN USART1_IRQn 0 */
  uint32_t tmp_flag = 0;
  uint32_t temp;
  /* USER CODE END USART1_IRQn 0 */
  HAL_UART_IRQHandler(&huart1);
  /* USER CODE BEGIN USART1_IRQn 1 */
  tmp_flag = __HAL_UART_GET_FLAG(&huart1, UART_FLAG_IDLE);
  if(tmp_flag != RESET)
  {
    __HAL_UART_CLEAR_IDLEFLAG(&huart1);
    temp = huart1.Instance->SR;     // 读取串口状态寄存器
    temp = huart1.Instance->DR;     // 读取串口数据寄存器
    HAL_UART_DMAStop(&huart1);
    temp = hdma_usart1_rx.Instance->CNDTR;  // 读取DMA剩余传输数量
    rx_length = BUFFER_SIZE - temp;
    recv_end_flag = 1;
  }
  /* USER CODE END USART1_IRQn 1 */
}

/* main.c中定义全局变量 */
#define BUFFER_SIZE 64                          // 缓冲区大小
uint8_t Receive_Buffer[64];                     // 串口接收缓冲区
volatile uint16_t recv_end_flag = 0;            // 串口接收完成的标记
volatile uint8_t rx_length = 0;                 // 串口接收的数据的长度
uint8_t Instruction[5] = {0};                   // 命令分离命令字
uint8_t Addr[8] = {0};                          // 命令分离地址参数
uint8_t Data[8] = {0};                          // 命令分离数据参数

/* main.c的main函数中加入 */
__HAL_UART_ENABLE_IT(&huart1, UART_IT_IDLE);                    // 开启串口空闲中断
HAL_UART_Receive_DMA(&huart1, Receive_Buffer, BUFFER_SIZE);    // 启动串口DMA接收

/* main.c的main函数while循环中加入 */
uint8_t i = 0, j;
if(recv_end_flag == 1)
{
  i = 0;
  /* 分离出指令的命令字 */
  while(Receive_Buffer[i] != ' ')
  {
      Instruction[i] = Receive_Buffer[i];
      i++;
  }
  sprintf(output, )
  HAL_UART_Transmit_DMA(&huart1, (uint8_t*)Instruction, strlen(Instruction));
  /* 分离出peek命令的地址参数 */
```

```
  if(!strcmp(Instruction, "peek"))
  {
      j = 7;          // 8位地址
      while(j > 0)
      {
          Addr[j--] = Receive_Buffer[++i];
      }
      Addr[0] = Receive_Buffer[++i];
      sprintf(output, "Instruction: %s; Addr: %#08x \r\n", Instruction, addr);
      HAL_UART_Transmit_DMA(&huart1, (uint8_t*)output, strlen((char*)output));
  }
  /* 分离出poke命令的地址参数和数据参数 */
  else if(!strcmp(Instruction, "poke"))
  {
      j = 7;          // 8位地址
      while(j > 0)
      {
          Addr[j--] = Receive_Buffer[++i];
      }
      Addr[0] = Receive_Buffer[++i];
      HAL_UART_Transmit_DMA(&huart1, (uint8_t*)Addr, strlen((char*)Addr));
      i+=2;
      j = 7;          // 8位数据
      while(j > 0)
      {
          Data[j--] = Receive_Buffer[i++];
      }
      Data[0] = Receive_Buffer[i++];
      sprintf(output, "Instruction: %s; Addr: %#08x; Data: %#08x \r\n",
Instruction, addr, data);
      HAL_UART_Transmit_DMA(&huart1, (uint8_t*)output, strlen((char*)output));
  }
  /* 分离出print命令的地址参数 */
  else if(!strcmp(Instruction, "print"))
  {
      j = 7;          // 8位地址
      while(j > 0)
      {
          Addr[j--] = Receive_Buffer[++i];
      }
      Addr[0] = Receive_Buffer[++i];
      sprintf(output, "Instruction: %s; Addr: %#08x \r\n", Instruction, addr);
      HAL_UART_Transmit_DMA(&huart1, (uint8_t*)output, strlen((char*)output));
  }
  /* 初始化，准备下一次接收 */
  for(i = 0; i < 5; i++)
  {
    Instruction[i] = 0;
  }
  rx_length = 0;
  recv_end_flag = 0;
  HAL_UART_Receive_DMA(&huart1, Receive_Buffer, BUFFER_SIZE);
}
```

能通过串口执行两条最简单的指令:

`peek <addr>` 以一个字为单位读取内存中`addr`位置的数据(`addr`是4字节对齐,十六进制的形式,长度为8位十六进制,没有引导字符，例如`00008000`),并以十六进制的形式输出结果，输出结果为自然序(高位在前)

- 代码

```
if(!strcmp(Instruction, "peek"))
{
  j = 7;
  while(j > 0)
  {
    i++;
    if((Receive_Buffer[i] == '0' && Receive_Buffer[i+1] == 'x') ||
Receive_Buffer[i] == 'x')
      continue;
    else
      Addr[j--] = Receive_Buffer[i];
  }
  Addr[0] = Receive_Buffer[++i];
  int addr = Array_to_Hex(Addr);                    // 字符串转换为整型
  sprintf((char*)Data, "%08X", *((int*)addr));     // 读取指定地址的数据
  HAL_UART_Transmit_DMA(&huart1, (uint8_t*)Data, strlen((char*)Data));
}
```

`poke <addr> <data>` 以一个字为单位修改内存中`addr`位置的数据为`data`(`addr`是4字节对齐,十六进制的形式,长度为8位十六进制，`data`也是十六进制的形式,长度为8位十六进制，为自然序高位在前)

- 代码

```
else if(!strcmp(Instruction, "poke"))
{
  j = 7;
  while(j > 0)
  {
    i++;
    if((Receive_Buffer[i] == '0' && Receive_Buffer[i+1] == 'x') ||
Receive_Buffer[i] == 'x')
      continue;
    else
      Addr[j--] = Receive_Buffer[i];
  }
  Addr[0] = Receive_Buffer[++i];
  int addr = Array_to_Hex(Addr);          // 字符串转换为整型
  i+=2;
  j = 7;
  while(j > 0)
  {
```

```
    if((Receive_Buffer[i] == '0' && Receive_Buffer[i+1] == 'x') ||
Receive_Buffer[i] == 'x')
    {
      i++;
      continue;
    }
    else
      Data[j--] = Receive_Buffer[i];
    i++;
  }
  Data[0] = Receive_Buffer[i++];
  int data = Array_to_Hex(Data);        // 字符串转换为整型
  *((int*)addr) = data;                 // 修改指定地址的数据为指定数据
}
```

注释

```
/* main.c中使用到的字符数组类型转换整型的函数 */
int Array_to_Hex(uint8_t Addr[])
{
    int addr = 0;
    int i;
    for(i = 0; i < 8; i++)
    {
        if(Addr[i] >= '0' && Addr[i] <= '9')
        {
            addr += (Addr[i] - '0') * pow(16, i);
        }
        else if(Addr[i] >= 'a' && Addr[i] <= 'z' || Addr[i] >= 'A' && Addr[i] <=
'Z')
        {
            addr += (Addr[i] - 'A' + 10) * pow(16,i);
        }
    }
    return addr;
}
```

扩展内容

`print <addr>`，**输出从addr开始的C语言字符串，addr不需要是4字节对齐的。**

- 代码

```
else if(!strcmp(Instruction, "print"))
{
  j = 7;
  while(j > 0)
  {
```

```
    i++;
    if((Receive_Buffer[i] == '0' && Receive_Buffer[i+1] == 'x') ||
Receive_Buffer[i] == 'x')
      continue;
    else
      Addr[j--] = Receive_Buffer[i];
  }
  Addr[0] = Receive_Buffer[++i];
  int addr = Array_to_Hex(Addr);    // 字符串转换为整型
  sprintf(output, "%s ", addr);     // 打印指定地址的字符串
  HAL_UART_Transmit_DMA(&huart1, (uint8_t*)output, strlen((char*)output));
}
```

实验结果

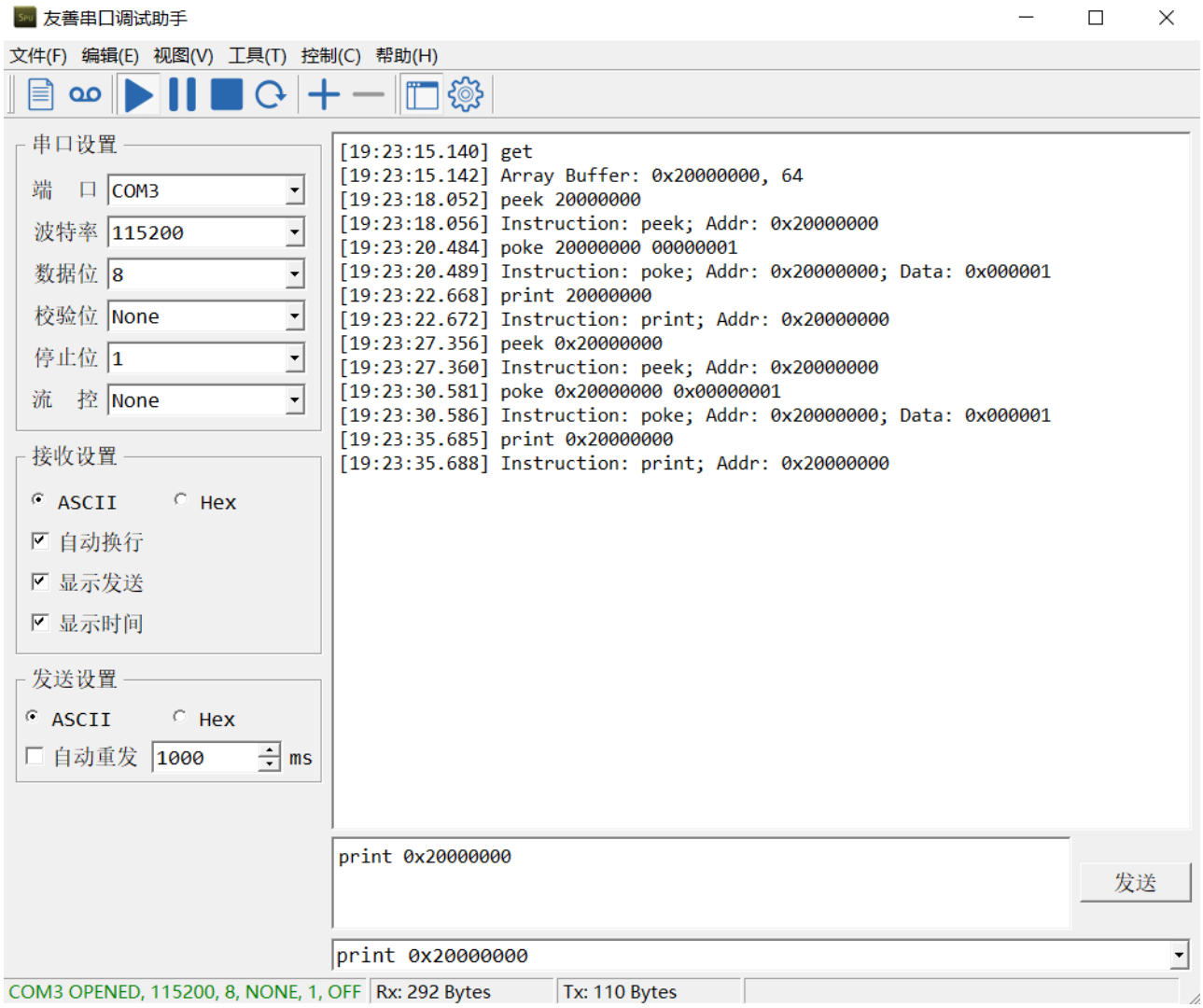
测试准备

- 为了验证实验完成的三个指令是否有效，添加以下代码进行测试。

```
/* main.c中加入  */
char Buffer[64] = "Hello World";                    // 用于测试的内存空间
uint8_t str[100], length;
else if(!strcmp(Instruction, "get"))                // 获取用于测试内存空间的首地
址
{
  length = sprintf(str, "Array Buffer: %p, %d\r\n", Buffer, 64);
  HAL_UART_Transmit_DMA(&huart1, str, length);
}
```

结果分析

- 指令分解



  - 指令get发送后，MCU返回用于测试的内存首地址是0x20000000，大小是64个字节;
  - 指令peek 200000000发送后，返回的是指令分离后得到的命令字peek和地址0x20000000;
  - 指令poke 200000000 00000001发送后，返回的是指令分离后得到的命令字poke和地址0x20000000，数据0x000000001;
  - 指令print 200000000发送后，返回的是指令分离后得到的命令字print和地址0x20000000;
  - 最后三条指令测试的是输入地址或数据的时候，输入格式中是否有"0x"表示十六进制都不影响指令分解结果，但要保证输入的数据是十六进制。

- peek指令和print指令

```
[19:30:46.837] get
[19:30:46.838] Array Buffer: 0x20000000, 64
[19:30:53.892] print 0x20000000
[19:30:53.897] Hello World
[19:31:00.188] peek 0x20000000
[19:31:00.191] 6C6C6548
[19:31:04.085] peek 0x20000004
[19:31:04.088] 6F57206F
[19:31:07.284] peek 0x20000008
[19:31:07.288] 00646C72
```

  - 指令get发送后，MCU返回用于测试的内存首地址是0x20000000，大小是64个字节;
  - 指令print 0x200000000发送后，返回的是存放在内存空间中的C语言字符串Hello World;

    - 字符串在存储的时候，是从左往右按顺序依次放在内存空间中，因此左边字符的地址比右边字符的地址低。
        - 指令`peek 0x200000000`发送后，返回的是存放在地址`0x20000000-0x20000003`的数据，由于是自然序输出(左边高地址右边低地址)，得到的数据为`6C6C6548`，查阅ASCII码表对应出字符串`lleH`;
        - 指令`peek 0x200000004`发送后，返回的是存放在地址`0x20000004-0x20000007`的数据，由于是自然序输出(左边高地址右边低地址)，得到的数据为`6F57206F`，查阅ASCII码表对应出字符串`ow o`;
        - 指令`peek 0x200000008`发送后，返回的是存放在地址`0x20000008-0x2000000B`的数据，由于是自然序输出(左边高地址右边低地址)，得到的数据为`00646C72`，查阅ASCII码表对应出字符串`dlr`;
- `poke`指令

```
[19:45:02.925] get
[19:45:02.926] Array Buffer: 0x20000000, 64
[19:45:06.949] peek 0x20000000
[19:45:06.952] 6C6C6548
[19:45:11.524] poke 0x20000000 0x4C4C4568
[19:45:17.772] print 0x20000000
[19:45:17.776] hELLo World
```

    - 指令`get`发送后，MCU返回用于测试的内存首地址是`0x20000000`，大小是64个字节;
    - 指令`peek 0x200000000`发送后，返回的是存放在地址`0x20000000-0x20000003`的数据，由于是自然序输出(左边高地址右边低地址)，得到的数据为`6C6C6548`，查阅ASCII码表对应出字符串`lleH`;
    - 指令`poke 0x200000000 0x4C4C4568`发送，用来修改地址`0x20000000`处的数据`0x6C6C6548`为`0x4C4C4568`，即从`lleH`变成`LLEh`。
    - 指令`print 0x200000000`发送后，返回的是存放在内存空间中的C语言字符串`hELLo World`;

实验心得

- 本次实验实现了简单的通过指令进行的串口交互。由于实验比较简单，所以没有遇到非常困难的地方。总结一下本次实验的要点：
    - 首要实现MCU对不定长字符串的接收。
    - 其次需要完成对指令的分解。
    - 接着需要对不同指令完成不同的操作。