**哈尔滨工程大学信通学院**

**实习手册**

**学生姓名 付丁一**

**学 号 2018080405**

**实习单位 中国高科**

**实习班级**

**指导老师 李北明 (校内)**

**王亚涛 (校外)**

**二○二一年七月**

|  |  |  |  |
| --- | --- | --- | --- |
| **时间** | **7.13** | **实习内容** | **STM32的开发（2）** |
| day7 1. 中断  2. 串口 中断与内部中断中断嵌套     假设有三个优先级012，0是最高。 当前正在处理1级别中断。 此时如果：   1. 0级别中断唤起    1. 当前中断被打断。记录当前位置后去处理0中断；之后返回1中断继续处理，最后回到正常流程中去。 2. 2级别中断唤起    1. 2中断挂起。等待1处理完之后再来处理2。 3. 同级别中断被唤起    1. 挂起。除非同时有效，则比较子优先级。  使用HAL库实现按键下降沿触发中断   Callback function：      然后Translate-Build-Download即可下载到板卡。  现象：按下按键后LED电平翻转。 原理与分析     [Nested Vectored Interrupt Controller\_Cortex-M3 Devices Generic User Guide](https://developer.arm.com/documentation/dui0552/a/cortex-m3-peripherals/nested-vectored-interrupt-controller?lang=en)  Tail-chaining是异常的背对背处理，没有中断之间的状态保存和恢复开销。当退出一个ISR并进入另一个ISR时，处理器跳过八个寄存器的弹出和八个寄存器的推入，因为这对堆栈内容没有影响。   * ISR：中断服务寄存器 * 为了加快响应（而产生的M3的功能）   向量表     * 中断整体流程     [微机原理个人总结之中断学习\_CSDN](https://blog.csdn.net/qq_40944242/article/details/105771065)  微机原理确实创世之源 Callback函数溯源中断流程中Callback处于某位置            串口数据收发通信和通信协议的分类通信的分类     单工、全双工、半双工示意图：    [What’s the Difference between Simplex and Duplex Fibre Optic Cable?\_Black Box Network](https://www.blackbox.be/en-be/page/25062/Resources/Technical-Resources/Black-Box-Explains/fibre-optic-cable/Simplex-vs-Duplex-Fibre) 通信协议   典型同步通信协议SPI/SSI举例：    [ADS1118 具有内部基准和温度传感器的兼容 SPI的 16 位模数转换器 (Rev. F)\_Texas Instruments](https://www.ti.com.cn/document-viewer/cn/ADS1118/datasheet/45-ZHCSEE1F#SBAS4573655)  异步串行通信举例：    [异步串行通信\_Wikipedia®](https://zh.wikipedia.org/wiki/%E5%BC%82%E6%AD%A5%E4%B8%B2%E8%A1%8C%E9%80%9A%E4%BF%A1) 单片机常用串行通信协议对比  |  |  |  |  |  | | --- | --- | --- | --- | --- | | 协议 | 名称 | 引脚说明 | 同步信号 | 通信方向 | | UART | Universal Asynchronous Receiver/Transmitter | TXD：发送端 RXD：接收端 GND：共地 | 异步通信 | 全双工 | | 1-wire | 单总线 | DQ：发送/接收端 | 异步通信 | 半双工 | | SPI | 串行外围设备接口 | SCK：同步时钟；MISO：主机输入，从机输出；MOSI：主机输出，从机输入；CS：若干片选信号 | 同步通信 | 全双工 | | IIC | 集成电路总线 | SCK：同步时钟；SDA：数据输入/输出 | 同步通信 | 半双工 |   单片机使用的异步串行通信字符格式如图 4-4 所示，一般情况下是 1 位起始位，8 位数据 位，无奇偶校验，1 位停止位。   STM32的串口资源  * 详见CubeMX  引脚复用与片内外设重映射  * “内部外设”  串口数据发送函数  * 工作方式   + 轮询     - 中断：接收或发送若干字节的数据就会进入中断     - DMA：传输过程不需要CPU干预 * HAL库   + 阻塞与非阻塞     - 发送：阻塞    使用CubeMX配置串口   连线：         串口重定向  * 改写C语言库函数，当连接器检查到用户编写了与 C 库函数同名的函数时，将优先使用用户编写的函数，从而实现对库函数的修改 * printf() 函数内部通过调用 fputc() 函数来实现数据输出，用户可以改写 fputc() 函数来实现串口重定向。   C  /\*\*  \* @brief: 重定向c库函数printf到DEBUG\_USARTx  \*/  int fputc(int ch, FILE \*f) {  HAL\_UART\_Transmit(&huart1, (uint8\_t \*)&ch, 1, 0xffff);  return ch;  }  /\*\*  \* @brief: 重定向c库函数getchar,scanf到DEBUG\_USARTx  \*/  int fgetc(FILE \*f) {  uint8\_t ch = 0;  HAL\_UART\_Receive(&huart1, &ch, 1, 0xffff);  return ch;  } main.c 在按键按下后，将Hello world！输出至串口。（轮询）  C  /\* Infinite loop \*/  /\* USER CODE BEGIN WHILE \*/  while (1) {  if(!KEY1){  HAL\_Delay(10);  while(!KEY1)  ;  printf( "Hello world!\r\n");  HAL\_Delay(10);  }  /\* USER CODE END WHILE \*/   应声虫实验 实验现象：“回声”。 开启Rx中断     C  void HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart) {  if(huart->Instance==USART1) {  Uart1ReceiveBuf[Uart1ReceiveCnt] = Uart1Temp[0];  Uart1ReceiveCnt++;  if(0x0a == Uart1Temp[0]) {  Uart1ReceiveFlag = 1;  }  HAL\_UART\_Receive\_IT(&huart1,(uint8\_t \*)Uart1Temp,REC\_LENGTH);  }  } main.c   C  while (1) {  if(Uart1ReceiveFlag) {  HAL\_UART\_Transmit(&huart1,Uart1ReceiveBuf,Uart1ReceiveCnt,0x10); //  for(int i = 0; i<Uart1ReceiveCnt; i++)  Uart1ReceiveBuf[i] = 0;  Uart1ReceiveCnt = 0;  Uart1ReceiveFlag = 0;  }  /\* USER CODE END WHILE \*/  /\* USER CODE BEGIN 3 \*/  } 现象  用串口控制LED的状态 配置同前。  实验现象：通过自定的协议，向串口发送指定指令，控制LED的状态。 实现判断指令逻辑   C  if(Uart1ReceiveFlag) {  printf( "The Received command is ：");  HAL\_UART\_Transmit(&huart1,Uart1ReceiveBuf,Uart1ReceiveCnt,0x10);  if(('L' == Uart1ReceiveBuf[0]) && ('E' == Uart1ReceiveBuf[1]) && ('D' == Uart1ReceiveBuf[2])) {  switch(Uart1ReceiveBuf[3]) {  case '0':  LED\_OFF;PRT\_SUCCESS  break;  case '1':  LED\_ON;PRT\_SUCCESS  break;  case '2':  LED\_TOG;PRT\_SUCCESS  break;  default:  PRT\_ERROR\_BIT\_MSG  break;  }  } else {  PRT\_ERROR\_HEAD\_MSG  }  //清除接收数组、计数器、标志位  for(int i = 0; i<Uart1ReceiveCnt; i++)  Uart1ReceiveBuf[i] = 0;  Uart1ReceiveCnt = 0;  Uart1ReceiveFlag = 0;  } uart.h / uart.c 除了一些常规的Uart需要的缓存字符串等之外，定义了发送指令错误时的回音。  C  //UART.h  #define REC\_LENGTH 1  #define MAX\_REC\_LENGTH 1024  #define PRT\_ERROR\_HEAD\_MSG printf( "ERROR order. Please check the head bit.\r\n");  #define PRT\_ERROR\_BIT\_MSG printf( "ERROR order. Please check the state bit.\r\n");  #define PRT\_SUCCESS printf( "success\r\n");  extern unsigned char Uart1ReceiveBuf[MAX\_REC\_LENGTH]; //UART1 存储接收数据  extern unsigned char Uart1ReceiveFlag; //UART1 接收完成标志  extern unsigned int Uart1ReceiveCnt; //UART1 接受数据计数器  extern unsigned char Uart1Temp[REC\_LENGTH]; //UART1 接收数据缓存  C  //UART.c  #include "uart.h"  unsigned char Uart1ReceiveBuf[MAX\_REC\_LENGTH] = {0}; //UART1 存储接收数据  unsigned char Uart1ReceiveFlag = 0; //UART1 接收完成标志  unsigned int Uart1ReceiveCnt = 0; //UART1 接受数据计数器  unsigned char Uart1Temp[REC\_LENGTH] = {0}; //UART1 接收数据缓存 main.h 在main.h之中，定义了一些有助于增加可读性的宏定义。  C  #define KEY1\_Pin GPIO\_PIN\_5  #define KEY1\_GPIO\_Port GPIOC  #define LED\_Pin GPIO\_PIN\_2  #define LED\_GPIO\_Port GPIOD  /\* USER CODE BEGIN Private defines \*/  #define KEY1 HAL\_GPIO\_ReadPin(KEY1\_GPIO\_Port,KEY1\_Pin)  #define LED\_ON HAL\_GPIO\_WritePin(LED\_GPIO\_Port,LED\_Pin,GPIO\_PIN\_RESET)  #define LED\_OFF HAL\_GPIO\_WritePin(LED\_GPIO\_Port,LED\_Pin,GPIO\_PIN\_SET)  #define LED\_TOG HAL\_GPIO\_TogglePin(LED\_GPIO\_Port,LED\_Pin) 实验现象 | | | |
| **指导教师** |  | **分数** |  |
| **实习评语:** | | | |