**中断与串口通讯实验报告**

**中英1702 菲华·帕兰斯 U201711458**

**一．实验目的**

1、 熟悉Keil开发环境，学习使用Go to Definition功能，学习和调用中断、串口等相关函数，并了解相关初始化操作；

2、 熟悉与实验相关的硬件电路，包括LED、按键、通用IO口电路，并掌握初始化及配置方法；

3、 熟练使用串口调试助手；

4、 加深对中断控制方式的理解；比较查询控制方式和中断控制方式，提出各自的适用条件。

**二．实验要求**

1、 编写程序，使LED0以2s的周期慢闪烁；

2、 采用库函数方式，编写串口通讯程序，进行串口通讯（MCU通过串口与上位机的对话，在收到上位机发过来的字符串后，随即返还给上位机）， 通讯时LED0快闪烁 （闪烁周期建议不小于0.1s），以表示进入串口工作状态；字符串长度自定，内容自选。（建议字符串长度不小于100）

3、 利用中断控制方式（注意中断初始化）同时实现上述两点要求；执行串口通讯中断子程序时LED0快速闪烁，无外部中断时执行主程序，保持LED0慢速闪烁。

4、 尝试利用程序控制方式实现上述功能，并对两种实现方法进行比较总结。

**三．程序流程图**



图1 - 左边为轮询方式，右边为中断方式

**四．硬件图**

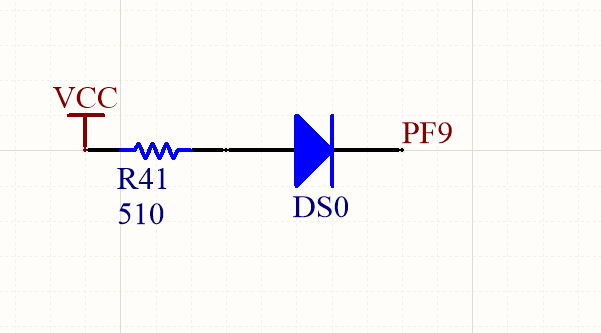
****

图2 – LED外设接线

**五．程序清单及注释**

**1.main.c**

#include "main.h"

#include "dma.h"

#include "tim.h"

#include "usart.h"

#include "gpio.h"

#include "led.h"

#include "stdbool.h"

#include "string.h"

#ifdef DMASolution

//BufferSize = 500 while using DMA method

#define bufferSize 500

//This array constantly equals to zero(for comparing)

const uint8\_t temp[bufferSize] = {0};

#endif

#ifdef ITSolution

//BufferSize = 1 while using IT method

#define bufferSize 1

//false => usart is not working. true => usart is in use

bool state = false;

#endif

//Buffer to save the data received

uint8\_t rxBuffer[bufferSize] = {0};

void SystemClock\_Config(void);

int arrayCompare(uint8\_t\* Array1, uint8\_t\* Array2, int size);

//DMA handle

extern DMA\_HandleTypeDef hdma\_usart1\_rx;

//counter for LED toggles

extern int counter;

int main(void)

{

//System Initializations

HAL\_Init();

SystemClock\_Config();

MX\_GPIO\_Init();

MX\_DMA\_Init();

MX\_USART1\_UART\_Init();

MX\_TIM7\_Init();

#ifdef DMASolution

//Using DMA to save the received data into memory

HAL\_UART\_Receive\_DMA(&huart1,rxBuffer,sizeof(rxBuffer));

#endif

#ifdef ITSolution

//Save the received data into memory

HAL\_UART\_Receive\_IT(&huart1,rxBuffer,sizeof(rxBuffer));

//Start timer7 IT

HAL\_TIM\_Base\_Start\_IT(&htim7);

#endif

//Loop

while (1)

{

#ifdef DMASolution

//Get DMA Transport Status

if(!arrayCompare((uint8\_t \*)temp,rxBuffer,bufferSize))

{

//Transmit the received data

HAL\_UART\_Transmit(&huart1,rxBuffer,sizeof(rxBuffer),0xffff);

//Clear the buffer

memcpy(rxBuffer,temp,bufferSize);

//Led toggle 5 times

for(int i=0;i<5;i++)

{

ledOn(LED);

HAL\_Delay(100);

ledOff(LED);

HAL\_Delay(100);

}

}

else

{

//Led works in regular state

ledOn(LED);

HAL\_Delay(1000);

ledOff(LED);

HAL\_Delay(1000);

}

#endif

}

}

/\*

\* @brief Initialize system clock

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE1);

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI;

RCC\_OscInitStruct.PLL.PLLM = 8;

RCC\_OscInitStruct.PLL.PLLN = 168;

RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;

RCC\_OscInitStruct.PLL.PLLQ = 4;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK|RCC\_CLOCKTYPE\_SYSCLK

|RCC\_CLOCKTYPE\_PCLK1|RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV4;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV2;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_5) != HAL\_OK)

{

Error\_Handler();

}

}

#ifdef DMASolution

/\*

\* @brief Compare two arrays

\* @retval equal => 1

\*/

int arrayCompare(uint8\_t\* array1, uint8\_t\* array2, int size){

for(int i = 0; i < size; i++){

if(array1[i] != array2[i])

return 0;

}

return 1;

}

#endif

#ifdef ITSolution

//Usart IT call back function

void HAL\_UART\_RxCpltCallback(UART\_HandleTypeDef \*huart){

if(huart->Instance == USART1)

{

//Transmit the received data

HAL\_UART\_Transmit(&huart1,rxBuffer,sizeof(rxBuffer),0);

//Restart the usart IT

HAL\_UART\_Receive\_IT(&huart1,rxBuffer,sizeof(rxBuffer));

//Change the USART working state into true

state = true;

//Clear the LED counter

counter = 0;

}

}

#endif

//Error handler function

void Error\_Handler(void)

{

ledOn(LED);

}

**2.main.h**

#ifndef \_\_MAIN\_H

#define \_\_MAIN\_H

#ifdef \_\_cplusplus

extern "C" {

#endif

#include "stm32f4xx\_hal.h"

//Define DMASolution to use polling, define ITSolution to use IT

//Here im using DMA method

#define DMASolution

//#define ITSolution

//LED @PF9

#define LED GPIOF,GPIO\_PIN\_9

void Error\_Handler(void);

#ifdef \_\_cplusplus

}

#endif

#endif

**3.** **stm32f4xx\_it.c (只给出了定时器7中断回调函数部分代码)**

void TIM7\_IRQHandler(void)

{

#ifdef ITSolution

HAL\_TIM\_IRQHandler(&htim7);

//Counter add one after each IT callback

counter ++;

//If state = 1, LED toggles rapidly, if state = 0, LED works regularly

if(state)

{

ledToggle(LED);

if(counter == 8){

//After toggle, change the Usart working state back to false

state = false;

counter = 0;

}

}

else if(counter == 10)

{

ledToggle(LED);

counter = 0;

}

#endif

}

**4.led.c**

#include "led.h"

//Common Cathode

#ifdef cc

void ledOn(GPIO\_TypeDef \*GPIOPort, uint32\_t GPIOPin){

HAL\_GPIO\_WritePin(GPIOPort,GPIOPin,on);

}

void ledOff(GPIO\_TypeDef \*GPIOPort, uint32\_t GPIOPin){

HAL\_GPIO\_WritePin(GPIOPort,GPIOPin,off);

}

#endif

//Common Anode

#ifdef ca

void ledOn(GPIO\_TypeDef \*GPIOPort, uint32\_t GPIOPin){

HAL\_GPIO\_WritePin(GPIOPort,GPIOPin,off);

}

void ledOff(GPIO\_TypeDef \*GPIOPort, uint32\_t GPIOPin){

HAL\_GPIO\_WritePin(GPIOPort,GPIOPin,on);

}

#endif

void ledToggle(GPIO\_TypeDef \*GPIOPort, uint32\_t GPIOPin){

HAL\_GPIO\_TogglePin(GPIOPort,GPIOPin);

}

**六．实验结果及分析**

**实验结果：**

在本次试验中，我分别使用轮询方式及中断方式完成了题目中要求的任务，其中字符串在轮询方式中可正确传输200个字符，失误率较低，而中断方式可在响应速度极快的情况下正确传输超过200个字符，且几乎没有失误率。

**实验分析：**

串口协议作为两机通讯的经典协议，在数据传输中十分常用，其中，轮询方式采用的是主函数在循环过程中反复向CPU确认是否收到数据的方法，若CPU已经收到数据，则将标志位改变，主函数在下次询问时即可得知已经收到数据，随后进入if分支处理接收的数据。中断方式则是在串口传输完毕后，外设给CPU提供一个中断信号，CPU收到中断信号即得知数据接收完毕，并将主程序挂起，进入中断回调函数并处理数据。

这两种方式在响应速度上有一定区别，中断方式的响应速度更快，但是轮询方式可以仅在主程序中完成任务，综合比较来看，中断方式有很大优势，因为其响应速度快，准确率高，不易出现数据丢失。在实际应用中，使用中断方式也是最常见的。

**七．对实验的建议**

我认为，对于此试验涉及到的内容来说，中断是最常用且最好用的方式，故我认为对中断方式应该给予更多的分值和重视。另外可以加入一些板间通讯的内容，有效利用LCD资源。