# 微处理器实验报告

|  |  |  |
| --- | --- | --- |
| **班级** | **姓名** | **学号** |
| **电气810** | **聂永欣** | **2186113564** |

## 实验一：微处理器应用编程及基本输入/输出实验

### 实验目的

1. 熟练掌握开发环境及CPU、外设接口、数据的观察、调试等开发方法。
2. 通过LED、按键，学习、掌握I/O的工作原理及编程、应用方法。
3. C语言、机器指令相结合，观察指令、寄存器,理解、领会微处理器系统工作。

### 目标要求

1. 填充学号至sBUF，通过8段LED，轮流显示自己学号各位；
2. 按下UP键（PA0），倒序（或暂停）显示自己学号；
3. 根据学号个位数，调整更新间隔[0.5s+学号个位\*0.1s]；
4. 对sBUF前10个数据累加、结果存至sBUF[15]；
5. 最后设断点，在UP键按下时，可暂停至断点。

### 实验现象

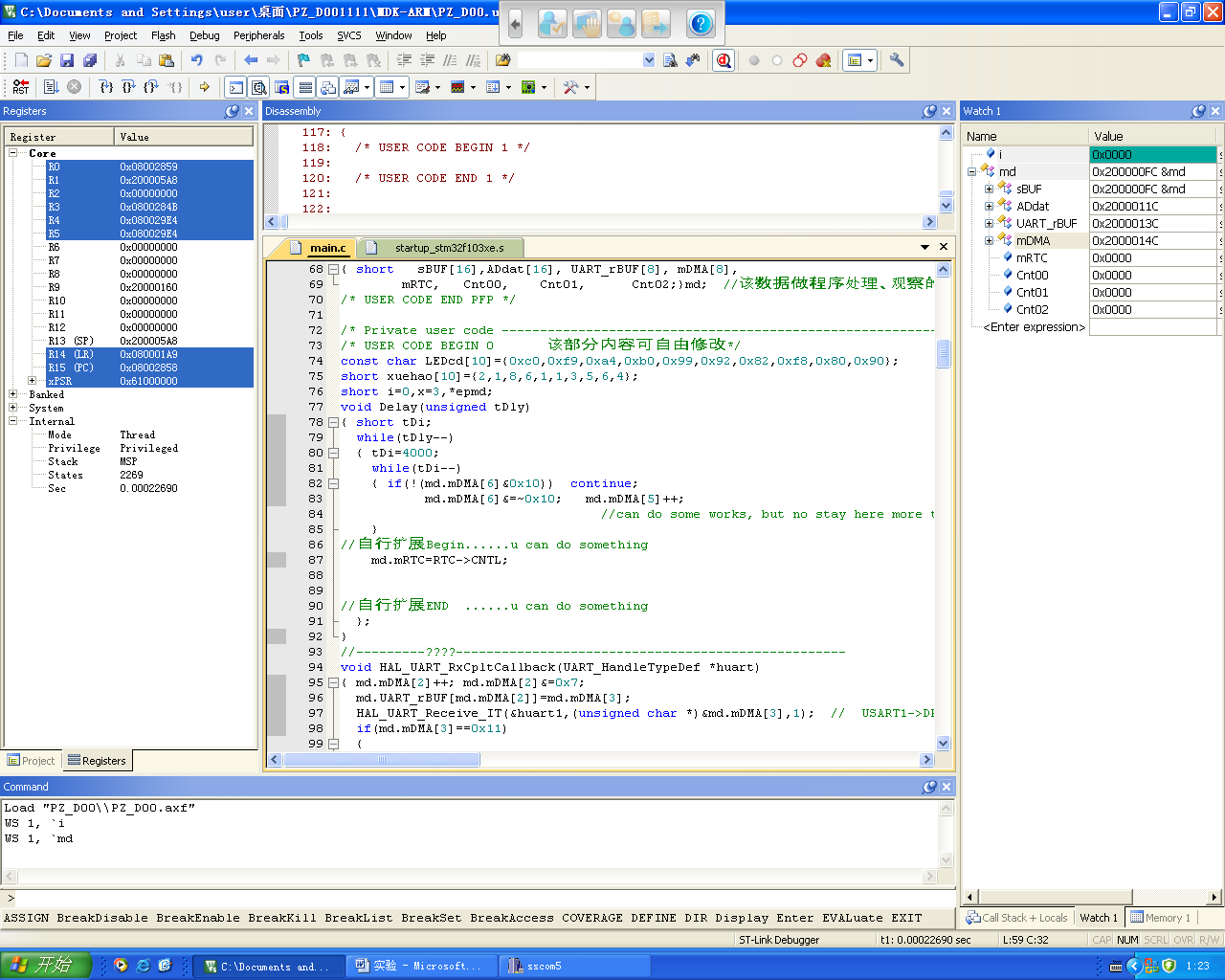


图1.1 程序界面截图

## 实验二 定时器及中断实验

### 实验要求

1. 了解STM32-F1系列处理器定时器及定时中断的工作原理及编程方法
2. 编写定时中断服务程序，完成周期性工作，并为其他模块提供时间控制。

### 实验内容

设定定时器周期，设计定时中断服务程序。以变量计数器观察其运行。

### 目标要求

不依靠软件延时Delay(unsigned tDly) ，在主程序实现1Hz及10Hz周期性简单处理任务（可通过计数变量如Cntx观察）。

### 实验程序

void TIM3\_IRQHandler(void)

{

/\* USER CODE BEGIN TIM3\_IRQn 0 ¸Ã²¿·ÖÄÚÈÝ¿É×ÔÓÉÐÞ¸Ä \*/

static unsigned short LEDpwm;

LEDpwm++; LEDpwm%=9800;

if(LEDpwm<4900) TIM3->CCR2= LEDpwm/2+88;

else TIM3->CCR2=4900- LEDpwm/2+88;

if(LEDpwm%10==1) { epmd[50]++; epmd[46]|=0x000f;}

if(LEDpwm%100==0) {

epmd[49]++;

epmd[46]|=0x00f0; } //Set a Click Flag

if(!(GPIOE->IDR&0x08)) //PE2(down Key) push

{ if(LEDpwm%8>3) GPIOB->BSRR|=1<<5; else GPIOB->BRR|=1<<5; } //Beep

/\* USER CODE END TIM3\_IRQn 0 \*/

HAL\_TIM\_IRQHandler(&htim3);

/\* USER CODE BEGIN TIM3\_IRQn 1 \*/

/\* USER CODE END TIM3\_IRQn 1 \*/

}

### 实验原理

**定时器中断：**

1. 打开相关外设的时钟。以定时器TIM3为例，由stm32的时钟树可以看到，TIM3时钟挂接在APB1上面，所以打开TIM3时钟时使用RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM3,ENABLE)，如果其中还使用到了其他外设，如GPIO等，再打开相关的外设时钟就可以了。

2. 清除中断挂起位。由于各种不可知的因素作用，在程序运行前要操作的定时器的中断挂起位有可能会被置位，这样就会导致在程序一开始就会进入定时器中断的中断服务程序。为了消除这种影响，我们在程序的一开始就将中断挂起位清除。在固件库中使用：

void TIM\_ClearITPendingBit(TIM\_TypeDef\*TIMx, u16 TIM\_IT)

来清除中断挂起位，该函数的具体使用参考固件库手册。

3. 定时器基本配置初始化。在这一步骤中主要确定定时器的预分频和设置自动重装载寄存器周期的值，并确定计数模式，这主要使用固件库中的 TIM\_TimeBaseInit()函数进行操作，该函数的原型为：

void TIM\_TimeBaseInit(TIM\_TypeDef\* TIMx,TIM\_TimeBaseInitTypeDef\* TIM\_TimeBaseInitStruct)

4. 使能定时器TIMx。这个简单，直接函数TIM\_Cmd()函数就可以了，比如使能定时器TIM3外设，则可用TIM\_Cmd(TIM3,ENABLE)。

5. 使能TIMx中断。调用函数即可。因为我们要使用 TIM3 的更新中断， 寄存器的相应位便可使能更新中断。 在库函数里面定时器中断使能是通过 TIM\_ITConfig 函数来实现的：

void TIM\_ITConfig(TIM\_TypeDef\* TIMx, uint16\_t TIM\_IT, FunctionalState NewState)；

第一个参数是选择定时器号，这个容易理解，取值为 TIM1~TIM17；第二个参数是用来指明我们使能的定时器中断的类型，定时器中断的；类型有很多种，包括更新中断 TIM\_IT\_Update，触发中断 TIM\_IT\_Trigger，以及输入捕获中断等等；第三个参数就很简单了， 就是失能还是使能。

如果要使能 TIM3 的更新中断，格式为：TIM\_ITConfig(TIM3,TIM\_IT\_Update,ENABLE );

6. 配置中断优先级，也就是配置嵌套向量终端控制器NVIC。进行本步骤首先需要配置优先级的分组，可以使用库函数NVIC\_PriorityGroupConfig()进行，分组的编号就是抢占优先级的位数，然后再配置NVIC初始化，使用函数NVIC\_Init()进行，这个函数的原型为void NVIC\_Init(NVIC\_InitTypeDef\* NVIC\_InitStruct)

7. 编写中断服务程序。首先要清除中断挂起位，接着再编写中断处理内容即可。

### 实验现象

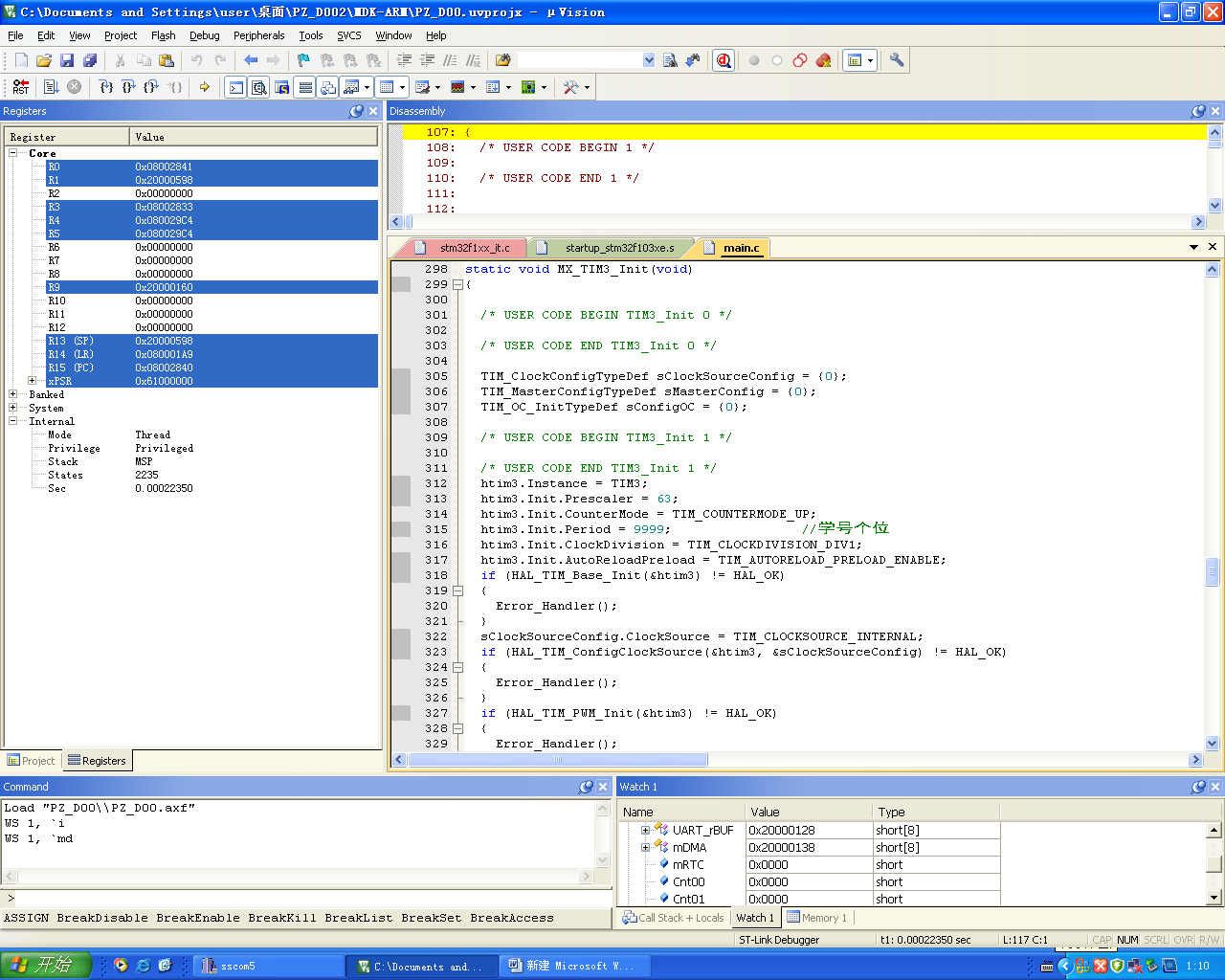


图2.1 程序界面截图1

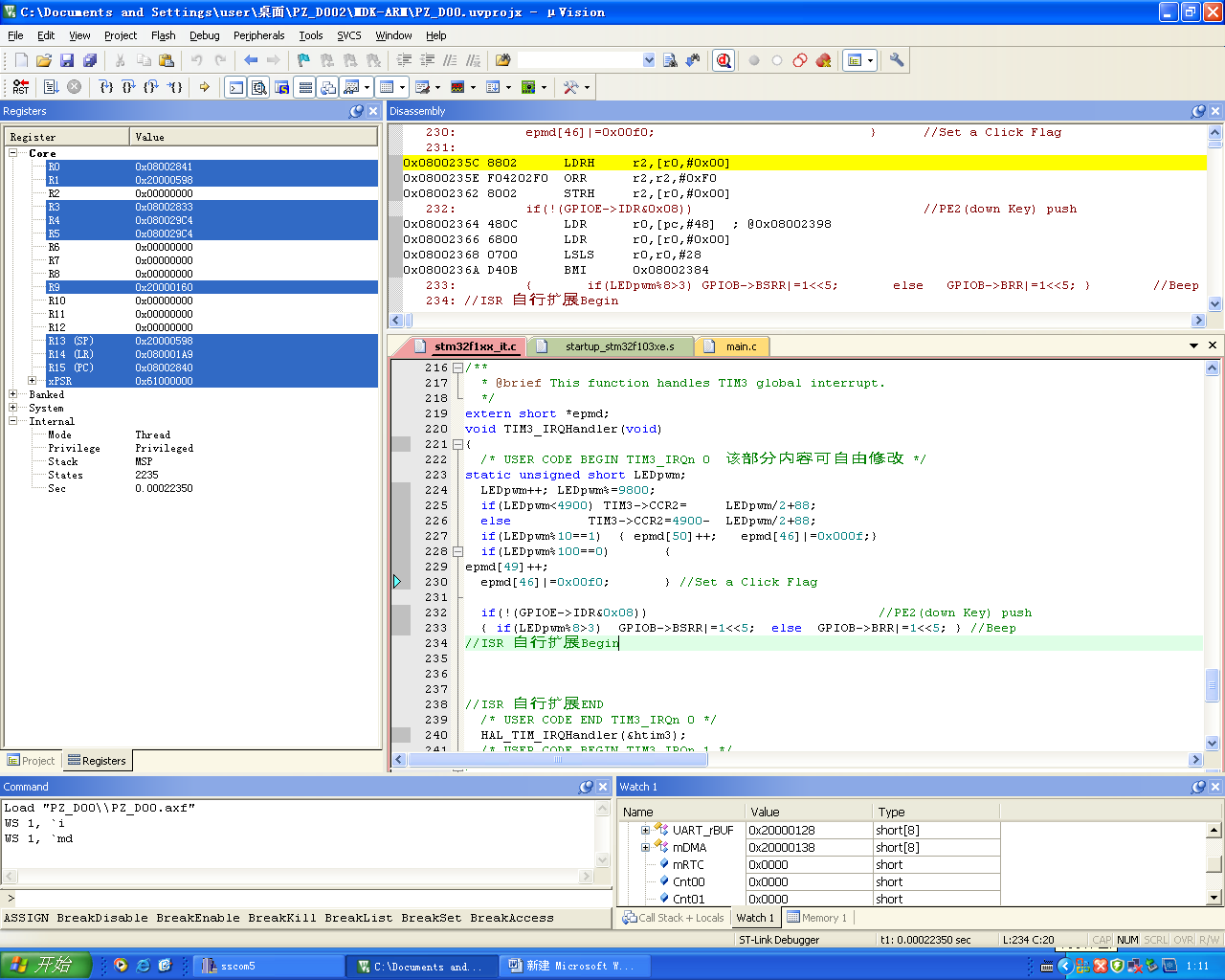


图2.2 程序界面截图2

## 实验三：ADC与DMA实验

### 实验目的

1. 了解STM32-F1系列处理器定时器+ADC+DMA工作原理及使用方法；
2. 对ADC数据进行简单的处理、计算。

### 实验内容

通过定时器3定时启动ADC，自DMA缓冲区读取ADC1.1结果，保存、计算。

### 目标要求

1. 设定恰当采样率（如2499+学号个位），以此采样率得到的ADC采样结果（PA1通道），陆续保存至循环缓冲区md.ADdat[0-7],并在定时中断处理程序计算8点数据平均值保存到ADdat[15]。
2. 调整电位器，观察实验结果。

### 实验程序

static void MX\_TIM3\_Init(void)

{

/\* USER CODE BEGIN TIM3\_Init 0 \*/

/\* USER CODE END TIM3\_Init 0 \*/

TIM\_ClockConfigTypeDef sClockSourceConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

TIM\_OC\_InitTypeDef sConfigOC = {0};

/\* USER CODE BEGIN TIM3\_Init 1 \*/

/\* USER CODE END TIM3\_Init 1 \*/

htim3.Instance = TIM3;

htim3.Init.Prescaler = 7;

htim3.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim3.Init.Period = 2399+4;

htim3.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim3.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_ENABLE;

if (HAL\_TIM\_Base\_Init(&htim3) != HAL\_OK)

{

Error\_Handler();

}

sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

if (HAL\_TIM\_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL\_OK)

{

Error\_Handler();

}

if (HAL\_TIM\_PWM\_Init(&htim3) != HAL\_OK)

{

Error\_Handler();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_OC1;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

if (HAL\_TIMEx\_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

sConfigOC.OCMode = TIM\_OCMODE\_PWM1;

sConfigOC.Pulse = 6;

sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;

sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;

if (HAL\_TIM\_PWM\_ConfigChannel(&htim3, &sConfigOC, TIM\_CHANNEL\_2) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM3\_Init 2 \*/

/\* USER CODE END TIM3\_Init 2 \*/

HAL\_TIM\_MspPostInit(&htim3);

}

### 实验原理

**ADC+DMA**

ADC（Analog-to-Digital Converter，模/数转换器）。是一种将模拟信号转换为数字信号进行处理的模块。在存储或传输时，模数转换器几乎必不可少。STM32在片上集成的ADC外设非常强大。

ADC配置：把ADC1的通道11使用的GPIO引脚PC1配置成模拟输入模式，在作为ADC的输入时，必须使用模拟输入。对于ADC通道，每个ADC通道对应一个GPIO引脚端口，GPIO的引脚在设为模拟输入模式后可用于模拟电压的输入。STM32F103VET6有三个ADC,这三个ADC公用16个外部通道。

DMA配置：使用DMA1的通道1，数据从ADC外设的数据寄存器（ADC1\_DR\_ Address）转移到内存（ADC\_ConvertedValue变量），内存外设地址都固定，每次传输的大小为半字（16位），使用DMA循环传输模式。

把ADC设置成连续转换模式，同时对应的DMA通道开启循环模式，这样ADC就一直在进行数据采集然后通过DMA把数据搬运至内存。但是这样做的话还得加一个定时中断，用来定时读取内存中的数据。使用ADC的定时器触发ADC转换的功能，然后使用DMA进行数据的搬运。这样只要设置好定时器的触发间隔，就能实现ADC定时采样转换的功能，然后可以在程序的死循环中一直检测DMA转换完成标志，然后进行数据的读取，或者使能DMA转换完成中断，这样每次转换完成就会产生中断

### 实验现象

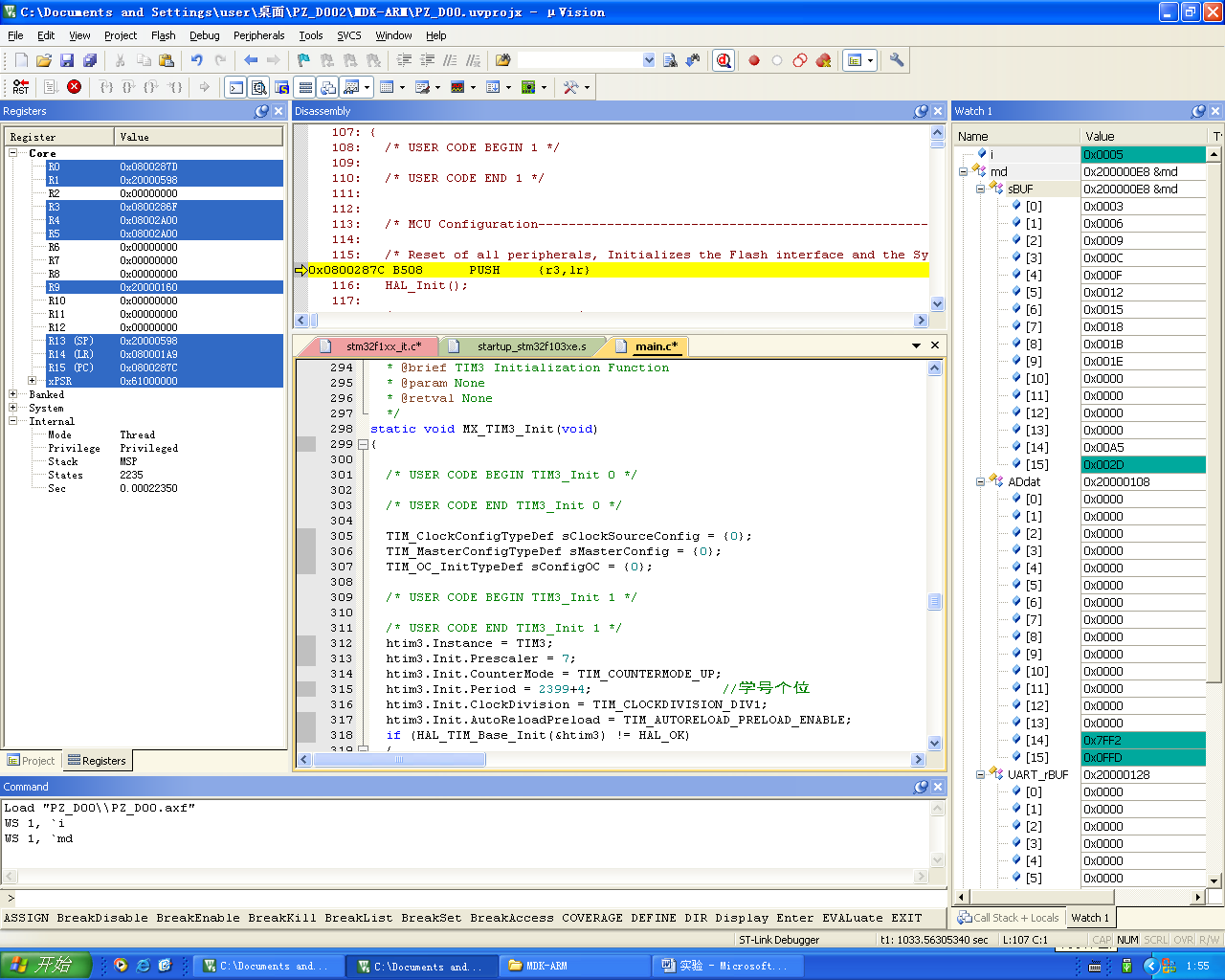


图3.1 程序界面截图

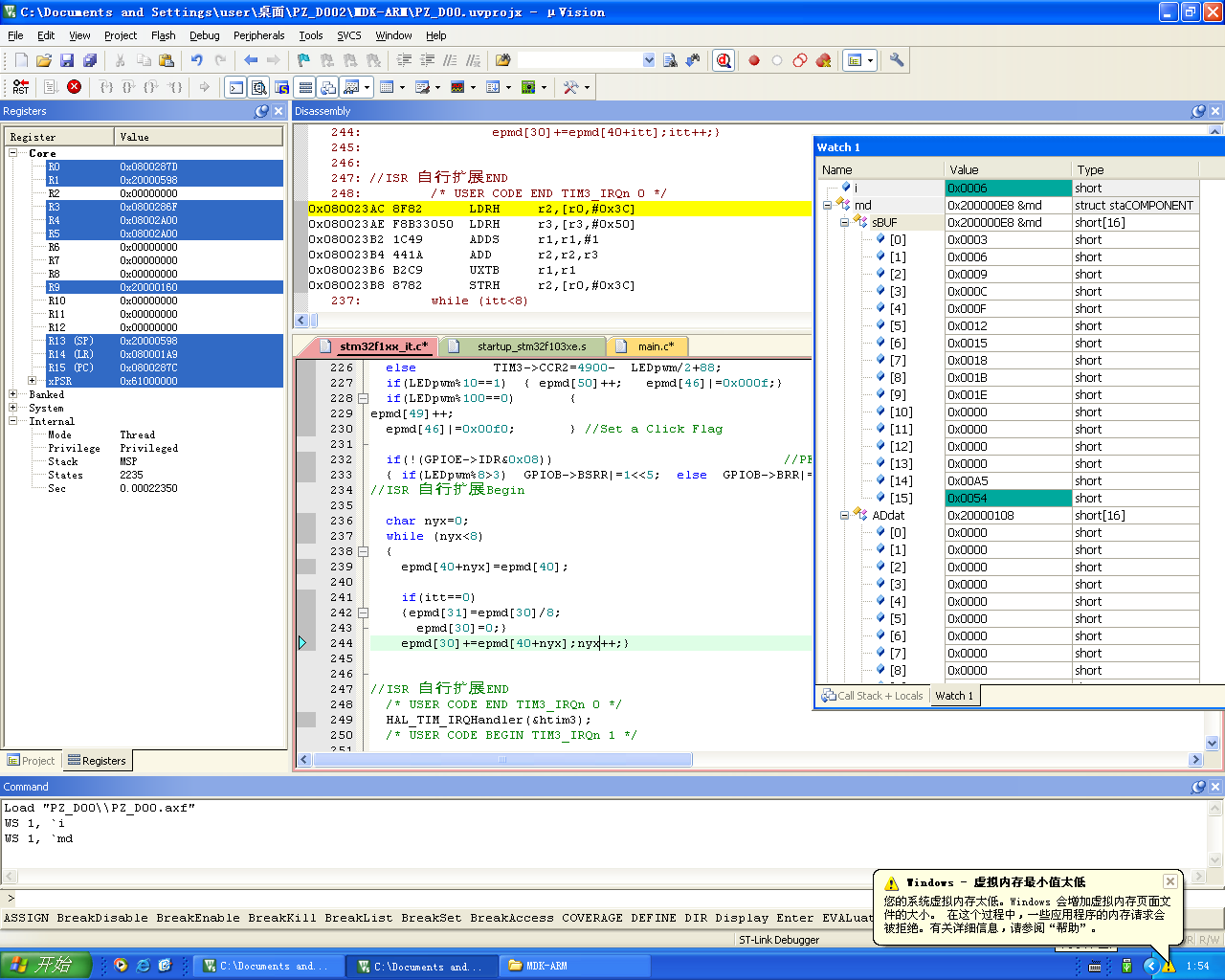


图3.2 程序界面截图

## 实验四：UART串行通讯实验

### 实验目的

1. 了解STM32-F1系列处理器UART的工作原理及编程方法
2. 对收、发内容进行简单处理。

### 实验内容

实现UART收、发功能。对收到的命令做处理，通过UART发送相应内容。

### 目标要求

根据接收的自行约定命令代码，通过UART分别发送学号或ADC结果（2字节）,在PC串口观察相应内容。

### 实验程序

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

{ md.mDMA[2]++; md.mDMA[2]&=0x7;

md.UART\_rBUF[md.mDMA[2]]=md.mDMA[3];

HAL\_UART\_Receive\_IT(&huart1,(unsigned char \*)&md.mDMA[3],1); // USART1->DR=md.mDMA[3]+1;

if(md.mDMA[3]==0x11)

{

for(int i=0;i<10;i++)

{

HAL\_UART\_Transmit(&huart1,(unsigned char \*)&md.sBUF[i],1,5000);

}

}

else

{

md.mDMA[4]=md.mDMA[3]+1;

while(HAL\_UART\_Transmit(&huart1,(unsigned char \*) & md.mDMA[4], 1, 5000) != HAL\_OK);

}

}

### 实验原理

STM32的串口通信接口有两种，分别是：UART（通用异步收发器）、USART（通用同步异步收发器）。而对于大容量STM32F10x系列芯片，分别有3个USART和2个UART。

STM32的UART特点：

1. 全双工异步通信；
2. 分数波特率发生器系统，提供精确的波特率。发送和接受共用的可编程波特率，最高可达4.5Mbits/s；
3. 可编程的数据字长度（8位或者9位）；
4. 可配置的停止位（支持1或者2位停止位）；
5. 可配置的使用DMA多缓冲器通信；
6. 单独的发送器和接收器使能位；
7. 检测标志：① 接受缓冲器 ②发送缓冲器空 ③传输结束标志；
8. 多个带标志的中断源，触发中断；
9. 其他：校验控制，四个错误检测标志。

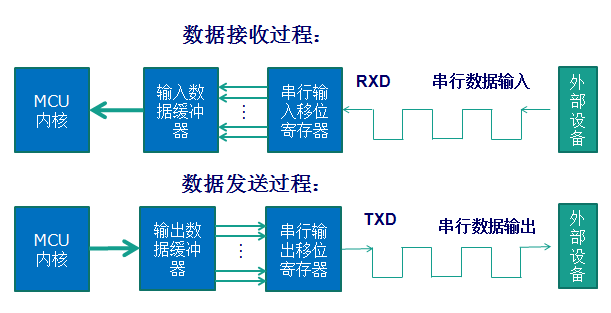


图4.1 串口通信过程

串口通讯的数据包由发送设备通过自身的TXD接口传输到接收设备的RXD接口，通讯双方的数据包格式要规约一致才能正常收发数据。STM32中串口异步通信需要定义的参数：起始位、数据位（8位或者9位）、奇偶校验位（第9位）、停止位（1,15,2位）、波特率设置。

UART串口通信的数据包以帧为单位，常用的帧结构为：1位起始位+8位数据位+1位奇偶校验位（可选）+1位停止位。

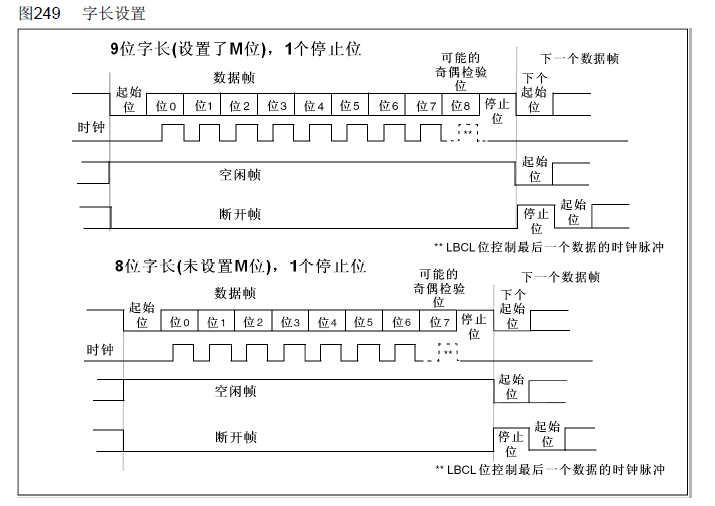


图4.2 常用的帧结构

### 实验现象

由程序可见，向串口发送“11”，返回学号，

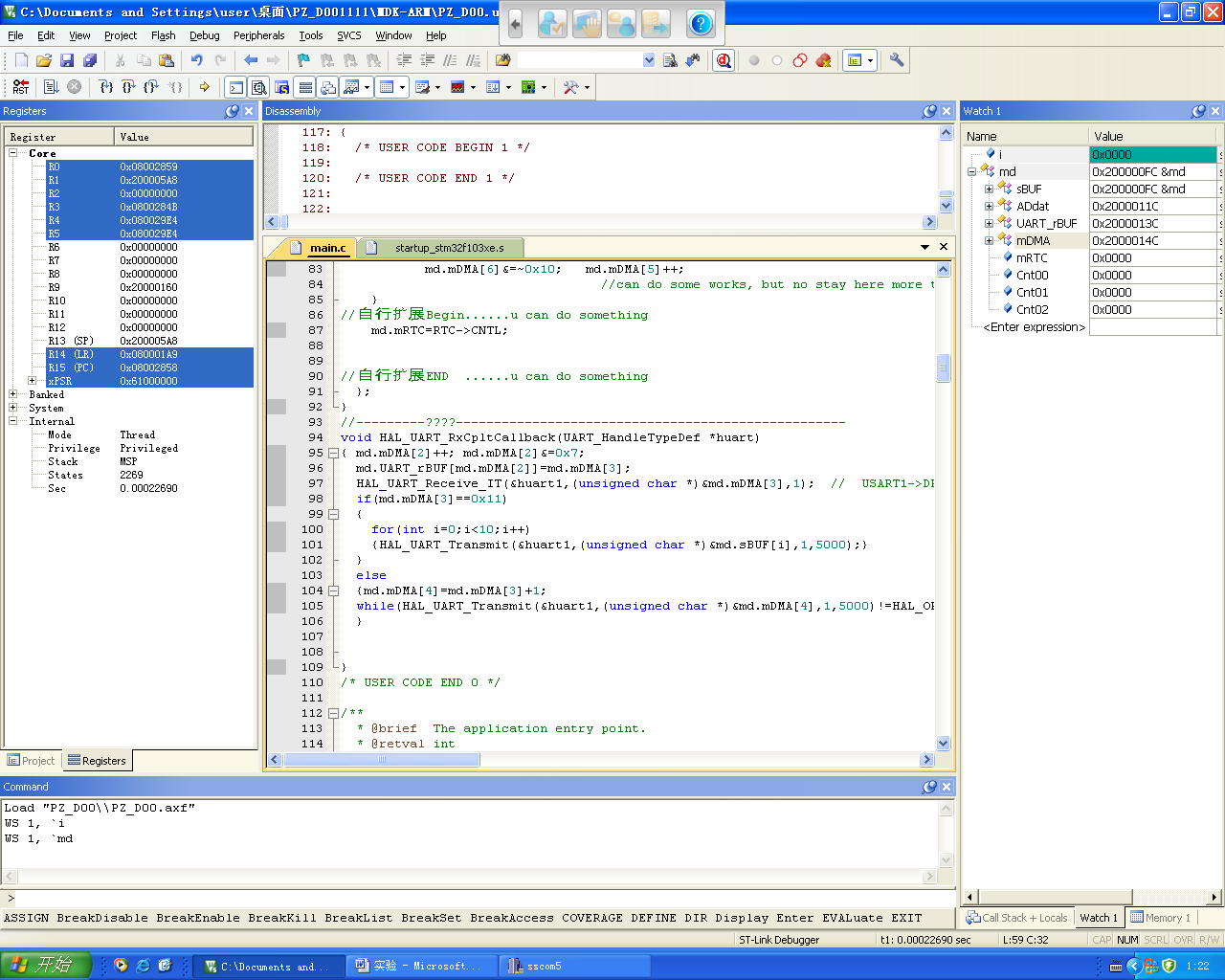


图4.3 程序界面截图

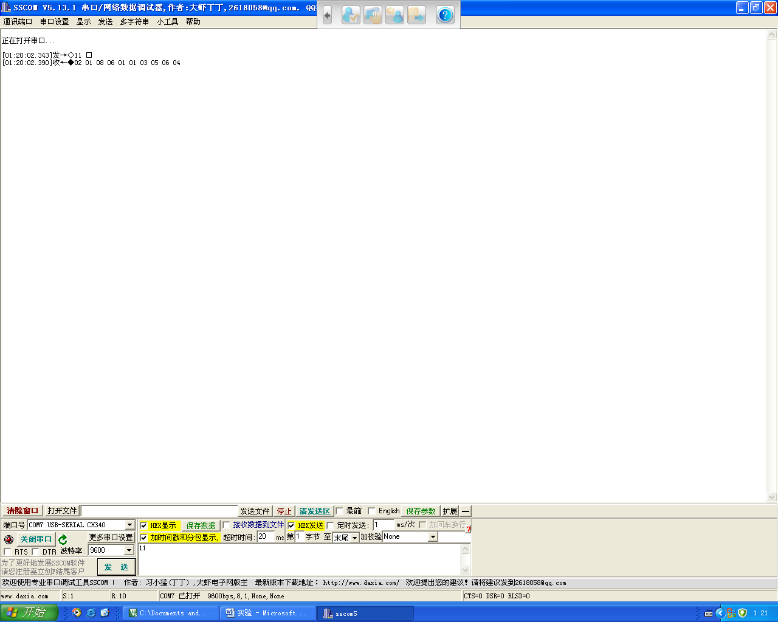


图4.4 SSCOM串口通信现象

由截图可知，向串口发送11，返回学号2186113564。由于每个数字必须要用两位数字储存，故返回结果为02 01 08 06 01 01 03 05 06 04，可得结果如上