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

1. 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。

2. 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。

3. 实验现象

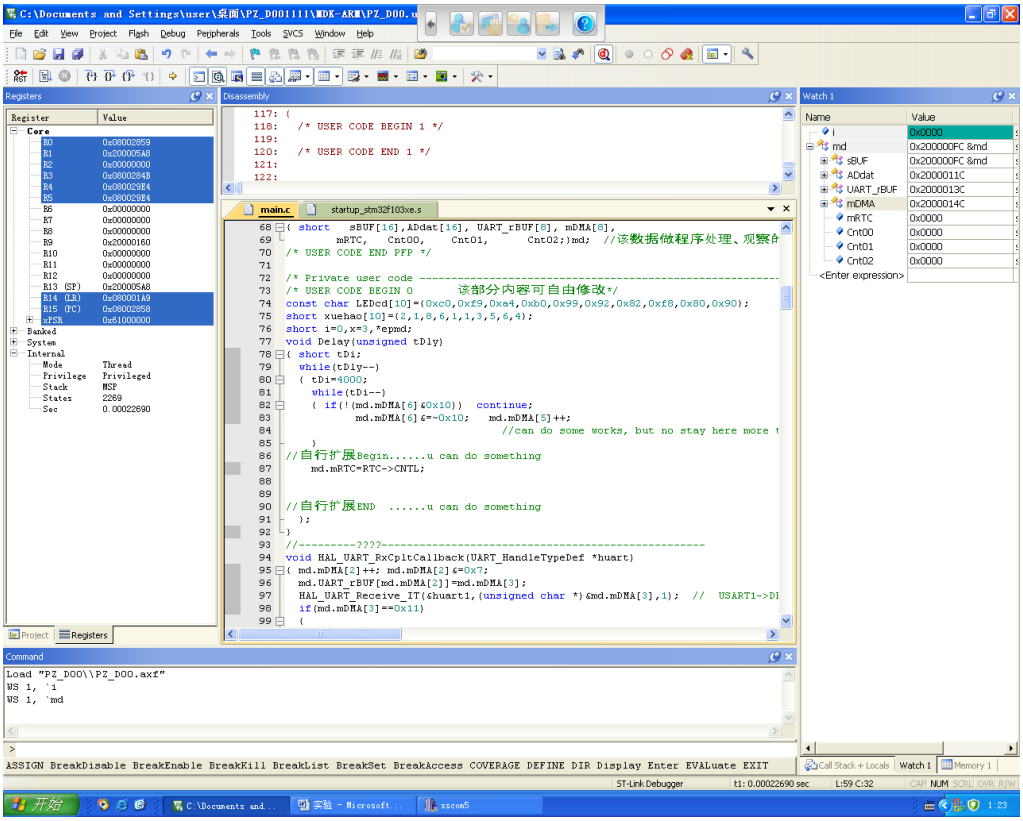

图 1.1 程序界面截图

实验二 定时器及中断实验

1. 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。

2. 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。

3. 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）。

4. 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 */
}
```

5. 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。

6. 实验现象

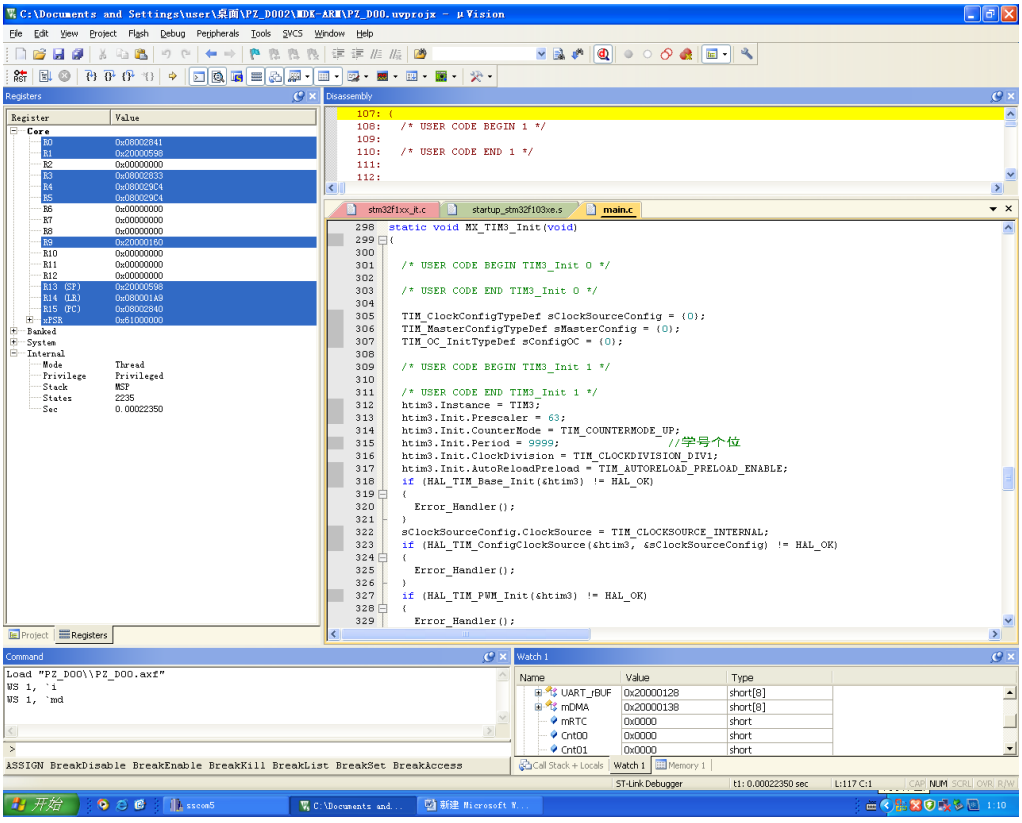

图 2.1 程序界面截图 1

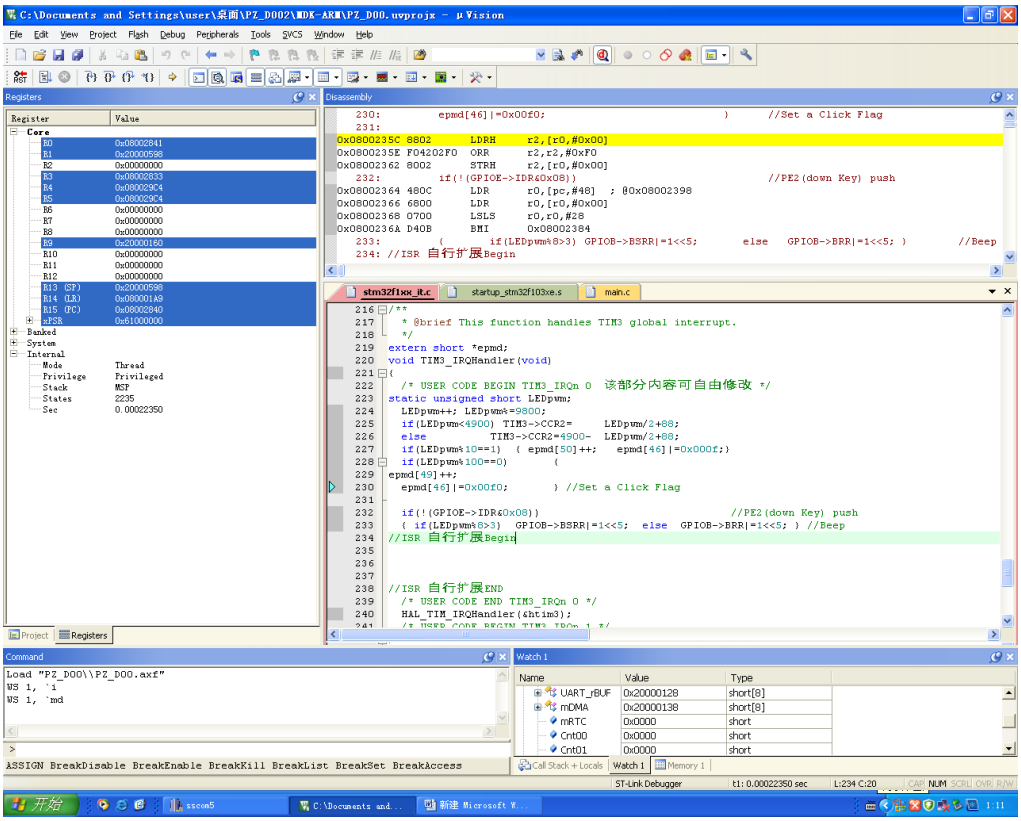
C:\Documents and Settings\user\桌面\PZ_D002\MDK-ARM\PZ_D00.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Core
R0 0x08002841
R1 0x20000538
R2 0x00000000
R3 0x08002833
R4 0x080029C4
R5 0x080029C4
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x20000160
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x20000598
R14 (LR) 0x080001A9
R15 (PC) 0x08002840
xPSR 0x61000000
Banked
System
Internal
Mode Thread
Privilege Privileged
Stack MSP
States 2235
Sec 0.00022350
Disassembly
230: epmd[46]|=0x00f0;) //Set a Click Flag
231:
0x0800235C 8802 LDRH r2,[r0,#0x00]
0x0800235E F04202F0 ORR r2,r2,#0xF0
0x08002362 8002 STRH r2,[r0,#0x00]
232: if(!(GPIOE->IDR&0x08)) //PE2(down Key) push
0x08002364 480C LDR r0,[pc,#48] ; @0x08002398
0x08002366 6800 LDR r0,[r0,#0x00]
0x08002368 0700 LSLS r0,r0,#28
0x0800236A D40B BMI 0x08002384
233: (if(LEDpwm%8>3) GPIOB->BSRR|=1<<5; else GPIOB->BRR|=1<<5;) //Beep
234: //ISR 自行扩展Begin
stm32f1xx_it.c
startup_stm32f103xe.s
main.c
216 /**
217 * @brief This function handles TIM3 global interrupt.
218 */
219 extern short *epmd;
220 void TIM3_IRQHandler(void)
221 {
222 /* USER CODE BEGIN TIM3_IRQn 0 该部分内容可自由修改 */
223 static unsigned short LEDpwm;
224 LEDpwm++; LEDpwm%=9800;
225 if(LEDpwm<4900) TIM3->CCR2= LEDpwm/2+88;
226 else TIM3->CCR2=4900- LEDpwm/2+88;
227 if(LEDpwm%10==1) { epmd[50]++; epmd[46]|=0x000f;}
228 if(LEDpwm%100==0) {
229 epmd[49]++;
230 epmd[46]|=0x00f0; } //Set a Click Flag
231
232 if(!(GPIOE->IDR&0x08)) //PE2(down Key) push
233 { if(LEDpwm%8>3) GPIOB->BSRR|=1<<5; else GPIOB->BRR|=1<<5; } //Beep
234 //ISR 自行扩展Begin
238 //ISR 自行扩展END
239 /* USER CODE END TIM3_IRQn 0 */
240 HAL_TIM_IRQHandler(&htim3);
Project Registers
Command
Load "PZ_D00\\PZ_D00.axf"
WS 1, `i
WS 1, `rmd
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess
Watch 1
Name Value Type
UART_rBUF 0x20000128 short[8]
mDMA 0x20000138 short[8]
mRTC 0x0000 short
Cnt00 0x0000 short
Cnt01 0x0000 short
Call Stack + Locals Watch 1 Memory 1
ST-Link Debugger t1: 0.00022350 sec L:234 C:20
开始
sscom5
新建 Microsoft W...
1:11

图 2.2 程序界面截图 2

实验三：ADC 与 DMA 实验

1. 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。

2. 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。

3. 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。

4. 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);

}
```

5. 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

6. 实验现象

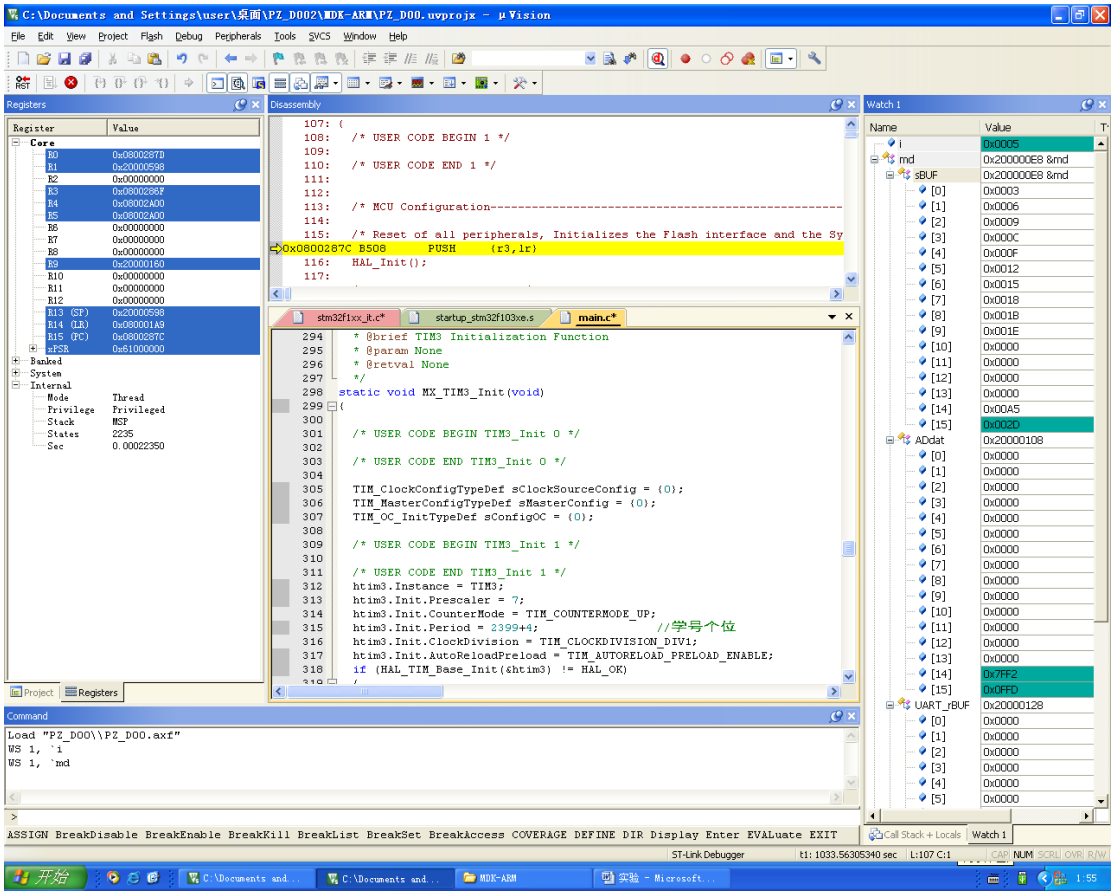

图 3.1 程序界面截图

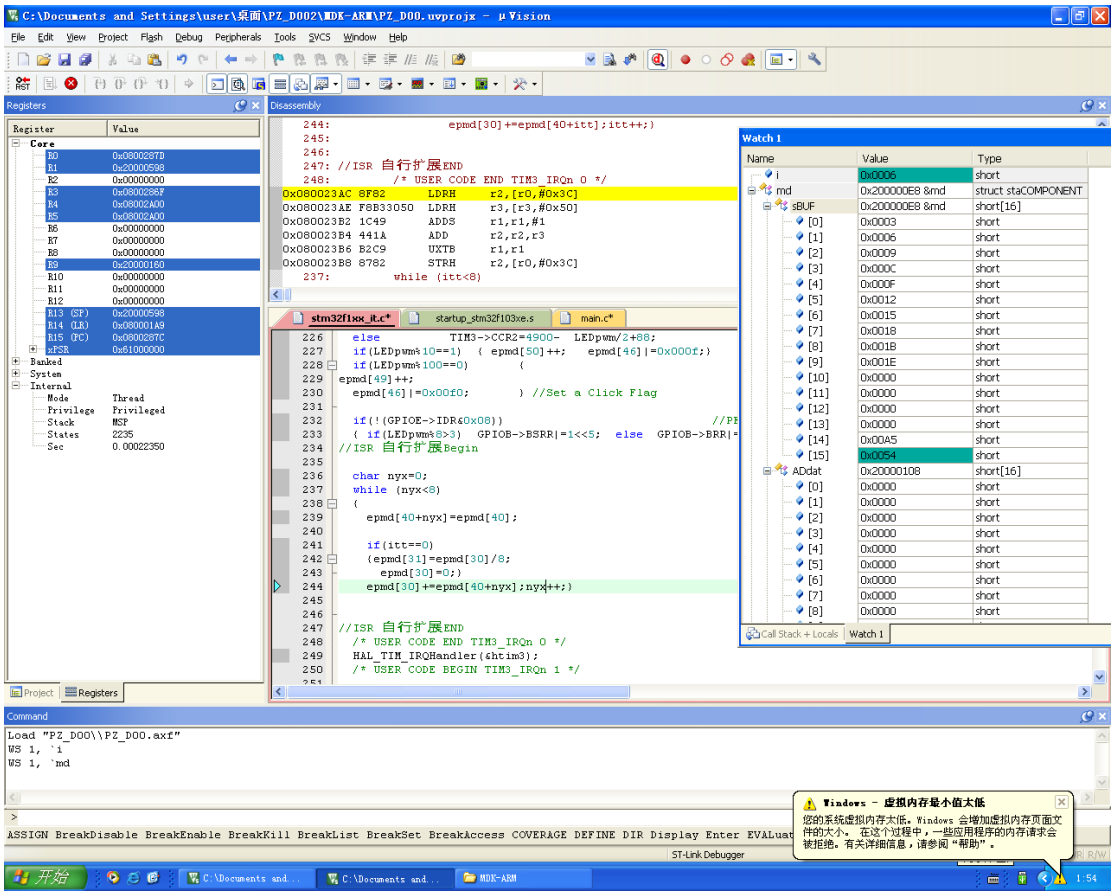
C:\Documents and Settings\user\桌面\PZ_D002\MDK-ARM\PZ_D00.uvprojx - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Core
R0 0x0800287D
R1 0x20000598
R2 0x00000000
R3 0x0800286F
R4 0x08002A00
R5 0x08002A00
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x20000160
R10 0x00000000
R11 0x00000000
R12 0x00000000
R13 (SP) 0x20000598
R14 (LR) 0x080001A9
R15 (PC) 0x0800287C
xPSR 0x61000000
Banked
System
Internal
Mode Thread
Privilege Privileged
Stack MSP
States 2235
Sec 0.00022350
Disassembly
244: epmd[30]+=epmd[40+itt];itt++;)
245:
246:
247: //ISR 自行扩展END
248: /* USER CODE END TIM3_IRQn 0 */
0x080023AC 8F82 LDRH r2,[r0,#0x3C]
0x080023AE F8B33050 LDRH r3,[r3,#0x50]
0x080023B2 1C49 ADDS r1,r1,#1
0x080023B4 441A ADD r2,r2,r3
0x080023B6 B2C9 UXTB r1,r1
0x080023B8 8782 STRH r2,[r0,#0x3C]
237: while (itt<8)
stm32f1xx_it.c* startup_stm32f103xe.s main.c*
226 else TIM3->CCR2=4900- LEDpwm/2+88;
227 if(LEDpwm%10==1) { epmd[50]++; epmd[46]|=0x000f;)
228 if(LEDpwm%100==0) {
229 epmd[49]++;
230 epmd[46]|=0x00f0;) //Set a Click Flag
231
232 if(!(GPIOE->IDR&0x08)) //PE
233 { if(LEDpwm%8>3) GPIOB->BSRR|=1<<5; else GPIOB->BRR|=
234 //ISR 自行扩展Begin
235
236 char nyx=0;
237 while (nyx<8)
238 {
239 epmd[40+nyx]=epmd[40];
240
241 if(itt==0)
242 {epmd[31]=epmd[30]/8;
243 epmd[30]=0;}
244 epmd[30]+=epmd[40+nyx];nyx++;)
245
246
247 //ISR 自行扩展END
248 /* USER CODE END TIM3_IRQn 0 */
249 HAL_TIM_IRQHandler(&htim3);
250 /* USER CODE BEGIN TIM3_IRQn 1 */
Watch 1
Name Value Type
i 0x0006 short
md 0x200000E8 &md struct staCOMPONENT
sBUF 0x200000E8 &md short[16]
[0] 0x0003 short
[1] 0x0006 short
[2] 0x0009 short
[3] 0x000C short
[4] 0x000F short
[5] 0x0012 short
[6] 0x0015 short
[7] 0x0018 short
[8] 0x001B short
[9] 0x001E short
[10] 0x0000 short
[11] 0x0000 short
[12] 0x0000 short
[13] 0x0000 short
[14] 0x00A5 short
[15] 0x0054 short
ADdat 0x20000108 short[16]
[0] 0x0000 short
[1] 0x0000 short
[2] 0x0000 short
[3] 0x0000 short
[4] 0x0000 short
[5] 0x0000 short
[6] 0x0000 short
[7] 0x0000 short
[8] 0x0000 short
Call Stack + Locals Watch 1
Project Registers
Command
Load "PZ_D00\\PZ_D00.axf"
WS 1, `i
WS 1, `md
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE DEFINE DIR Display Enter EVALuat
ST-Link Debugger
Windows - 虚拟内存最小值太低
您的系统虚拟内存太低。Windows 会增加虚拟内存页面文件的大小。 在这个过程中，一些应用程序的内存请求会被拒绝。有关详细信息，请参阅“帮助”。
开始
C:\Documents and...
C:\Documents and...
MDK-ARM
1:54

图 3.2 程序界面截图

实验四：UART 串行通讯实验

1. 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。

2. 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。

3. 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。

4. 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);
     }
}
```

5. 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。

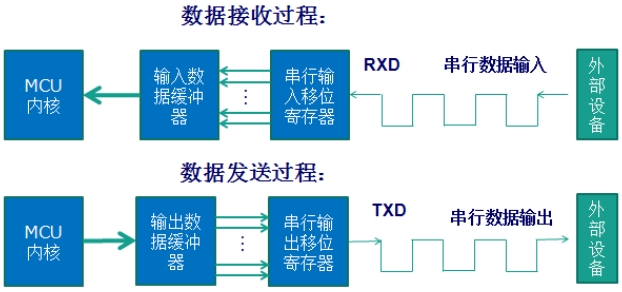

图 4.1 串口通信过程

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249 字长设置

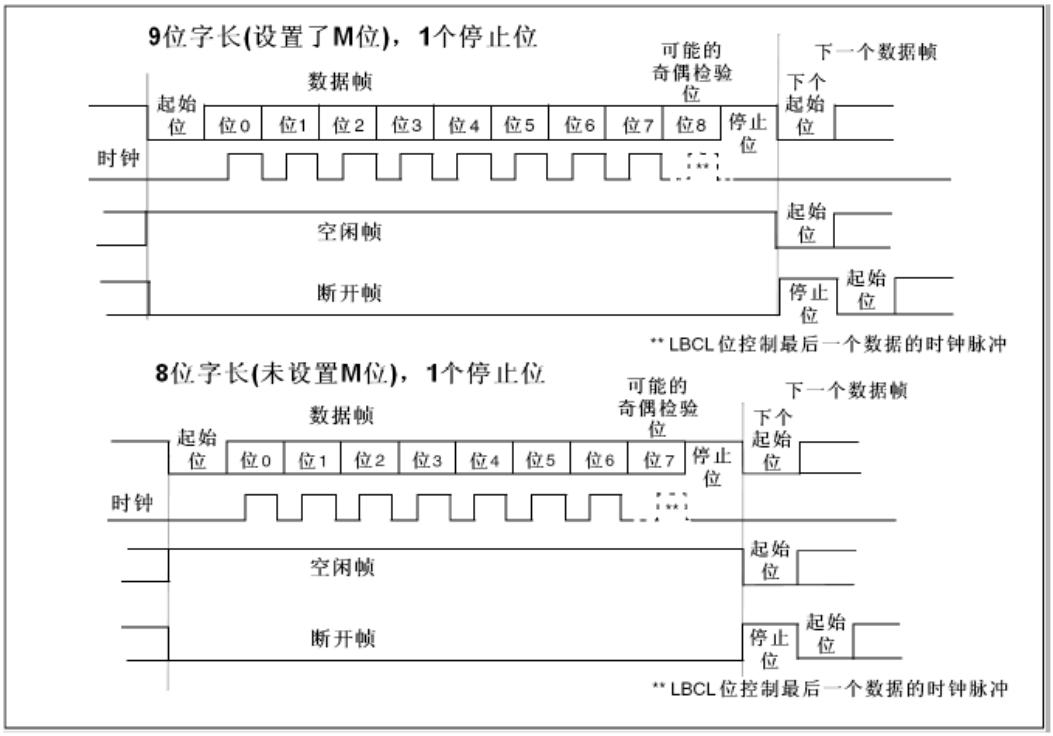

图 4.2 常用的帧结构

6. 实验现象

由程序可见，向串口发送“11”，返回学号，

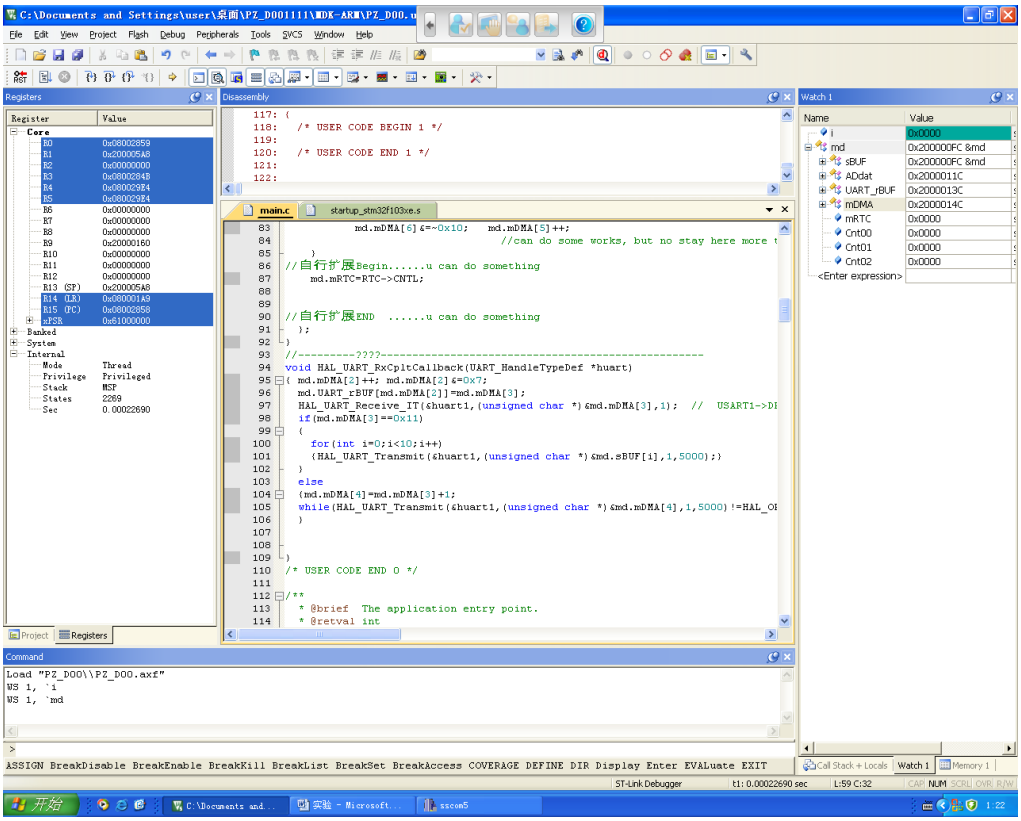

图 4.3 程序界面截图

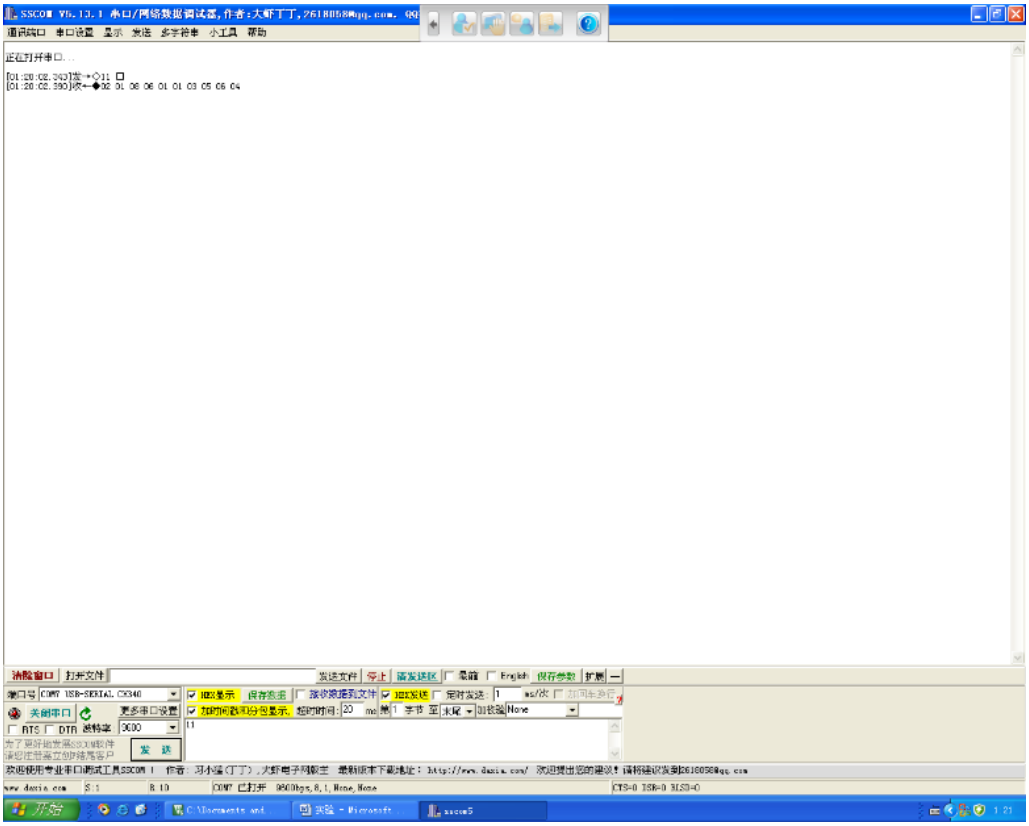

图 4.4 SSCOM 串口通信现象

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