第3章　程序测试

3.1　简单闪烁LED程序测试

如果要实现汇编版本的LED程序的正常运行，必须先保证flash内容的读取，AXI与gpio模块的正常通信。汇编版的测试程序如下，参考其中注释。

```
    .text
    .align 2
    .globl  main
    .set nomips16
    .set nomicromips
main:
    nop
    lui $1,0xd000
    ori $1,$1,0x0004
    lui $2,0
    sw $2,0x0(($1)      #gpio使能为输出

label1:
    lui $1,0xd000
    ori $1,0x0000
    lui $2,0xffff
    ori $2,0xffff       #gpio全部输出1
    sw  $2,0x0($1)
    lui $3,0x01
label2:                 #相当于延迟的作用
    addiu $3,$3,-1
    move $5,$3
    bne $5,$0,label2
    nop
    nop

    lui $1,0xd000
    ori $1,0x0000
    lui $2,0xff3f
    ori $2,0xffff       #gpio连接LED的两个引脚为输出0
    sw  $2,0x0($1)
    lui $3,0x01
label3:                 #相当于延迟的作用
```

```
addiu $3,$3,-1
move $5,$3
bne $5,$0,label3
nop
nop
j label1
nop
nop
addiu $3,$3,-1
move $5,$3
bne $5,$0,label3
nop
nop
j label1
nop
nop
```

如果能观察到两个一闪一闪的 LED 灯，则说明成功了。但需要注意到，本汇编程序都是在寄存器中操作的，并没有涉及存储空间上的读取。这是因为在本文的设计中 flash 被定位为 ROM，不可写。涉及内存读写需要保证另外一个模块 RAM 与 AXI 的通信。

C 语言版本的 LED 程序测试，目的是检查最基本的启动文件编写和链接脚本是否正确。因为 C 语言程序涉及变量、堆栈，以及内存的读写操作，这就要求链接脚本对只读 flash、可读可写的 RAM 划分要明确，启动文件关于数据搬运的部分要正确。目的只有一个：编译器要清晰地知道变量放哪，堆栈应该在哪，如何进行压栈。在这个过程中，分析反汇编的 asm 文件就十分重要了。C 语言的测试代码如下。

```
int main(void)
{
    unsigned int * gpio_tri_addr  = GPIO_TRI_ADDR;
    unsigned int * gpio_data_addr = GPIO_DATA_ADDR;
    *gpio_tri_addr = 0x0;
    while(1){
        *gpio_data_addr = 0xffffffff;
        udelay(200000);
        *gpio_data_addr = 0xff3fffff;
        udelay(200000);
    }
    return 0;
}
```

3.2 简单时钟程序测试

C 语言版本的时钟程序测试的目的是验证中断是否可用，验证链接脚本在固定位置存放例外代码是否正确，验证例外函数的汇编代码编写是否合理，验证 C 语言中嵌入汇

编代码是否正确。在这个过程中，一定要对寄存器的使用做到心中有数，明白进入例外函数后，还能跳回进入异常之前的状态吗？（这是编者的经验之谈，只有清楚寄存器的使用情况才不容易放错值。）在这个过程中需要关注的代码如下。

```
/* mytest.ld */
. = 0x00000380;
        __isr_vector = .;

/* startuo_mytest.s */
addiu   $29,$29,-16
    sw    $31,12($29)
    sw    $2,8($29)
    sw    $3,4($29)
    jal   timer_interrupt
    nop
    lw    $31,12($29)
    lw    $2,8($29)
    lw    $3,4($29)
    addiu $29,$29,16
    eret

/* system.c */
asm volatile(
        "mfc0 $26,$11 \n\t"
        "la $27,0x30D40 \n\t"
        "addu $26,$26,$27 \n\t"
        "mtc0 $26,$11 \n\t"
);
```

3.3 仿真的一点小技巧

在仿真的过程中需要输入一些指令，如果是自己对照着 MIPS 指令集仿照着计算一条条指令实在是太麻烦了。办法一：可以编写一些汇编指令，再反汇编一下生成 *.asm 文件。可以看到 asm 文件的内容如图 3.1 所示。

```
Disassembly of section .text:

00000000 <_start>:
   0:   3403eeff    li   v1,0xeeff
   4:   a0030003    sb   v1,3(zero)
   8:   00031a02    srl  v1,v1,0x8
   c:   a0030002    sb   v1,2(zero)
  10:   3403ccdd    li   v1,0xccdd
  14:   a0030001    sb   v1,1(zero)
  18:   00031a02    srl  v1,v1,0x8
  1c:   a0030000    sb   v1,0(zero)
  20:   80010003    lb   at,3(zero)
  24:   90010002    lbu  at,2(zero)
  28:   00000000    nop
```

圈出来的就是指令对应的数据

图 3.1　asm 文件内容示例

可能这还是太麻烦了,办法二:使用《自己动手写 CPU》书籍资料包中的工具。也许还有更好的方法,读者可自行尝试。参考具体注释。

Cmd 中输入命令:

```
./Bin2Mem.exe -f led_asm.bin -o led_asm.data

/* Verilog 中读入指令文件,仿真 */
//reg[31:0] memory[0:2000];
//initial
//begin
//    $readmemh("./led_asm.data",memory);
//end
//也许联想到了将指令数据初始化在 reg 中(ram 中),flash 都不需要了,每次改 FPGA
//程序就行了,这个可以尝试一下,未考虑过,但应该不是仿真这种初始化方法
```