计算机系统实验报告

实验人: 李京桥、王一凡、傅祺睿

学号: 201840150、201840349、201840286

实验时间: 2022/5/27 ~ 2022/6/9

目录

1 实验目的

本实验的目标是在 DE10-Standard 开发板的 FPGA 上实现一个简单的计算机系统，能够运行简单的指令，并处理一定量的输入输出。在所有功能开发完毕后，希望能够完成基本的 terminal 功能，即键盘输入命令，并在显示器上输出结果。

需要实现的功能包括：

1. 接受键盘输入并回显在屏幕上
2. 支持换行、删除、滚屏等操作
3. 命令分析：根据键盘输入命令执行对应的子程序，并将执行结果输出到屏幕上。
    - 打入 hello ，显示 Hello World!
    - 打入 time ，显示时间
    - 打入 fib n ，计算斐波那契额数列并显示结果
    - 打入未知命令，输出 Unknown Command

在此之上，在本实验中还实现了下列拓展功能：

1. 增加外设种类，支持对板载 LED 、七段数码管显示的控制
2. 实现了简单的计算器，输入表达式可以计算结果
3. 实现了开机自启动

2 实验环境与器材

- DE10 Standard 开发平台
- Quartus 17.1 Lite
- Modelsim 仿真工具
- 带有 PS2 接口的键盘
- 带有 VGA 接口的屏幕显示器

3 实验原理

硬件部分主要是利用上两个实验中已经实现的 CPU ，将其实例化并与自己实现的指令、数据存储器相连，从而得到一个可以正确工作的 CPU 。

软硬交互的关键在于对屏幕、开发板信号的输入输出如何转化为指令，这里主要依靠外设的内存映射。具体来说就是将不同的外设映射到不同的内存区域，指令中访存地址为 32 位，我们将高 12 位也就是 16 进制下的高三位作为区分不同外设内存空间的标志。每个外设“忠实”地将自己输入的内容写入这片对应的内存或者输出这片内存的内容，这样我们对外设的控制就变成了对这篇内存的控制，与外设的交互就变成了调用更改、查询这些内存的函数。

有了与外设的交互（尤其是输入输出函数），软件的实现便与日常开发软件差异不大。可以用软件控制的外设有：主存、屏幕、LED 灯、七段数码管、板上自己实现的计时器。通过对这些外设控制，我们就可以实现所有的基础功能即扩展功能。

4 实验步骤

4.1 确定整体架构

上面已经说过，在 CPU 已经有比较完善的版本可以直接使用时，内存映射是整体设计中比较关键的部分。我们首先确认了内存映射的情况和各个外设的使用方式，图纸如下：

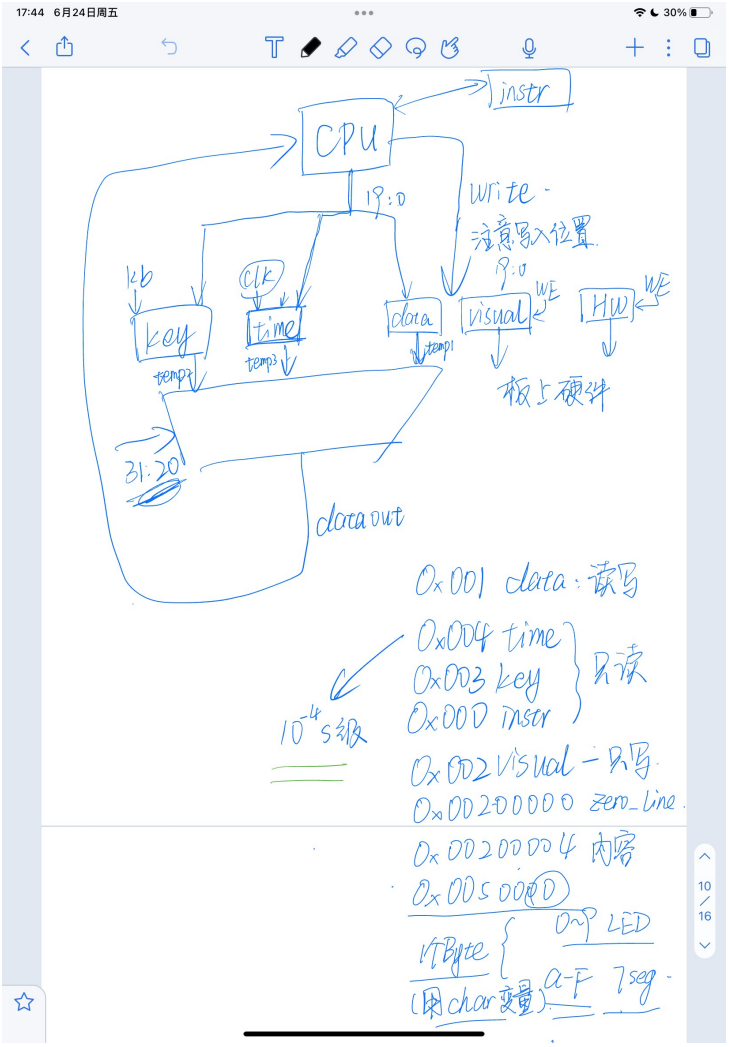

图 1: 手画图纸，略显草率

可以看到，我们为每个需要控制的外设都分配了一定大小的内存空间，16 进制下 8 位地址中高 3 位既是用来区分不同的外设。同时，我们确认了每片内存空间由谁来读写，比如显存空间就由 CPU 写而显示器读————每片空间的读（写）者都必须唯一。

4.2 硬件的实现

之前的 CPU 是只读指令存储器和读写数据存储器，可是现在我们要控制的内存空间变得复杂，因此需要修改 CPu 的一些实现。

首先是写数据时，每块内存模块都会判断写入的地址高 12 位是否与自己的标志相同，确认相同即是要往自己内部写入时，才会写入数据。也就是说逻辑上各个内存是按照标志连在一起的，但是物理上它们却是分开的，这种差异使得各存储模块在写入时需要判定一下是否在向自己写入。以

数据存储器为例，其使能端不再是 enable ，而是需要地址高 12 位判断一下是否正在向其中写入；同时真正的模块内地址是“虚拟地址”的低 20 位。其实例化接口如图：

```
135  dmem data_mem(
136      .addr (dmemaddr[19:0]),
137      .dataout (memtemp1),
138      .datain (dmemdatain),
139      .rdclk (dmemrdclk),
140      .wrclk (dmemwrclk),
141      .memop (MemOp),
142      .we (dmemaddr[31:20]==12'h001 && enable)
143
144  );
145
```

图 2: 数据存储器的实现

其次是读数据时，从设计图种看出，从数据存储器读出 temp1 ，从键盘模块读出 temp2 ，从计时器模块读出 temp3 ，CPU 要根据读地址的高 12 位选择这三个数据，这个多路选择器的实现如下：

```
 96  always @ (dmemaddr) begin
 97      case (dmemaddr[31:20])
 98          12'h001 : dmemdataout = memtemp1;
 99          12'h003 : dmemdataout = memtemp2;
100          12'h004 : dmemdataout = memtemp3;
101          default : dmemdataout = 12'b0;
102      endcase
103  end
104
```

图 3: CPU 读数多路选择器

可以看到，每次读数时 CPU 读到的 dmemdataout 都是该选择器的输出结果，这也在设计图中有所反映。

综上，在利用之前已有的 CPU 时，需要类似上面例子的方式更改各模块的实现，从而能实现本实验中的各种外设操作。

4.3 软件与硬件的连接

软件与硬件的连接其实就是按照约定的地址标示实现若干从内存读写数据的函数，这些函数本质上会由 CPU 的 sw 或 ld 指令完成，从而在软硬件之间建立起关系。

```
#define VGA_ZERO 0x00200000
#define VGA_START 0x00200004
#define VGA_LINE_O 0x00210000
#define VGA_MAXLINE 30
#define VGA_MASK 0x003f
#define VGA_MAXCOL 70

#define KB_DATA 0X00300000

#define TIME 0x00400000
#define T_01s 1000
#define T_08s 8000

#define LEDR_START 0x00500000

#define SEG_START 0x0050000a
```

图 4: 硬件地址划分在软件中的宏定义

硬件对软件提供了许多接口，这些接口的实现对于程序员来说是不可见的，他们只需要调用这些接口控制外设即可。接口如下：

1、putch(char ch)、putstr(char* str): 在屏幕上显示/退格/换行。实现原理: 以输出正常字符为例，在屏幕上输出字符的硬件实质就是把一个字节的数据放到显存内的一个地址。

```
vga_start[(((vga_line + temp_zero)%VGA_MAXLINE)<<7)+vga_ch] = ch;
vga_ch++;
if(vga_ch>=VGA_MAXCOL){
    vga_ch = 0;
    if(vga_line == VGA_MAXLINE-1){
        for(int i=0; i<VGA_MAXCOL; i++){
            vga_start[(temp_zero<<7) + i] = 0;
        }
        temp_zero = (temp_zero + 1)%30;
        *zero_line = temp_zero;

    }

    else vga_line++ ;
}


vga_start[(((vga_line + temp_zero)%VGA_MAXLINE)<<7)+vga_ch] = 0x5F;
```

图 5: 普通字符的输出

2、getstr(char *str): 输入字符串实现原理：对键盘按下/松开进行忙等待检查，在合适时间将按键的 ascii 码传入。last_ascii 记录上次时钟周期的 ascii 码，cur_ascii 记录当前时钟周期的 ascii 码，press_begin 记录按下的时间。

last=0 cur≠0: 刚刚按下

last≠0 cur≠0: 按住不放（按住 0.8s 后每 0.1s 输入一次）

last≠0 cur=0: 松手

last=0 cur=0: 没按

```
if(cur_ascii!=0)
{
    if(last_ascii==0)//刚刚按下
    {
        press_begin=gettime();
        if(cur_ascii==13)//enter 检测到回车
        {
            putch(10);//显存里存换行符10
            cmd[i]='\0';//字符串中存结束符
            int j=0;
            while(j<2500)
            {
                j++;
            }
            return;
        }
```

图 6: 以按下回车为例

3、gettime(): 获取系统时间 (单位：10^{-4}s)，settime(int time_set): 系统时间设置
实现原理：从硬件中指定位置直接读出/存入，代码如下：

```c
int gettime()//单位：10^-4s
{
    int* t=(int*)TIME;
    return *t;
}
```

(a) 获取系统时间

```c
void settime(int time_set)//将内部时间设置为time_set (10^-4s)
{
    int* t=(int*)TIME;
    *t=time_set;
}
```

(b) 设置系统时间

图 7: 时间控制代码

4、open/close_LEDR(int n)// 亮/灭第 n 个灯
代码如下：

```c
void open_LEDR(int n)//亮第n个灯
{
    if(n>6)
    {
        return;
    }
    char* LEDR=(char*)(LEDR_START+n);
    *LEDR=(char)1;
}
```

(a) 亮灯代码

```c
void close_LEDR(int n)//灭第n个灯
{
    if(n>6)
    {
        return;
    }
    char* LEDR=(char*)(LEDR_START+n);
    *LEDR=(char)0;
}
```

(b) 灭灯代码

图 8: 控制 LED 灯的代码

4.4 软件的实现

有了上面对外设的控制，软件的开发与普通的 C 语言开发就十分接近了。实现一个终端并且每次对输入的指令进行处理即可。在终端中，一共实现了八条终端指令，依次如下：

1、help 用来打印各种功能
只需要将能够实现的指令打印至屏幕上即可

```c
void printHelp()
{
    putstr("hello\n");
    putstr("fib n\n");
    putstr("time\n");
    putstr("set time\n");
    putstr("cal\n");
    putstr("open/close led n\n");
    putstr("set seg n num\n");

}
```

图 9

2、hello：定义字符串，然后打印在屏幕上

```
void printHello(){
    char hello[] = "Hello World!";
    putstr(hello);
    putch('\n');
}
```

图 10

3、time

通过 gettime() 函数获取此时的时间，返回值是一个 32 位的整数，精度为 10^{-4}s ，采用相除、取模的方式分别得到秒、分、时，并按顺序输出：

```
void printTime(int tim){//precision is 0.0001s
    int t, hour, min, sec;
    t = tim / 10000;
    sec = t % 60;
    t /= 60;
    min = t % 60;
    t /= 60;
    hour = t % 24;

    putInt(hour);
    putch(':');
    putInt(min);
    putch(':');
    putInt(sec);
    putch('\n');
}
```

图 11

4、set time

提示用户依次输入时、分、秒，接收到的输入为字符串，挨个转换为对应的整数后，将它们合为一个精度为 10^{-4} 秒的 32 位整数，调用 settime() 设置：

```
void seTtime(){
    putstr("please input hour, min, sec with space:");
    char t[N];
    getstr(t);
    int hour, min, sec;
    hour = min = sec = 0;
    //split, and judge whether wrong parameter
    int len = stringlen(t);
    int i;
    for(i=0; t[i]!=' '; i++)
        hour = 10*hour+t[i]-'0';
    for(i++; t[i]!=' '; i++)
        min = 10*min + t[i] - '0';
    for(i++; t[i]!=' ' && i!=len; i++)
        sec = 10*sec+t[i]-'0';
    //judge ERROR:
    if(hour > 24 || sec > 59 || min > 59)
        error(ERROR_PARA);
    else  {
        int num = 0;
        num = (sec+60*min+3600*hour)*10000;
        settime(num);
    }
}
```

图 12

5、fib n

利用斐波那契数列的生成原理，采用一个 while 循环计算数列第 n 项并输出即可：

```
void fib_n(int n){
    int a0 = 0, a1 = 1, an;
    int index = 1;
    if(n == 0)
        an = 0;
    else if (n == 1)
        an = 1;
    while ((index < n)){
        an = a0 + a1;
        a0 = a1;
        a1 = an;
        index++;
    }
    putInt(an);
    putch('\n');
}
```

图 13

6、open/close LED n

该命令从输入的字符串中找到数字，并调用 open_LEDR 或者 close_LEDR 函数控制 LED

灯。（处理函数较长，代码随文件夹一同提交）

7、set seg n num

该命令的处理函数从字符串中取出对应的数字 n 与 num ，调用 set_SEG 函数将数码管写入对应数值。（处理函数较长，代码随文件夹一同提交）

在软件的 main 函数中，只需要读入各个命令并且分析、输出结果即可：

```
char instr[N];
getstr(instr);
if(stringcmp(instr, "help", stringlen("help"))==0){
    printHelp();
}
else if(stringcmp(instr, "hello", stringlen("hello"))==0)
    printHello();
else if(stringcmp(instr, "time", stringlen("time"))==0){
    int time = gettime();
    printTime(time);
}
else if(stringcmp(instr, "set time", stringlen("set time"))==0){
    seTtime();
}
```

图 14

5 遇到的问题及解决方法

5.1 代码复用

计算器是自己之前用 C++ 写的，转换为纯纯的 C 语言过程中出现了一系列的问题，比如没有 vector ，需要自己手写数组来实现等等。最严重的是自己在进行转换的过程中，对使用 ctrl+f 将 int& 全部替换为 int，编译是通过了，可是忘了自己使用引用的目的，函数调用后，并没有将参数的值传递回去，因此每次计算表达式时都会报错。debug 时也无从下手，直到自己挨个将错误处有关函数实现一一分析后才找到问题。

5.2 fib 函数的报错

验证 fib 数列时，发生某一个数 n 的值比 n+1 对应的值还要小，有时候还会输出负数，经同学提醒发现自己没有考虑指数型增长的斐波那契数列会很快地增长到 32 位整数之外，因此添加了对 n 的大小判断。

5.3 屏幕显示错乱

依次编译运行后屏幕上出现了参差不齐的大量 h 字符。通过 RTLviewer 对电路图检查发现，编译器对 addr«7+i 解释顺序为 addr«(7+i) 。将其修改为 (addr«7)+i 就恢复了正常显示。

5.4 软件 make 编译报错

因为 RV32I 指令中没有常用的乘法指令，但是 gcc 在编译过程中如果遇到整数乘法会直接调用软件乘法函数 ___mulsi3 来通过软件完成整数乘法操作。所以需要添加乘除法函数 ___divsi3、___modsi3 链接到主函数。

如下面两图：

```
unsigned int __modsi3(unsigned int a, unsigned int b) {
    unsigned int bit = 1;
    unsigned int res = 0;

    while (b < a && bit && !(b & (1UL << 31))) {
        b <<= 1;
        bit <<= 1;
    }
    while (bit) {
        if (a >= b) {
            a -= b;
            res |= bit;
        }
        bit >>= 1;
        b >>= 1;
    }
    return a;
}
```

图 15: 取模指令

```
unsigned int __divsi3(unsigned int a, unsigned int b) {
    unsigned int bit = 1;
    unsigned int res = 0;

    while (b < a && bit && !(b & (1UL << 31))) {
        b <<= 1;
        bit <<= 1;
    }
    while (bit) {
        if (a >= b) {
            a -= b;
            res |= bit;
        }
        bit >>= 1;
        b >>= 1;
    }
    return res;
}
```

图 16: 除法指令

5.5 编译时间过长、资源不够

因为使用了手写的 regs 作为指令存储器、数据存储器，整个项目难以综合，编译时间过长。后来将两个存储器的实现改为 IP 核实现：指令存储器采用 1-PORT ROM，数据存储器改为 2-PORT RAM 即可。

6 实验得到的启示

1. 软硬件结合推进，模块化编写。使得 debug 可以定位到某一具体的模块（虽然 de 一个 bug 也需要很久）

2. 团队合作中需要建立完善的版本管理方法如 git 等（我们的团队未考虑此问题采用了最笨的一种：QQ 群文件 (。﹏。)，导致实验后期版本混乱）。做完实验回收站长这个样：

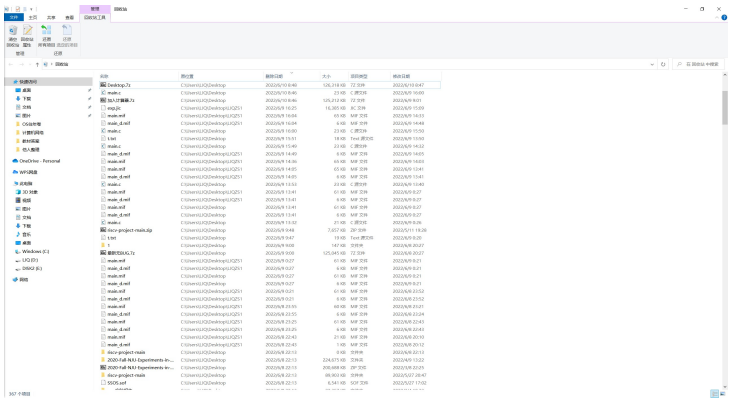

图 17: 一片祥和的回收站

3. C 语言、C++ 和 Verilog 在语法上十分相似，但是在性质、思想和运用场合上是天差地别。需要在运用时把握他们的特点，不能混在一起。

7 意见与建议

希望以后的课程在这门实验课之前能提供简单的 Verilog 语言教学，重点是硬件电路的设计思想和方法。在完全没有学习过这种设计的情况下直接开始写各个实验，会感到十分困难。

同时希望老师提供的虚拟机能带文件拖拽、剪贴板共享、GUI（每次 make 都要用 fstp 传到主机还是有点麻烦的）。