exp12

计算机系统

201840114 张嘉伟 三班
201220069 周心同 一班
200220027 原博洋 三班

2021 年 12 月 30 日

目录

完成功能 [Top]

一.基础功能 [Top]

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，计算斐波那契数列并显示结果
    - 打入未知命令，输出 Unknown Command。

二.扩展功能 [Top]

1. **支持led外设**
2. **支持help，clear等基本命令，有报错功能**
3. **命令行显示，光标显示**
4. **表达式计算器，支持加减乘除和取负运算，支持括号**
5. **游戏模块**

    - g2048 (2048)
    - snake (贪吃蛇)
    - chess (五子棋)
    - russia (俄罗斯方块)
    - mazz (走迷宫)

组内分工 [Top]

• 完成硬件部分的编写和对软件的数据支持

• 完成实验报告的硬件部分

201220069 周心同

• 完成软件部分的编写和对游戏的接口支持

• 完成实验报告的软件部分

201220027 原博洋

• 完成游戏部分的编写和部分软件内容，并辅助前两位同学debug

• 完成实验报告的排版

硬件部分 [Top]

本实验需要完成一个简易的计算机系统。它的硬件部分由五个主要模块组成，分别是CPU、指令及数据存储器、VGA、键盘、特殊寄存器及LED控制模块。这五个模块组合在一起，形成了具有完整功能的计算机系统硬件。以下将对这五个模块的设计进行简单的介绍。

一、CPU [Top]

本实验中的CPU是一个单周期CPU，沿用了实验十一中的CPU设计，没有进行改动，主要的接口定义如下：

```
module rv32is(
    input   clock,
    input   reset,
    output [31:0] imemaddr,
    input  [31:0] imemdataout,
    output  imemclk,
    output [31:0] dmemaddr,
    input  [31:0] dmemdataout,
    output [31:0] dmemdatain,
    output  dmemrdclk,
    output  dmemwrclk,
    output [2:0] dmemop,
    output  dmemwe,
    output [31:0] dbgdata);
```

由于没有采用五段流水线设计，所以CPU的时钟没有直接采用CLOCK_50，因为过高的时钟频率可能会导致CPU来不及处理工作，所以将50MHZ的CLOCK_50降频为25MHZ的时钟信号作为CPU的时钟，也就是整个计算机系统的主时钟。

以imem开头的三个接口主要负责与指令存储器进行交互，在一个时钟周期开始（CPU时钟信号的下降沿）时从指令寄存器中取出将要执行的下一条指令，通过imemdataout传入CPU进行指令的解析。

以dmem开头的单个接口主要负责与数据存储器进行交互。这里的数据存储器不仅指严格意义上的数据存储器，事实上，包括VGA模块、键盘模块等需要CPU读或者写的具有存储功能的模块都可以被视作数据存储器，只需要给这些模块划分出不同的地址，就能够通过软件控制CPU读取和写入这些不同的模块。下面给出dmemdataout接口的实现，可以很清晰地体现出这一思想：

```
assign dmemdataout = (dmemaddr[31:20]==12'h1)?(dmemdata):
                             (dmemaddr[31:20]==12'h3)?({{24{1'b0}},kmemdata}):
                             (dmemaddr[31:20]==12'h4)?(rmemdata): 32'b0;
```

当地址的高十二位为1时，读入的就是数据存储器的内容，为3时，读入的就是键盘缓冲区的内容，为4时，读入的就是特殊寄存器的内容。至于是否需要写入某个存储器模块，也是通过判定dmemaddr是否符合该模块地址要求以及总的写使能信号dmemwe是否为1来决定的。

二、指令及数据寄存器 [Top]

指令寄存器是使用IP核生成的单口ROM，CPU只能读取数据，不能写入，在顶层模块中的例化如下：

```
imem my_imem(imemaddr[16:2], imemclk, imemdataout);
```

imemaddr的低两位被舍弃了，因为imem的每个字被设置为4字节，一次能够读取32位，而CPU提供的地址是按照字节编址的。

数据存储器则不能简单地直接使用IP核生成，而是要在IP核生成的RAM之外再套一个模块。这是因为根据memop的不同，数据存储器需要进行32位、16位或8位数据的读取和写入。数据存储器的设计代码如下：

```
module datamem(
    input [2:0] memop,
    input [31:0] data,
    input [16:0] rdaddr,
    input rdclk,
    input [16:0] wraddr,
    input wrclk,
    input wren,
    output reg [31:0] dataout
);
    wire [31:0]q;
    reg [31:0]datain;
    wire [3:0]byteena_a;
    wire [7:0]byteout;
    wire [15:0]wordout;

    assign wordout = (rdaddr[1]==1'b1)? q[31:16]:q[15:0];
    assign byteout = (rdaddr[1]==1'b1)? ((rdaddr[0]==1'b1)?q[31:24]:q[23:16]):
((rdaddr[0]==1'b1)?q[15:8]:q[7:0]);

    dmem my_dmem(byteena_a, datain, rdaddr[16:2], rdclk, wraddr[16:2], wrclk,
wren, q);

   byteen_generate my_byteen(memop, wraddr[1:0], byteena_a);

    always @ (*)    // read
    begin
        case(memop)
            3'b000 : begin
                dataout = {{24{byteout[7]}}, byteout};
            end
            3'b001 : begin
                dataout = {{16{wordout[15]}}, wordout};
```

```
            end
            3'b010 : begin
                dataout = q;
            end
            3'b100 : begin
                dataout = {{24{1'b0}}, byteout};
            end
            3'b101 : begin
                dataout = {{16{1'b0}}, wordout};
            end
            default :begin
                dataout = 32'b0;
            end
        endcase
    end

    always @ (*) //write
    begin
        if(wren)
        begin
            case(memop)
                3'b000 : begin
                    datain = {{4{data[7:0]}}};
                end
                3'b001 : begin
                    datain = {{2{data[15:0]}}};
                end
                default : begin
                    datain = data;
                end
            endcase
        end else begin
            datain <= 32'b0;
        end
    end
endmodule
```

dmem是由IP核生成的，带有byteen_a接口的双口RAM。byteen_generate模块会根据不同的memop和写入地址的低两位生成不同的byteen_a。dmem读取出的输出也需要根据不同的memop和读取地址的低两位来进行裁剪与拼接，然后才能作为最终结果输出。

数据存储器是一个CPU可读可写的模块，高十二位为1的所有地址空间都属于它，也就是说它理论上有1MB的容量，但由于IP核生成的RAM有大小限制，它实际上只有一半的理论容量，但也足够用了。

三、VGA [Top]

本实验中的VGA模块与实验九中的VGA模块类似，都拥有一个字符显存RAM和一个存放了字符颜色信息的ROM。和实验九相比，主要增加了滚屏的功能。字符显存设定的大小为64*70，也就是六十四行，每行可以存放70个字符。VGA模块接受一个六位的输入start_line，这个输入指示VGA从字符显存的哪一行开始显示，一共显示三十行，具体的实现方式如下：

```
assign row_out = (start_line + v_addr[8:4]) & 6'h3f;
assign row_ascii = v_addr[3:0];

assign col_out = h_addr / 9;
assign col_ascii = h_addr % 9;
```

通过修改start_line的值，软件可以很方便地控制显示器显示字符显存的哪三十行，也就很方便地实现了滚屏的功能。

此外，光标功能也是通过硬件来实现的，VGA模块内部会保存CPU最后写入的位置，然后在该位置后显示一个闪烁的光标。

VGA是一个CPU只写的模块，高十二位为2的所有地址空间都属于它。

四、键盘 [Top]

和实验九不同，本实验中的键盘模块不能直接修改VGA字符显存中的内容，它只需要把键盘读取到的扫描码保存在一个缓冲区中，让CPU来读取，然后通过软件进行处理。键盘的顶层模块设计如下：

```
module keyboard(
    input clk,
    input clrn,
    input [19:0] rdaddr,
    input ps2_clk,
    input ps2_data,
    input we,
    output [7:0] key_code
);

    reg [7:0] buffer [34:0];
    wire [7:0] tail;

    assign tail = buffer[32];

    wire sign_shift;
    wire sign_caps;
    wire sign_ctrl;
    wire [7:0] cur_key;
    wire [7:0] key_count;
    wire [7:0] ascii_key;
    wire ready;

    mykeyboard mykey(clk, clrn, ps2_clk, ps2_data, key_count, cur_key,
ascii_key, sign_shift, sign_caps, sign_ctrl, ready);

    reg [7:0] count = 8'b00000000;

    // write to the buffer
    always @ (posedge ps2_clk)
    begin

        buffer[33] <= sign_shift;
        buffer[34] <= sign_caps;

        if (count == 1)
```

```
        begin
            if (cur_key != 8'h00 && cur_key != 8'hf0 && cur_key != 8'h12 &&
cur_key != 8'h59 && cur_key != 8'h58)
            begin
                count <= count + 1;
                buffer[tail] <= cur_key;
                if (buffer[32] == 8'h1f)
                begin
                    buffer[32] <= 0;
                end
                else
                begin
                    buffer[32] <= buffer[32] + 1;
                end
            end
        end
        else if (count == 8'h0a)
        begin
            count <= 0;
        end
        else
        begin
            count <= count + 1;
        end
    end

    // read to the cpu
    assign key_code = buffer[rdaddr];

endmodule
```

该模块内部定义了一个35*8位的寄存器堆。其中0到31号这32个寄存器是用来储存读取到的扫描码的，而32，33，34号寄存器分别用来存储tail指针，sign_shift和sign_caps。tail指针始终指向键盘向缓冲区写入的最后一个位置的下一个位置，它与head指针搭配，可以很容易地判断缓冲区中是否还有未读取的扫描码。tail指针由硬件维护，而head指针交给软件维护。

该模块中例化的mykeyboard模块与实验九中实现的键盘模块相同。

键盘是一个CPU只读的模块，高十二位为3的所有地址空间都属于它。

五、特殊寄存器及LED控制模块 [Top]

特殊寄存器是存放一些特殊的值的模块。该模块的设计如下：

```
module special_reg(
    input rdclk,
    input wrclk,
    input [31:0] wraddr,
    input wren,
    input [31:0] datain,
    output [31:0] dataout,
    output [5:0] line
    );

    reg [5:0] line_ctrl = 0;
```

```
    wire [31:0] time_read;
    timer mytime(rdclk, time_read);

    assign dataout = time_read;
    assign line = line_ctrl;

    always @ (posedge wrclk)
    begin
        if(wren)
        begin
            line_ctrl <= datain[5:0];
        end
    end
endmodule
```

该模块输出的line被接入VGA模块作为start_line，所以CPU只要修改该模块中的line_ctrl就能够控制滚屏。并且该模块还提供了一个time数据，这个time是系统启动后所经过的毫秒数，由一个计数器timer生成，主要是为了支持time指令而设置的。高十二位为4的所有地址空间都属于这个特殊寄存器。

LED控制模块顾名思义，是用来控制开发板LED灯开关的。它是一个CPU只写的模块，高十二位为5的所有地址空间都属于该模块。它有一个十位的输出，分别控制十个LED灯。软件可以通过向该模块中写入特定的值来控制LED灯的亮和灭。

软件部分 [Top]

一. 基础部分 [Top]

1.1 软件与硬件数据的传递 [Top]

在sys.h中，定义一些数据在硬件中的地址

```
#define VGA_START    0x00200000
//#define VGA_LINE_O   0x00210000
#define VGA_MAXLINE  64
//#define LINE_MASK    0x003f
#define VGA_MAXCOL   70
#define KEY_TAIL     0x00300020
#define KEY_START    0x00300000
#define SIGN_SHIFT   0x00300021
#define SIGN_CAPS    0x00300022
//#define TIME_ms      0x00400001
#define TIME_ADDR    0x00400000
#define LINE_0       0x00400001
#define SCR_MAXLINE  30//滚屏支持的行数
#define COLOR_ADDR   0x00480000  //001
#define LED_0        0x00500000  //0-9
```

在软件中，通过 char *类型来访问

```
char* led=(char *)LED_0;
char *vga_start = (char*) VGA_START;
char *key_tail = (char *)KEY_TAIL;
char *key_buf = (char *)KEY_START;
char *time =(char *) TIME_ADDR;
//char *time_ms=(char *) TIME_ms;
char *sign_caps=(char *)SIGN_CAPS;
char *sign_shift=(char *)SIGN_SHIFT;
char *line_0=(char *)LINE_0;
```

1.2 数据类型的转换 [Top]

在输入和输出数据的时候，数据类型都是字符型的，然而要做计算处理时，数据类型则需要int（或uint）型的，因而这里有一些数据类型转换函数。

```
char *itoa(int i, char *a);//十六进制数字转换成十六进制字符串
int htob(char *h);//十六进制字符串变为十进制数字
char* itoa_10(char* res,int num); //把十进制数字转换成字符串
char* itoa_long_10(char* res,long long int num);
```

实现方式比较简单，以htob为例，如下：

```
int htob(char *h)//十六进制字符串变为十进制数字
{
  int number =0;
  int len;
  for(len=0; h[len]!='\0';len++);
  int base=1;
  for(int i=len-1;i>=0;i--)
  {
     if(h[i]>='0'  && h[i]<='9')
     {
       number += (h[i]-'0')*base;
     }
     else if(h[i]>='a' && h[i]<='f')
     {
       number+= (h[i]-'a'+10)*base;
     }
     base = base <<4;

  }
  return number;
}
```

二. 基础模块与基础功能 [Top]

2.1 输入模块 [Top]

在main函数中，做完初始化工作后，进入while（1）无限循环，不断接受输入并输出回应。

```
int main()
{
    vga_init();
    char mingling[]=">>>";
    putstr(mingling);
    while(1)
    {
       char ch=getch();
       putcom(ch);
    }
    return 0;
}
```

在接受输入的函数里面，通过`while (head == *key_tail);`这一步，不断比对当前head和硬件传来的tail，当不相等时说明有新的输入，此时进行读取。

读取到的为扫描码，将它转换为ascii码，由于打表上有一点小失误，需要多一句`if(tmp == ' ' && c != 0x29 ) tmp ='\0';`

在读取时，将输入保存到command中，以便对一行读取的内容进行处理。

```
char getch()//转换ascii
{
  int c;
  char tmp;
  while (head == *key_tail)
    ;
  do {
    c = key_buf[head];
    key_buf[head] = '\0';
  } while (c == 0x12 || c == 0x58);

  tmp = tran_ascii(c);
  if(tmp == ' ' && c != 0x29 ) tmp ='\0';

  if(tmp!='\0' && tmp != 8)
  {
    command[cnt]=tmp;
    cnt++;
  }

  if (head == 0x1f || *key_tail == 0)
    head = 0;
  else
    ++head;
  return tmp;
}
```

其中ascii码转换模块如下，这里的sign是硬件传过来的（打表过长，这里不贴）

```
char tran_ascii(int key_code)
{
  if(*sign_caps&&*sign_shift) return caps_shift[key_code];
  else if(*sign_shift) return shift[key_code];
  else if(*sign_caps) return caps[key_code];
  else return normal[key_code];
}
```

2.2 输出模块 [Top]

2.2.1 系统输出 [Top]

系统输出是指软件自己打印到vga上的而非键盘输入的，这代表有很多内容都不需要去处理。

例如系统输出是不会有退格键的，如果有就报错。

```
void putch(char ch)//系统输出
{
  if(ch==8) //backspace
  {
    char wrong[]="assert wrong!\n";
    putstr(wrong);
     return;
  }
  if(ch==10) //enter
  {
    vga_line_add();
    shell_line=vga_line;

    vga_ch=0;
    return;
  }
  vga_start[ ((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch] = ch;
  vga_ch++;
  if(vga_ch>=VGA_MAXCOL)
  {
    vga_line_add();
    vga_ch=0;
  }
  return;
}
```

配合系统输出的可以输出字符串的函数：

```
void putstr(char *str)
{
    for(char* p=str;*p!=0;p++)
      putch(*p);
}
```

2.2.2 键盘输出 [Top]

键盘输出需要考虑退格 光标 命令行 回车 滚屏 操作复杂 下面就拆分开来解析

```
void putcom(char ch)//按键
{
    if(ch==8) //backspace
  {
    if(vga_ch>3)
    {
      vga_ch--;
      vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch]='\0';
      if(vga_ch==3 && vga_line==shell_line)
      vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch-1]='>';
      else
      vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch-1]=command[cnt-2];
      //char h =vga_start[(vga_line<<6)+(vga_line<<2)+(vga_line<<1)+vga_ch-1];
      //vga_start[(vga_line<<6)+(vga_line<<2)+(vga_line<<1)+vga_ch-1]=h;
      cnt--;
      command[cnt]='\0';
    }
    else if(vga_line!=shell_line)
    {
      if(vga_ch>0)
      {
        vga_ch--;
        vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch]='\0';
        vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch-1]=command[cnt-2];
        cnt--;
        command[cnt]='\0';
      }
      else
      {
        vga_ch=VGA_MAXCOL-1;
        vga_line--;
        vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch]='\0';
        vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch-1]=command[cnt-2];
        cnt--;
      command[cnt]='\0';
      }
    }
    // else
    // {
    //   if(vga_line>0)
    //   {
    //     vga_line--;
    //     vga_ch=VGA_MAXCOL-1;
    //   }
    // }
      return;
```

```
  }
  if(ch==10) //enter
  {
    command[cnt] = '\0';
    //TODO
      vga_line_add();
      shell_line=vga_line;
    //滚屏待处理
    vga_ch=0;
    deal_com(command);
    char mingling[]=">>>";
    putstr(mingling);
    // char startline[]="startline:";
    // char vgaline[]="vgaline:";
    // char buf[100];
    // putstr(startline);
    // putstr(itoa(start_line&0x3f,buf));
    // putstr(vgaline);
    // putstr(itoa(vga_line&0x3f,buf));
    cnt=0;
    command[cnt]='\0';
    return;
  }
  vga_start[ ((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch] = ch;
  vga_ch++;
  if(vga_ch>=VGA_MAXCOL)
  {
    //TODO
    vga_line_add();
    vga_ch=0;
  }
  return;
}
```

2.2.2.1 退格及其相关 [Top]

退格的基本思路是将前一个字符写入'\0'。

但是要考虑光标的问题，硬件中光标的实现是，将光标放在写入的字符的后一个位置，因为需要将删除的字符的前一个字符重新写入，由于硬件上不支持软件读取vga，所以之前保存的command就有用了，可以读取command重新写入，另外要注意到最前面的时候重新写入的是命令行'>'。

考虑到命令行，记录一个shell_line标记命令行行数，这样就可以判断当前行前三个是不是命令行了，防止误删。

另外注意在删除的时候更新command。

```
 if(ch==8) //backspace
  {
    if(vga_ch>3)
    {
      vga_ch--;
      vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch]='\0';
      if(vga_ch==3 && vga_line==shell_line)
      vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch-1]='>';
```

```
      else
      vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch-1]=command[cnt-2];
      cnt--;
      command[cnt]='\0';

    }
    else if(vga_line!=shell_line)
    {
      if(vga_ch>0)
      {
        vga_ch--;
        vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch]='\0';

        vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch-1]=command[cnt-2];

        cnt--;
        command[cnt]='\0';


      }
      else
      {
        vga_ch=VGA_MAXCOL-1;
        vga_line--;
        vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch]='\0';
        vga_start[((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch-1]=command[cnt-2];
        cnt--;
      command[cnt]='\0';
      }
    }
      return;
  }
```

2.2.2.2 回车及其相关 [Top]

回车要注意处理command命令

```
if(ch==10) //enter
  {

    command[cnt] = '\0';

    vga_line_add();
    shell_line=vga_line;
    vga_ch=0;

    deal_com(command);

    char mingling[]=">>>";
    putstr(mingling);
```

```
    cnt=0;
    command[cnt]='\0';

    return;
  }
```

2.2.2.3 纯字符 [Top]

最基本的一些处理

```
  vga_start[ ((vga_line&0x3f)<<6)+((vga_line&0x3f)<<2)+((vga_line&0x3f)
<<1)+vga_ch] = ch;
  vga_ch++;
   if(vga_ch>=VGA_MAXCOL)
  {
    vga_line_add();
    vga_ch=0;
  }
```

2.2.2.4 滚屏 [Top]

注意到上文每当要到下一行的时候，我都是用vga_line_add()而不是vga_line++来处理的，这是滚屏的关键。

在硬件中，显示的位置是从start_line开始往后三十行的，而不是vga_line，所以当vga_line上限时，start_line要++，从而实现滚屏。而vga_line控制的是写入的位置。（在硬件中start_line只给了我写入权限 但没有给我读取权限 所以要自己写一个start_line 然后再用`*line_0=(char)(start_line&0x3f);`写入）

我在有关地址的计算中，都是与0x3f取与（即mod64），这样就能够实现无限滚屏。

值得注意的是，每次往上滚一行时，都要把滚上去的那行清零，防止到时候循环回来还会显示。

```
void vga_line_add()
{
   vga_line++;
   if((vga_line&0x3f)>= ((SCR_MAXLINE+start_line) & 0x3f) )
   {
     start_line++;
     *line_0=(char)(start_line&0x3f);
     for(int i=0;i<VGA_MAXCOL;i++)
      {
        vga_start[(((start_line&0x3f)-1)<<6)+(((start_line&0x3f)-1)<<2)+
(((start_line&0x3f)-1)<<1)+i] ='\0';
      }
   }
}
```

2.3 命令模块 [Top]

命令模块写的不怎么好，但是为了防止出bug，姑且就这样写了。

对于每一个command， 例如在屏幕上输出hello再按下回车，此时在deal_command函数内处理时，command为 hello\n\0,cnt为6。

我们以hello为例，我会首先对cnt进行判断，然后进行逐个字符的比较，确保为hello\n\0的情况下，调用对应的函数（或直接处理）进行应对。

```
int flag[14]={0};  //0:  hello 1:time 2:fib  3:wrong 4:color 5:cal 6:help
7:clear 8:led 9:g2048 10:snake 11:chess 12:russia 13:mazz
```

对于各个命令，我设了一个flag数组，除了flag[3]是特殊的报错函数外，其他的都会在flag[i]为0时，调用对应的函数进行处理，而若是flag[3]为1时，会调用报错函数，表明你使用了该command，但是输入不符合规范。

输入规范可以通过输入help命令查看：

```
        putstr("==================COMMAND HELP=====================\n");
        putstr("help     : print COMMAND HELP\n");
        putstr("hello    : print hello\n");
        putstr("time     : print time\n");
        putstr("fib n    : calculate fib(n)\n");
        putstr("cal expr : calculate expression\n");
        putstr("clear    : clear the screen\n");
        putstr("led n    : turn on/off the led\n");
        putstr("g2048    : play the game 2048\n");
        putstr("snake    : play the game snake\n");
        putstr("chess    : play the game chess\n");
        putstr("russia   : play the game russia\n");
        putstr("mazz     : play the game mazz\n");
        putstr("====================================================\n");
```

三. 一些命令函数的实现 [Top]

3.1 hello [Top]

```
char temp[]="Hello World!\n";
putstr (temp);
```

3.2 time [Top]

硬件将传过来一个以毫秒为单位的十六进制的time（为后面游戏作准备）

```
char buf[100];

int c=*time;
c=c/1000;

putstr(itoa_10( buf,  htob(itoa(c,buf))  ));

putstr(endl);
```

3.3 fib [Top]

一开始我是通过for循环计算的，后来发现可以打表。两种实现在代码里都保留了下来，这里就不贴了。

3.4 cal（表达式计算器） [Top]

整个表达式计算器是纯自己写的，略微复杂。

首先创建一个全局的tokens数组，每次进行cal的时候先初始化，带一个nr_token标记数组元素个数，由于是字符串数组这里用二维字符数组代替。

第一步 先将表达式分解成token存在tokens数组，具体做法就是分为 +-*/()以及数字七种情况，如果有其他情况报错。

第二步 辨别特殊运算符-，因为-有可能是取负而不是减号，所以需要扫一遍tokens数组，若‘-’符号前面是+-*/符号，说明这是取负符号，改为@以作区分。

第三步 递归运算，简单来说，对于一个表达式，先去掉外层的括号（如果有的话），然后找到剩下的运算符中，最晚算的那一个，然后递归运算改运算符左右两边的value。

具体实现已经贴出来，这个表达式计算器我写的还是可以的。

```
char tokens[32][32];
int nr_token=0;
int expr(char *e,int *success)
{
   nr_token=0;
    if(make_token(e)==0)
    {
      *success=0;//putstr("tiaoshi1");
      return 0;

    }
    for(int i=0;i<nr_token;i++)
    {
      if(tokens[i][0]== '-')
      {
        if(i==0 || tokens[i-1][0] == '+' || tokens[i-1][0]=='-' || tokens[i-1]
[0]=='/'   || tokens[i-1][0]=='*' )
        tokens[i][0]='@';
      }
    }
    return eval(0,nr_token-1,success);
}

int make_token(char *e)
{
  for(int i=0;e[i]!='\0';i++)
  {

     if(e[i]>='0' && e[i]<='9')
     {
        int j=0;
        while(e[i] >= '0' && e[i]<= '9')
        {
           tokens[nr_token][j]=e[i];
           j++;i++;
        }
        i--;
        tokens[nr_token][j]='\0';
        nr_token++;
     }
     else if(e[i]=='+' || e[i]=='-' || e[i]=='*' || e[i]=='/'|| e[i] =='(' ||
e[i]==')')
     {
       tokens[nr_token][0]=e[i];
       tokens[nr_token][1]='\0';
       nr_token++;
     }
```

```
     else
     {
      // putstr("tiaoshi6:");
       //putch(e[i]);
       //putstr(endl);
       return 0;
     }
  }
  return 1;
}
int eval(int p,int q,int * success)
{
  if(*success == 0) return 0;
  if(p>q)
  {
    *success=0;return 0;

  }
  else if(p==q)
  {
     int number=0;
     for(int i=0;tokens[p][i]!='\0';i++)
     {
       number = (number<<3) + (number<<1) + tokens[p][i] - '0';
     }
     return number;
  }
  else if(check_parentheses(p, q,success) == 1)
  {
    return eval(p+1,q-1,success);
  }
  else
  {
    int cnt=0;
    char op_type='s';
    int op_position;
    for( int i=p;  i<q;  i++ )
    {
        if(tokens[i][0] == '(' ) cnt++;
        else if(tokens[i][0] == ')' ) cnt--;
        //cnt<0情况在前面处理了
        if(cnt>0) continue;
        //cnt==0
        if( isp(tokens[i][0]) <=  isp(op_type) )
        {
            op_type=tokens[i][0];
            op_position=i;
        }
    }


    if(op_type=='s') {*success=0;return 0;}
    if(op_type=='@')
    {
        int val=eval(op_position+1,q,success);
        return -val;
    }
    else
```

```
    {
        int val1 = eval(p, op_position - 1,success);
        int val2 = eval(op_position + 1, q,success);

        //printf (" %d %d    ",val1 ,val2);

        switch(op_type)
        {
            case '+': return val1 + val2;
            case '-': return val1 - val2;
            case '*': return val1 * val2;
            case '/': return val1 / val2;
            default: *success=0;return 0;
        }
    }
  }
}
int check_parentheses(int p,int  q,int *success)
{
  if(tokens[p][0]!='(') return 0;
  if(tokens[q][0]!=')') return 0;
  int cnt=0;
  for(int i=p+1;i<q;i++)
  {
      if(tokens[i][0]=='(') cnt++;
      else if(tokens[i][0]==')') cnt--;
      if(cnt<0) {*success=0;return 0;}
  }
  return 1;
}
int isp( char op_type)
{
    switch (op_type)
    {
        case '+':return 0;
        case '-':return 0;
        case '*':return 1;
        case '/':return 1;
        case '@':return 2;
        case '&':return 2;
        case 's':return 3;//初始值
        default:return 3;//不是符号
    }
}
```

3.5 led [Top]

没什么好说的，软件配合一下硬件就可以了，由于实现意义不大，七段数码管就懒得实现了。

```
*(led+x) = (char)1;
```

3.6 help [Top]

上文贴过

3.7 clear [Top]

不难理解其实清屏就是vga_init函数。

注意要把shell_line，start_line（*line_0）也都清零了。

```
void vga_init(){
    vga_line = 0;
    vga_ch =0;
    shell_line=0;
    start_line=0;
    *line_0 = 0;
    for(int i=0;i<VGA_MAXLINE;i++)
        for(int j=0;j<VGA_MAXCOL;j++)
            vga_start[ (i<<6)+(i<<2)+(i<<1)+j ] =0;//64+4+2
}
```

四. 游戏模块 [Top]

4.1 基础函数 [Top]

游戏模块有一些基础函数，由于不能调用库函数，需要自己提供接口，如下：

```
void srand()
{
  seed= 41;
}
int rand()
{
  seed = ((seed * 214013 + 2531011) >> 16) & 0x7fff;
    if (seed == 41) seed = 40;
    return seed;
}
void sleep()
{
  int t=*time;
  int c=*time;
  while(   t+100>=c   ) c=*time;
  return;
}
void sleep2()
{
  int t=*time;
  int c=*time;
  while(   t+50>=c   ) c=*time;
  return;
}

int _kbhit()
{
if (head!= *key_tail) return 1;
else return 0;
}
```

随机数用线性同余法实现，sleep用传过来的time实现，_kbhit判断方式与之前同理，清屏就用vga_init，输入输出就用之前的函数。

4.2 g2048 （2048） [Top]

以下所有游戏主要代码的编写是由原博洋同学完成的。

2048游戏虽然也废了一番功夫，但是比起后面的来说，这可以算是最简单的一个了。

如果玩到十万以上的话，显示上就不能对齐了（实际上不可能玩到十万以上。

另外这里的随机函数并不是随便写一个就可以的。

wasd控制 按q退出

```
int score=0;int k=0;int seed=0;
void move(int i, int v, int map[])
{
    if (i + v < 0 || 15 < i + v || !map[i])
        return;
    if ((v == 1 || v == -1) && i / 4 - (i + v) / 4)
        return;
    if (map[i + v] == map[i])
  {
        map[i + v] *= -2; score += map[i]; map[i] = k = 0;
  }
    if (!map[i + v])
  {
        map[i + v] = map[i];map[i] = k = 0;move(i + v, v, map);
  }
}
void order(int end, int i, int v, int map[])
{
    for (int begin = end == 16 ? 0 : 15; begin - end; begin += i)
        move(begin, v, map);
}
void g2048()
{
    int map[16] ;
  score=0;k=0;seed=0;
  for(int i=0;i<=15;i++) map[i]=0;
  int game = 1,  i, j;
  char c;
    const char *wall[4] = { "\n----- ----- ----- -----\n","|","|","|" };
    for (srand(),
   c ='0', j = 0; game; j = k = 1, c = getch()
   ) {
        if (c == 'a' || c == 'w')
            order(16, 1, c == 'a' ? -1 : -4, map);
        else if (c == 'd' || c == 's')
            order(-1, -1, c == 'd' ? 1 : 4, map);
    else if(c=='q')
    {
      vga_init();
      return ;
    }
```

```
        for (i = 0; i < 16; i++)
    {
      if(map[i]<0) map[i] = -map[i];
      //map[i] || (j = 0);
      if(map[i]==0) j=0;
    }
        do if (i = rand() % 16, j || k)
            break;
        while (map[i] || (map[i] = rand() % 5 ? 2 : 4, 0));
        vga_init();
        for (i = 0; j && (i < 15 || (game = 0)); i++)
            if (   (i < 12 && map[i] == map[i + 4]) ||
                 ( ((i + 1 )& 3) && map[i] == map[i + 1] )   )
        break;
        if (!game)
    {
      char x[]="Game over!";
            putstr(x);
    }
        else
    {
            //cout << "score:" << score;
      char x[]="score: ";
      putstr(x);
      char buf[100];
      putstr( itoa_10 (buf,score));
    }
        for (i = 0; i < 16; i++)
    {
        //  cout << wall[i & 3] <<"    "<< map[i];
      putstr(wall[i&3]);
      char x[]="     ";
      char x1[]="    ";
      char x2[]="   ";
      char x3[]="  ";
      char x4[]=" ";
      char x5[]="";

      if(map[i]>10000) putstr(x5);
      else if(map[i]>1000) putstr(x4);
      else if(map[i]>100) putstr(x3);
      else if(map[i]>10) putstr(x2);
      else if(map[i]>0) putstr(x1);
      else putstr(x);

      char buf[100];
      putstr(itoa_10(buf,map[i]));
    }
    if(!game)
    {
      putstr(endl);
    }
    }
}
```

4.3 snake （贪吃蛇） [Top]

wasd控制 按q退出

```
void snake() {
    int W = 20, S = W * W, z[2] = { 0 }, l = 3, i,   * p, f;
    char C;char c='D';
    int m[1000];
    for (int i = 0; i < S; i++)
        m[i] = 0;
    for (C = m[1] = -1; C - 27; sleep())
    {
        if (head!= *key_tail)
        {
            char t=getch();
            C = t & 95;
            C - 65 && C - 68 && C - 83 && C - 87 || (C ^ c) & 20 ^ 4 && (c = C);
            if(t=='q') {vga_init();return ;}
        }
        p = z + !!(c & 2);
        *p += c / 3 & 2;
        *p = (-- * p + W) % W;
        f = 1;
        vga_init();
        *(p = m + *z + z[1] * W) > 0 && (C = 27);
        for (; *p && (m[i = rand() % S] || (--m[i], ++l, --f)););
        for (i = 0, *p = l; i < S; )
        {

           if (m[i] > 0)
            {
                m[i] -= f;
                char tmp[]= "()";
                putstr(tmp);
            }
            else
                if (m[i])
                {
                  char tmp[]="OO";
                  putstr(tmp);
                }
                else
                {
                  char tmp[]="  ";
                  putstr(tmp);
                }


            if(!(++i % W))  {char tmp[]="|\n";putstr(tmp);}

        }
    }
}
```

4.4 chess （五子棋） [Top]

wasd控制 按空格下棋 按q退出

```
int abs(int i)
{
  if(i>=0) return i;
  else return -i;
}
int sum(int v, int l, int* W, int* S, int* m)
{
    return (abs(v % *W - (v + l) % *W) - *W + 1 && v + l >= 0 && v + l <*S) &&
        m[v] == m[v + l] ? 1 + sum(v + l, l, W, S, m) : 0;
}
void chess()
{
    int W = 13, S, z = 0, c = 1, r = 2, i, j;
    int m[1000];
    S = W * W;
    for (int i = 0; i < S; i++)
        m[i] = 0;
    for (; r < 4 && c - 27; )
    {
        c - 68 || ++z, c - 65 || --z, c - 83 || (z += W), c - 87 || (z -= W);
        vga_init();
        if (z = (z + S) % S, i = 2, !c && !m[z])
            for (m[z] = r ^= 3; j = i % 3 - 1 + i / 3 * W, i < 6; ++i)
                sum(z, j, &W, &S, m) + sum(z, -j, &W, &S, m) > 3 && (r ^= 3, i =
r += 4);
        for (i = 0; i < S; )
        {
            // cout << " >"[i == z] << " O@"[m[i]];
            if (i == z)
            {
                putstr(">");
            }
            else

            {
                putstr(" ");
            }
            if (m[i]==1)
            {
                putstr("O");
            }
            else if (m[i] == 2)
            {
                putstr("X");
            }
            else
            {
                putstr(" ");
            }
            if(!(++i%W))
            {
              char buf[10];
              putstr(itoa_10(buf,i/W));
```

```
              putstr(endl);
            }
        }
        for (i = 0; i < W; ++i)
            {
              putstr(" ");
              char tmp=(char)(97+i);
              putch(tmp);
            }
        if (r & 1)
            putstr("\nWhite ");
        else
            putstr("\nBlack ");
        if (r >= 4)
            putstr("win!\n");

        char t=getch();
        if(t=='q') {vga_init();return;}
        c = t  & 95;
    }
}
```

4.5 russia（俄罗斯方块） [Top]

w切换 ad控制位置 s加速

```
#define Get(C) for (C, i = n[T]; j = X + i % 4, k2 = Y + i / 4 % 4, i; i >>= 4)
int W = 10, H = 25, S, i, j, k2, c, d = 0, X = 0, Y = 0, T = 0, n[] = {
    51264, 12816, 21520, 21520, 25872, 34113, 21537, 38208, 25921, 38481,
    38484, 38209, 25922, 43345, 34388, 38160, 25920, 38177, 42580, 38993 },
m[1000];
int move2(int* v, int l)
{
    Get(*v += l)(j < 0 || j >= W || k2 >= H || m[k2 * W + j]) && (c = 0);
    return c ? 1 : (*v -= l, v == &Y && (c = -1));
}
void russia()
{
  W = 10; H = 25; d = 0; X = 0; Y = 0; T = 0;
    for (int i = 0; i < 1000; i++)
        m[i] = 0;
    S = W * H;
    for (; c - 27; sleep2(),vga_init() )
    {
    if(_kbhit())
    {
       char t=getch();
       if(t=='q') {   vga_init();return ;}
       c=t&95;
    }
    else
    {
      c=1;
    }

    i = n[T];
    for (; j = X + i % 4, k2 = Y + i / 4 % 4, i; i >>= 4)
```

```
     m[k2 * W + j] = 0;
        c ^ 65 || move2(&X, -1), c ^ 68 || move2(&X, 1), c ^ 83 || move2(&Y, 1);
        c ^ 87 || (i = T < 8 ? 1 : 3, move2(&T, T& i^ i ? 1 : -i));
        Get(++d - 10 || (d = 0, c = 1, move2(&Y, 1)))
    m[k2 * W + j] = 1;
        if (c == -1 && !(Y || (c = 27), T = rand() % 20, Y = X = 0))
            for (j = W, i = S - 1; j -= m[i], i; i-- % W || (j = W))
                for (j || (k2 = i += W); !j && (--k2 >= W); m[k2] = m[k2 - W]);
        for (; i < S; //++i % W || _cputs("|\n")
    )
    {
        if(m[i])
        {
          putstr("[]");
        }
        else
        {
          putstr("  ");
        }
        if(!(++i%W))
        {
          putstr("|\n");
        }

    }

    }
}
```

4.6 mazz（走迷宫） [Top]

wasd移动 q刷新迷宫 e退出

```
int W3 = 15, S3, c3 = 81, i3, j3, k3, l3[4] = { 1, -1 }, m3[1000], L3[1000];
int init() {
    for (i3 = S3; i3-- || (c3 = m3[i3 = rand() % S3] == 2, k3 = 0);)
        m3[i3] = 2 + ((i3 + 1) % W3 < 2) + (i3 % (S3 - W3) < W3);
    for (c3 && (L3[k3++] = i3); k3 && (j3 = L3[i3 = rand() % k3], 1);)
        for (L3[i3] = L3[--k3], i3 = m3[j3] & 3 ? m3[j3] = 0 : 4; i3 < 4; ++i3)
            c3 = j3 + l3[i3], m3[c3] & 3 && ++m3[c3], m3[c3] & 3 && (L3[k3++] =
c3);
    for (i3 = c3 ? W3 : S3; i3 >= S3 ? init(), 0 : m3[i3 - W3]; i3 += W3)
        m3[i3] & 3 | m3[S3 - 1 - i3] & 3 || (m3[j3 = i3] = m3[k3 = S3 - 1 - i3]
= 0);
    return 0;
}
void mazz()
{
  W3 = 15;c3 = 81;
    for (int i1 = 0; i1 < 1000; i1++)
        m3[i1] = 0;
    for (int i2 = 0; i2 < 1000; i2++)
        L3[i2] = 0;
    S3 = W3 * W3;
    srand();
    l3[2] -= l3[3] = W3;
    for (; c3 - 81 && j3 - k3 ? 0 : init(), c3 - 27; )
```

```
    {
        c3 - 87 || m3[j3 - w3] || (j3 -= w3), c3 - 68 || m3[j3 + 1] || ++j3;
        c3 - 83 || m3[j3 + w3] || (j3 += w3), c3 - 65 || m3[j3 - 1] || --j3;
    vga_init();
        for (i3 = 0; i3 < S3; )
    {

      if(i3-j3==0)
      {
        putstr("<>");
      }
      else
      {
        if(m3[i3])
        {
          putstr("[]");
        }
        else
        {
          putstr("  ");
        }
      }


      if(!(++i3%w3))
      {
        putstr(endl);
      }
    }
      char t=getch();
    if(t=='e')
    {
      vga_init();return ;
    }
    c3 = t& 95;
  }
}
```

遇到的问题与解决方案 [Top]

1. **软件内容进行编译时太多warning覆盖error**

   尝试了make|less 和 make>debug.txt都没用，后来发现可以用 make 2>debug.txt 。

2. **许许多多杂糅显示问题，如退格与命令行，退格与光标，回车与滚屏**

   不断进行bug测试，定位软件还是硬件问题，然后进行解决。

   遇到的问题过多，不赘述。

3. **游戏接口问题**

   由于这次写游戏不能调用库函数，所以要自己实现一些游戏接口，最麻烦的就是随机数和sleep。

   随机数在电脑中是用伪随机的序列来完成的，因而采用线性同余法的方式，由于没有类，把seed作为全局变量，一开始的seed计算方式比较随便，有可能出现多次随机到同一数的情况，游戏就会出现问题，最后经过查询，采用如下的seed计算方式：

   `seed = ((seed * 214013 + 2531011) >> 16) & 0x7fff;`

   同时在srand里将seed初始化为41，计算seed时再加一句当seed为41时，置为40。

   经过测试，这样实现的伪随机函数可以实现一长串的伪随机序列。

   sleep的实现有两种，一种是通过软件循环百万次的方式，让程序运行sleep的时间，一种是通过硬件传过来的time来计算等待时间，这里采取后面一种，个人觉得我的写法也比较巧妙。

实验的启示与建议 [Top]

一定要多和队友交流，考虑问题的来源。

如果出现了不能理解的bug，首先考虑自己的代码问题，而不要怀疑开发板有问题。

```
If it is the second case, remember:
The machine is always right!
Every line of untested code is always wrong!
```

另外，这里的软硬件结合的编写是特殊的，尤其是软件的特殊之处在于，有一些数据的值会自己改变。