同济大学

TONGJI UNIVERSITY

《计算机系统实验》

实验报告

实验名称	实验 3：GUI 程序开发
实验成员	王钧涛 2050254
日　　期	二零二三年 6 月 28 日

1、实验目的

应用程序开发：以 GUI 程序为主

检查方式：图形界面

2、实验内容

① VGA WishBone 控制器开发

② PS2 键盘 WishBone 控制器开发

③ 系统前置函数设计

④ “星穹铁道 MINI”游戏开发

3、实验步骤

① VGA WishBone 控制器开发：

VGA（Video Graphics Array)视频图形阵列是 IBM 于 1987 年随 PS/2 机一起推出的一种使用模拟信号的视频传输标准，具有分辨率高、显示速率快、颜色丰富等优点，在彩色显示器领域得到了广泛的应用。不支持热插拔，不支持音频传输。

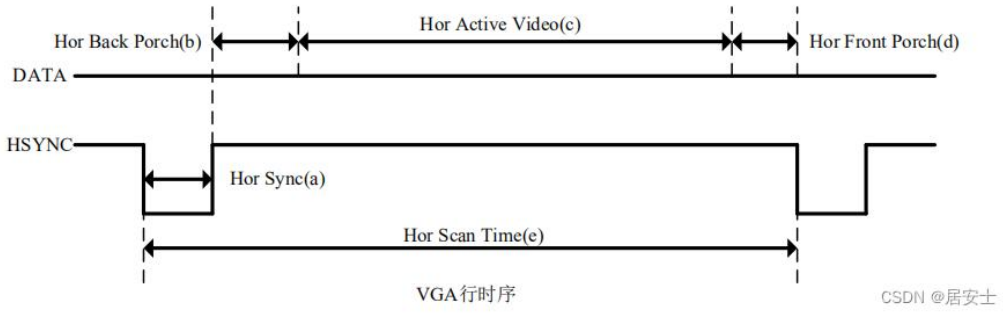
VGA 显示图像使用扫描的方式，从第一行的第一个像素开始，逐渐填充，第一行第一个、第一行第二个、、、、第二行第一个、第二行第二个、、、、第 n 行最后一个。通过这种方式构成一帧完整的图像，当扫描速度足够快，加之人眼的视觉暂留特性，我们会看到一幅完整的图片，而不是一个个闪烁的像素点。这就是 VGA 显示的原理。

表 2 常见刷新率时序表：

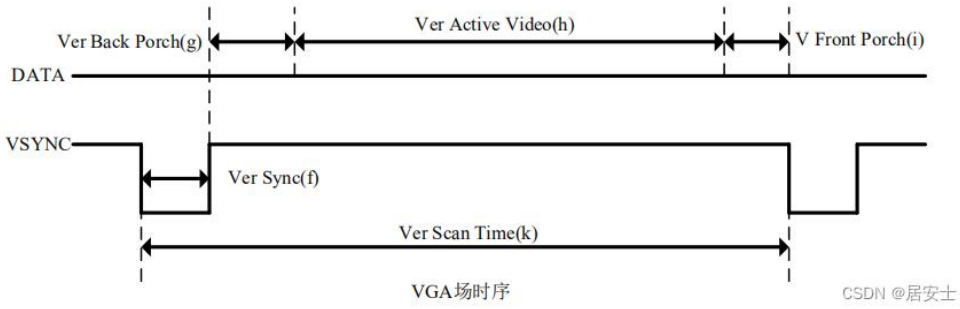
显示模式	时钟	行时序（像素数）					帧时序（行数）				
	（MHz）	a	b	c	d	e	o	p	q	r	s
640x480@60	25.175	96	48	640	16	800	2	33	480	10	525
640x480@75	31.5	64	120	640	16	840	3	16	480	1	500
800x600@60	40.0	128	88	800	40	1056	4	23	600	1	628
800x600@75	49.5	80	160	800	16	1056	3	21	600	1	625
1024x768@60	65	136	160	1024	24	1344	6	29	768	3	806
1024x768@75	78.8	176	176	1024	16	1312	3	28	768	1	800
1280x1024@60	108.0	112	248	1280	48	1688	3	38	1024	1	1066
1280x800@60	83.46	136	200	1280	64	1680	3	24	800	1	828
1440x900@60	106.47	152	232	1440	80	1904	3	28	900	1	932

本次试验采用的是 1024x768@60hz 的分辨率

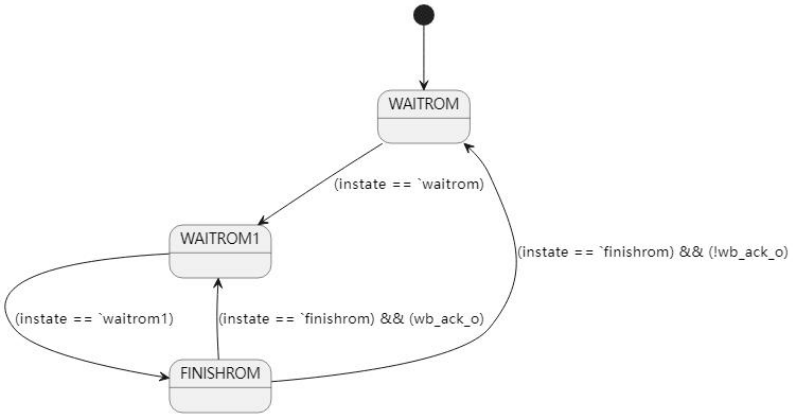
一行数据包括：Hor Sync（行同步）、Hor Back Porch（行消隐）、Hor Active Video（行规频有效）和 Hor Front Porch（行前肩），VGA 的行信号时序图如下图所示。



场时序与行信号时序类似：



在数字逻辑开发的 VGA 显示器的基础上，为了适应 CPU 的读写，在其上包装 WishBone 状态机。



然而，出现了这样的问题：原来 dis blk 大小为 1024x768，完整写入需要 1m 的指令，而 CPU 由于时序问题运行的时钟降频到了 20mhz，刷新一次 VGA 需要半秒钟，无法实现实时的 30hz 至少的刷新率

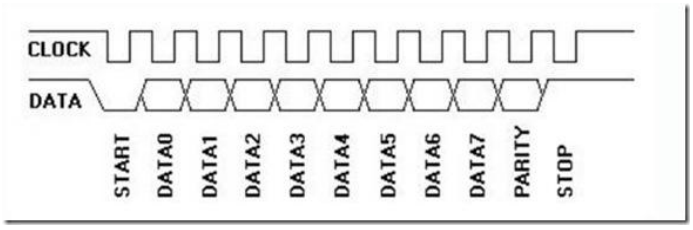
解决的方法是通过划分显示器单元，以 8*16 的单元格作为基本输入输出大小，将点阵输出变成 ascii 字符阵的输出。

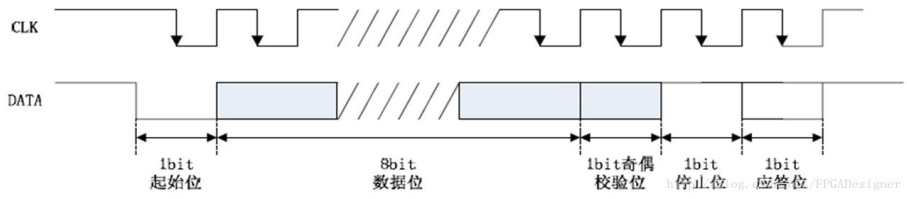
这样一来，显示器刷新区域降为 128 * 48，只需 10k 左右的指令即可完成刷新。

② PS2 键盘 WishBone 控制器开发：

1981 年 IBM 推出了 IBM PC/XT 键盘及其接标准。经过多年演变成 6 mini-DiN 连接器接口，封装上更小巧，用双向串行通讯协议并且提供有可选择的第三套键盘扫描码集，同时支持 17 个主机到键盘的命令。现在市面上的键盘都和 PS/2 及 AT 键盘兼容。

PS2 协议和串口接收差不多，只是数据要多一个奇偶校验位。其协议时序如下，只考虑 PS2 接收，即只接收外部 PS2 数据：





有两根线，一根时钟线，一根数据线。在没有数据传输的时候，两根线都是高电平。传输的时候，PS2 器件会开始启动时钟和数据变换，进行数据的传输。

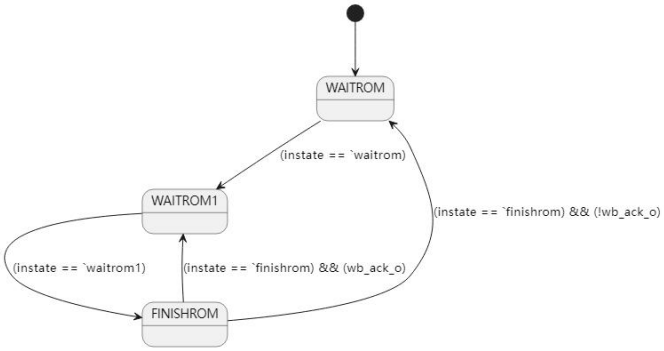
从时序可以看出，数据再时钟的上升沿变化，所以对于接收，要在时钟的**下降沿**进行数据的采集。

本次试验直接使用数字逻辑开发的键盘 PS2 驱动(FIFO)，并在其上包装 WishBone 状态机。为了开发方便，做了一定的简化，只检测第一扫描码，不检测第二扫描码，这样带来的代价是无法读取控制字符，但是相对开发还是够用的。

不接入地址，只要检测到对 PS2 从机的访问，就直接将扫描到的数据返回。

需要注意的是，实现的 PS2 WishBone 控制器是带有中断输出 INT 的，但是需要向 μOS 系统的汇编代码中写入中断向量寄存表和相应的中断向量处理函数，如果不进行写入就会造成中断输入一直卡死在中断信号位上，使得进程不断重复下台-准备-再上台的循环，从而大幅影响运行速度。这里为了方便，在 Verilog 中设置开关 J16 允许在板上选择是否开启键盘中断。

WISHBONE 状态机如下：



③ 系统前置函数设计：

对于游戏而言，随机数生成器是必不可少的一样工具，它允许程序生成随机地图、随机窗口等等关乎游戏体验的内容。然而在 MIPS 环境下并没有这样一个随机数种子。因此，往系统函数中添加随机数生成系列函数 void randseed(int) 和 int rand() 两个函数，通过线性同余法进行生成和计算。

```
INT32U rand_seed = 1;
void randseed(INT32U seed){
    rand_seed = seed;
}

INT32U rand() {
    rand_seed = rand_seed * 1103515245 + 12345;
    return (INT32U)(rand_seed / 65536) % (G_JMP << 1 + 1);
}
```

④ “星穹铁道 MINI”游戏开发：

“星穹铁道 MINI”的游戏规则如下：

用户进入游戏后游戏立刻开始，用户所控制的光标在游戏自动生成的地图中进行

装 订 线

探索，使用 ASWD 键进行操作，每隔 tick ms（tick 会随着游戏时间的增加不断缩短，从而保证游戏体验）后，用户所处在的星穹铁道都会向前进发。用户需要保持所控制的光标在黑色区域（铁道）内，如果自己主动或者被动地装上白色区域（星穹）则视为游戏失败。在游戏失败后会显示 Game OVer，此时游戏暂停，在此时再按下任意按键即可重新开始游戏。

实验设计的代码如下：

```
INT8U map[G_X][G_Y] = {0};
INT8U t = 14, mx = 1, my = 14;
void Gwrite(INT8U x, INT8U y, INT8U c){
    map[x][y] = c;
    REG8(0x40000000 + x + (y+3) * 128) = c;
}
INT8U absminus(INT8U a, INT8U b){
    return a > b ? a - b: b - a;
}

char last_key = 0;
char keyboard_in(){
    INT32U c = REG32(0x50000000);
    char ch = (c & 0xff) | (c >> 24);
    if(last_key == ch) return 0;
    last_key = ch;
    return ch;
}

void Ginit(){
    INT8U x = 0, y = 0;
     for(x = 0; x < G_X; x++)
        for(y = 0; y < G_Y; y++)
            Gwrite(x, y, absminus(y,t) <= G_WINDOW ? G_EMPTY : G_BLOCK);

    char info[8] = "Time: 000";
    for(x = 0; x < 8; x++)
        REG8(0x40000000 + 4 + x + 128) = info[x];
    for(x = 0; x < 60; x++)
        REG8(0x40000000 + x) = ' ';
    Gwrite(mx, my, G_PERSON);
}

void GTimeUpdate(INT8U time){
    REG8(0x40000000 + 10 + 128) = (time / 100) + '0';
    REG8(0x40000000 + 11 + 128) = ((time / 10) % 10) + '0';
    REG8(0x40000000 + 12 + 128) = time % 10 + '0';
}
```

```
38.
39. INT8U Gupdate(){
40.     INT8U x = 0, y = 0;
41.     for(x = 1; x < G_X; x++)
42.         for(y = 0; y < G_Y; y++){
43.             Gwrite(x - 1, y, map[x][y] == G_PERSON ? G_EMPTY : map[x][y]
    );
44.             if(x == mx + 1 && y == my){
45.                 if(map[x][y] == G_BLOCK)
46.                     return 1;
47.                 Gwrite(mx, my, G_PERSON);
48.             }
49.         }
50.     INT8U r = myrand();
51.     if(t + r > G_JMP + 1 + G_WINDOW && t + r < G_Y - G_JMP - G_WINDOW -
    1)
52.         t = t + r - G_JMP;
53.     for(y = 0; y < G_Y; y++)
54.         Gwrite(95, y, absminus(y,t) <= G_WINDOW ? G_EMPTY : G_BLOCK);
55.     if(map[mx][my] == G_BLOCK)
56.         return 1;
57.
58.     return 0;
59. }
60.
61. INT8U Gmove(INT8U p){
62.     if(p == 1){
63.         if(map[mx][my - 1] == G_BLOCK){
64.             return 1;
65.         } else {
66.             Gwrite(mx, my, G_EMPTY);
67.             Gwrite(mx, my-1, G_PERSON);
68.             my -= 1;
69.         }
70.     }else if(p == 2){
71.         if(map[mx][my + 1] == G_BLOCK){
72.             return 1;
73.         } else {
74.             Gwrite(mx, my, G_EMPTY);
75.             Gwrite(mx, my+1, G_PERSON);
76.             my += 1;
77.         }
78.     } else if(mx > 0 && p == 3){
79.         if(map[mx - 1][my] == G_BLOCK){
80.             return 1;
81.         } else {
82.             Gwrite(mx-1, my, G_EMPTY);
83.             Gwrite(mx, my, G_PERSON);
```

装 订 线

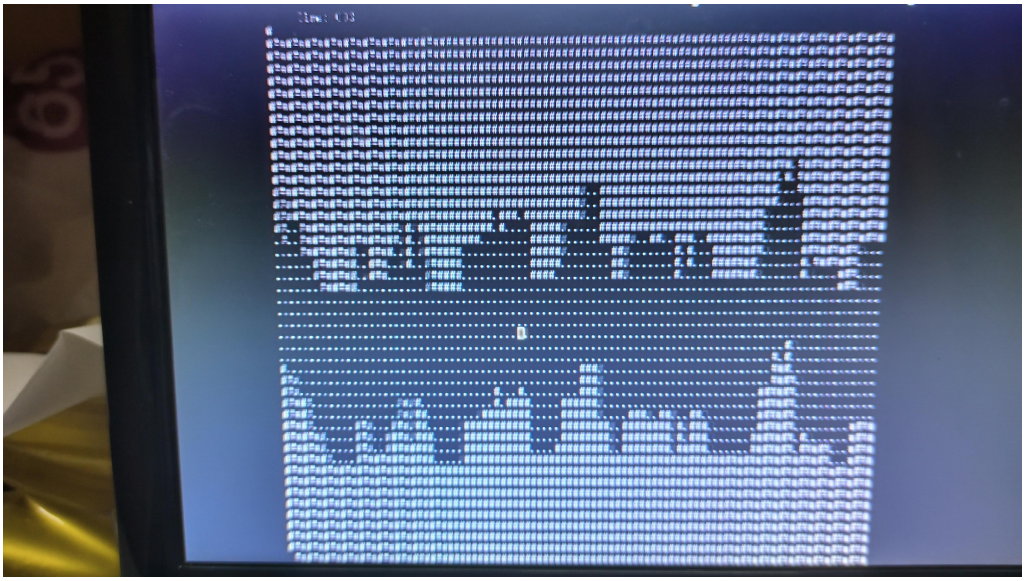
```
84.             mx -= 1;
85.         }
86.     } else if(p == 4){
87.         if(map[mx + 1][my] == G_BLOCK){
88.             return 1;
89.         } else {
90.             Gwrite(mx+1, my, G_EMPTY);
91.             Gwrite(mx, my, G_PERSON);
92.             mx += 1;
93.         }
94.     }
95.     return 0;
96. }
97.
98. void Ggameover(){
99.     INT8U x, y;
100.       for(x = 40; x < 61; x++) // 50
101.           for(y = 20; y < 27; y++) // 23
102.               Gwrite(x, y, ' ');
103.       char show[10] = "Game Over";
104.       for(x = 0; x < 9; x++){
105.           Gwrite(x+50-4, 23, show[x]);
106.       }
107.   }
108.
109.   void  TaskStart (void *pdata)
110.   {
111.       INT32U count = 0;
112.       pdata = pdata;
113.       OSInitTick();
114.       for (;;) {
115.          if(count <= 102)
116.           {
117.            uart_putc(Info[count]);
118.            uart_putc(Info[count+1]);
119.           } else {
120.               break;
121.           }
122.           gpio_out(count);
123.           count=count+2;
124.           OSTimeDly(10);
125.       }
126.       //start
127.       INT8U ch = 0;
128.       INT32U i = 0, tick = 0;
129.       while(1){
130.           mx = 1, my = t;
131.           Ginit();
```

```
132.          for(i = 0;;i++){
133.              ch = keyboard_in() | uart_getc();
134.              if(ch == 'w' || ch == 'W' || ch == 's' || ch == 'S' || c
   h == 'a' || ch == 'A' || ch == 'd' || ch == 'D')
135.                  if(Gmove((ch == 'w' || ch == 'W') ? 1 : (ch == 's' |
   | ch == 'S') ? 2 : (ch == 'a' || ch == 'A') ? 3 : (ch == 'D' || ch == '
   d') ? 4 : 0))
136.                      break;
137.              int c = i / 300;
138.              tick = (5 - (c > 4 ? 4 : c));
139.              if(i % tick == 0)
140.                  if(Gupdate())
141.                      break;
142.              if(i % 100 == 1){
143.                  GTimeUpdate(i / 100);
144.              }
145.              OSTimeDly(1);
146.          }
147.          //GameOver
148.          Ggameover();
149.          for(i = 0;;i++){
150.              ch = uart_getc() || keyboard_in();
151.              if(ch) break;
152.              OSTimeDly(10);
153.          }
154.      }
155.  }
```

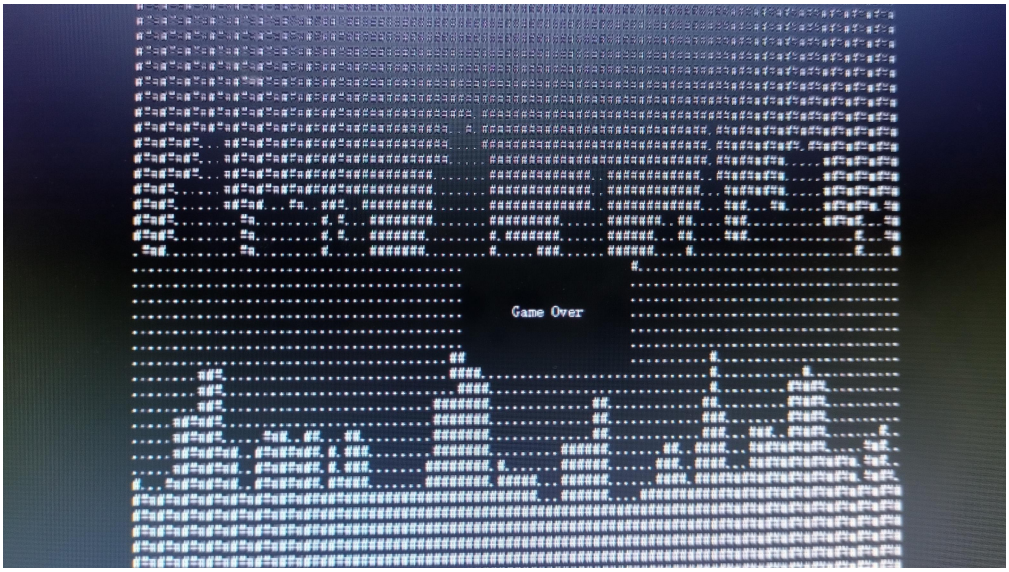
4、实验结果

观察 VGA 图像：

游戏运行时：使用键盘 ADSW 控制光标进行移动，让光标保持在黑色通道内如果撞到了白色墙面就判失败。左上角显示时间，游戏会根据时间的加速而加速。



撞到白色墙壁后游戏结束，显示 GAME OVER 字样：



5、实验总结

本次试验实现了 VGA 和键盘的 Wishbone 驱动，使得 CPU 可以通过对 wishbone 总线的读写控制 VGA 和接受键盘输入的中断和信号内容。这两个模块其实在大二上学期进行数字逻辑开发的时候就已经使用过了，并且同时也实现了一个比较精美的游戏（某种意义上来说，比本次试验实现的游戏要精美许多）。但是相较而言，工作量显然是不可同日而语的，为什么会出现这种现象？一个简单的 flppy bird 游戏在迁移到通用 CPU 上却无法实现全分辨率的输出，只好退而求其次进行字符 CLI 屏下的游戏

模拟。这让我深刻的意识到用硬件实现和软件实现之间的差距：如果一个程序用硬件实现，可以做到效率和速度的最大化，可取而代之的就是通用性的降低，一台硬件设备只能运行一个程序；而如果用软件实现，为了通用化，在时序、协议等等方面浪费了大量的硬件资源，但是其优势就是能够实现最大化通用性能，只要符合 MIPS89 字符集的软件和符合 wishbone 总线的硬件都可以自由的安装在 CPU 上并进行执行。

整体而言，本次计算机系统实验总算是胜利结束了，从第一行 MIPS CPU 的开发开始到完整实现的系统，开发时间差不多在 200 小时左右，贯通了数字逻辑、计算机组成原理和系统结构、编译原理和操作系统等等课程，进一步强化了我对硬件和软硬件结合体系的乐趣，是一门非常有乐趣、有挑战性、收获满满的课！