一、实验内容

1. 项目名称

基于 VS1003B、支持 PS/2 协议的键盘和 VGA 的超级马里奥游戏

2. 项目内容概括

基于 VS1003B、支持 PS/2 协议的键盘和 VGA，模拟任天堂 FC 版本的超级马里奥游戏的第一关，在开发板上实现了一个简化版本的复刻。下板后，首先在 VGA 屏幕上显示出开始界面（如图 1），同时播放该游戏的主题曲；此时游戏开始，即控制马里奥越过障碍并且撞击带有问号的方块获得积分，在七段数码管上显示（如图 2）；到达终点后游戏结束。

图 1

图 2

3. 操作与规则说明

FPGA 开发板连接 VGA，PS/2 键盘和耳机（或音响），游戏开始前，玩家需要通过开发板上的拨码开关重置游戏以开始游戏。游戏开始，玩家需要通过 PS/2 键盘上的“W”、“S”和“D”键，分别控制马里奥的跳跃，左移和右移。马里奥不能穿过管道、置空方块和阶梯，当碰到带有问号的方块时，游戏得分会增加 100，开发板上的数码管实时显示当前得分。通过开发板上最右侧的拨码开关可以实现全部数据复位功能。

4. 器件简介：

①Nexys4 DDR Artix-7——由 Xilinx 公司开发出的一款现场可编程门阵列（FPGA）开发板

②VGA——VGA(Video Graphics Array)是 IBM 在 1987 年随 PS/2 机一起推出的一种视频传输标准，具有分辨率高、显示速率快、颜色丰富等优点；

③豹击蓝键键盘——使用 PS/2 协议的一款有线键盘。

④小米方盒子蓝牙音箱 2——用于输出音频的设备。

二、马里奥游戏数字系统总框图

1. 系统组成总框图

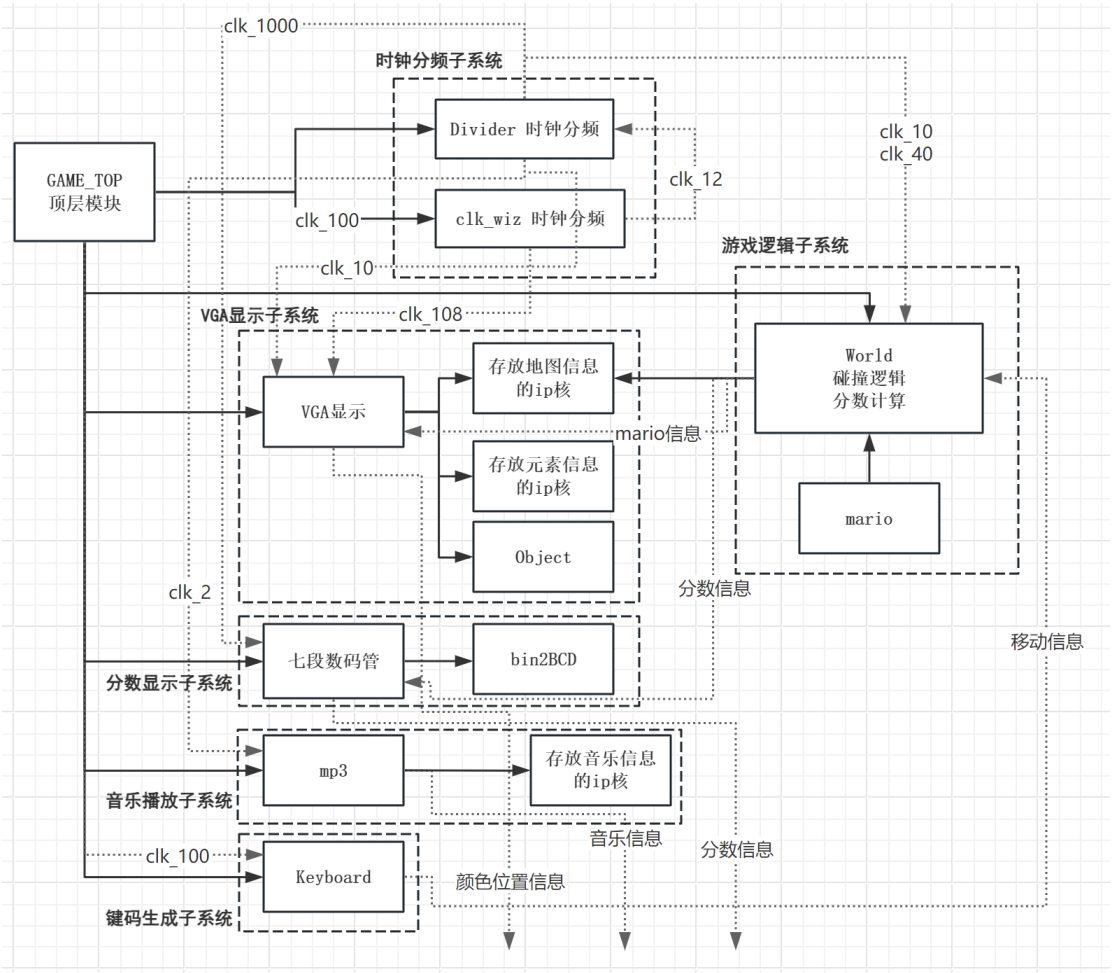

图 3 系统组成总框图

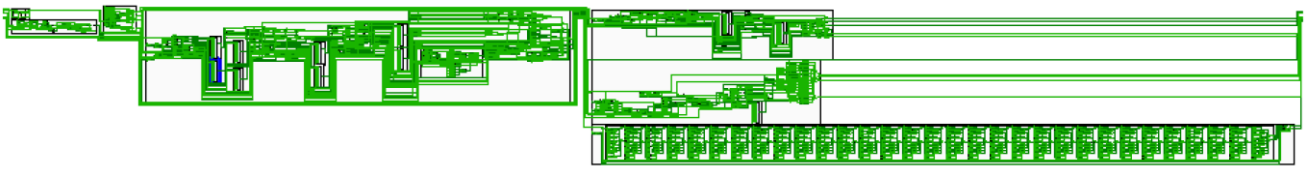

图 4 RTL 图

2. 子系统功能简介

①控制器子系统 GAME_TOP

控制器子系统负责控制整个马里奥游戏系统中各个子系统的输入输出，与数据交互，协调各子系统有条不紊的工作。

②时钟分频子系统

时钟分频子系统将 100MHz 的系统时钟分频为 108MHz、12MHz、10Hz、40Hz、2MHz、1000Hz 以供其余模块后续使用。

③VGA 显示子系统

VGA 显示子系统从 ROM 中读取 RGB444 的像素信息，在控制器子系统的控制下进行地图和人物显示，坐标的索引和转换。

④分数显示子系统

七段数码管显示子系统接收来自游戏逻辑子系统的分数，以十进制形式在七段数码管中显示呈现。

⑤音乐播放子系统

音乐播放子系统从 ROM 中读取 mid 文件信息，并通过状态机进行 VS1003B 的硬件复位和软件复位，其次进行寄存器的配置，调整播放参数，以便更好地播放音乐。

⑥键码生成子系统

键码生成子系统接收来自 PS/2 键盘的信息，将键盘返回的有效键值转换为按键字母对应的 ASCII 码，供游戏逻辑子系统后续使用。

⑦游戏逻辑子系统

游戏逻辑子系统从键码生成子系统中获取人物控制信息。首先根据移动信息，进行人物动画的关键帧切换，其次对人物进行坐标变换，将坐标变换信息传入 VGA 显示子系统进行显示；之后计算增加分数，将游戏得分传入分数显示子系统进行显示。

三、系统控制器设计

1. ASM 流程图

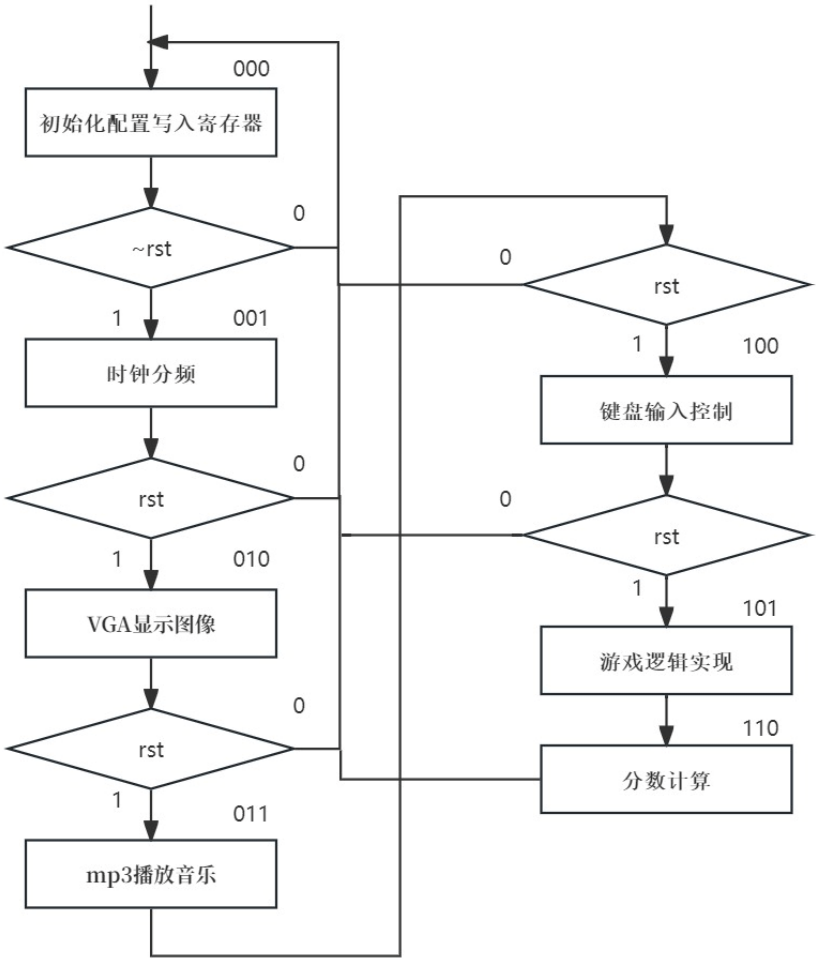

图 5

2. 状态转移真值表

PS（现态）			NS（次态）			转换条件
C	B	A	C	B	A	
0	0	0	0	0	0	rst
			0	0	1	~rst
0	0	1	0	0	0	~rst

			0	1	0	rst
0	1	0	0	0	0	~rst
			0	1	1	rst
0	1	1	0	0	0	~rst
			1	0	0	rst
1	0	0	0	0	0	~rst
			1	0	1	rst
1	0	1	1	1	0	
1	1	0	0	0	0	

3. 次态激励函数表达式

$$A^{n+1} = (\bar{A}\bar{B}\bar{C}) \cdot \overline{rst} + (\bar{A}B\bar{C} + \bar{A}\bar{B}C) \cdot rst$$

$$B^{n+1} = (A\bar{B}\bar{C} + \bar{A}B\bar{C}) \cdot rst + A\bar{B}C$$

$$C^{n+1} = (AB\bar{C} + \bar{A}\bar{B}C) \cdot rst + A\bar{B}C$$

4. 控制命令逻辑表达式

$$INIT = \bar{A}\bar{B}\bar{C}$$

$$DIVIDER = A\bar{B}\bar{C}$$

$$VGA = \bar{A}B\bar{C}$$

$$MP3 = AB\bar{C}$$

$$KEY = \bar{A}\bar{B}C$$

$$WORLD = A\bar{B}C$$

$$SCORE = \bar{A}BC$$

5. logisim 控制器逻辑方案图

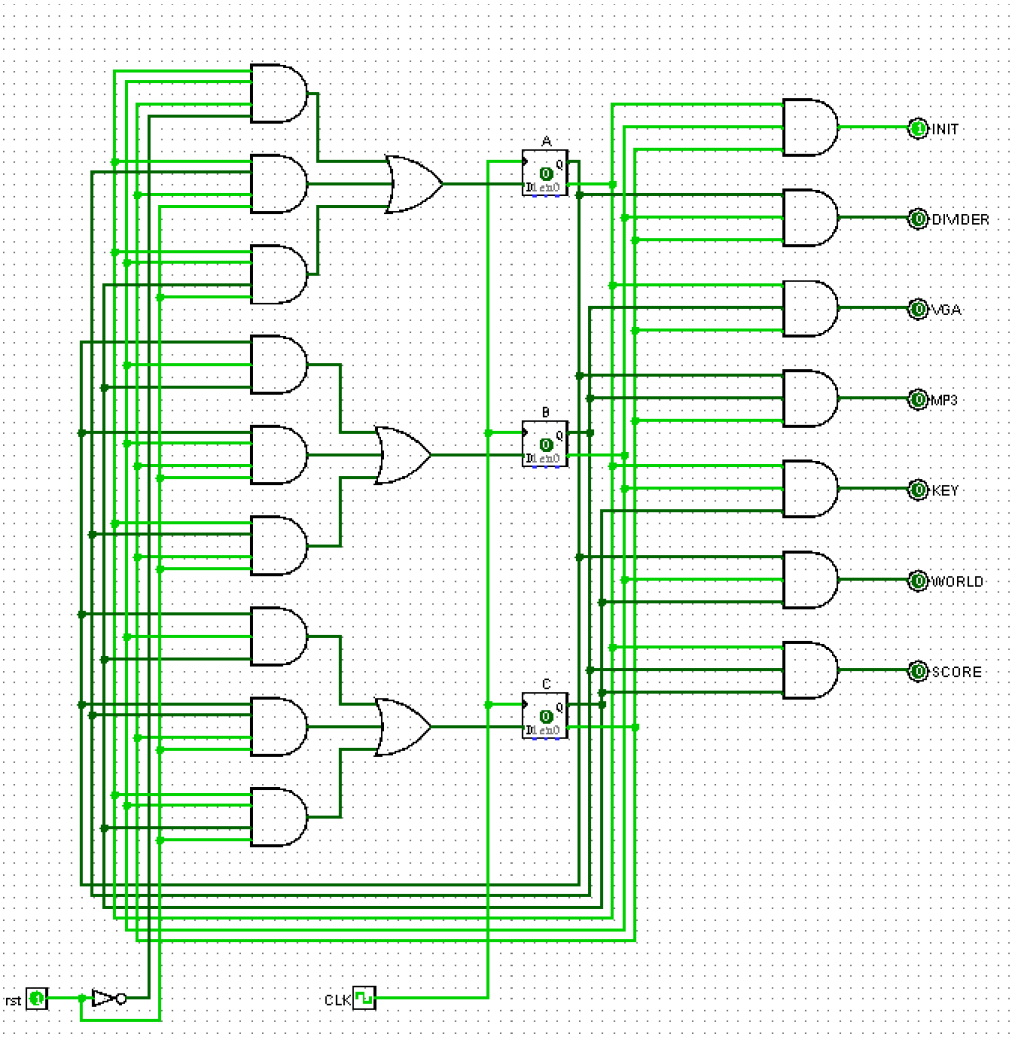

图 6

四、子系统模块建模

本系统划分了 6 个子系统模块，共包括：时钟分频模块、VGA 显示模块、分数显示模块、音乐播放模块、键码生成模块和游戏逻辑模块。下面将分别具体介绍之。

1. 时钟分频模块 1（clk_wiz_0）

（1）描述：利用 Vivado 自带的 IP 核：Clocking Wizard，实例化生成了一个将开发板 100MHz 的系统时钟分频成多种频率时钟的 PLL 时钟分频器。其中，108MHz 的时钟分配给 VGA 显示模块，用于驱动 VGA 显示器；12MHz 的时钟分配给 Divider 模块，即时钟分频模块 2，用于时钟的进一步分频。

（2）功能框图：

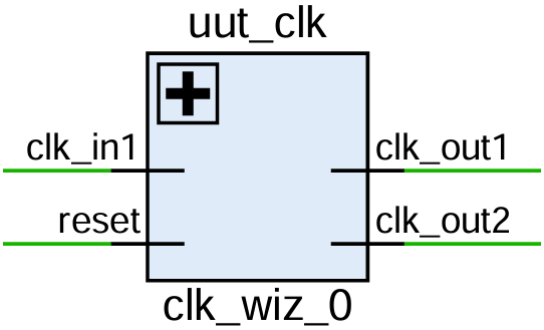

图 7

（3）接口信号定义：

接口名称	接口属性	接口描述
clk_in1	input	开发板系统时钟，100MHz
reset	input	复位信号，高电平有效
clk_out1	output	VGA 驱动显示时钟，108MHz
clk_out2	output	Divider 分频时钟，12MHz

（4）实现思路：

该模块为实例化 IP 核的模块。

2. 时钟分频模块 2（Divider）

（1）描述：实例化生成将开发板 12MHz 的时钟分频成多种频率时钟的时钟分频器。其中，2MHz 的时钟分配给音乐播放模块，用于驱动 VS1003B；10Hz 的时钟分配给 VGA 显示模块，用于视野的转换，即地图的移动，同时分配给游戏逻辑模块，用于人物动作状态的切换；40Hz 的时钟分配给游戏逻辑模块，用于碰撞检测；1000Hz 的时钟分配给分数显示模块，用于实时显示分数。

（2）功能框图：

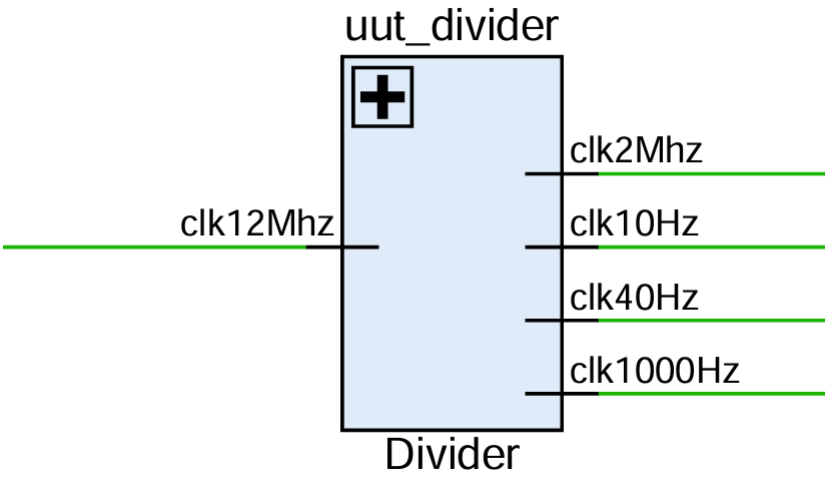

图 8

（3）接口信号定义：

接口名称	接口属性	接口描述
clk12Mhz	input	待分频的时钟，12MHz
clk2Mhz	output	音乐播放模块驱动时钟，2MHz
clk10Hz	output	VGA 显示视野转换时钟、游戏逻辑模块人物动作状态切换时钟，10Hz
clk40Hz	output	游戏逻辑模块碰撞检测时钟，40Hz
clk1000Hz	output	分数显示模块实时显示模块，1000Hz

（4）实现思路：

定义一个计数器，根据分频频率计算计数器计数周期，一旦计数达到一个周期，则将分频得到的时钟信号取反，从而模拟出分频效果。

3. VGA 显示模块

（1）描述：该模块为根据输入的像素数据、行场驱动与时钟驱动，来进行对数据的 VGA 显示。该模块负责以 1280*1024 的分辨率输出 VGA 时序逻辑以供屏幕显示。VGA 时序参数等具体参数在代码段参数部分列出。该模块以 108MHz 的时钟输入一行像素输出后拉高行同步信号以同步数据，场时序同理。

（2）功能框图：

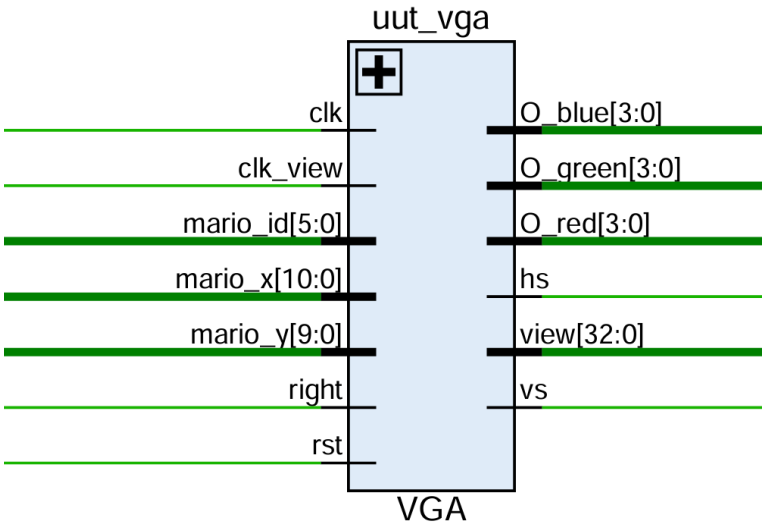

图 8

（3）接口信号定义：

接口名称	接口属性	接口描述
clk	input	输入的驱动时钟，108MHz
clk_view	input	用于切换视野的时钟，10Hz
mario_id	input	需要显示的游戏人物的元素 id
mario_x	input	当前游戏人物左上角的横坐标
mario_y	input	当前游戏人物左上角的纵坐标
right	input	右移信号
rst	input	复位信号，低电平有效

O_blue	output	RGB 蓝颜色信号
O_red	output	RGB 红颜色信号
O_green	output	RGB 绿颜色信号
view	output	当前视野的横坐标
hs	output	行同步信号
vs	output	列同步信号

（4）实现思路：

首先查询有关资料，得到 VGA 的行时序常数和场时序常数，并且根据这些常数定义行时序计数器和列时序计数器。（如图 9）

VGA 常用分辨率时序参数

显示模式	时钟	行时序参数(单位：像素)					列时序参数(单位：行)				
	/MHz	a	b	c	d	e	f	g	h	i	k
640x480@60Hz	25.175	96	48	640	16	800	2	33	480	10	525
800x600@60Hz	40	128	88	800	40	1056	4	23	600	1	623
1024x768@60Hz	65	136	160	1024	24	1344	6	29	768	3	806
1280x720@60Hz	74.25	40	220	1280	110	1650	5	20	720	5	750
1280x1024@60Hz	108	112	248	1280	48	1688	3	38	1024	1	1066
1920x1080@60Hz	148.5	44	148	1920	88	2200	5	36	1080	4	1125

图 9

其次，根据 VGA 的详细时序产生行时序和场时序。并定义有效区域标志，当有效区域信号为高时，颜色信息才会显示到屏幕上。（如图 10）

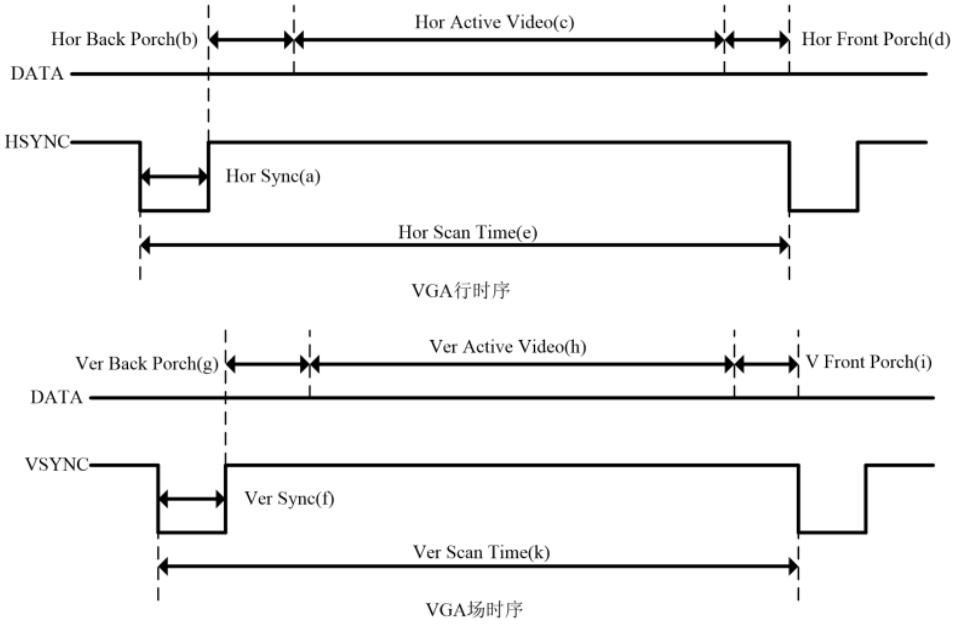

图 10

在设置好 VGA 的各个参数后，我们将需要使用的游戏图片素材导入到 ROM 中。游戏素材来源于网址 https://www.mariomayhem.com/downloads/sprites/。考虑到开发板的 ROM 空间有限，我们截取需要使用到的游戏素材，在 windows 画图软件中进行编辑，每一种不同的方格仅出现一次（如图 11），当显示游戏画面时，将根据素材的索引计算偏移地址查找到该素材。

图 11

考虑到原始图片素材大小过小，在 VGA 上进行显示，难以看清；因此，我们使用 C 语言将 bmp 图像等比例放大，得到 640×320 像素的 bmp 图像，程序代码见附录。其次，我们将 bmp 图像通过 matlab 转化为 COE 文件，程序代码见附录。

获得素材之后，我们将根据任天堂版本的游戏地图（如图 12）来生成游戏地图文本文件（如图 13），按照 coe 的格式编辑好后导入 IP 核。

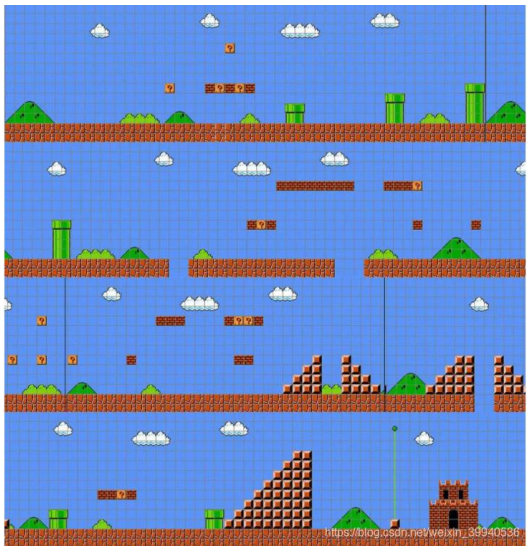

图 12

```
memory_initialization_radix=10;
memory_initialization_vector=
40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 20, 21, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
40, 40, 40, 40, 40, 40, 40, 40, 20, 21, 40, 40, 40, 40, 40, 40, 40, 40, 40, 30, 31, 40, 40, 40, 40, 40, 40, 40, 20, 21, 40,
40, 40, 40, 40, 40, 40, 40, 40, 30, 31, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 30, 31, 40,
40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 0,  40, 40, 40, 40, 40, 40, 40, 40,
40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 0,  40, 40, 40, 2,  0,  2,  0,  2,  40, 40, 40, 40, 40, 40,
40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40,
40, 40, 15, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 15, 40, 40, 40, 40, 40, 40, 40, 40, 40, 40, 38, 39, 40,
40, 24, 25, 26, 40, 40, 40, 40, 40, 40, 40, 40, 40, 22, 23, 40, 24, 25, 26, 40, 40, 40, 40, 22, 23, 40, 40, 40, 48, 49, 40,
3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,  3,
```

图 13

整个 VGA 显示模块含有子模块 Object，该模块将素材的 id 转换为素材在 ROM 中的偏移地址。最后，我们建立当前显示像素与地图文件中所存储的 id 和 ROM 中存储的素材的对应关系式，读出 RGB565 数据，并只选取高位 RGB444 作为显示像素的 RGB 值。在超出游戏范围的屏幕显示区域，我们默认显示黑色。

VGA 显示模块还会获取当前游戏人物的坐标值，当显示坐标与人物坐标重合时，不再读取地图中的信息，而读取当前人物的 id，在屏幕上显示出该人物。

最后 10Hz 的时钟驱动 VGA 显示模块，进行视野的转换。也就是说，当人物超过屏幕的 1/2 时，将视野向右移动 8 个像素。

4. 分数显示模块

（1）描述：该模块由 1000Hz 的时钟驱动，输入 32 位二进制数，代表当前游戏得分。该模块将实现二进制到十进制的转换，并且选择相应的数码管进行输出。

（2）功能框图：

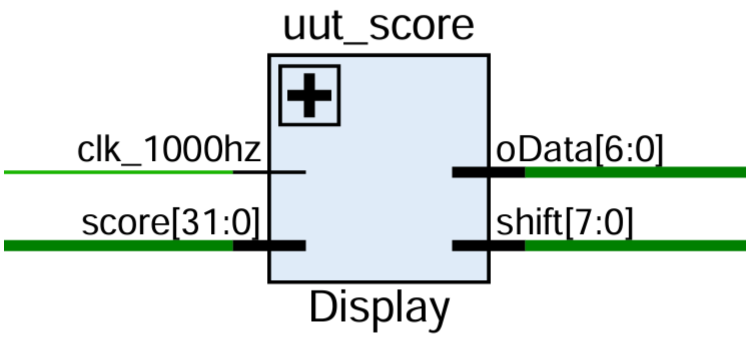

图 14

（3）接口信号定义：

接口名称	接口属性	接口描述
clk_1000Hz	input	驱动分数显示模块的 1000Hz 时钟
score	input	当前得分，二进制形式表示
oData	output	数码管的显示信号
shift	output	数码管的片选输出

（4）实现思路：

首先将二进制的游戏得分转化为 8421BCD 码。BCD 码范围在 0000~1001 之间，满 10 进位，只能表示十进制数 0~9。当二进制数 ＞0100b， 即 ＞4 时，左移以后就 ≥ 1010，即

≥ 10，对应的 BCD 码就需要表示个位和十位，对于一个 4 位的二进制数，先输入的高 3 位在 ＞4 的时候，要进行处理，使得最低位输入进来后，表示十位的 BCD 码为 0001。

因此采用加 3 移位法，在最低位输入前，如果高 3 位的 abc＞4 时，对其加上 3 最低位 d 输入，使得加过 3 的高 3 位整体左移一位。

这相当于（abc + 0011）*2+d，即 abc*2 + 6 + d，超过了 4 位 2 进制数表示的范围， 向更高位进一位，此时表示十位的 BCD 码为 0001。每次当新的一位输入前，都需要对连续 4bit 数据判断大小并进行加 3 移位处理。

最后，每次选择一个数码管进行输出，即可实现游戏得分的实时显示。

5. 音乐播放模块

（1）描述：该模块由 2MHz 的时钟驱动，输入开始播放信号与音乐文件信息，该模块将实现 VS1003B 的驱动，包括软件复位、硬件复位、寄存器配置并播放音乐。

（2）功能框图：

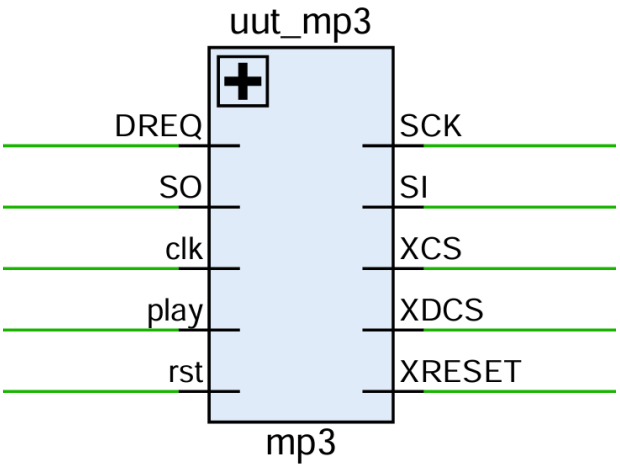

图 15

（3）接口信号定义：

接口名称	接口属性	接口描述
clk	input	驱动音乐播放模块的 2MHz 时钟
rst	input	复位信号，低电平有效
play	input	开始播放请求
SO	input	SPI 通信接口，串行输出
DREQ	input	数据请求，高电平时可传输数据
SCK	output	SPI 通信接口，串行时钟输入
SI	output	SPI 通信接口，串行数据输入
XCS	output	低电平有效片选输入
XDCS	output	低电平有效片选输入
XRESET	output	硬件复位，低电平有效

（4）实现思路：

首先在 vivado 中载入音乐文件。将音乐转化为 coe 文件，注意，音乐文件必须是 mid 格式且为单音轨。我们在 https://themushroomkingdom.net/media/smb/mid 上下载 mid 文件，并使用 Ableton Live 11 Suite 音乐制作软件将多余音轨删除（如图 16）

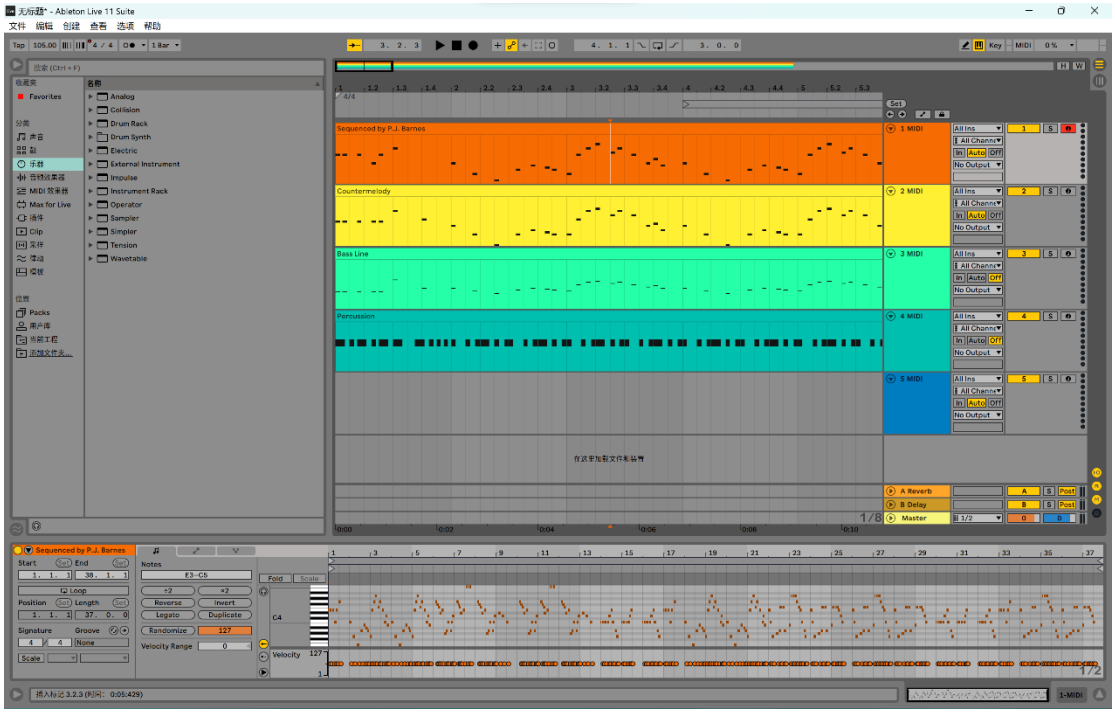

图 16

两个逗号之间有 32 位数据，用来初始化一个简单单端口 ROM，设置为写入宽度为 32 位，读出宽度为 32 位。

其次初始化 MP3。我们先对 VS1003B 进行复位，包括硬件复位和软件复位，我们对 VS1003B 相关寄存器进行配置。查阅相关资料可知 MP3 的 SCI 写入时序（如图 17）

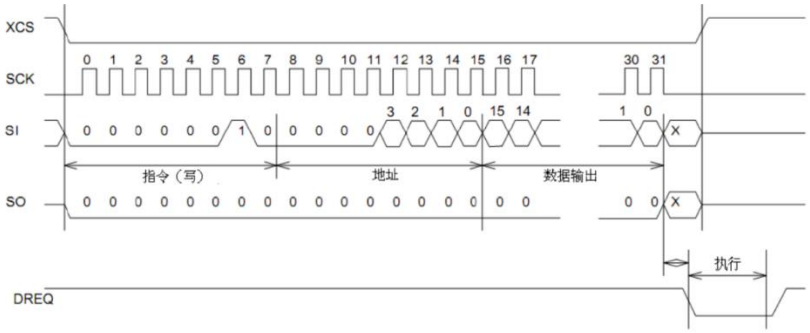

图 17

由图可知，使 XCS=0，SCK 模拟一个时钟。当 DREQ 为高电平时，向 SI 逐位传值。

根据 MODE 寄存器中各位的功能，我们设置传入指令 32'h02_00_08_04，进行软件复位，并且打开 VS1003B 的本地模式（如图 18）。

位	名称	功能	值	说明
0	SM_DIFF	差分	0 1	正常的相同音频 左通道想相反
1	SM_LAYER12	允许 MPEG layers I&II 解码	0 1	不允许 允许
2	SM_RESET	软件复位	0 1	不复位 复位
3	SM_CANCEL	取消当前文件的解码	0 1	不取消 取消
4	SM_EARSPEAKER_LO	EarSpeaker 低位设定	0 1	关闭 激活
5	SM_TESTS	允许 SDI 测试	0 1	不允许 允许
6	SM_STREAM	流模式	0 1	不是 是
7	SM_EARSPEAKER_HI	EarSpeaker 高位设定	0 1	关闭 激活
8	SM_DACT	DCLK 的有效边沿	0 1	上升沿 下降沿
9	SM_SDIORD	SDI 位顺序	0 1	MSB 在前 MSB 在后
10	SM_SDISHARE	SM_SDISHARE	0 1	不共享 共享
11	SM_SDINEW	VS1002 本地 SPI 模式	0 1	非本地模式 本地模式（新模式）
12	SM_ADPCM	ADPCM 录音激活	0 1	不激活 激活
13	SM_ADPCM_HP	ADPCM 高通滤波允许	0 1	不允许 允许
14	SM_LINE_IN	ADPCM 音源选择	0 1	麦克风 线路输入
15	SM_CLK_RANGE	SM_CLK_RANGE	0 1	12…13Mhz 24…26Mhz

图 18

同样地，我们设置 BASS，CLOKF，VOL 寄存器。

配置完毕后，即开始播放音乐，需要我们发送音频数据，根据 SCI 读时序（如图 19）

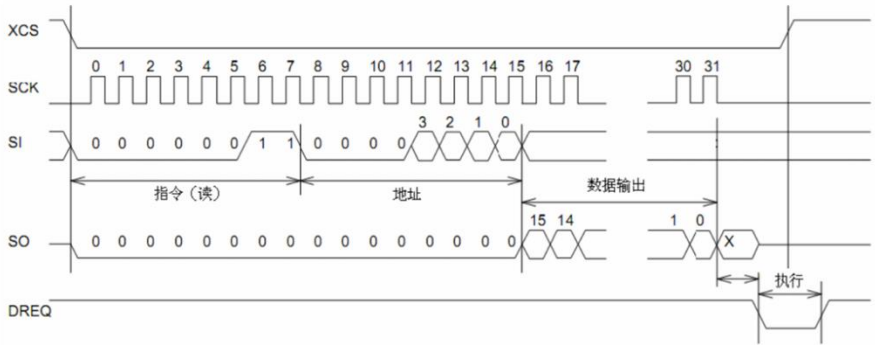

图 19

通过先拉低 XCS，然后发送读指令，随后再发送一个地址，把音频数据从 ROM 里 32 位位宽地读出来。最后，我们在 SO 线上就可以读到输出的数据，同时 SI 上的数据将被忽略。

6. 键码生成模块

（1）描述：该模块由 1000Hz 的时钟驱动，接收从键盘发送过来的信号，将键盘键码转化为 ASCII 码。

（2）功能框图：

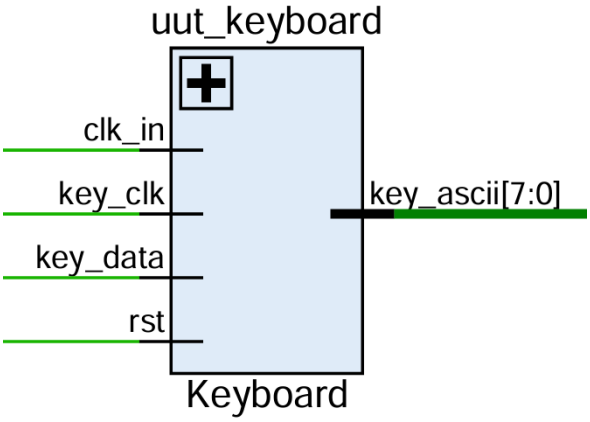

图 20

（3）接口信号定义：

接口名称	接口属性	接口描述
clk_in	input	驱动键码生成模块的 1000Hz 时钟
rst	input	复位信号，低电平有效
key_clk	input	PS2 键盘时钟输入
key_data	input	PS2 键盘数据输入
key_ascii	output	按键键值对应的 ASCII 编码

（4）实现思路：

开发板与键盘的通信满足 PS2 通信协议。当按下一个键时，键盘送出相应键的扫描码；松开时，也送出扫描码。键盘各个按键对应的扫描码如下图所示（本实验用到的键用红框标出）

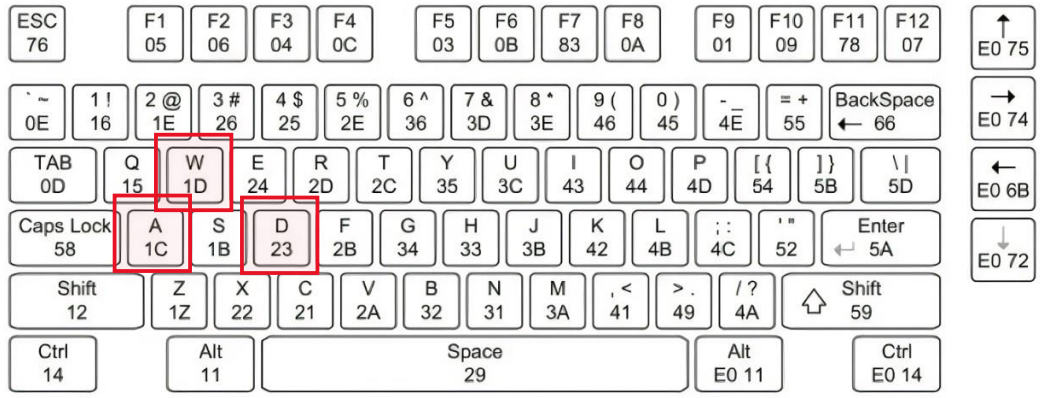

图 21

根据通码和断码判定按键的当前是按下还是松开，收到断码（8'hf0）表示按键松开，下一个数据为断码，设置断码标示为 1，当断码标示为 0 时，表示当前数据为按下数据，输出键值并设置按下标示为 1，当断码标示为 1 时，标示当前数据为松开数据，断码标示和按下标示都清 0。

PS/2 键盘向主机发送的数据为：

1 个起始位（总是逻辑 0）、8 个数据位（LSB 低位在前）、1 个奇偶校验位（采用奇校验）和 1 个停止位（总是逻辑 1）。

时钟高电平时键盘开始送数据，时钟低电平时主机开始读数据，时序如下图

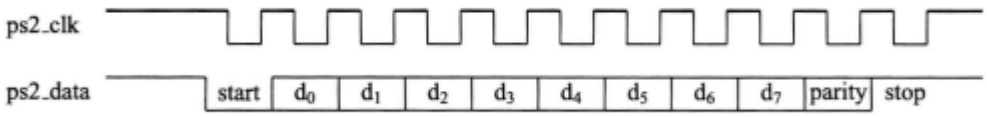

图 22

最后将键盘返回的有效键值转换为按键字母对应的 ASCII 码。

7. 游戏逻辑模块

（1）描述：该模块输入游戏控制信号（人物动作信号）与当前视野位置信息，该模块将实现游戏的碰撞检测、人物动作关键帧的切换、游戏得分的计算和人物的移动。

（2）功能框图：

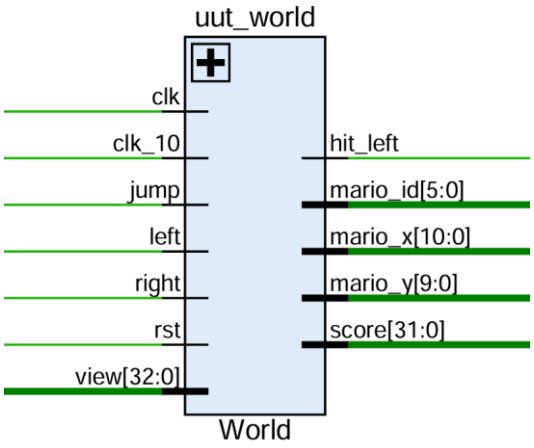

图 23

（3）接口信号定义：

接口名称	接口属性	接口描述
clk	input	实现碰撞检测的 40Hz 时钟
clk_10	input	实现人物状态机的 10Hz 时钟
jump	input	跳跃信号
left	input	左移信号
right	input	右移信号
rst	input	复位信号，低电平有效
view	input	当前视野位置信息
hit_left	output	是否碰撞物体左边缘信号
mario_id	output	当前人物素材的 id
mario_x	output	当前游戏人物左上角的横坐标
mario_y	output	当前游戏人物左上角的纵坐标
score	output	当前游戏的得分

（4）实现思路：

首先我们进行碰撞检测。在该游戏中，碰撞检测的逻辑为：在每一次时钟上升沿到来时，判断人物方框上下左右共八个点的背景 id 是否为碰撞物体 id，通过与 VGA 显示类似的坐标变换原理实现判断。如果是，则置相应标志位为 1，大致示意图如下。

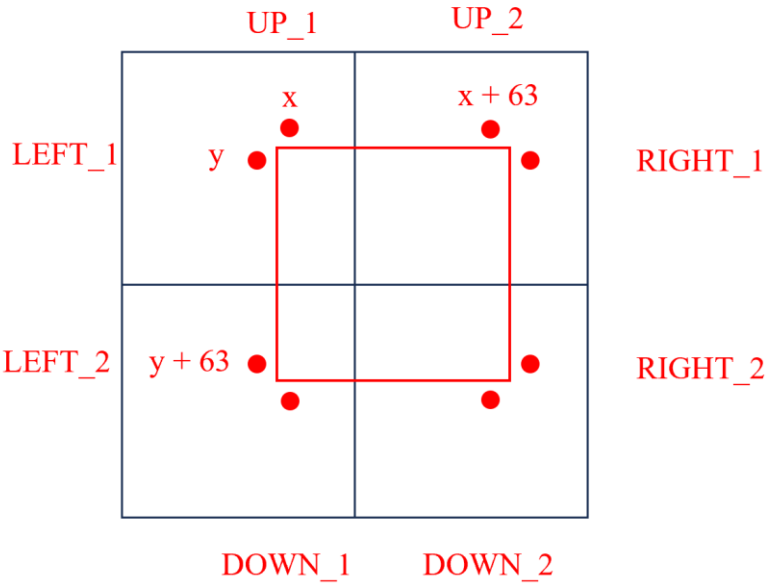

图 24

其次我们需要进行走路帧和跳跃帧的切换，写在子模块 Mario 中。走路与跳跃的状态机的状态转换图如图 25、26、27 所示。

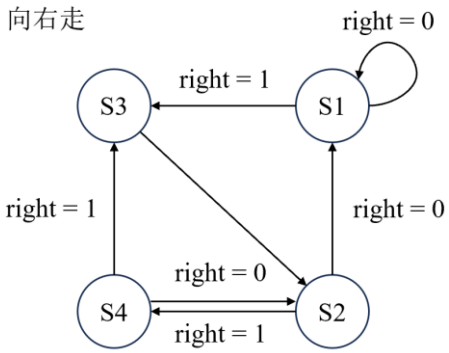

图 25

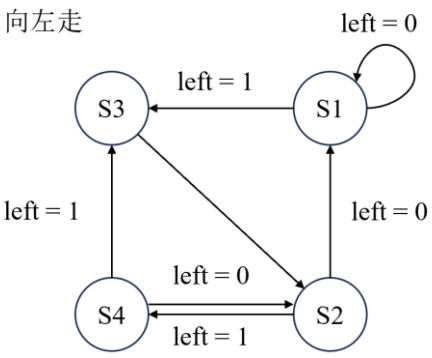

图 26

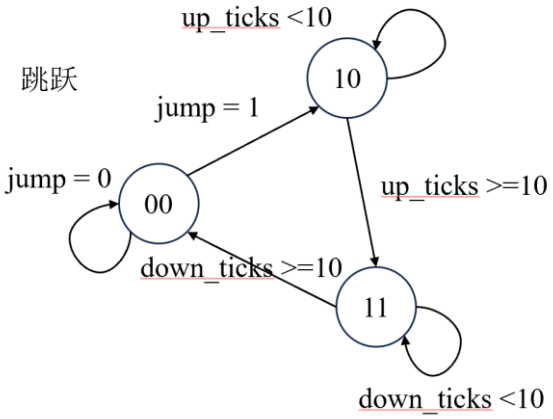

图 27

根据状态的变换，进行人物朝向的切换和 id 的切换。结合状态机和碰撞检测，我们可以在碰到方块下边沿时实现游戏得分的增加。

对于游戏人物，移动的逻辑略微复杂。首先，系统将检测游戏人物是否处于跳跃状态（jump_state）。若是，则根据上移或下降状态进行 y 坐标的处理。若游戏人物处于上升状态（jump_state = 10）且发生了碰撞，或者是上升计数（jump_ticks）达到了给定的目标值，则将上升状态改为下降状态（jump_state＝11），之后游戏人物将发生下降。同时，若游戏人物处于上升状态，且与上方某个未知箱子发生了碰撞，则实现加分。

当游戏人物处于下降状态时，系统将进行下方物体的碰撞检测。若发生了碰撞，则将游戏人物的 jump_state 变为 00，从而将游戏人物跳跃状态归位。

对游戏人物下方物体的碰撞检测是一直进行的。即使游戏人物不处于下降状态，系统能会判断下方是否存在支撑物。若否，则将 jump_state 置为 11，实现游戏人物的掉落处理。

对于游戏人物的左右移动同样是进行碰撞检测。倘若不存在障碍物，则游戏人物的 x 坐标发生了改变，实现左右移动。

五、测试模块建模

以下 tb 模块代码均为在各个子系统模块未连接时的单独测试模块功能的代码，各个模块连接综合后的总系统暂无 tb 模块。

1. VGA 测试模块

首先模拟一个系统时钟，每隔 50ns 取反。对于复位信号，先设置为低电平有效，一段时间后拉高该信号。其次，模拟人物操作信息，在这里，我们将人物的状态在向右走与停止状态反复切换。仿真后观察 VGA 输出的 RGB444 信号。

2. World 主世界测试模块

首先模拟一个系统时钟，每隔 50ns 取反。对于复位信号，先设置为低电平有效，一段时间后拉高该信号。其次，模拟出人物状态变换的时钟，时钟周期为 0.1s。我们设置视野固定在 640 像素的位置，并且使人物状态在向右走与跳跃之间反复切换。仿真后观察人物的坐标变换信息。

3. mario 状态机测试模块

首先模拟一个系统时钟，每隔 1ns 取反。对于复位信号，先设置为低电平有效，一段时间后拉高该信号。其次，模拟出人物状态变换的时钟，每隔 50ns 取反。在这里，我们将人物的状态在向右走与停止状态反复切换。仿真后观察人物的 id 变换信息。

六、实验结果

1. VGA 测试模块仿真波形图

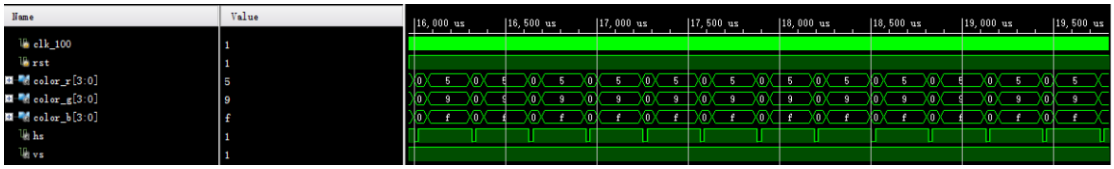

图 28

2. VGA 与 World 联合测试模块仿真波形图

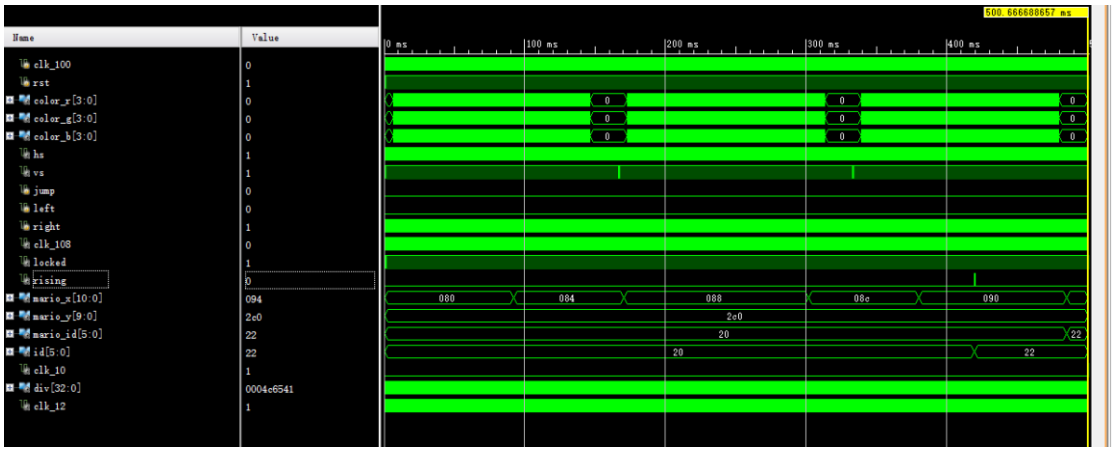

图 29

3. World 测试模块仿真波形图

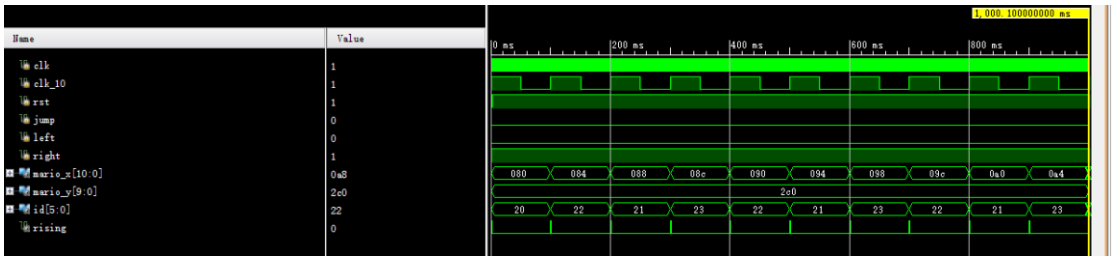

图 30

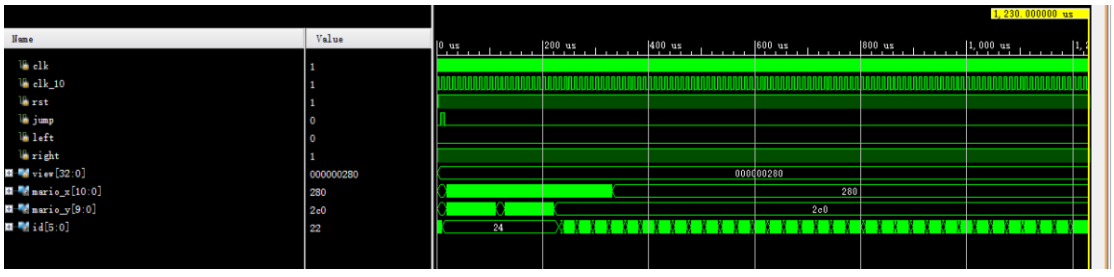

图 31

4. mario 测试模块仿真波形图

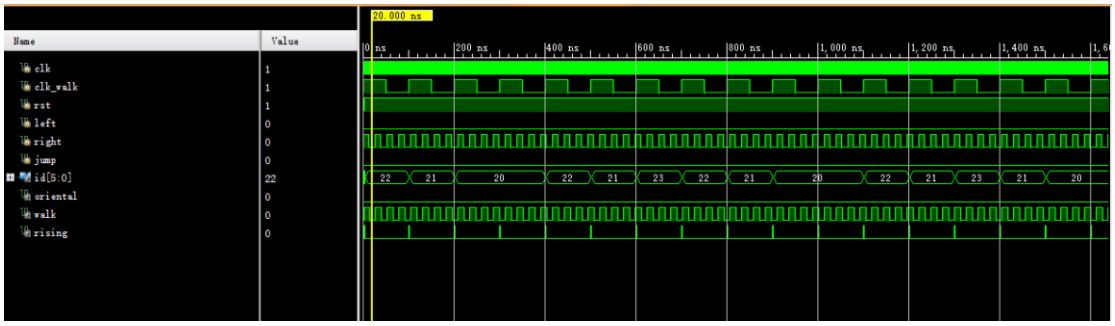

图 32

5. VGA 显示模块下板结果图

图 32

图 33

6. GAME_TOP 模块下板结果图

图 34

七、结论

本次实验中，研究了 VGA、mp3、PS/2 键盘以及七段数码管的时序及控制方法，通过调用 ip 核存储图片、音乐等数据，并编写复杂的控制逻辑，实现了一个较为完备的马里奥小游戏。各个子模块均能够按照预期协调正常运作，数字系统工作正常。在方向键的操作下，游戏人物可以进行正常的移动与碰撞，各动画显示流畅，游戏体验感较好。

八、附录

1. mid 文件转 coe 的 c++代码

```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;

int usage()
{
    cerr << "文件名以下形式均可以:" << endl;
    cerr << "    " << "a.txt                    : 不带路径形式" << endl;
    cerr << "    " << "..\\data\\b.dat            : 相对路径形式" << endl;
    cerr << "    " << "C:\\Windows\\System32\\c.dat : 绝对相对路径形式" << endl;

    return 0;
}

int main(int argc, char** argv)
{
    //以二进制方式读取
    usage();
```

```
    cerr << "请输入文件名 : ";
    char name[100];
    cin >> name;

    ifstream in(name, ios::in | ios::binary);
    if (in.is_open() == false) {
        cout << "输入文件" << name << "打开失败!" << endl;
        return -1;
    }

    char str[16] = { 0 };
    //先获取文件大小，移动文件指针
    in.seekg(0, ios::end);
    int size = (int)(in.tellg());
    in.seekg(0, ios::beg); // 复原

    cout << "memory_initialization_radix=16;\n";
    cout << "memory_initialization_vector=\n";

    unsigned char ch;
    for (int i = 1; i <= size; i++) {

        ch = in.get();

        cout << setiosflags(ios::uppercase);
        cout << hex << setw(2) << setfill('0') << int(ch);
        cout << setfill(' ');
        cout << resetiosflags(ios::uppercase);

        if (i % 4 == 0)
            cout << ',' << endl;
    }

    in.close();
    return 0;
}
```

2. bmp 图像等比例放大 c 语言程序

```
#define _CRT_SECURE_NO_WARNINGS
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
```

```
typedef unsigned char BYTE;
typedef unsigned short WORD;
typedef unsigned int DWORD;
typedef unsigned long   LDWORD;
typedef long LONG;

#pragma pack(1)/* 必须在结构体定义之前使用,这是为了让结构体中各成员按1字节对齐*/
typedef struct tagBITMAPFILEHEADER
{
    WORD bfType; //保存图片类型。 'BM'
    DWORD bfSize; //位图文件的大小，以字节为单位（3-6字节，低位在前）
    WORD bfReserved1;//位图文件保留字，必须为0(7-8字节）
    WORD bfReserved2;//位图文件保留字，必须为0(9-10字节）
    DWORD bfOffBits;   //RGB数据偏移地址,位图数据的起始位置，以相对于位图（11-14字节，低位在前）
}BITMAPFILEHEADER;

typedef struct tagBITMAPINFOHEADER
{
    DWORD biSize;   //本结构所占用字节数（15-18字节）
    DWORD biWidth;  //位图的宽度，以像素为单位（19-22字节）
    DWORD biHeight; //位图的高度，以像素为单位（23-26字节）
    WORD biPlanes;  //目标设备的级别，必须为1(27-28字节）
    WORD biBitCount;//每个像素所需的位数，必须是1（双色）（29-30字节）,4(16色），8(256色）16(高彩色)或24（真彩色）之一

    DWORD biCompression;//位图压缩类型，必须是0（不压缩），（31-34字节）
    //1(BI_RLE8压缩类型）或2(BI_RLE4压缩类型）之一

    DWORD biSizeImage;  //位图的大小(其中包含了为了补齐行数是4的倍数而添加的空字节)，以字节为单位（35-38字节）

    DWORD biXPelsPerMeter;//位图水平分辨率，每米像素数（39-42字节）
    DWORD biYPelsPerMeter;//位图垂直分辨率，每米像素数（43-46字节)
    DWORD biClrUsed;//位图实际使用的颜色表中的颜色数（47-50字节）
    DWORD biClrImportant;   //位图显示过程中重要的颜色数（51-54字节）
}BITMAPINFOHEADER;

int Bmp_Bigger_And_Smaller(const char* old_bmp_path, const char* new_bmp_path)
{

    //定义原照片信息结构体
    BITMAPFILEHEADER head;
    BITMAPINFOHEADER info;
```

```
    //将结构体清空
    memset(&head, 0, sizeof(BITMAPFILEHEADER));
    memset(&info, 0, sizeof(BITMAPINFOHEADER));

    FILE* fpr1 = fopen(old_bmp_path, "rb");
    FILE* fpw2 = fopen(new_bmp_path, "wb");


    if (fpr1 == NULL || fpw2 == NULL)
    {
        printf("图片打开失败!\n");
        return -1;
    }
    //读取原照片的头信息
    fread(&head, sizeof(BITMAPFILEHEADER), 1, fpr1);
    fread(&info, sizeof(BITMAPINFOHEADER), 1, fpr1);

    unsigned int old_width = info.biWidth;//获取原图片的宽
    unsigned int old_height = info.biHeight;//获取原图片的高

    //获取原图片的位图数据
    unsigned char* src_data = (unsigned char*)malloc(old_width * old_height *
3);
    fseek(fpr1, 54, SEEK_SET);
    fread(src_data, old_width * old_height * 3, 1, fpr1);

    printf("原图片的宽:%d\n", old_width);
    printf("原图片的高:%d\n", old_height);

    //修改原照片的宽高
    unsigned int new_Width, new_Height;
    printf("请输入新图片的宽:\n");
    scanf("%d", &new_Width);
    printf("请输入新图片的高:\n");
    scanf("%d", &new_Height);


    head.bfSize = new_Width * new_Height * 3 + 54;
    info.biWidth = new_Width;
    info.biHeight = new_Height;

    //将修改过的头信息写进新照片
    fwrite(&head, sizeof(BITMAPFILEHEADER), 1, fpw2);
```

```
    fwrite(&info, sizeof(BITMAPINFOHEADER), 1, fpw2);

    int i = 0, j = 0;
    unsigned long dwsrcX, dwsrcY;
    unsigned char* pucDest;
    unsigned char* pucSrc;
    unsigned char* dest_data = (unsigned char*)malloc(new_Width * new_Height *
3);

    for (i = 0; i < new_Height; i++)
    {
        dwsrcY = i * old_height / new_Height;
        pucDest = dest_data + i * new_Width * 3;
        pucSrc = src_data + dwsrcY * old_width * 3;
        for (j = 0; j < new_Width; j++)
        {
            dwsrcX = j * old_width / new_Width;
            memcpy(pucDest + j * 3, pucSrc + dwsrcX * 3, 3);//数据拷贝
        }
    }
    fseek(fpw2, 54, SEEK_SET);
    fwrite(dest_data, new_Width * new_Height * 3, 1, fpw2);
    printf("生成新图片成功!\n");

    //释放堆空间
    free(dest_data);
    free(src_data);

    //关闭文件
    fclose(fpr1);
    fclose(fpw2);

    return 0;
}

int main()
{
    Bmp_Bigger_And_Smaller("sprites.bmp", "new_scale.bmp");
    return 0;
}
```

3. 将 bmp 图像转换为 coe 文件的 matlab 程序

```
clear
clc

% 利用 imread 函数把图片转化为一个三维矩阵
image_array = imread('new_scale.bmp');

% 利用 size 函数把图片矩阵的三个维度大小计算出来
% 第一维为图片的高度，第二维为图片的宽度，第三维为图片的 RGB 分量
[height,width,z]=size(image_array);   % 128*128*3

% imshow(image_array); % 显示图片

red   = image_array(:,:,1); % 提取红色分量，数据类型为 uint8
green = image_array(:,:,2); % 提取绿色分量，数据类型为 uint8
blue  = image_array(:,:,3); % 提取蓝色分量，数据类型为 uint8

% 把上面得到了各个分量重组成一个 1 维矩阵，由于 reshape 函数重组矩阵的
% 时候是按照列进行重组的，所以重组前需要先把各个分量矩阵进行转置以
% 后在重组
% 利用 reshape 重组完毕以后，由于后面需要对数据拼接，所以为了避免溢出
% 这里把 uint8 类型的数据扩大为 uint32 类型
r = uint32(reshape(red'   , 1 ,height*width));
g = uint32(reshape(green' , 1 ,height*width));
b = uint32(reshape(blue'  , 1 ,height*width));

% 初始化要写入.coe 文件中的 RGB 颜色矩阵
rgb=zeros(1,height*width);

% 因为导入的图片是 24-bit 真彩色图片，每个像素占用 24-bit，其中 RGB 分别占用 8-bit
% 而我这里需要的是 16-bit，其中 R 为 5-bit，G 为 6-bit，B 为 5-bit，所以需要在这里对
% 24-bit 的数据进行重组与拼接
% bitshift()函数的作用是对数据进行移位操作，其中第一个参数是要进行移位的数据，第
二个参数为负数表示向右移，为
% 正数表示向左移，更详细的用法直接在 Matlab 命令窗口输入 doc bitshift 进行查看
% 所以这里对红色分量先右移 3 位取出高 5 位，然后左移 11 位作为 ROM 中 RGB 数据的第 15-
bit 到第 11-bit
% 对绿色分量先右移 2 位取出高 6 位，然后左移 5 位作为 ROM 中 RGB 数据的第 10-bit 到第
5-bit
% 对蓝色分量先右移 3 位取出高 5 位，然后左移 0 位作为 ROM 中 RGB 数据的第 4-bit 到第 0-
bit
for i = 1:height*width
```

```
    rgb(i) = bitshift(bitshift(r(i),-3),11) + bitshift(bitshift(g(i),-2),5)
+ bitshift(bitshift(b(i),-3),0);
end

fid = fopen( 'image.coe', 'w+' );

% .coe 文件的最前面一行必须为这个字符串，其中 16 表示 16 进制
fprintf( fid, 'memory_initialization_radix=16;\n');

% .coe 文件的第二行必须为这个字符串
fprintf( fid, 'memory_initialization_vector =\n');

% 把 rgb 数据的前 height*width-1  个数据写入.coe 文件中，每个数据之间用逗号隔开
fprintf( fid, '%x,\n',rgb(1:end-1));

% 把 rgb 数据的最后一个数据写入.coe 文件中，并用分号结尾
fprintf( fid, '%x;',rgb(end));

fclose( fid ); % 关闭文件指针
```

4. GAME_TOP 模块 Verilog 代码

```
`timescale 1ns / 1ns
module GAME_TOP(
    input   clk_100,              //100Mhz
    input   rst,                   //复位

    //VGA
    output [3:0]    color_r,    //R
    output [3:0]    color_g,    //G
    output [3:0]    color_b,    //B
    output              hs,         //行同步
    output              vs,         //场同步

    //MP3
    input       SO,             //传出
    input       DREQ,           //数据请求，高电平时可传输数据
    output      XCS,            //SCI 传输读写指令
    output      XDCS,           //SDI 传输数据
    output      SCK,            //时钟
    output      SI,             //传入 mp3
    output      XRESET,         //硬件复位，低电平有效
```

```
//键盘
input   key_clk,                //键盘时钟
input   key_data,               //键盘输入数据

//七段数码管
output [7:0] shift,
output [6:0] oData
);

wire [10:0] mario_x;
wire [9:0] mario_y;
wire [5:0] mario_id;
wire [32:0] view;
wire [31:0] score;
wire hit_left;

//时钟
wire clk_108, clk_12, clk_10, clk_40, clk_1000, locked;

//键盘输入
wire [8:0] keys;
wire key_state;
wire left;
wire jump;
wire right;

assign left = (keys == 65);
assign right = (keys == 68);
assign jump = (keys == 87);

clk_wiz_0 uut_clk(
    .reset(~rst),
    .locked(locked),
    .clk_in1(clk_100),
    .clk_out1(clk_108),
    .clk_out2(clk_12)
);

VGA uut_vga(
    .clk(clk_108),
    .clk_view(clk_10),
    .rst(rst),
    .mario_x(mario_x),
    .mario_y(mario_y),
```

```
    .mario_id(mario_id),
    .right(right && !hit_left),
    .O_red(color_r),
    .O_green(color_g),
    .O_blue(color_b),
    .hs(hs),
    .vs(vs),
    .view(view)
);

Divider uut_divider(
    .clk12Mhz(clk_12),
    .clk2Mhz(clk_2),
    .clk10Hz(clk_10),
    .clk40Hz(clk_40),
    .clk1000Hz(clk_1000)
);

mp3 uut_mp3(
    .clk(clk_2),
    .rst(rst),
    .play(1'd1),
    .SO(SO),
    .DREQ(DREQ),
    .XCS(XCS),
    .XDCS(XDCS),
    .SCK(SCK),
    .SI(SI),
    .XRESET(XRESET)
);

World uut_world(
    .clk(clk_40),
    .clk_10(clk_10),
    .rst(rst),
    .jump(jump),
    .left(left),
    .right(right),
    .view(view),
    .mario_x(mario_x),
    .mario_y(mario_y),
    .mario_id(mario_id),
    .score(score),
    .hit_left(hit_left)
```

```
    );

    Keyboard uut_keyboard(
        .clk_in(clk_100),
        .rst(rst),
        .key_clk(key_clk),
        .key_data(key_data),
        .key_state(key_state),
        .key_ascii(keys)
    );

    //数码管显示分数
    Display uut_score(
        .clk_1000hz(clk_1000),
        .score(score),
        .shift(shift),//第几个数码管(片选)
        .oData(oData)
    );

endmodule
```

5. VGA 显示模块 Verilog 代码

```
`timescale 1ns / 1ns
module VGA(
    input  clk   ,     // 时钟
    input  clk_view,
    input  rst   ,     // 复位,低电平有效
    input  [10:0] mario_x, // mario 左上角的横坐标
    input  [9:0]  mario_y, // mario 左上角的纵坐标
    input  [5:0]  mario_id,// mario id
    input  right,          // 右移信号
    output   reg   [3:0]    O_red   , // VGA 红色分量
    output   reg   [3:0]    O_green , // VGA 绿色分量
    output   reg   [3:0]    O_blue  , // VGA 蓝色分量
    output                    hs    , // 行同步信号
    output                    vs    , // 场同步信号
    output   reg   [32:0]    view     // 当前视野的位置
    );

    //行时序常数
    parameter  HS_SYNC     = 112,
               HS_BACK     = 248,
               HS_ACTIVE   = 1280,
               HS_FRONT    = 48;
```

```
    //场时序常数
    parameter  VS_SYNC     = 3,
               VS_BACK     = 38,
               VS_ACTIVE   = 1024,
               VS_FRONT    = 1;
    //最大行列
    parameter  COL = 1688,
               ROW = 1066;

    parameter      COLOR_BAR_WIDTH   =   HS_ACTIVE / 8  ;

    reg [10:0]      h_cnt         ; // 行时序计数器
    reg [10:0]      v_cnt         ; // 列时序计数器

    wire            active        ; // 激活标志，当这个信号为 1 时 RGB 的数据
可以显示在屏幕上
    wire [10:0] x;
    wire [9:0]  y;//坐标

    parameter      IMAGE_WIDTH       =   1280    ,
                   IMAGE_HEIGHT      =   896     ,
                   IMAGE_PIX_NUM     =   204800  ;
//////////////////////////////////////////////////////////////////////
// 功能：产生行时序
//////////////////////////////////////////////////////////////////////
    always @(posedge clk or negedge rst)
    begin
        if(!rst)
            h_cnt <=  12'd0   ;
        else if(h_cnt == COL - 1'b1)
            h_cnt <=  12'd0   ;
        else
            h_cnt <=  h_cnt + 1'b1  ;
    end

    assign hs =   (h_cnt < HS_SYNC) ? 1'b0 : 1'b1    ;

//////////////////////////////////////////////////////////////////////
// 功能：产生场时序
//////////////////////////////////////////////////////////////////////
    always @(posedge clk or negedge rst)
    begin
        if(!rst)
            v_cnt <=  12'd0   ;
```

```
        else if(v_cnt == ROW - 1'b1)
            v_cnt <=  12'd0   ;
        else if(h_cnt == COL - 1'b1)
            v_cnt <=  v_cnt + 1'b1  ;
        else
            v_cnt <=  v_cnt ;
    end

    assign vs =   (v_cnt < VS_SYNC) ? 1'b0 : 1'b1    ;
//////////////////////////////////////////////////////////////////////
// 产生有效区域标志，当这个信号为高时往 RGB 送的数据才会显示到屏幕上
//////////////////////////////////////////////////////////////////////
    assign active =     (h_cnt >= (HS_SYNC + HS_BACK           ))  &&
                       (h_cnt <= (HS_SYNC + HS_BACK + HS_ACTIVE))  &&
                       (v_cnt >= (VS_SYNC + VS_BACK           ))  &&
                       (v_cnt <= (VS_SYNC + VS_BACK +
HS_ACTIVE))  ;

    reg  [11:0]         rom_addr_map; // 存放地图的 rom 地址
    wire [5:0]               typeid; // 元素的 id
    wire [11:0]      map_addr_offset; // 地图偏移地址
    wire [17:0]     rom_addr_sprites; // 存放元素的 rom 地址
    wire [17:0]           mario_addr; // 马里奥动作在 rom 中的起始地址
    wire [17:0]    mario_addr_offset; // 马里奥偏移地址
    wire [17:0]  sprites_addr_offset; // 元素偏移地址
    wire [15:0]         sprites_data; // 元素的 rgb 信息
    wire [15:0]           mario_data; // mario 的 rgb 信息
    reg  [4:0]               map_row; // 当前显示地图方格为当前地图的第几行
    reg  [4:0]               map_col; // 当前显示地图方格为当前地图的第几列
    reg  [5:0]           sprites_row; // 当前显示像素为当前元素的第几行
    reg  [5:0]           sprites_col; // 当前显示像素为当前元素的第几列

    assign map_addr_offset     = rom_addr_map     + 212 * map_row     +
map_col; // 当前的地图块元素位置
    assign sprites_addr_offset = rom_addr_sprites + 640 * sprites_row +
sprites_col; // 当前的像素位置
    assign mario_addr_offset   = mario_addr     + 640 * (y - mario_y) +
x - mario_x; // 当前马里奥像素位置
    assign x = (active) ? h_cnt - (HS_SYNC + HS_BACK):0;
    assign y = (active) ? v_cnt - (VS_SYNC + VS_BACK):0; //当前 VGA 中显示
的像素在 VGA 上的位置

    blk_mem_gen_map map(
        .clka(clk),             // input clka
```

```
    .addra(map_addr_offset), // input [11 : 0] addra
    .douta(typeid)           // output [5 : 0] douta
);

Object object_ground(
    .id(typeid),
    .addr(rom_addr_sprites)
);

Object object_mario(
    .id(mario_id),
    .addr(mario_addr)
);

blk_mem_gen_0 sprites(
    .clka(clk),                  // input clka
    .addra(sprites_addr_offset), // input [17 : 0] addra
    .douta(sprites_data),        // output [15 : 0] douta
    .clkb(clk),                  // input clkb
    .addrb(mario_addr_offset),   // input [17 : 0] addrb
    .doutb(mario_data)           // output [15 : 0] doutb
);

always @(posedge clk or negedge rst)
begin
    if(!rst)
    begin
         rom_addr_map <=  12'd0 ;
              map_row <=   5'd0 ;
              map_col <=   5'd0 ;
          sprites_row <=   6'd0 ;
          sprites_col <=   6'd0 ;
    end
    else if(active)
    begin
        if(h_cnt >= (HS_SYNC + HS_BACK                    )  &&
           h_cnt <= (HS_SYNC + HS_BACK + IMAGE_WIDTH  - 1'b1)  &&
           v_cnt >= (VS_SYNC + VS_BACK                    )  &&
           v_cnt <= (VS_SYNC + VS_BACK + IMAGE_HEIGHT - 1'b1)  )
        begin
            rom_addr_map <= (view - 11'd640) / 64;
            map_row <= y / 64;
            map_col <= (x + (view - 11'd640) % 64) / 64;
            sprites_row <=  y % 64;
```

```
                sprites_col <=  (x - 2 + (view - 11'd640) % 64) % 64; //
减二消除毛刺
                if((x <= mario_x + 63) && (x >= mario_x) && (y >=
mario_y) && (y <= mario_y + 63) && mario_data != 16'd23743) begin
                    O_red       <=  mario_data[15:12];
                    O_green     <=  mario_data[10:7];
                    O_blue      <=  mario_data[4:1];
                end
                else begin
                    O_red       <=  sprites_data[15:12];
                    O_green     <=  sprites_data[10:7];
                    O_blue      <=  sprites_data[4:1];
                end
            end
            else
            begin
                O_red       <=  4'd0;
                O_green     <=  4'd0;
                O_blue      <=  4'd0;
            end
        end
        else
        begin
            O_red       <=  4'd0;
            O_green     <=  4'd0;
            O_blue      <=  4'd0;
        end
    end

    always @(posedge clk_view or negedge rst) begin
       if(!rst)
           view <=  32'd640 ;
       else if (mario_x >= 640 && right == 1) begin
           view <= view + 32'd8;
           if(view >= 12800)
                view <= view;
       end
       else
           view <= view;
    end

endmodule
```

6. 偏移地址转换模块 Object Verilog 代码

```
`timescale 1ns / 1ns
// 功能：Map object id to address in the rom.
//box: 0
//boxempty: 1
//block: 2
//ground: 3

//castle1: 4
//castle2: 5
//castle3: 6
//castle4: 7
//castle5: 8
//castle6: 9

//goomba1: 10
//goomba2: 11
//goomba3: 12

//obstacle: 13

//mushroom: 14

//hill_up: 15
//hill_left: 24
//hill_down: 25
//hill_right: 26

//coin1: 16
//coin2: 17
//coin3: 18
//coin4: 19

//cloud1: 20
//cloud2: 21
//cloud3: 30
//cloud4: 31

//grass_left: 22
//grass_right: 23

//flag: 27
//pillar: 28
```

```
//ball: 29

//player1l: 32
//player1r: 42
//player2l: 33
//player2r: 43
//player3l: 34
//player3r: 44
//player4l: 35
//player4r: 45
//player5l: 36
//player5r: 46
//player_die: 37

//tunnel1: 38
//tunnel2: 39
//tunnel3: 48
//tunnel4: 49

//sky: 40

module Object(
    input [5:0] id, // 0 - 49 (64)
    output reg [17:0] addr
);

    always@(*) begin
        case(id)
            0: addr <= 0;
            1: addr <= 64 * 1;
            2: addr <= 64 * 2;
            3: addr <= 64 * 3;
            4: addr <= 64 * 4;
            5: addr <= 64 * 5;
            6: addr <= 64 * 6;
            7: addr <= 64 * 7;
            8: addr <= 64 * 8;
            9: addr <= 64 * 9;
            10: addr <= 640 * 64;
            11: addr <= 640 * 64 + 64 * 1;
            12: addr <= 640 * 64 + 64 * 2;
            13: addr <= 640 * 64 + 64 * 3;
            14: addr <= 640 * 64 + 64 * 4;
            15: addr <= 640 * 64 + 64 * 5;
```

```
            16: addr <= 640 * 64 + 64 * 6;
            17: addr <= 640 * 64 + 64 * 7;
            18: addr <= 640 * 64 + 64 * 8;
            19: addr <= 640 * 64 + 64 * 9;
            20: addr <= 640 * 64 * 2;
            21: addr <= 640 * 64 * 2 + 64 * 1;
            22: addr <= 640 * 64 * 2 + 64 * 2;
            23: addr <= 640 * 64 * 2 + 64 * 3;
            24: addr <= 640 * 64 * 2 + 64 * 4;
            25: addr <= 640 * 64 * 2 + 64 * 5;
            26: addr <= 640 * 64 * 2 + 64 * 6;
            27: addr <= 640 * 64 * 2 + 64 * 7;
            28: addr <= 640 * 64 * 2 + 64 * 8;
            29: addr <= 640 * 64 * 2 + 64 * 9;
            30: addr <= 640 * 64 * 3;
            31: addr <= 640 * 64 * 3 + 64 * 1;
            32: addr <= 640 * 64 * 3 + 64 * 2;
            33: addr <= 640 * 64 * 3 + 64 * 3;
            34: addr <= 640 * 64 * 3 + 64 * 4;
            35: addr <= 640 * 64 * 3 + 64 * 5;
            36: addr <= 640 * 64 * 3 + 64 * 6;
            37: addr <= 640 * 64 * 3 + 64 * 7;
            38: addr <= 640 * 64 * 3 + 64 * 8;
            39: addr <= 640 * 64 * 3 + 64 * 9;
            40: addr <= 640 * 64 * 4;
            42: addr <= 640 * 64 * 4 + 64 * 2;
            43: addr <= 640 * 64 * 4 + 64 * 3;
            44: addr <= 640 * 64 * 4 + 64 * 4;
            45: addr <= 640 * 64 * 4 + 64 * 5;
            46: addr <= 640 * 64 * 4 + 64 * 6;
            48: addr <= 640 * 64 * 4 + 64 * 8;
            49: addr <= 640 * 64 * 4 + 64 * 9;
            default:
                addr <= 640 * 64 * 4;
        endcase
    end

endmodule
```

7. 键码生成模块 Verilog 代码

```
`timescale 1ns / 1ps
module Keyboard//将键盘键码转化为 ASCII 码
(
```

```
    input                   clk_in,             //系统时钟
    input                   rst,                //系统复位，低有效
    input                   key_clk,            //PS2 键盘时钟输入
    input                   key_data,           //PS2 键盘数据输入
    output  reg             key_state,          //键盘的按下状态，按下置 1，
松开置 0
    output  reg     [7:0]   key_ascii           //按键键值对应的 ASCII 编码
);

reg     key_clk_r0 = 1'b1, key_clk_r1 = 1'b1;
reg     key_data_r0 = 1'b1,key_data_r1 = 1'b1;
//对键盘时钟数据信号进行延时锁存
always @ (posedge clk_in or negedge rst) begin
    if(!rst) begin
        key_clk_r0 <= 1'b1;
        key_clk_r1 <= 1'b1;
        key_data_r0 <= 1'b1;
        key_data_r1 <= 1'b1;
    end else begin
        key_clk_r0 <= key_clk;
        key_clk_r1 <= key_clk_r0;
        key_data_r0 <= key_data;
        key_data_r1 <= key_data_r0;
    end
end

//键盘时钟信号下降沿检测
wire    key_clk_neg = key_clk_r1 & (~key_clk_r0);

reg             [3:0]   cnt;
reg             [7:0]   temp_data;
//根据键盘的时钟信号的下降沿读取数据
always @ (posedge clk_in or negedge rst) begin
    if(!rst) begin
        cnt <= 4'd0;
        temp_data <= 8'd0;
    end else if(key_clk_neg) begin
        if(cnt >= 4'd10) cnt <= 4'd0;
        else cnt <= cnt + 1'b1;
        case (cnt)
            4'd0: ; //起始位
            4'd1: temp_data[0] <= key_data_r1;  //数据位 bit0
            4'd2: temp_data[1] <= key_data_r1;  //数据位 bit1
            4'd3: temp_data[2] <= key_data_r1;  //数据位 bit2
```

```
            4'd4: temp_data[3] <= key_data_r1;  //数据位 bit3
            4'd5: temp_data[4] <= key_data_r1;  //数据位 bit4
            4'd6: temp_data[5] <= key_data_r1;  //数据位 bit5
            4'd7: temp_data[6] <= key_data_r1;  //数据位 bit6
            4'd8: temp_data[7] <= key_data_r1;  //数据位 bit7
            4'd9: ; //校验位
            4'd10:; //结束位
            default: ;
        endcase
    end
end

reg                     key_break = 1'b0;
reg             [7:0]   key_byte = 1'b0;
//根据通码和断码判定按键的当前是按下还是松开
always @ (posedge clk_in or negedge rst) begin
    if(!rst) begin
        key_break <= 1'b0;
        key_state <= 1'b0;
        key_byte <= 1'b0;
    end else if(cnt==4'd10 && key_clk_neg) begin
        if(temp_data == 8'hf0) key_break <= 1'b1;   //收到断码（8'hf0）表
示按键松开，下一个数据为断码，设置断码标示为 1
        else if(!key_break) begin   //当断码标示为 0 时，表示当前数据为按下
数据，输出键值并设置按下标示为 1
            key_state <= 1'b1;
            key_byte <= temp_data;
        end else begin  //当断码标示为 1 时，标示当前数据为松开数据，断码标示
和按下标示都清 0
            key_state <= 1'b0;
            key_break <= 1'b0;
            key_byte<=0;
        end
    end
end

//将键盘返回的有效键值转换为按键字母对应的 ASCII 码
always @ (key_byte) begin
    case (key_byte)    //translate key_byte to key_ascii
            8'h1d: key_ascii = 8'h57;   //W
            8'h1c: key_ascii = 8'h41;   //A
            8'h1b: key_ascii = 8'h53;   //S
            8'h23: key_ascii = 8'h44;   //D
        default: key_ascii=8'h0;        //nothing
```

```
    endcase
end

endmodule
```

8. 时钟分频模块 Verilog 代码

```
`timescale 1ns / 1ns
module Divider(
    input clk12Mhz,
    output reg clk2Mhz = 0,
    output reg clk10Hz = 0,
    output reg clk40Hz = 0,
    output reg clk1000Hz = 0
    );
    integer cnt1 = 32'd0;
    integer cnt2 = 32'd0;
    integer cnt3 = 32'd0;
    integer cnt4 = 32'd0;
    always @(posedge clk12Mhz) begin
        if (cnt1 < 6 / 2 - 1)
            cnt1 <= cnt1 + 1'b1;
        else begin
            cnt1 <= 32'd0;
            clk2Mhz <= ~clk2Mhz;
        end

        if (cnt2 < 1200000 / 2 - 1)
            cnt2 <= cnt2 + 1'b1;
        else begin
            cnt2 <= 32'd0;
            clk10Hz <= ~clk10Hz;
        end

        if (cnt3 < 300000 / 2 - 1)
            cnt3 <= cnt3 + 1'b1;
        else begin
            cnt3 <= 32'd0;
            clk40Hz <= ~clk40Hz;
        end

        if (cnt4 < 12000 / 2 - 1)
            cnt4 <= cnt4 + 1'b1;
        else begin
            cnt4 <= 32'd0;
```

```
            clk1000Hz <= ~clk1000Hz;
        end

    end

endmodule
```

9. 分数显示模块 Verilog 代码

```
`timescale 1ns / 1ps
module Display(
    input clk_1000hz,
    input [31:0] score,
    output reg [7:0] shift,//第几个数码管(片选)
    output reg [6:0] oData
);
    wire [3:0] Data[7:0];
    reg  [3:0]     cnt=0;//计数器

    //转换为 BCD
    bin2BCD uut_bin2BCD(
        .number(score),
        .bcd0(Data[0]),
        .bcd1(Data[1]),
        .bcd2(Data[2]),
        .bcd3(Data[3]),
        .bcd4(Data[4]),
        .bcd5(Data[5]),
        .bcd6(Data[6]),
        .bcd7(Data[7])
    );

    //片选输出
    always@(posedge clk_1000hz)begin
        if(cnt == 4'd8)
            cnt <= 0;
        else
            cnt <= cnt + 1;
        shift <= 8'b1111_1111;
        shift[cnt] <= 0;//选择一个数码管进行输出

        case (Data[cnt])
            4'b0000: oData <= 7'b1000000;
            4'b0001: oData <= 7'b1111001;
            4'b0010: oData <= 7'b0100100;
```

```
            4'b0011: oData <= 7'b0110000;
            4'b0100: oData <= 7'b0011001;
            4'b0101: oData <= 7'b0010010;
            4'b0110: oData <= 7'b0000010;
            4'b0111: oData <= 7'b1111000;
            4'b1000: oData <= 7'b0000000;
            4'b1001: oData <= 7'b0010000;
            default: oData <= 7'b1111111;
        endcase
    end

endmodule
```

10. 二进制转 BCD 码模块 Verilog 代码

```
`timescale 1ns / 1ps
//BCD 转换
module bin2BCD(
    input [31:0] number,     //数字
    output [3:0] bcd0,
    output [3:0] bcd1,
    output [3:0] bcd2,
    output [3:0] bcd3,
    output [3:0] bcd4,
    output [3:0] bcd5,
    output [3:0] bcd6,
    output [3:0] bcd7
    );

    reg [31:0]  bin;
    reg [31:0]  result;
    reg [31:0]  bcd;

    //转换为 BCD 码
    always @(number) begin
        bin = number[31:0];
        result = 32'd0;
        repeat (31)
        begin
            result[0] = bin[31];
            if (result[3:0] > 4)
                result[3:0] = result[3:0] + 4'd3;
            else
                result[3:0] = result[3:0];
            if (result[7:4] > 4)
```

```
            result[7:4] = result[7:4] + 4'd3;
        else
            result[7:4] = result[7:4];
        if (result[11:8] > 4)
            result[11:8] = result[11:8] + 4'd3;
        else
            result[11:8] = result[11:8];
        if (result[15:12] > 4)
            result[15:12] = result[15:12] + 4'd3;
        else
            result[15:12] = result[15:12];
        if (result[19:16] > 4)
            result[19:16] = result[19:16] + 4'd3;
        else
            result[19:16] = result[19:16];

        if (result[23:20] > 4)
            result[23:20] = result[23:20] + 4'd3;
        else
            result[23:20] = result[23:20];

        if (result[27:24] > 4)
            result[27:24] = result[27:24] + 4'd3;
        else
            result[27:24] = result[27:24];
        if (result[31:28] > 4)
            result[31:28] = result[31:28] + 4'd3;
        else
            result[31:28] = result[31:28];
        result = result << 1;
        bin = bin << 1;
    end
    result[0] = bin[31];
    bcd = result;
end

assign bcd0 = bcd[3:0];
assign bcd1 = bcd[7:4];
assign bcd2 = bcd[11:8];
assign bcd3 = bcd[15:12];
assign bcd4 = bcd[19:16];
assign bcd5 = bcd[23:20];
assign bcd6 = bcd[27:24];
assign bcd7 = bcd[31:28];
```

```
endmodule
```

11. 音乐播放模块 Verilog 代码

```
`timescale 1ns / 1ns
module mp3(
    input       clk,            //12.288/6MHZ 时钟
    input       rst,
    input       play,           //开始播放请求
    input       SO,             //传出
    input       DREQ,           //数据请求，高电平时可传输数据

    output reg  XCS,            //SCI 传输读写指令
    output reg  XDCS,           //SDI 传输数据
    output      SCK,            //时钟
    output reg  SI,             //传入 mp3
    output reg  XRESET          //硬件复位，低电平有效
    );
    parameter  H_RESET     = 4'd0,         //硬复位
               S_RESET     = 4'd1,         //软复位
               SET_CLOCKF  = 4'd2,         //设置时钟寄存器
               SET_BASS    = 4'd3,         //设置音调寄存器
               SET_VOL     = 4'd4,         //设置音量
               WAIT        = 4'd5,         //等待
               PLAY        = 4'd6,         //播放
               END         = 6'd7;         //结束

    reg [3:0]       state       = WAIT ;            //状态
    reg [31:0]      cntdown     = 32'd0;            //延时
    reg [31:0]      cmd         = 32'd0;            //指令与地
    reg [7:0]       cntData     = 8'd32;            //SCI 指令地址位数计数

    reg [31:0]      music_data  = 32'd0;           //音乐数据
    reg [31:0]      cntSended   = 32'd32;          //SDI 当前 4 字节已传送
BIT

    reg  [9:0]      addra       = 10'd0;           //ROM 中的地址
    wire [31:0]     data;                         //ROM 传出

    reg             ena         = 0;

    assign SCK = (clk & ena);
    //速度控制
```

```
    reg [31:0] mp3Speed = 1700000;//延迟

    always @(negedge clk) begin
        if(!rst) begin
            XDCS <= 1'b1;
            ena <= 0;
            SI <= 1'b0;
            XCS <= 1'b1;
            state <= WAIT;
            XRESET <= 1'b1; // 硬件不复位
            addra <= 17'd0;
            cntSended <= 32'd32;
            music_data <= 32'd0;
        end
        else begin
            case (state)
                /*----------------等待---------------*/
                WAIT:begin
                        if(cntdown > 0)
                            cntdown <= cntdown - 1'b1;
                        //转到硬复位
                        else begin
                            cntdown <= 32'd1000;
                            state <= H_RESET;
                        end
                    end
                /*-----------------硬复------------------*/
                H_RESET:begin
                            if(cntdown > 0)
                                cntdown <= cntdown - 1'b1;
                            else begin
                                XCS <= 1'b1;
                                XRESET <= 1'b0;
                                cntdown <= 32'd16700;               //复位
后延时一段时

                                state <= S_RESET;                   //转移
到软复位
                                cmd <= 32'h02_00_08_04;            //软复
位指
                                cntData <= 8'd32;                  //指令、
地、数据长度
                            end
                        end
```

```
                /*------------------软复-----------------*/
                S_RESET:begin
                            if(cntdown > 0) begin
                                XRESET <= (cntdown < 32'd16650);
                                cntdown <= cntdown - 1'b1;
                            end
                            else if(cntData == 0) begin           //软复位
结
                                cntdown <= 32'd16600;

                                state <= SET_VOL;                  //转移
到设置 VOL
                                cmd <= 32'h02_0b_00_00;
                                cntData <= 8'd32;

                                XCS <= 1'b1;                       //拉高
XCS
                                ena <= 1'b0;                       //关闭
输入时钟
                                SI <= 1'b0;
                            end
                            else if(DREQ) begin                    //当
DREQ 有效时开始软复位
                                XCS <= 1'b0;
                                ena <= 1'b1;
                                SI <= cmd[cntData - 1];
                                cntData <= cntData - 1'b1;
                            end
                        else begin
                                XCS <= 1'b1;                       //DREQ
无效时继续等
                                ena <= 1'b0;
                                SI <= 1'b0;
                        end
                    end

                /*----------播放音乐----------*/
                PLAY:begin
                        if(cntdown > 0)
                            cntdown <= cntdown - 1'b1;
                        else if(play)begin
                            XDCS <= 1'b0;
                            ena <= 1'b1;
```

```
                        if(cntSended == 0) begin              //传输 4
字节
                            XDCS <= 1'b1;                    //拉高 XDCS
                            ena <= 1'b0;
                            SI <= 1'b0;
                            cntSended <= 32'd32;
                            music_data <= data;
                            addra <= addra + 1'b1;
                        end
                        else begin
                        //当 DREQ 有效 或当前字节尚未发送完 则继续传
                            if(DREQ || (cntSended != 32 && cntSended !=
24 && cntSended != 16 && cntSended != 8)) begin
                                SI <= music_data[cntSended - 1];
                                cntSended <= cntSended - 1'b1;
                                ena <= 1;
                                XDCS <= 1'b0;
                            end
                        else begin      //DREQ 拉低，停止传
                            ena <= 1'b0;
                            XDCS <= 1'b1;
                            SI <= 1'b0;
                        end
                    end
                end
                else;
                end
                /*---------------------寄存器配------------------*/
                default:
                if(cntdown > 0)
                    cntdown <= cntdown - 1'b1;
                else if(cntData == 0) begin          //结束次 SCI 写入
                    if(state == SET_CLOCKF) begin
                        cntdown <= mp3Speed;//32'd1700000;
                        state <= PLAY;
                    end
                    else if(state == SET_BASS) begin
                        cntdown <= 32'd2100;
                        cmd <= 32'h02_03_70_00;
                        state <= SET_CLOCKF;
                    end
                    else begin //SET_VAL
                        cntdown <= 32'd2100;
                        cmd <= 32'h02_02_00_00;
```

```
                    state <= SET_BASS;
                end
                cntData <= 8'd32;
                XCS <= 1'b1;
                ena <= 1'b0;
                SI <= 1'b0;
            end
            else if(DREQ) begin                    //写入 SCI 指令、
地、数
                XCS <= 1'b0;
                ena <= 1'b1;
                SI <= cmd[cntData - 1];
                cntData <= cntData - 1'b1;
            end
            else begin                             //DREQ 拉低，等
                XCS <= 1'b1;
                ena <= 1'b0;
                SI <= 1'b0;
            end
        endcase
    end
end

blk_mem_gen_maintheme maintheme (
    .clka(clk),             // 时钟
    .addra(addra),          // 地址
    .douta(data)           // 数据输出
);

endmodule
```

12. 游戏逻辑模块 Verilog 代码

```
`timescale 1ns / 1ns
module World(
    input clk, // 40hz
    input clk_10, //10hz
    input rst,
    input jump,
    input left,
    input right,
    input [32:0] view,

    output reg [10:0] mario_x,
    output reg [9:0] mario_y,
```

```
output reg [5:0] mario_id = 6'd32,
output [5:0] m_id,
output reg [31:0] score,
output reg death,
output reg hit_left
);

// Common data
wire clk_walk = clk_10; // 0.1s

// Mario
parameter init_mario_x = 11'd128;
parameter init_mario_y = 10'd704;
reg [1:0] jump_state; // 00: reset; 10: up; 11: down
reg [5:0] up_ticks;   // 64: nomove -> up
wire mario_oriental;
wire mario_walk;

// 碰撞检测部分
reg hit_up;
reg hit_down;
//reg hit_left;
reg hit_right;
wire [11:0] map_addr_offset[7:0]; // 地图偏移地址
reg  [4:0]          map_row[7:0]; // 当前马里奥方格为当前地图的第几行
reg  [4:0]          map_col[7:0]; // 当前马里奥方格为当前地图的第几列
reg  [11:0]         rom_addr_map; // 存放地图的 rom 地址
wire [5:0]           typeid[7:0]; // 碰撞检测用

parameter UP_1 = 0;
parameter UP_2 = 1;
parameter DOWN_1 = 2;
parameter DOWN_2 = 3;
parameter LEFT_1 = 4;
parameter LEFT_2 = 5;
parameter RIGHT_1 = 6;
parameter RIGHT_2 = 7;

// 碰撞检测
always @(posedge clk or negedge rst) begin
     if(!rst) begin
            rom_addr_map <=  12'd0 ;
            map_row[UP_1] <=   5'd0 ;
            map_row[UP_2] <=   5'd0 ;
```

```
                map_col[UP_1] <=   5'd0 ;
                map_col[UP_2] <=   5'd0 ;
                map_row[DOWN_1] <=   5'd0 ;
                map_row[DOWN_2] <=   5'd0 ;
                map_col[DOWN_1] <=   5'd0 ;
                map_col[DOWN_2] <=   5'd0 ;
                map_row[LEFT_1] <=   5'd0 ;
                map_row[LEFT_2] <=   5'd0 ;
                map_col[LEFT_1] <=   5'd0 ;
                map_col[LEFT_2] <=   5'd0 ;
                map_row[RIGHT_1] <=   5'd0 ;
                map_row[RIGHT_2] <=   5'd0 ;
                map_col[RIGHT_1] <=   5'd0 ;
                map_col[RIGHT_2] <=   5'd0 ;
                hit_up <= 0;
                hit_down <= 0;
                hit_left <= 0;
                hit_right <= 0;
            end
        else begin
                rom_addr_map <= (view - 11'd640) / 64;
                map_row[UP_1] <= (mario_y - 1) / 64;
                map_col[UP_1] <= (mario_x + (view - 11'd640) % 64) / 64;
                map_row[UP_2] <= (mario_y - 1) / 64;
                map_col[UP_2] <= (mario_x + 63 + (view - 11'd640) % 64)
/ 64;

                map_row[DOWN_1] <= (mario_y + 64) / 64;
                map_col[DOWN_1] <= (mario_x + (view - 11'd640) % 64) /
64;
                map_row[DOWN_2] <= (mario_y + 64) / 64;
                map_col[DOWN_2] <= (mario_x + 63 + (view - 11'd640) %
64) / 64;

                map_row[LEFT_1] <= (mario_y) / 64;
                map_col[LEFT_1] <= (mario_x - 1 + (view - 11'd640) % 64)
/ 64;
                map_row[LEFT_2] <= (mario_y + 63) / 64;
                map_col[LEFT_2] <= (mario_x - 1 + (view - 11'd640) % 64)
/ 64;

                map_row[RIGHT_1] <= (mario_y) / 64;
                map_col[RIGHT_1] <= (mario_x + 64 + (view - 11'd640) %
64) / 64;
```

```
                map_row[RIGHT_2] <= (mario_y + 63) / 64;
                map_col[RIGHT_2] <= (mario_x + 64 + (view - 11'd640) %
64) / 64;

                if((typeid[UP_1] == 0 || typeid[UP_1] == 1 ||
typeid[UP_1] == 2) ||
                   (typeid[UP_2] == 0 || typeid[UP_2] == 1 ||
typeid[UP_2] == 2))
                    hit_down <= 1;
                else
                    hit_down <= 0;

                if((typeid[DOWN_1] == 0 || typeid[DOWN_1] == 1 ||
typeid[DOWN_1] == 2 || typeid[DOWN_1] == 3 || typeid[DOWN_1] == 13 ||
typeid[DOWN_1] == 38 || typeid[DOWN_1] == 39) ||
                   (typeid[DOWN_2] == 0 || typeid[DOWN_2] == 1 ||
typeid[DOWN_2] == 2 || typeid[DOWN_2] == 3 || typeid[DOWN_2] == 13 ||
typeid[DOWN_2] == 38 || typeid[DOWN_2] == 39))
                    hit_up <= 1;
                else
                    hit_up <= 0;

                if((typeid[LEFT_1] == 0 || typeid[LEFT_1] == 1 ||
typeid[LEFT_1] == 2 || typeid[LEFT_1] == 3 || typeid[LEFT_1] == 13 ||
typeid[LEFT_1] == 39 || typeid[LEFT_1] == 49) ||
                   (typeid[LEFT_2] == 0 || typeid[LEFT_2] == 1 ||
typeid[LEFT_2] == 2 || typeid[LEFT_2] == 3 || typeid[LEFT_2] == 13 ||
typeid[LEFT_2] == 39 || typeid[LEFT_2] == 49))
                    hit_right <= 1;
                else
                    hit_right <= 0;

                if((typeid[RIGHT_1] == 0 || typeid[RIGHT_1] == 1 ||
typeid[RIGHT_1] == 2 || typeid[RIGHT_1] == 3 || typeid[RIGHT_1] == 13
|| typeid[RIGHT_1] == 38 || typeid[RIGHT_1] == 48) ||
                   (typeid[RIGHT_2] == 0 || typeid[RIGHT_2] == 1 ||
typeid[RIGHT_2] == 2 || typeid[RIGHT_2] == 3 || typeid[RIGHT_2] == 13
|| typeid[RIGHT_2] == 38 || typeid[RIGHT_2] == 48))
                    hit_left <= 1;
                else
                    hit_left <= 0;
            end
        end
```

```
    genvar i;
    generate
        for(i = 0; i <= 7; i = i + 1) begin: calculate
           blk_mem_gen_map map(
            .clka(clk),              // input clka
            .addra(map_addr_offset[i]), // input [11 : 0] addra
            .douta(typeid[i])           // output [5 : 0] douta
            );
           assign map_addr_offset[i] = rom_addr_map + 212 * map_row[i] +
map_col[i]; // 当前的地图块元素位置
        end
    endgenerate

    // 实现走路帧
    Mario mario(
        .clk_walk(clk_walk),
        .rst(rst),
        .left(left),
        .right(right),
        .jump(jump),
        .id(m_id),
        .oriental(mario_oriental),
        .walk(mario_walk)
    );

    //mario 走路跳跃状态机
    always@(posedge clk_10, negedge rst) begin
        if (!rst) begin
            mario_x <= init_mario_x;
            mario_y <= init_mario_y;

            up_ticks <= 6'd0;
//          down_ticks <= 6'd0;
            jump_state <= 2'b00;

            mario_id <= 6'd32;
            death <= 0;
        end
        else begin

                    // Let Mario walks!
                    if (mario_walk && !death) begin
                       if((hit_right == 1 && left == 1)|| (hit_left == 1
&& right == 1))
```

```
                        mario_x <= mario_x;
                    else
                     if(mario_x >= 640 && right == 1)
                        mario_x <= 640;
                     else
                        mario_x <= mario_oriental ? mario_x - 11'd16 :
mario_x + 11'd16;

                     if(jump_state == 2'b10 || jump_state == 2'b11)
                        mario_id <= mario_oriental ? 6'd46 : 6'd36;
                     else if(right == 0 && left == 0)
                        mario_id <= mario_oriental ? 6'd42 : 6'd32;
                     else
                        mario_id <= m_id;
                end

                //die
                if(mario_y >= 832) begin
                   death <= 1;
                   mario_id <= 37;
                end

                // Let Mario jumps!
                if(death == 0)
                case(jump_state)
                   2'b00: begin
                       if(jump == 1) begin
                           jump_state <= 2'b10;
                           mario_id <= mario_oriental ? 6'd46 :
6'd36;
                       end
                       else if(hit_up == 0) begin
                           jump_state <= 2'b11;
                       end
                       else begin
                           jump_state <= jump_state;
                       end
                   end
                   2'b10: begin
                       if(up_ticks == 6'd10) begin
                           up_ticks <= 0;
                           jump_state <= 2'b11;
                           mario_id <= mario_oriental ? 6'd46 :
6'd36;
```

```
                        end
                        else begin
                            if(hit_down == 1 || mario_y == 0) begin
                                up_ticks <= 0;
                                jump_state <= 2'b11;
                            end
                            else begin
                                up_ticks <= up_ticks + 1;
                                mario_y <= mario_y - 10'd32;
                                jump_state <= jump_state;
                            end
                        end
                    end
                    2'b11: begin
                        if(hit_up == 1) begin
                            jump_state <= 2'b00;
                            mario_id <= mario_oriental ? 6'd42 :
6'd32;
                        end
                        else begin
                            mario_y <= mario_y + 10'd32;
                            jump_state <= jump_state;
                        end
                    end
                endcase
        end
    end

    // 加分机制
    always@(posedge clk_10, negedge rst) begin
       if(!rst) begin
           score <= 0;
       end
       else begin
           if(typeid[UP_1] == 0 || typeid[UP_2] == 0)
               score <= score + 50;
       end
    end

endmodule
```

13. mario 人物状态机模块 Verilog 代码

```
`timescale 1ns / 1ns
module Mario(
```

```
    // 用于判断左右方向等
    input clk_walk, // 用于切换 mario 的动作
    input rst,
    input left,  // 向左走
    input right, // 向右走
    input jump,

    output reg [5:0] id,
    output reg oriental, // 0: right 1: left
    output reg walk      // 0: no 1: yes
    );

    //state machine walk
    parameter walk1 = 2'd0;
    parameter walk2 = 2'd1;
    parameter walk3 = 2'd2;
    parameter walk4 = 2'd3;

    // 便于搜索 id
    parameter player1r = 32;
    parameter player1l = 42;
    parameter player2r = 33;
    parameter player2l = 43;
    parameter player3r = 34;
    parameter player3l = 44;
    parameter player4r = 35;
    parameter player4l = 45;

    reg [1:0] walk_state; // 0/1/2/3 //行走帧
    always@(posedge clk_walk, negedge rst) begin
        if (!rst)
        begin
            oriental <= 0;
            walk <= 0;
            walk_state <= walk1;
            id <= player1r;
        end
        else
        begin
            if (~oriental & left & ~right) // 判断方向，左右全为 0or1，则方
向不变
                oriental <= 1;
            else if (oriental & ~left & right)
                oriental <= 0;
```

```
walk <= left ^ right; // 全为0/1就不走

  if(jump == 0)
    if (left != 1) // 没向左走
            case (walk_state)
                walk1: begin
                          if(right == 1) begin
                              walk_state <= walk3;
                              id <= player3r;
                          end
                          else begin
                              walk_state <= walk1;
                              id <= player1r;
                          end
                       end
                walk2: begin
                          if(right == 1) begin
                              walk_state <= walk4;
                              id <= player4r;
                          end
                          else begin
                              walk_state <= walk1;
                              id <= player1r;
                          end
                       end
                walk3: begin
                          walk_state <= walk2;
                          id <= player2r;
                       end
                walk4: begin
                          if(right == 1) begin
                              walk_state <= walk3;
                              id <= player3r;
                          end
                          else begin
                              walk_state <= walk2;
                              id <= player2r;
                          end
                       end
           endcase
     else
    case (walk_state)
    walk1: begin
```

```
                        if(right == 0) begin
                            walk_state <= walk3;
                            id <= player3l;
                        end
                        else begin
                            walk_state <= walk1;
                            id <= player1l;
                        end
                    end
                walk2: begin
                        if(right == 0) begin
                            walk_state <= walk4;
                            id <= player4l;
                        end
                        else begin
                            walk_state <= walk1;
                            id <= player1l;
                        end
                    end
                walk3: begin
                        walk_state <= walk2;
                        id <= player2l;
                    end
                walk4: begin
                        if(right == 0) begin
                            walk_state <= walk3;
                            id <= player3l;
                        end
                        else begin
                            walk_state <= walk2;
                            id <= player2l;
                        end
                    end
                endcase
            end
        end

endmodule
```

14. VGA 显示测试模块 Verilog 代码

```
`timescale 10ns / 1ns
module vga_testbench;
    reg   clk_100;              //100
    reg   rst;
```

```
    wire [3:0]    color_r;    //R
    wire [3:0]    color_g;    //G
    wire [3:0]    color_b;    //B
    wire          hs;
    wire          vs;
    reg jump;
    reg left;
    reg right;

    wire clk_108,locked, rising;

    clk_wiz_0 uut_clk(
        .reset(~rst),
        .locked(locked),
        .clk_in1(clk_100),
        .clk_out1(clk_108),
        .clk_out2(clk_12)
    );

    wire [10:0] mario_x;
    wire [9:0] mario_y;
    wire [5:0] mario_id;
    wire [5:0] id;

    //时钟
    wire clk_10;

    wire [32:0] div; // 用于游戏界面的时钟
    clkdiv uut_clkdiv(
        .clk(clk_100),
        .rst(rst),
        .clkdiv(div)
    );

    Divider uut_divider(
        .clk12Mhz(clk_12),
        .clk10Hz(clk_10)
    );

//    VGA
    VGA uut_vga(
        .clk(clk_108),
        .clk_view(clk_10),
        .rst(rst),
```

```
    .mario_x(mario_x),
    .mario_y(mario_y),
    .mario_id(mario_id),
    .O_red(color_r),
    .O_green(color_g),
    .O_blue(color_b),
    .hs(hs),
    .vs(vs)
);

World uut_world(
    .clkdiv(div),
    .rst(rst),
    .jump(jump),
    .left(left),
    .right(right),
    .mario_x(mario_x),
    .mario_y(mario_y),
    .mario_id(mario_id),
    .m_id(id),
    .rising(rising)
);

initial
begin
    clk_100 = 1;
    forever
    begin
        #5 clk_100 = 0;
        #5 clk_100 = 1;
    end
end

initial
begin
    rst = 0;
    #100 rst = 1;
end

initial
begin
    right = 1;
    forever
    begin
```

```
        #13 right = 0;
        #13 right = 1;
        end
    end

    initial
    begin
        left = 0;
    end

    initial
    begin
        jump = 0;
    end

endmodule
```

15. VGA 显示测试模块中所用时钟分频模块

```
`timescale 1ns / 1ns
module clkdiv(
    input clk,
    input rst,
    output reg [32:0] clkdiv
    );

    always @ (posedge clk, negedge rst) begin
        if (!rst) clkdiv <= 0;
        else clkdiv <= clkdiv + 1'b1;
    end

endmodule
```

16. mario 人物状态机测试模块 Verilog 代码

```
`timescale 1ns / 1ns
module mario_testbench();
    reg clk;
    reg clk_walk;
    reg rst;
```

```
reg left;
reg right;
reg jump;

wire [5:0] id;
wire oriental;
wire walk;
wire rising;

Mario mario(
    .clk(clk),
    .clk_walk(clk_walk),
    .rst(rst),
    .left(left),
    .right(right),
    .jump(jump),
    .id(id),
    .oriental(oriental),
    .walk(walk),
    .rising(rising)
);

initial
begin
    clk = 1;
    forever
    begin
        #1 clk = 0;
        #1 clk = 1;
    end
end

initial
begin
    clk_walk = 1;
    forever
    begin
        #50 clk_walk = 0;
        #50 clk_walk = 1;
    end
end

initial
begin
```

```
        rst = 0;
        #1 rst = 1;
    end

    initial
    begin
        right = 1;
        forever
        begin
        #13 right = 0;
        #13 right = 1;
        end
    end

    initial
    begin
        left = 0;
    end

    initial
    begin
        jump = 0;
    end

endmodule
```

17. World 主世界测试模块 Verilog 代码

```
`timescale 10ns / 1ns
module world_testbench();
    reg clk;
    reg clk_10;
    reg rst;
    reg jump;
    reg left;
    reg right;
    reg [32:0] view;

    wire [10:0] mario_x;
    wire [9:0]  mario_y;
    wire [5:0]  id;
//    wire rising;

    World uut_world(
        .clk(clk),
```

```
        .clk_10(clk_10),
        .rst(rst),
        .jump(jump),
        .left(left),
        .right(right),
        .view(view),
        .mario_x(mario_x),
        .mario_y(mario_y),
        .mario_id(id)
//        .rising(rising)
    );

    initial
    begin
        clk = 1;
        forever
        begin
            #5 clk = 0;
            #5 clk = 1;
        end
    end

    initial
    begin
        clk_10 = 1;
        forever
        begin
            #5000000 clk_10 = 0;
            #5000000 clk_10 = 1;
        end
    end

    initial
    begin
        rst = 0;
        #1 rst = 1;
    end

    initial
    begin
        right = 1;
    end

    initial
```

```
    begin
        left = 0;
    end

    initial
    begin
        jump = 1;
        view = 640;
    end

endmodule
```