1. 系统顶层模块设计

1.1 系统功能

通过控制电路按钮可实现三种模式的循环转换，并通过 LED 指示灯获知当前状态：

（1） 基础时钟：在数码管上显示一个数字时钟完成以 1Hz 为频率从 00-00-00 到 23-59-59 的计数功能，并在控制电路开关作用下可实现加速功能。

（2） 闹钟模式：在该模式下，通过控制电路开关按钮可实现闹钟的设置，时分秒的加减，并且在分钟同步时，播放音乐闹钟，可通过控制电路开关将闹钟打开或关闭。

（3） 游戏模式：在该模式下，通过控制电路开关控制游戏背景音乐的打开或关闭，可通过控制电路按钮控制角色的上下移动以及攻击并触发 LED 闪烁，通过 LED 显示角色生命值，并在角色收到撞击时减少生命值，当生命值降为 0 时，游戏停止，但可通过控制电路按钮将角色复活并重新获得三点生命值，切换至其他模式时记下当前生命值状态，再次进入游戏模式时将会继承。

1.2 设计思路

时钟设计层次结构图如图 1 所示，时钟主要具有三种模式：时间显示，闹钟模式，游戏模式。

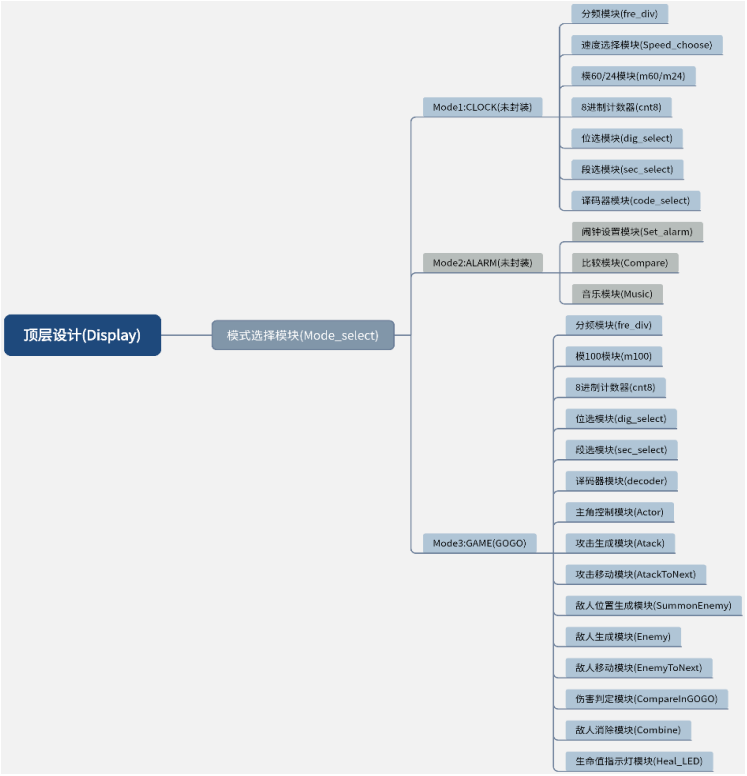

图 1 总体设计层次结构图

设计时采用分模块设计的思路，将各个模块的功能进行细化，确认设计需求，对单个小板块进行设计封装，然后并入整体设计中，检查运行状态是否正常，记录遇到的问题并分析成因，再进一步修改模块；若多次更改模块仍无法达到预期效果，分析是否模块设计思路错误，若是则改变实现方式，重复以上步骤，设计模块步骤如图 2 所示，总体设计思路与模块设计思路类似。

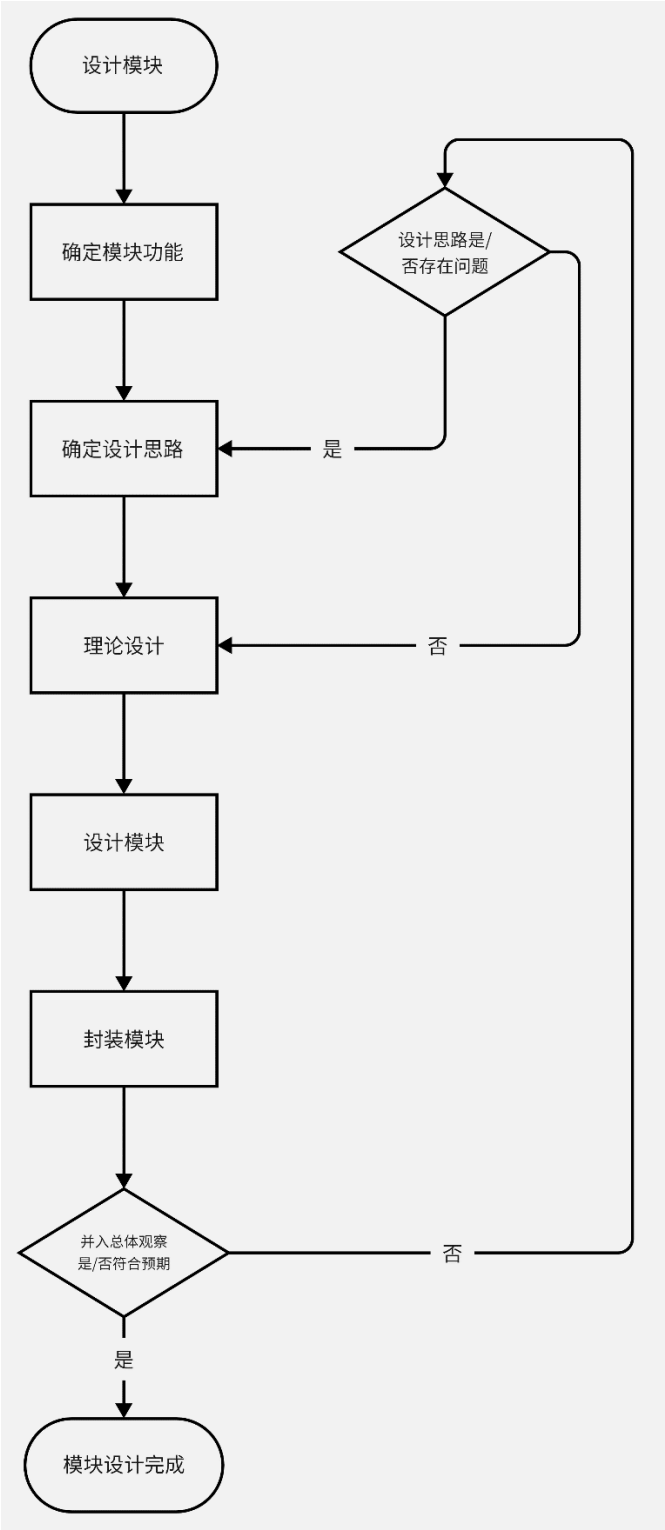

图 2 模块设计思路

1.3 顶层设计电路图

顶层设计电路总图如图 3-1 所示，为将各个模块内容更加清晰地展现，将电路图分为左上、右上、左下、右下共四个部分进行展示，分别对应图 3-2、图 3-3、图 3-4、图 3-5。

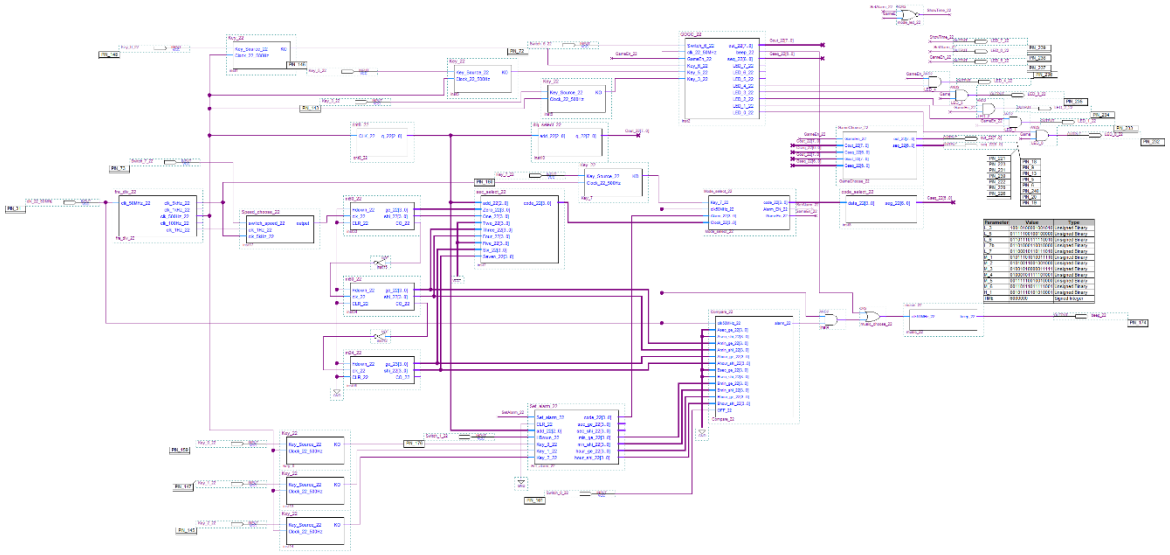

图 3-1 顶层模块设计电路接线图(总)

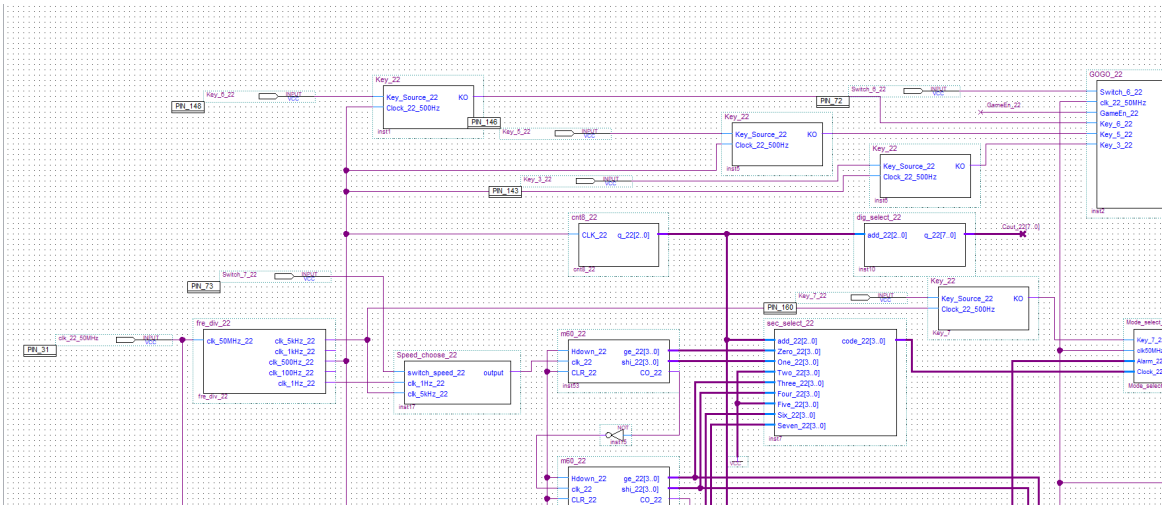

图 3-2 顶层模块设计电路接线图(左上)

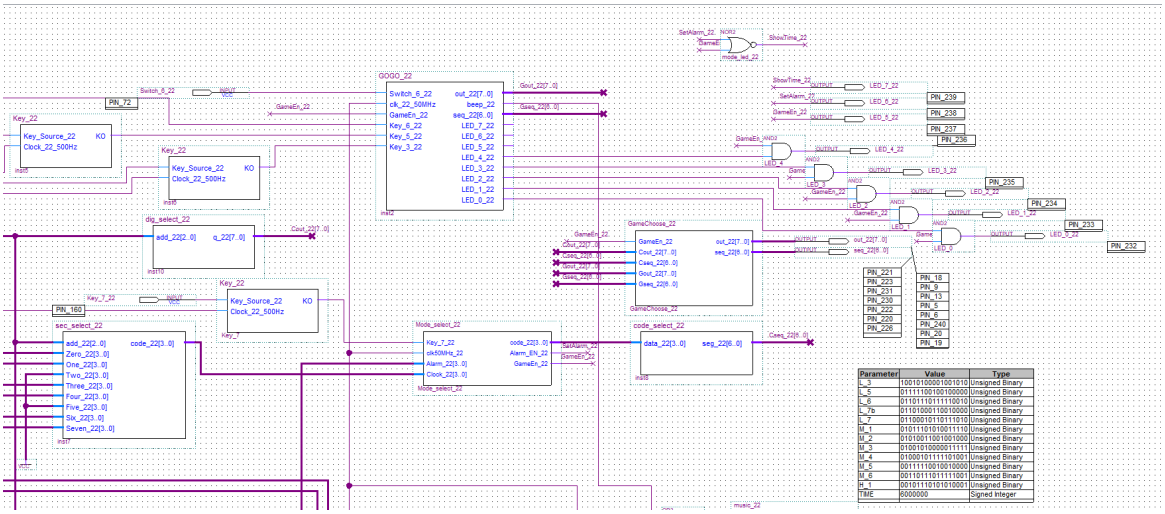

图 3-3 顶层模块设计电路接线图(右上)

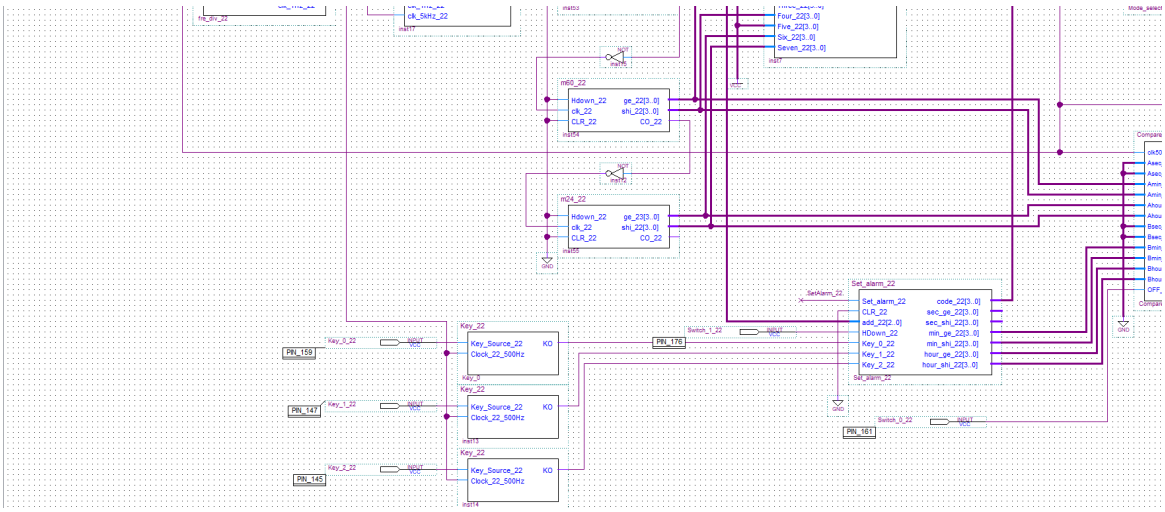

图 3-4 顶层模块设计电路接线图(左下)

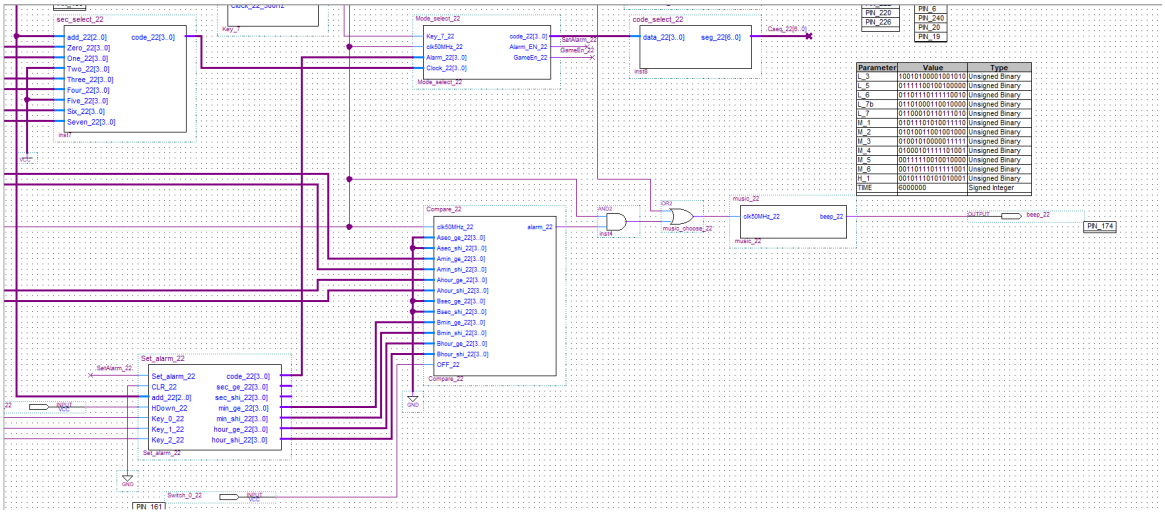

图 3-5 顶层模块设计电路接线图(右下)

2. 分频模块电路设计及仿真

2.1 模块功能

分频模块电路需要将由晶振振荡器产生的 50MHz 的时钟频率进行分频，最终输出所需要的时钟频率（5KHz、1KHz、500Hz、100Hz、1Hz）。

2.2 设计思路

确定模块功能，进行理论设计，50MHz 的频率可由如图 4 所示方式取模运算进行拆分获取所需要的频率。

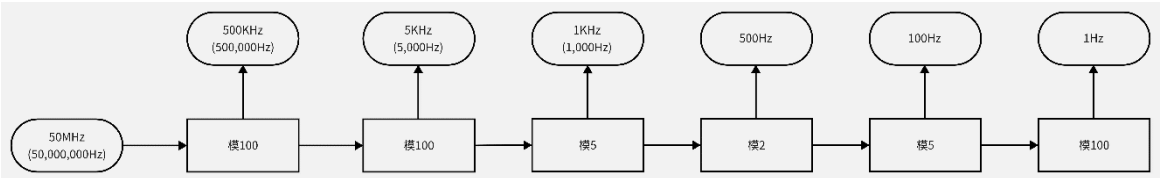

图 4 分频模块设计思路

根据所学内容，选择既能进行模 2 又能进行模 5 运算的 74390 芯片设计最佳。

2.3 设计结果

设计结果电路如图 5 所示，在设计电路时，因多次使用模 100 运算，所以可以将模 100 运算电路进行封装，以增强电路的可读性，模 100 电路如图 6 所示。

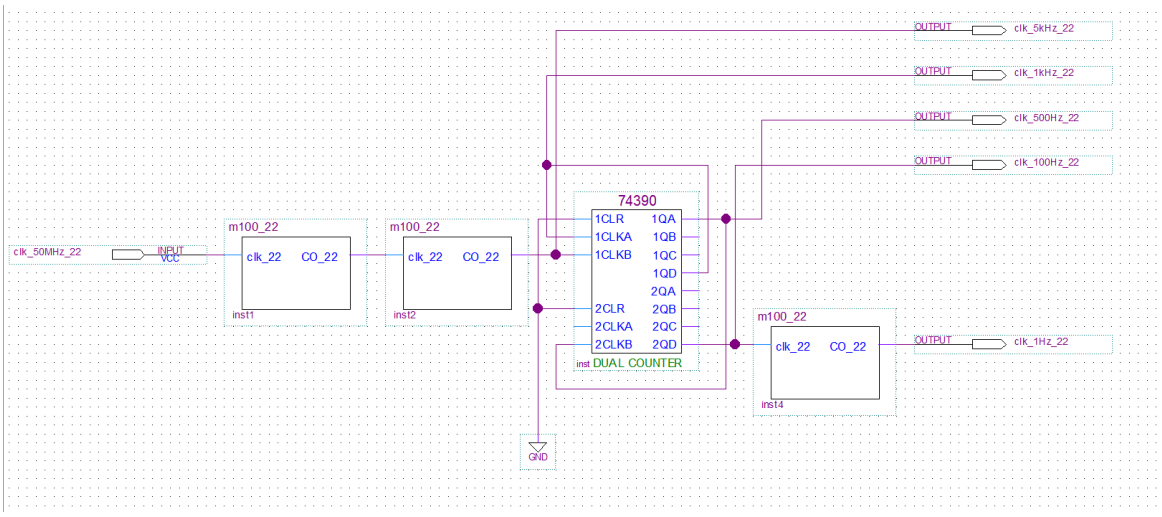

图 5 分频模块电路设计

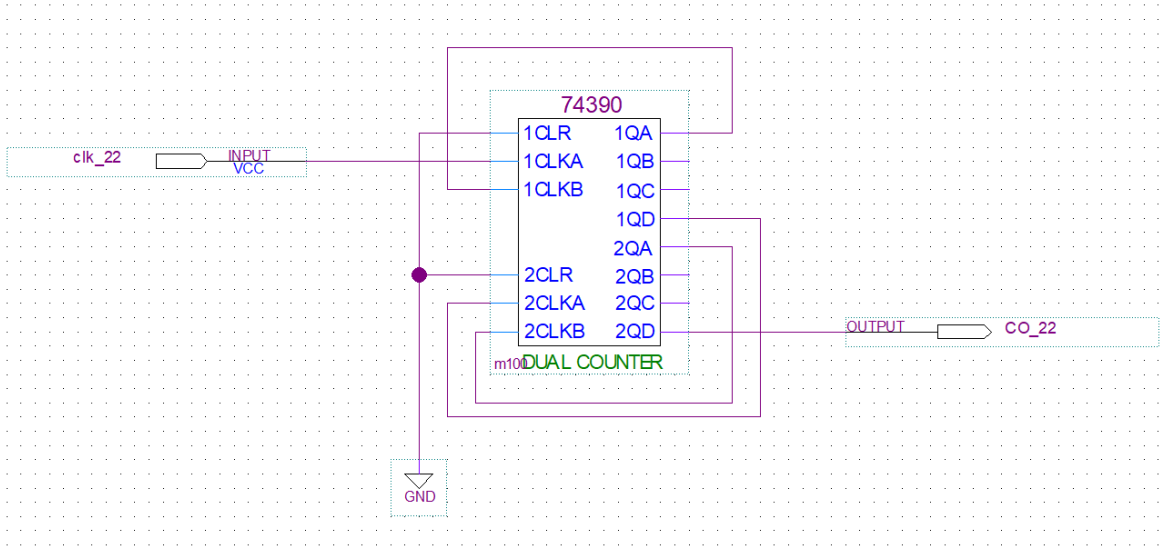

图 6 模 100 电路设计

2.4 测试结果

50MHz 分频为 1Hz 无法直接仿真，此处在 Quartus II 中对模 100 模块进行波形仿真，其波形如图 7 所示，每一百个输入时钟脉冲，将会触发一次输出脉冲，以达成将原频率分为原来的$\frac{1}{100}$的目的。

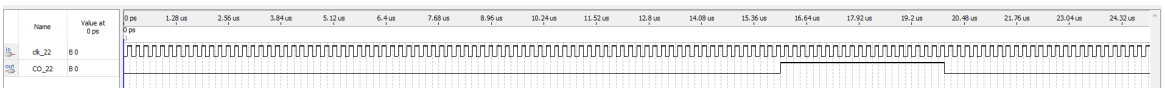

图 7 模 100 波形仿真

3. 计时模块设计及仿真

3.1 分、秒计时模块

3.1.1 模块功能

模块实现功能从 0 到 59 的模 60 计数，并在 59 变为 0 的时刻输出一个进位信号。在累加的基础上对设计进行修改，实现从 59 到 0 的递减计数。

3.1.2 设计思路

模块实现功能需要同时具有加法与减法，故选用 74192 芯片，其对应功能表如表 1 所示，可见为加计数模式时可计数到 1001（9）即十进制，故可采用两片 74192 芯片级联的方式完成模 100 以内的任何需求。由于减计数模式默认从 99（1001-1001）开始所以需要用到 7442 芯片检测第二片 1001 的状态输出一个置数信号，让 99 变为预先所设定好的 59（1001-0101），7442 芯片的功能表如表 2 所示，显然在 1001 时$\overline{O9N}$引脚输出低电平，我们将其取反作为置数信号。

表 1 74192 功能表

CLR	$\overline{LDN}$	UP	DN	D	C	B	A	QD	QC	QB	QA	$\overline{CON}$	$\overline{BON}$
1	×	×	×	×	×	×	×	0	0	0	0		
0	0	×	×	d	c	b	a	d	c	b	a		
0	1	⊔	1	×	×	×	×		加计数				
0	1	1	⊔	×	×	×	×		减计数				
×	×	⊔	1	×	×	×	×	1	0	0	1	⊔	1
×	×	1	⊔	×	×	×	×	0	0	0	0	1	⊔

表 2 7442 功能表

D	C	B	A	$\overline{O0N}$	$\overline{O1N}$	$\overline{O2N}$	$\overline{O3N}$	$\overline{O4N}$	$\overline{O5N}$	$\overline{O6N}$	$\overline{O7N}$	$\overline{O8N}$	$\overline{O9N}$
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1
							…						
1	1	1	1	1	1	1	1	1	1	1	1	1	1

由表 1 可知 74192 的加减计数模式是低有效，故可通过电路控制开关和与非门共同使用来控制加减法计数，为使默认状态为加计数，故采用与非门结合的方式，以达成 UP 端口为 CP 信号时 DN 端口持续为高电平信号，反之亦然。

级联方式为将$\overline{\text{CON}}$接到 UP，$\overline{\text{BON}}$接到 DN，即进位输出给加模式，借位输出给减模式。

清零信号可在第二片芯片处将 CB 相与，并用 D 触发器消除险象后作为清零信号，将清零信号与输入的 CLR 信号相或可在封装模块时完成清零信号的输入，清零信号需要同时接到两个芯片上。

第一片芯片为个位信号故将 DCBA 置为 1001（9），同理，第二片芯片为十位信号故将 DCBA 置为 0101（5），如此便可在置位信号到来时将减计数最高位置为 59。

3.1.3 设计结果

设计结果电路如图 8 所示。

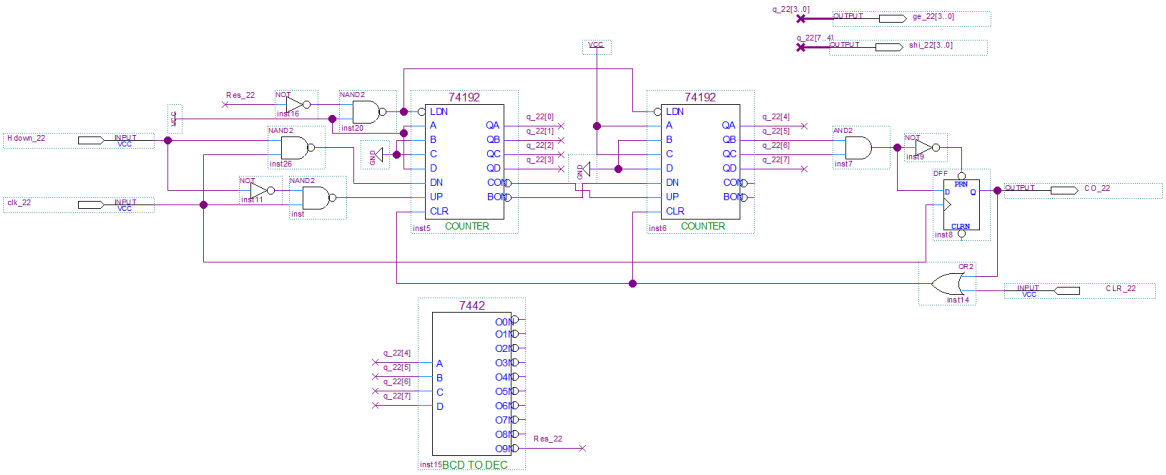

图 8 模 60 电路设计

3.1.4 仿真测试

进行的波形仿真主要由三部分组成：加计数及其进位，减计数，清零。

加计数及其进位波形仿真如图 9-1 所示，CO 为进位信号，Hdown 为加法减法控制信号，低电平为加计数模式。

减计数波形仿真如图 9-2 所示。

清零波形仿真如图 9-3 所示。

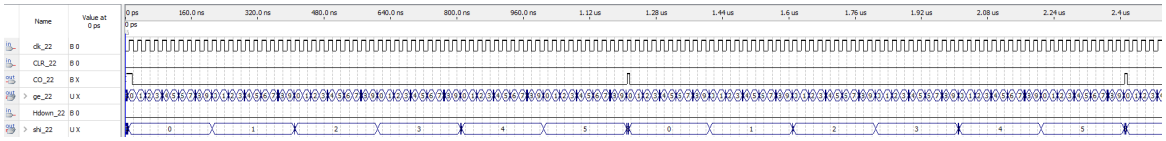

图 9-1 模 60 加计数及其进位波形仿真

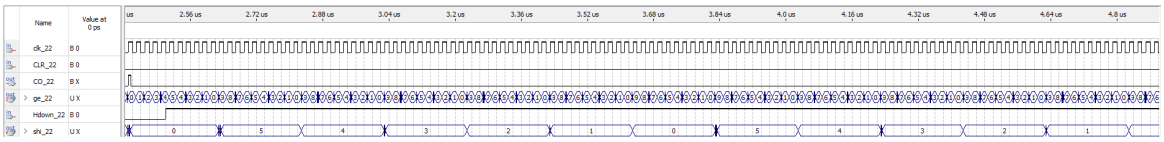

图 9-2 模 60 减计数波形仿真

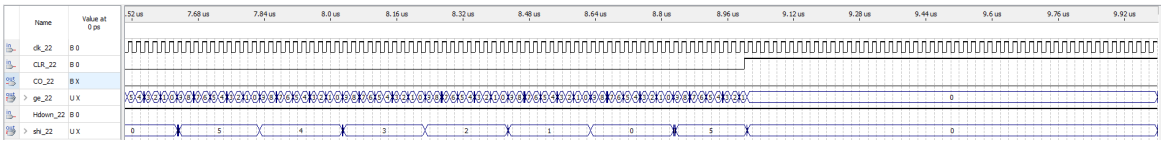

图 9-3 模 60 清零波形仿真

3.2 小时计时模块

3.2.1 模块功能

模块实现功能从 0 到 23 的模 24 计数。在累加的基础上对设计进行修改实现从 23 到 0 的递减计数。

3.2.2 设计思路

设计思路与模 60 模块思路相似，唯二不同之处分别是：清零信号和置位初始值。

电路设计仍然采用两片 74192 芯片级联以同时达成加计数与减计数，同样使用 7442 芯片将 99 变为预先设定好的 23。

当十位为 2(0010)，个位为 4（0100）时，输出一个清零信号，所以可以将第一片 74192（个位）的 DBA 取反，将第二片 74192（十位）的 DCA 取反，再将八个引脚相与作为清零信号。

减计数置位初始值变为 23（0011-0010），将第一片 74192 的置位初始值设置为 0010，第二片 74192 的置位初始值设置为 0011。

除此以外的电路设计可以完全沿用模 60 的设计。

3.2.3 设计结果

设计结果电路如图 10 所示。

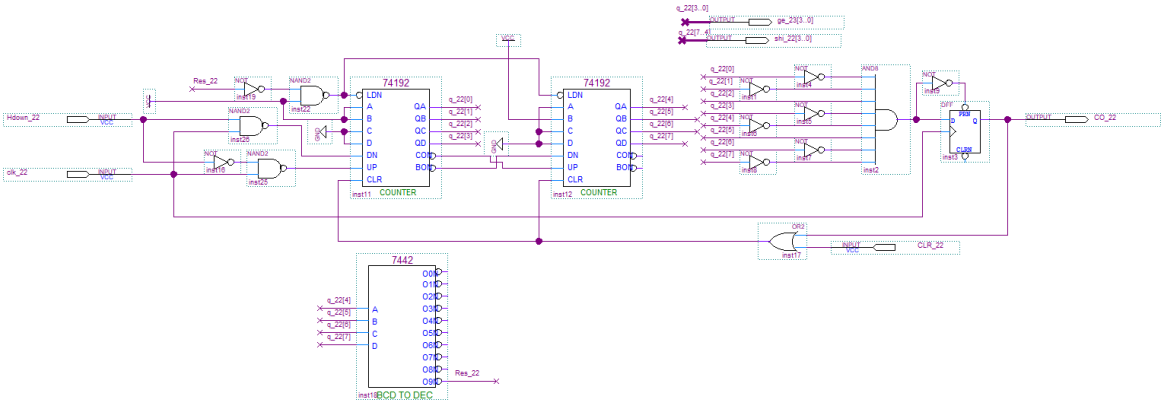

图 10 模 24 电路设计

3.2.4 仿真测试

进行的波形仿真主要由三部分组成：加计数及其进位，减计数，清零。

加计数及其进位波形仿真如图 11-1 所示，CO 为进位信号，Hdown 为加法减法控制信号，低电平为加计数模式。

减计数波形仿真如图 11-2 所示。

清零波形仿真如图 11-3 所示。

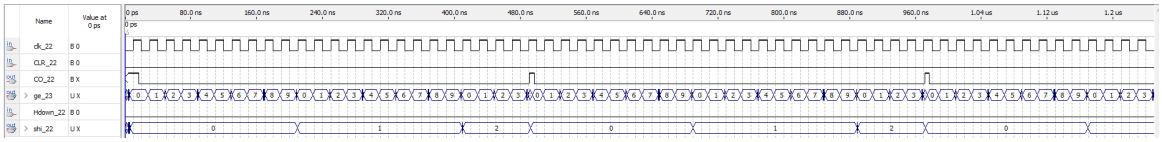

图 11-1 模 24 加计数及其进位波形仿真

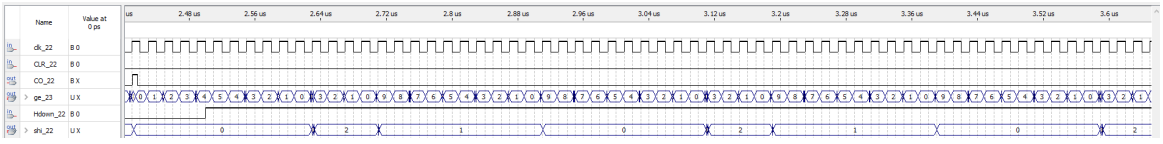

图 11-2 模 24 减计数波形仿真

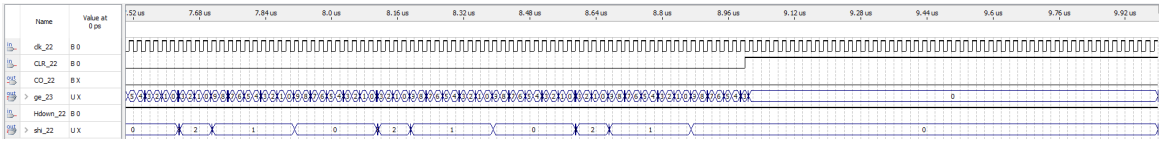

图 11-3 模 24 清零波形仿真

4. 数码管动态显示模块

4.1 动态显示模块的设计

动态显示模块主要由四个模块构成：扫描模块（cnt8）、位选模块（dig_select）、数据选择模块（sec_select）、译码模块（code_select）。由于在实验前期中并未透彻了解每个模块内容导致译码模块与数据选择模块命名易出现误解，所以本报告中尽量采用中文描述方式而非模块命名名称。

模块功能框图如图 12 所示：

扫描模块输入一个扫描频率，输出从 0 到 7 不断循环的位宽为三的二进制码（后省略位宽）到位选模块以及数据选择模块，以达成扫描时位选与段选同步的目的；

位选模块输入二进制码后，通过一个 3-8 译码器，选出当前对应的数码管，输出位选信号到各个引脚，即只有当前选中位为 0 其他位为 1，提高二进制码输入频率即可达成扫描效果，由于人眼具有“余晖效应”，观察到的数码管将会是常亮的；

数据选择模块输入段选信号、二进制码，根据二进制码进行数据选择，当选择到的数码管位选为 0 时，将其对应的段选信号选出并输出，且由于十进制数字由四位二进制码组成，所以段选信号的位宽为四；

译码模块将输入的四位段选信号进行译码，将对应的二进制码译为位宽为七的控制数码管各段亮灭的信号（若要加入小数点则位宽为八），并输出给各个引脚。

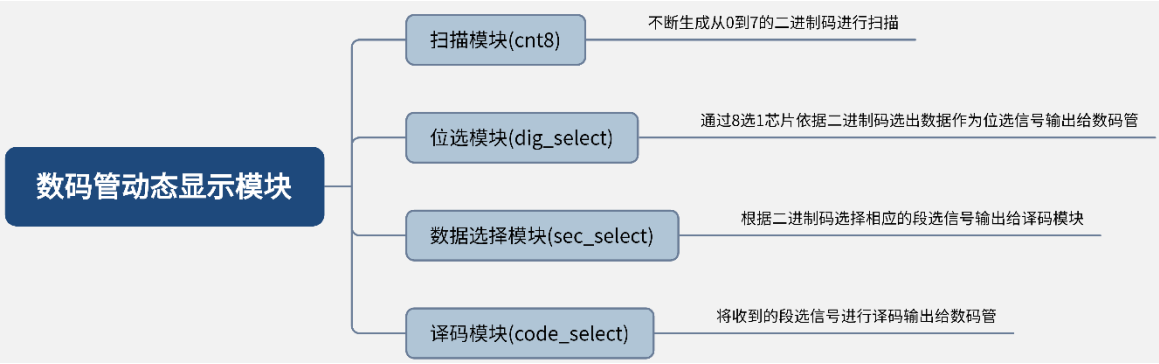

图 12 数码管动态显示模块功能框图

4.2 扫描模块

4.2.1 模块功能

扫描模块（cnt8）实现从 0 到 7 的计数，输入时钟为扫描频率，输出位宽为三的二进制码。

4.2.2 设计思路

利用异步双二-五-十进制加法计数器（74390）可完成从 0 到 7 的计数，并在 8（1000）时清零，以此往复循环生成从 0 到 7 对应的二进制码并输出，以达成不断扫描的效果。

4.2.3 设计结果

设计结果如图 13 所示。

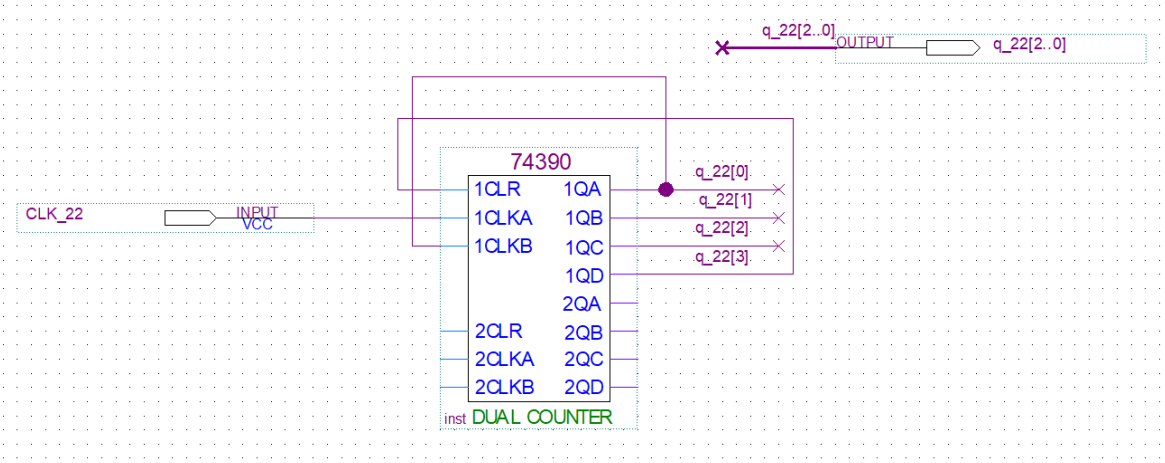

图 13 扫描模块（cnt8）电路设计

4.2.4 仿真测试

扫描模块波形图仿真，输入扫描频率（时钟脉冲），输出 0 到 7 的 8421BCD 码，如图 14 所示。

图 14 扫描模块（cnt8）波形图仿真

4.3 位选模块

4.3.1 模块功能

位选模块（dig_select）功能输入扫描模块的二进制码，通过 3-8 译码器选择需要开启的数码管，输出位宽为八的信号控制数码管的亮灭，由 11111110 到 01111111 进行扫描数码管。

4.3.2 设计思路

通过 3-8 译码器实现将二进制码转化为从 11111110 到 01111111 逐次左移的位选码。

4.3.3 设计结果

设计结果电路如图 15 所示。

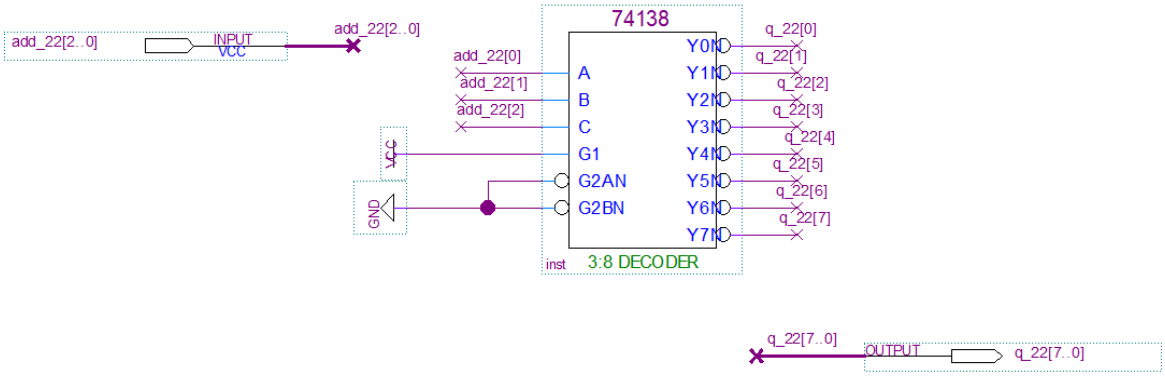

图 15 位选电路设计

4.3.4 仿真测试

输入由 0 到 7 的 8421BCD 码，输出当前选择的位选码，波形仿真结果如图 16 所示。

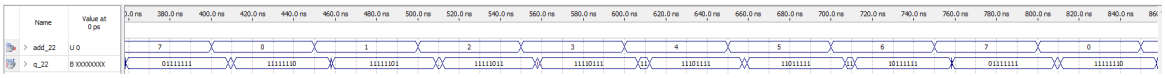

图 16 位选模块（dig_select）波形仿真

4.4 数据选择模块

4.4.1 模块功能

数据选择（sec_select）模块实现功能将秒（m60）、分（m60）、时（m24）计数器输入的 8421BCD 码根据扫描模块（cnt8）生成的二进制码，与位选模块（dig_select）相同步，再将对应的数码管的段选信号选择出来并输出给译码模块（code_select）。

4.4.2 设计思路

通过四片 8 选 1 数据选择器（74151），将输入的 8421BCD 码进行拆分，再根据扫描模块生成的二进制码各自选择一位数据，并将四位数据并为总线作为输出，输入的 8421BCD 码在 74151 上对应的位置与期望在数码管上显示的位置一致，即前文所提到的将位选模块与数据选择模块进行同步。

4.4.3 设计结果

数据选择模块电路设计如图 17 所示。

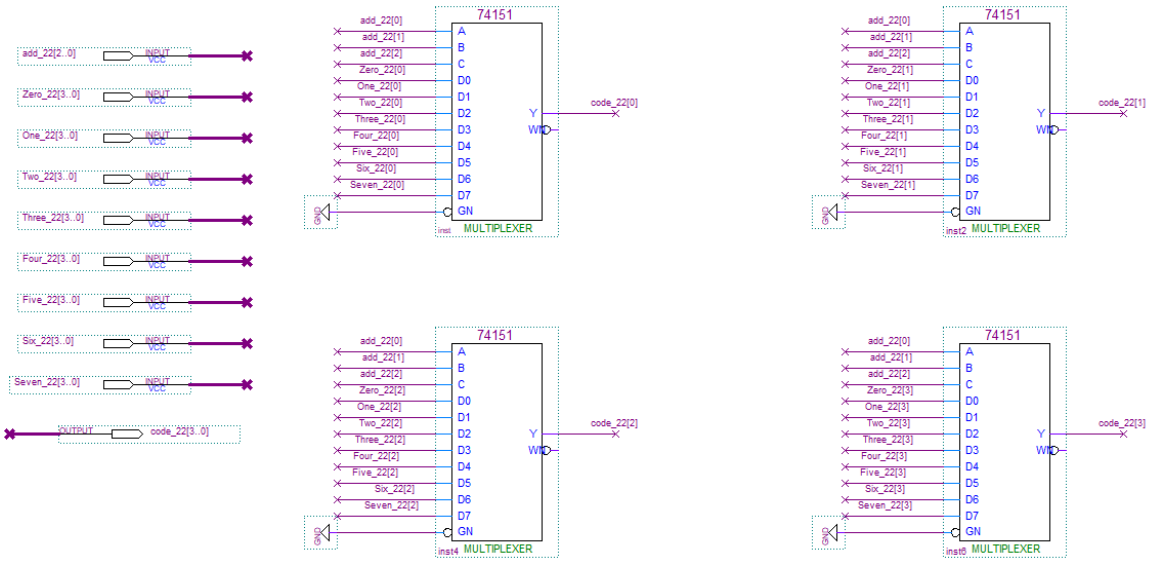

图 17 数据选择模块电路设计

4.4.4 仿真测试

输入由 0 到 7 的二进制码作为选择信号，再设置从四片 74151 上 D0 引脚到 D7 引脚共四位构成的总线初始值，观察选择出的数据，即 code 输出端，结果如图 18 所示。

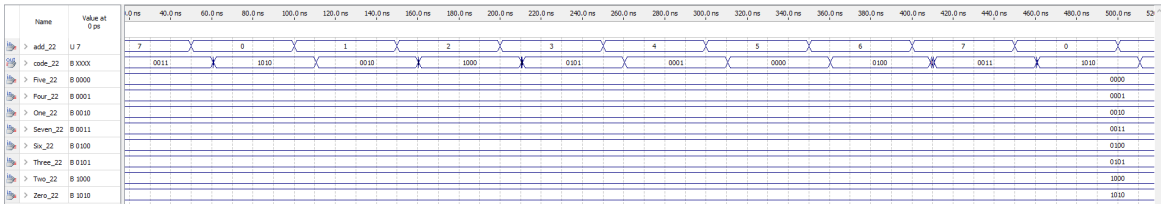

图 18 数据选择模块波形仿真

4.5 译码模块

4.5.1 模块功能

译码模块（code_select）将数据选择器选出的四位段选信号进行译码，将 8421BCD 码对应的数字译码为数字对应的数码管编码方式，并将其输出至数码管的各个对应引脚。

4.5.2 设计思路

共阴数码管为高有效，编码方式如表 3 所示（为 6、9 补段形式后）。由于表 3 编码方式为补段后编码，所以要根据 7448 功能表进行更改，进行补段输出，需要注意的是Q_a是低位Q_g是高位。

表 3 数码管对应数字编码方式

序号	Q_a	Q_b	Q_c	Q_d	Q_e	Q_f	Q_g
0	1	1	1	1	1	1	0
1	0	1	1	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

4.5.3 设计结果

设计结果电路如图 19 所示。

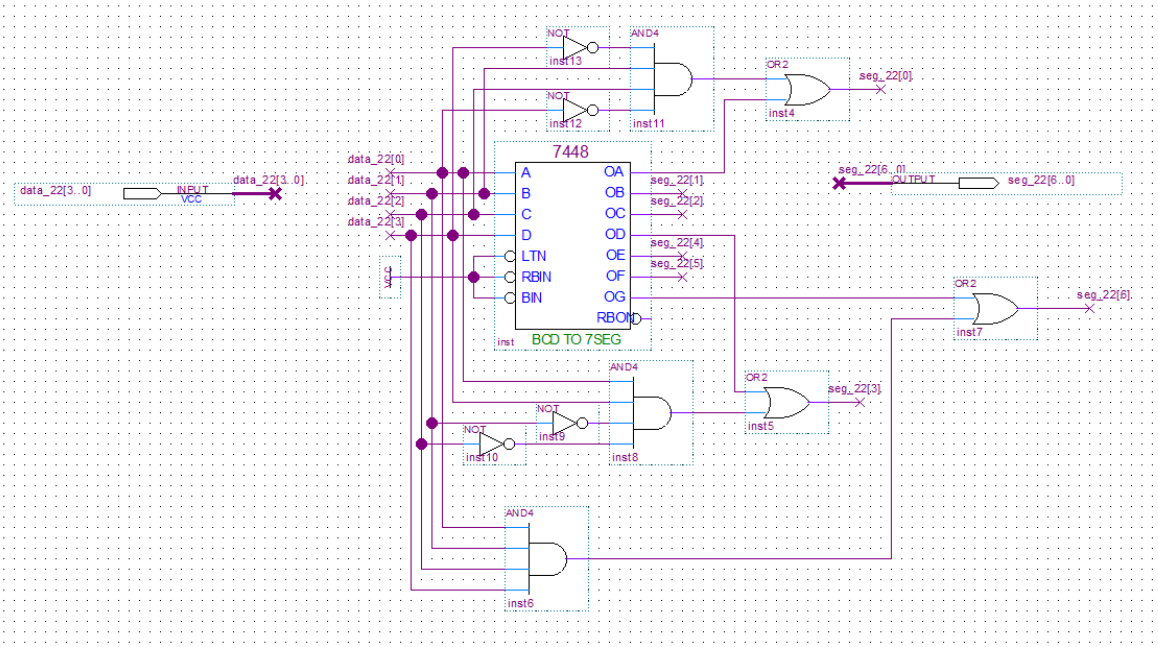

图 19 译码模块电路设计

4.5.4 仿真测试

输入所需要显示数字的 8421BCD 码，观察仿真波形，如图 20 所示。

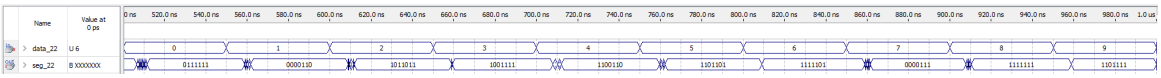

图 20 译码模块波形仿真

4.6 动态显示模块电路图

4.6.1 电路工作原理

动态显示模块工作原理，主要利用了人眼的视觉暂留效应，即“余晖效应”，使数码管以高频率扫描，以达到在人眼中呈现为常亮的状态，用摄影机、手机等设备拍摄数码管即可发现现实时数码管会闪烁，或是亮度呈脉冲扫描形式，从右往左进行扫描。

四个模块的主要工作方式：

由扫描模块进行主控，扫描模块生成的二进制码转换为十进制数字即为选中的那一位数码管，选择同一位数码管的位选及其段选（通过数据选择模块区分），当选中位的数码管亮起的同时，其段选信号也被发送出来，在当前数码管上进行显示，也就是将需要显示的数字，加快频率显示即为我们需要的数字钟。

4.6.2 实验现象

数码管从右往左按照输入的频率进行扫描，将两个模 60 与一个模 40 进行级联，输入 1Hz 的频率，通过人眼可以观察到常亮的数字钟，并以一秒为频率递增，正常进位。

4.6.3 方案分析

从结果观察以及波形仿真可知方案为正确可行的。

5. 其他拓展功能

5.1 调频功能

5.1.1 模块功能

调频模块（Speed_choose）实现选择输入时钟频率的功能，可以通过控制电路开关输出 1Hz 频率正常显示时间，也可以输出 5KHz 频率使时钟显示的速度加速。

5.1.2 设计思路

根据电路开关的高低电平选择输出一个引脚的信号，可以采用双 4 选 1 芯片（74153），将 B 接地，即 A 的高低电平起到选择作用，分别将 1Hz 和 5KHz 接入 1C0 和 1C1，故在低电平时输出 1Hz 的频率，高电平时输出 5KHz 的频率。

5.1.3 设计结果

设计结果电路如图 21 所示。

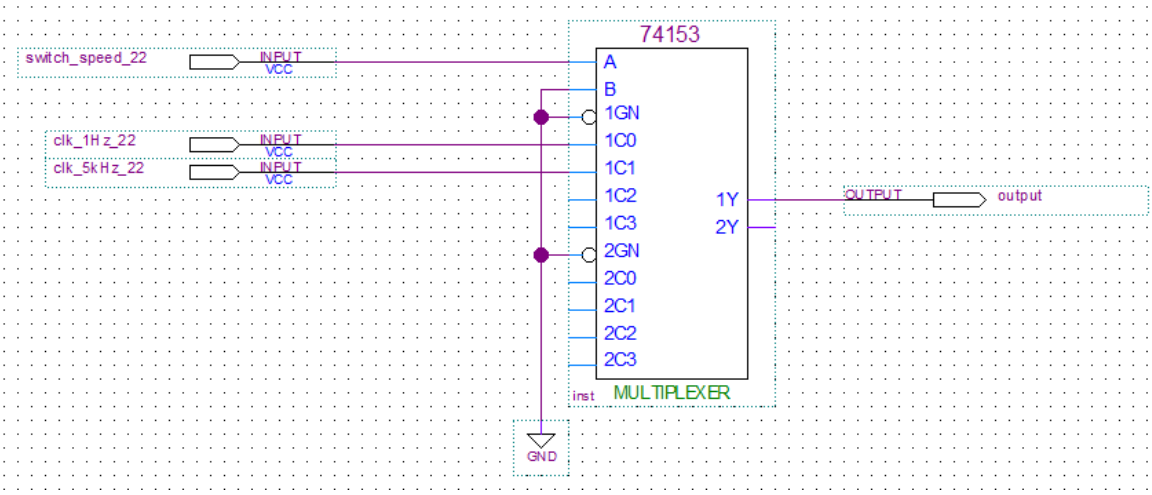

图 21 调频模块电路设计

5.1.4 仿真测试

仿真中无法同时输入 1Hz 与 5KHz 频率进行仿真（将 output 引脚改为 Fre_22 才能正常进行波形仿真），故采用既有区分性的的两个频率进行波形仿真，在 Switch_speed 为高电平时，输出频率更高的信号，为低电平时则正常输出频率较低的信号，仿真结果如图 22 所示。

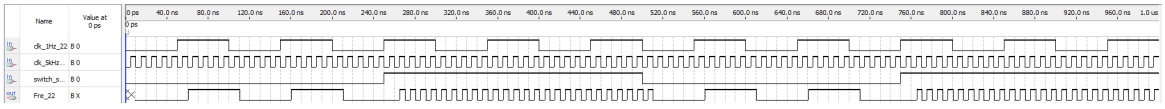

图 22 调频模块波形仿真

5.2 闹钟模式

5.2.1 模块功能

闹钟模块需要实现提前设置时间，并在时钟到达闹钟时间时响起音乐，并可通过控制电路开关控制闹钟的开关的功能，并且时间设置完成可加减功能。

5.2.2 设计思路

闹钟模式主要由以下三个板块构成：闹钟设置、比较模块、音乐模块。

考虑到音乐模块分频操作较为复杂繁琐，故使用 Verilog HDL 语言进行编写。

闹钟设置：将两个模 60 与一个模 24 级联，通过控制电路按钮作为时钟的脉冲计数，并通过电路开关控制加计数或减计数，输入位宽为三的由扫描模块生成的二进制码，并将模 60 与模 24 的输出接入数据选择模块（sec_select），该部分设计可参考基础时钟的显示；

比较模块：输入正常时钟的时、分、秒的个位与十位，以及设置闹钟的时、分、秒的个位与十位，将相对应的位数进行比较（可不设置秒数的比较，这样可使闹钟仅响一分钟且无需我们设计复位方案），若均相似则输出一个信号激活闹钟音乐。

音乐模块：音乐模块收到比较模块的高电平，并且由电路开关控制的闹钟处于打开状态时，播放音乐。音乐模块设计思路为输入一个 50MHz 的频率并进一步进行拆分，在已知各个音阶的频率的情况下如表 4 所示,可以通过如下公式算出各个音阶所占有的分频系数(音长计算同理)，

分频系数 = 时钟频率 / 2 / 音符频率……（1）

各个音符以及单个节拍音长使用 parameter 进行定义，定义一个计数寄存器 count，一个记录分频系数寄存器 count_end,一个节拍寄存器 countl,一个乐谱状态机 state,一个输出信号 beep_r。音乐模块主要由两个 always 语句组成：第一部分，当计数器 count 不等于 count_end 即音符对应的分频系数时，不断累加，每当计数器的 count 等于 count_end 时便使 beep_r 取反，可以用如下算式表示：

beep_r 取反频率 = 时钟频率 / 分频系数……（2）

而又由于 beep_r 为取反操作，激活蜂鸣器的频率应当减半，所以蜂鸣器的频率可由一下算式表示：

蜂鸣器频率 = 时钟频率 / 分频系数 / 2……（3）

不难得出式（1）与式（3）相等，即蜂鸣器频率 = 音符频率，此时蜂鸣器发出的被人耳听到的声音便是我们所需要的音符。第二部分，为乐谱部分，若 countl 节拍寄存器小于设定的单个音长的分频系数，则加 1 直到等于音长的分频系数，其代表着当前的音符发音持续时长，计算方法参考前文。每当 countl 到达一次 TIME，state 乐谱状态机加 1，也就是变为下一个存入的音符，此处操作 count_end 的值将我们需要的音符对应的分频系数赋值给他，便可完成音高的变化，将默认频率设为 0 即此时不发声，意为休止符。接下来就只需要将乐谱录入 case 语句中即可（乐谱如图 23 所示，来源于网络）。三连音本代码中处理为四分音符（一个 state），总的来说表现力也不错，同时也避免了进一步细化节奏带来的繁重工程量。

表 4 C 调音符与频率对照表

音符	频率（Hz）	音符	频率（Hz）	音符	频率（Hz）
低音 1	262	中音 1	523	高音 1	1046
低音 1#	277	中音 1#	554	高音 1#	1109
低音 2	294	中音 2	587	高音 2	1175
低音 2#	311	中音 2#	622	高音 2#	1245
低音 3	330	中音 3	659	高音 3	1318
低音 4	249	中音 4	698	高音 4	1397
低音 4#	370	中音 4#	740	高音 4#	1480
低音 5	392	中音 5	784	高音 5	1568
低音 5#	415	中音 5#	831	高音 5#	1661
低音 6	440	中音 6	880	高音 6	1760
低音 6#	466	中音 6#	932	高音 6#	1865
低音 7	494	中音 7	988	高音 7	1976

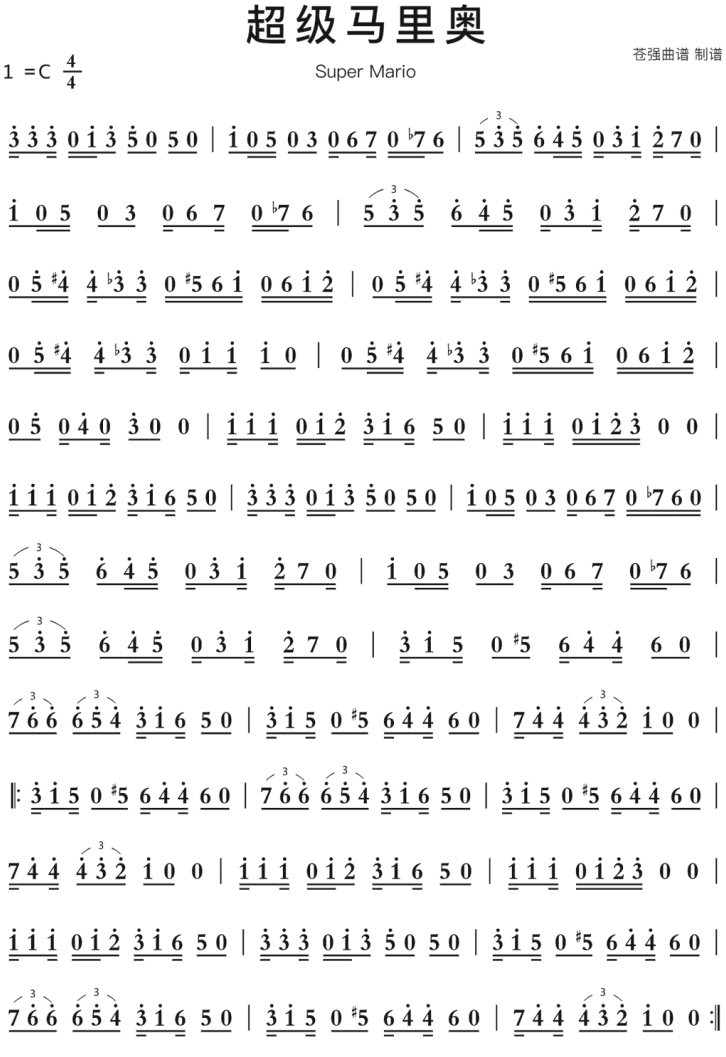

图 23 超级马里奥简谱乐谱

5.2.3 设计结果

闹钟设定模块（Set_alarm）设计结果电路如图 24 所示。

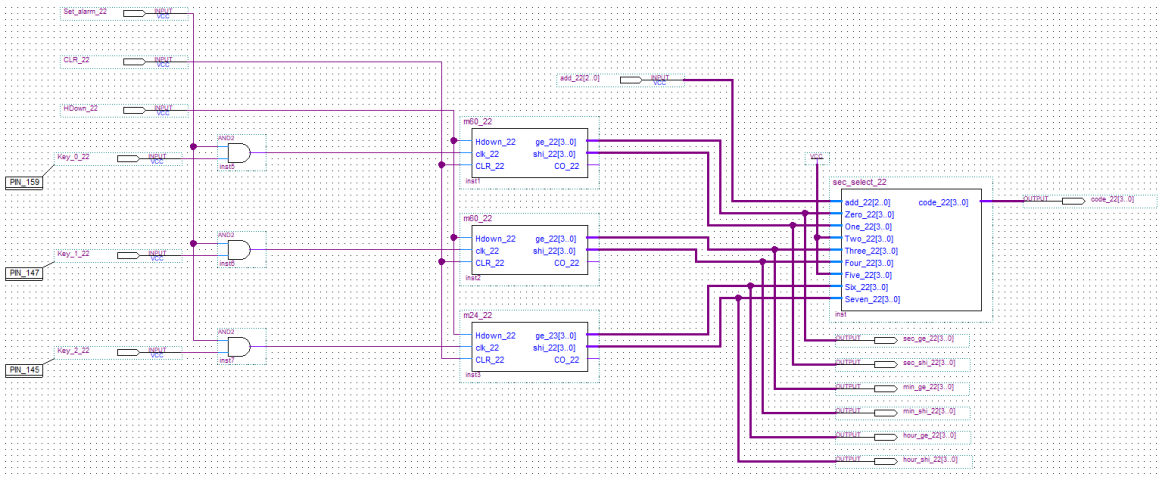

图 24 闹钟设定模块电路设计

比较模块采用了 Verilog HDL 语言以增强工程的可读性，减少放置大量的同或门等进行比较，设计结果代码如代码块 1 所示。

代码块 1 比较模块代码设计

```
1.  module  Compare_22(clk50MHz_22,Asec_ge_22,Asec_shi_22,Amin_ge_22,Amin_s
    hi_22,Ahour_ge_22,Ahour_shi_22,Bsec_ge_22,Bsec_shi_22,Bmin_ge_22,Bmin_s
    hi_22,Bhour_ge_22,Bhour_shi_22,OFF_22,alarm_22);
2.  input[3:0] Asec_ge_22;
3.  input[3:0] Asec_shi_22;
4.  input[3:0] Amin_ge_22;
5.  input[3:0] Amin_shi_22;
6.  input[3:0] Ahour_ge_22;
7.  input[3:0] Ahour_shi_22;
8.  input[3:0] Bsec_ge_22;
9.  input[3:0] Bsec_shi_22;
10. input[3:0] Bmin_ge_22;
11. input[3:0] Bmin_shi_22;
12. input[3:0] Bhour_ge_22;
13. input[3:0] Bhour_shi_22;
14. input clk50MHz_22;
15. input OFF_22;
16.
17. output alarm_22;
18.
19. reg     alarm;
20. assign alarm_22 = alarm;
21.
22. always@(posedge clk50MHz_22)begin
```

```
if(OFF_22==0)begin
    if(Amin_ge_22==Bmin_ge_22)begin
        if(Amin_shi_22==Bmin_shi_22)begin
            if(Ahour_ge_22==Bhour_ge_22)begin
                if(Ahour_shi_22==Bhour_shi_22)begin
                    if(Asec_ge_22==Bsec_ge_22)begin
                        if(Asec_shi_22==Bsec_shi_22)begin
                            alarm <= 1'b1;
                        end
                    end
                end
            end
         end
     end
else begin
    alarm <= 1'b0;
end
end
else begin
        alarm <= 1'b0;
end
end
endmodule
```

音乐模块设计如代码 2 所示。

代码 2 音乐模块代码设计

```
module  music_22(clk50MHz_22,beep_22);  //模块名称 song
input      clk50MHz_22;                                    //系统时钟 50MHz
output  beep_22;                                           //蜂鸣器输出端
reg             beep_r;                                    //寄存器
reg[15:0] state;                                           //乐谱状态机
reg[16:0]count,count_end;
reg[23:0]count1;
//乐谱参数:D=F/2K  (D:参数,F:时钟频率,K:音高频率)
parameter   L_3 = 17'd75850,  //低音 3
L_5 = 17'd63776,  //低音 5
L_6b= 17'd60241,  //低音 6 降调
L_6 = 17'd56818,    //低音 6
L_7b= 17'd53648,  //低音 7 降调
L_7 = 17'd50618,        //低音 7
M_1 = 17'd47774,        //中音 1
M_2 = 17'd42568,        //中音 2
M_3b= 17'd40193,  //中音 3 降调
M_3 = 17'd37919,        //中音 3
```

```
M_4 = 17'd35817,  //中音4
M_5b= 17'd33784,  //中音4升调
M_5 = 17'd31888,        //中音5
M_6b= 17'd30084,  //中音5升调
M_6 = 17'd28409,        //中音6
H_1 = 17'd23889;        //高音1
parameter       TIME = 6000000; //控制每一个音的长短(250ms/2)
assign beep_22 = beep_r;                          //输出音乐
always@(posedge clk50MHz_22) begin
count <= count + 1'b1;          //计数器加1
if(count == count_end) begin
count <= 17'h0;                 //计数器清零
beep_r <= !beep_r;              //输出取反
end
end

//曲谱 产生分频的系数并描述出曲谱
always @(posedge clk50MHz_22) begin
if(count1 < TIME)             //一个节拍250mS/2
count1 = count1 + 1'b1;
else begin
count1 = 24'd0;
if(state == 16'd707)
state = 16'd0;
else
state = state + 1'b1;
case(state)
//1.1
16'd0:count_end = M_3;
16'd1,16'd2:count_end = M_3;
16'd3:count_end = M_3;
//,
16'd5:count_end = M_1;
16'd6,16'd7:count_end = M_3;
16'd8,16'd9:count_end = M_5;
//,,
16'd12,16'd13:count_end = L_5;
//,,
//1.2
16'd16,16'd17:count_end = M_1;
//,
16'd19:count_end = L_5;
//,,
```

```
16'd22,16'd23:count_end = L_3;
//,
16'd25,16'd26:count_end = L_6;
16'd27:count_end = L_7;
//,
16'd29:count_end = L_7b;
16'd30,16'd31:count_end = L_6;
//1.3
16'd32:count_end = L_5;
16'd33:count_end = M_3;
16'd34:count_end = M_5;
16'd35,16'd36:count_end = M_6;
16'd37:count_end = M_4; 16'd38:count_end = M_5;
//,
16'd40,16'd41:count_end = M_3;
16'd42:count_end = M_1;
16'd43:count_end = M_2;
16'd44,16'd45:count_end = L_7;
//,
//2.1
16'd47,16'd48:count_end = M_1;
//,
16'd49:count_end = L_5;
//,,
16'd52,16'd53:count_end = L_3;
//,
16'd55,16'd56:count_end = L_6;
16'd57:count_end = L_7;
//,
16'd59:count_end = L_7b;
16'd60, 16'd61:count_end = L_6;
//2.2
16'd62:count_end = L_5;
16'd63:count_end = M_3;
16'd64:count_end = M_5;
16'd65,16'd66:count_end = M_6;
16'd67:count_end = M_4;
16'd68:count_end = M_5;
//,
   16'd70, 16'd71:count_end = M_3;
   16'd72:count_end = M_1;
   16'd73:count_end = M_2;
   16'd74,16'd75:count_end = L_7;
   //,
```

```
105.    //3.1
106.    //,,
107.    16'd79:count_end = M_5;
108.    16'd80:count_end = M_5b;
109.    16'd81:count_end = M_4;
110.    16'd82,16'd83:count_end = M_3b;
111.    16'd84:count_end = M_3;
112.    //,
113.    16'd86:count_end = M_6b;
114.    16'd87:count_end = L_6;
115.    16'd88:count_end = M_1;
116.    //,
117.    16'd90:count_end = L_6;
118.    16'd91:count_end = M_1;
119.    16'd92:count_end = M_2;
120.    //3.2
121.    //,,
122.    16'd95:count_end = M_5;
123.    16'd96:count_end = M_5b;
124.    16'd97:count_end = M_4;
125.    16'd98,16'd99:count_end = M_3b;
126.    16'd100:count_end = M_3;
127.    //,
128.    16'd102:count_end = M_6b;
129.    16'd103:count_end = L_6;
130.    16'd104:count_end = M_1;
131.    //,
132.    16'd106:count_end = L_6;
133.    16'd107:count_end = M_1;
134.    16'd108:count_end = M_2;
135.    //4.1
136.    //,,
137.    16'd111:count_end = M_5;
138.    16'd112:count_end = M_5b;
139.    16'd113:count_end = M_4;
140.    16'd114,16'd115:count_end = M_3b;
141.    16'd116:count_end = M_3;
142.    //,
143.    16'd117,16'd118:count_end = M_1;
144.    16'd119:count_end = M_1;
145.    16'd120,16'd121:count_end = M_1;
146.    //,,
147.    //4.2
148.    //,,
```

```
149.    16'd127:count_end = M_5;
150.    16'd128:count_end = M_5b;
151.    16'd129:count_end = M_4;
152.    16'd130,16'd131:count_end = M_3b;
153.    16'd132:count_end = M_3;
154.    //,
155.    16'd134:count_end = M_6b;
156.    16'd135:count_end = L_6;
157.    16'd136:count_end = M_1;
158.    //,
159.    16'd138:count_end = L_6;
160.    16'd139:count_end = M_1;
161.    16'd140:count_end = M_2;
162.    //5.1
163.    //,,
164.    16'd143,16'd144:count_end = M_5;
165.    //,
166.    16'd146,16'd147:count_end = M_4;
167.    //,
168.    16'd149,16'd150:count_end = M_3;
169.    //,,
170.    //,,,,
171.    //5.2
172.    16'd157:count_end = M_1;
173.    16'd158,16'd159:count_end = M_1;
174.    16'd160:count_end = M_1;
175.    //,
176.    16'd162:count_end = M_1;
177.    16'd163,16'd164:count_end = M_2;
178.    16'd165:count_end = M_3;
179.    16'd166,16'd167:count_end = M_1;
180.    16'd168:count_end = L_6;
181.    16'd169,16'd170:count_end = L_5;
182.    //,,
183.    //5.3
184.    16'd173:count_end = M_1;
185.    16'd174,16'd175:count_end = M_1;
186.    16'd176:count_end = M_1;
187.    //,
188.    16'd178:count_end = M_1;
189.    16'd179:count_end = M_2;
190.    16'd180:count_end = M_3;
191.    //,,,,
192.    //,,,,
```

```
193.    //6.1
194.    16'd189:count_end = M_1;
195.    16'd190,16'd191:count_end = M_1;
196.    16'd192:count_end = M_1;
197.    //,
198.    16'd194:count_end = M_1;
199.    16'd195,16'd196:count_end = M_2;
200.    16'd197:count_end = M_3;
201.    16'd198,16'd199:count_end = M_1;
202.    16'd200:count_end = L_6;
203.    16'd201,16'd202:count_end = L_5;
204.    //,,
205.    //6.2
206.    16'd205:count_end = M_3;
207.    16'd206,16'd207:count_end = M_3;
208.    16'd208:count_end = M_3;
209.    //,
210.    16'd210:count_end = M_1;
211.    16'd211,16'd212:count_end = M_3;
212.    16'd213,16'd214:count_end = M_5;
213.    //,,
214.    16'd217,16'd218:count_end = L_5;
215.    //,,
216.    //6.3
217.    16'd221,16'd222:count_end = M_1;
218.    //,
219.    16'd224:count_end = L_5;
220.    //,,
221.    16'd227,16'd228:count_end = L_3;
222.    //,
223.    16'd230,16'd231:count_end = L_6;
224.    16'd232:count_end = L_7;
225.    //,
226.    16'd234:count_end = L_7b;
227.    16'd235:count_end = L_6;
228.    //,
229.    //7.1
230.    16'd237:count_end = L_5;
231.    16'd238:count_end = M_3;
232.    16'd239:count_end = M_5;
233.    16'd240,16'd241:count_end = M_6;
234.    16'd242:count_end = M_4;
235.    16'd243:count_end = M_5;
236.    //,
```

```
237.    16'd245,16'd246:count_end = M_3;
238.    16'd247:count_end = M_1;
239.    16'd248:count_end = M_2;
240.    16'd249,16'd250:count_end = L_7;
241.    //,
242.    //7.2
243.    16'd252,16'd253:count_end = M_1;
244.    //,
245.    16'd255:count_end = L_5;
246.    //,,
247.    16'd258,16'd259:count_end = L_3;
248.    //,
249.    16'd261,16'd262:count_end = L_6;
250.    16'd263:count_end = L_7;
251.    //,
252.    16'd265:count_end = L_7b;
253.    16'd266,16'd267:count_end = L_6;
254.    //8.1
255.    16'd268:count_end = L_5;
256.    16'd269:count_end = M_3;
257.    16'd270:count_end = M_5;
258.    16'd271,16'd272:count_end = M_6;
259.    16'd273:count_end = M_4;
260.    16'd274:count_end = M_5;
261.    //,
262.    16'd276,16'd277:count_end = M_3;
263.    16'd278:count_end = M_1;
264.    16'd279:count_end = M_2;
265.    16'd280,16'd281:count_end = L_7;
266.    //,
267.    //8.2
268.    16'd283:count_end = M_3;
269.    16'd284,16'd285:count_end = M_1;
270.    16'd286:count_end = L_5;
271.    //,,
272.    16'd289,16'd290:count_end = L_6b;
273.    16'd291,16'd292:count_end = M_4;
274.    16'd293:count_end = M_4;
275.    16'd294,16'd295:count_end = L_6;
276.    //,,
277.    //9.1
278.    16'd298:count_end = L_7;
279.    16'd299:count_end = M_6;
280.    16'd300:count_end = M_6;
```

```
    16'd301:count_end = M_6;
    16'd302:count_end = M_5;
    16'd303:count_end = M_4;
    16'd304:count_end = M_3;
    16'd305,16'd306:count_end = M_1;
    16'd307:count_end = L_6;
    16'd308,16'd309:count_end = L_5;
    //,,
    //9.2
    16'd312:count_end = M_3;
    16'd313,16'd314:count_end = M_1;
    16'd315:count_end = L_5;
    //,,
    16'd318,16'd319:count_end = L_6b;
    16'd320:count_end = L_6;
    16'd321,16'd322:count_end = M_4;
    16'd323:count_end = M_4;
    16'd324,16'd325:count_end = L_6;
    //,,
    //9.3
    16'd328:count_end = L_7;
    16'd329,16'd330:count_end = M_4;
    16'd331:count_end = M_4;
    16'd332:count_end = M_4;
    16'd333:count_end = M_3;
    16'd334:count_end = M_2;
    16'd335:count_end = M_1;
    //,,
    //,,,,
    //||:
    //10.1
    16'd342:count_end = M_3;
    16'd343,16'd344:count_end = M_1;
    16'd345:count_end = L_5;
    //,,
    16'd348,16'd349:count_end = L_6b;
    16'd350:count_end = L_6;
    16'd351,16'd352:count_end = M_4;
    16'd353:count_end = M_4;
    16'd354,16'd355:count_end = L_6;
    //,,
    //10.2
    16'd358:count_end = L_7;
    16'd359:count_end = M_6;
```

```
    16'd360:count_end = M_6;
    16'd361:count_end = M_6;
    16'd362:count_end = M_5;
    16'd363:count_end = M_4;
    16'd364:count_end = M_3;
    16'd365,16'd366:count_end = M_1;
    16'd367:count_end = L_6;
    16'd368,16'd369:count_end = L_5;
    //,,
    //10.3
    16'd372:count_end = M_3;
    16'd373,16'd374:count_end = M_1;
    16'd375:count_end = L_5;
    //,,
    16'd378,16'd379:count_end = L_6b;
    16'd380:count_end = L_6;
    16'd381,16'd382:count_end = M_4;
    16'd383:count_end = M_4;
    16'd384,16'd385:count_end = L_6;
    //,,
    //11.1
    16'd388:count_end = L_7;
    16'd389,16'd390:count_end = M_4;
    16'd391:count_end = M_4;
    16'd392:count_end = M_4;
    16'd393:count_end = M_3;
    16'd394:count_end = M_2;
    16'd395,16'd396:count_end = M_1;
    //,,
    //,,,,
    //11.2
    16'd403:count_end = M_1;
    16'd404,16'd405:count_end = M_1;
    16'd406:count_end = M_1;
    //,(,)
    16'd409:count_end = M_1;
    16'd410,16'd411:count_end = M_2;
    16'd412:count_end = M_3;
    16'd413,16'd414:count_end = M_1;
    16'd415:count_end = L_6;
    16'd416,16'd417:count_end = L_5;
    //,,
    //11.3
    16'd420:count_end = M_1;
```

```
    16'd421,16'd422:count_end = M_1;
    16'd423:count_end = M_1;
    //,
    16'd425:count_end = M_1;
    16'd426:count_end = M_2;
    16'd427:count_end = M_3;
    //,,
    //,,
    //12.1
    16'd432:count_end = M_1;
    16'd433,16'd434:count_end = M_1;
    16'd435:count_end = M_1;
    //,
    16'd437:count_end = M_1;
    16'd438,16'd439:count_end = M_2;
    16'd440:count_end = M_3;
    16'd441,16'd442:count_end = M_1;
    16'd443:count_end = L_6;
    16'd444,16'd445:count_end = L_5;
    //,,
    //12.2
    16'd448:count_end = M_3;
    16'd449,16'd450:count_end = M_3;
    16'd451:count_end = M_3;
    //,
    16'd453:count_end = M_1;
    16'd454,16'd455:count_end = M_3;
    16'd456,16'd457:count_end = M_5;
    //,,
    16'd460,16'd461:count_end = L_5;
    //,,
    //12.3
    16'd464:count_end = M_3;
    16'd465,16'd466:count_end = M_1;
    16'd467:count_end = L_5;
    //,,
    16'd470,16'd471:count_end = L_6b;
    16'd472:count_end = L_6;
    16'd473,16'd474:count_end = M_4;
    16'd475:count_end = M_4;
    16'd476,16'd477:count_end = L_6;
    //,,
    //13.1
    16'd480:count_end = L_7;
```

```
413.    16'd481:count_end = M_6;
414.    16'd482:count_end = M_6;
415.    16'd483:count_end = M_6;
416.    16'd484:count_end = M_5;
417.    16'd485:count_end = M_4;
418.    16'd486:count_end = M_3;
419.    16'd487,16'd488:count_end = M_1;
420.    16'd489:count_end = L_6;
421.    16'd490,16'd491:count_end = L_5;
422.    //,,
423.    //13.2
424.    16'd494:count_end = M_3;
425.    16'd495,16'd496:count_end = M_1;
426.    16'd497:count_end = L_5;
427.    //,,
428.    16'd500,16'd501:count_end = L_6b;
429.    16'd502,16'd503:count_end = M_4;
430.    16'd504:count_end = M_4;
431.    16'd505,16'd506:count_end = L_6;
432.    //,,
433.    //13.3
434.    16'd509:count_end = L_7;
435.    16'd510,16'd511:count_end = M_4;
436.    16'd512:count_end = M_4;
437.    16'd513:count_end = M_4;
438.    16'd514:count_end = M_3;
439.    16'd515:count_end = M_2;
440.    16'd516,16'd517:count_end = M_1;
441.    //,(,)
442.    //,,,,
443.    //:||
444.    //10.1
445.    16'd524:count_end = M_3;
446.    16'd525,16'd526:count_end = M_1;
447.    16'd527:count_end = L_5;
448.    //,,
449.    16'd530,16'd531:count_end = L_6b;
450.    16'd532:count_end = L_6;
451.    16'd533,16'd534:count_end = M_4;
452.    16'd535:count_end = M_4;
453.    16'd536,16'd537:count_end = L_6;
454.    //,,
455.    //10.2
456.    16'd540:count_end = L_7;
```

```
457.    16'd541:count_end = M_6;
458.    16'd542:count_end = M_6;
459.    16'd543:count_end = M_6;
460.    16'd544:count_end = M_5;
461.    16'd545:count_end = M_4;
462.    16'd546:count_end = M_3;
463.    16'd547,16'd548:count_end = M_1;
464.    16'd549:count_end = L_6;
465.    16'd550,16'd551:count_end = L_5;
466.    //,,
467.    //10.3
468.    16'd554:count_end = M_3;
469.    16'd555,16'd556:count_end = M_1;
470.    16'd557:count_end = L_5;
471.    //,,
472.    16'd560,16'd561:count_end = L_6b;
473.    16'd562:count_end = L_6;
474.    16'd563,16'd564:count_end = M_4;
475.    16'd565:count_end = M_4;
476.    16'd566,16'd567:count_end = L_6;
477.    //,,
478.    //11.1
479.    16'd570:count_end = L_7;
480.    16'd571,16'd572:count_end = M_4;
481.    16'd573:count_end = M_4;
482.    16'd574:count_end = M_4;
483.    16'd575:count_end = M_3;
484.    16'd576:count_end = M_2;
485.    16'd577,16'd578:count_end = M_1;
486.    //,,
487.    //,,,,
488.    //11.2
489.    16'd585:count_end = M_1;
490.    16'd586,16'd587:count_end = M_1;
491.    16'd588:count_end = M_1;
492.    //,
493.    16'd590:count_end = M_1;
494.    16'd591,16'd592:count_end = M_2;
495.    16'd593:count_end = M_3;
496.    16'd594,16'd595:count_end = M_1;
497.    16'd596:count_end = L_6;
498.    16'd597,16'd598:count_end = L_5;
499.    //,,
500.    //11.3
```

```
501.    16'd601:count_end = M_1;
502.    16'd602,16'd603:count_end = M_1;
503.    16'd604:count_end = M_1;
504.    //,(,)
505.    16'd607:count_end = M_1;
506.    16'd608:count_end = M_2;
507.    16'd609:count_end = M_3;
508.    //,,
509.    //,,
510.    //12.1
511.    16'd614:count_end = M_1;
512.    16'd615,16'd616:count_end = M_1;
513.    16'd617:count_end = M_1;
514.    //,
515.    16'd619:count_end = M_1;
516.    16'd620,16'd621:count_end = M_2;
517.    16'd622:count_end = M_3;
518.    16'd623,16'd624:count_end = M_1;
519.    16'd625:count_end = L_6;
520.    16'd626,16'd627:count_end = L_5;
521.    //,,
522.    //12.2
523.    16'd630:count_end = M_3;
524.    16'd631,16'd632:count_end = M_3;
525.    16'd633:count_end = M_3;
526.    //,
527.    16'd635:count_end = M_1;
528.    16'd636,16'd637:count_end = M_3;
529.    16'd638,16'd639:count_end = M_5;
530.    //,,
531.    16'd642,16'd643:count_end = L_5;
532.    //,,
533.    //12.3
534.    16'd646:count_end = M_3;
535.    16'd647,16'd648:count_end = M_1;
536.    16'd649:count_end = L_5;
537.    //,,
538.    16'd652,16'd653:count_end = L_6b;
539.    16'd654:count_end = L_6;
540.    16'd655,16'd656:count_end = M_4;
541.    16'd657:count_end = M_4;
542.    16'd658,16'd659:count_end = L_6;
543.    //,,
544.    //13.1
```

```
    16'd662:count_end = L_7;
    16'd663:count_end = M_6;
    16'd664:count_end = M_6;
    16'd665:count_end = M_6;
    16'd666:count_end = M_5;
    16'd667:count_end = M_4;
    16'd668:count_end = M_3;
    16'd669,16'd670:count_end = M_1;
    16'd671:count_end = L_6;
    16'd672,16'd673:count_end = L_5;
    //,,
    //13.2
    16'd673:count_end = M_3;
    16'd677,16'd678:count_end = M_1;
    16'd679:count_end = L_5;
    //,,
    16'd682,16'd683:count_end = L_6b;
    16'd684,16'd685:count_end = M_4;
    16'd686:count_end = M_4;
    16'd687,16'd688:count_end = L_6;
    //,,
    //13.3
    16'd691:count_end = L_7;
    16'd692,16'd693:count_end = M_4;
    16'd694:count_end = M_4;
    16'd695:count_end = M_4;
    16'd696:count_end = M_3;
    16'd697:count_end = M_2;
    16'd698,16'd699:count_end = M_1;
    //,(.)
    //,,,,
    default: count_end = 16'h0;
    endcase
    end
    end
    endmodule
```

5.2.4 仿真测试

本模块使用软件仿真过于复杂，采用实物检测的方式进行检查。

5.3 游戏模式

5.3.1 模块功能

游戏模式主要实现功能通过控制电路按钮操控主角上下移动，并发射子弹攻击不断撞向主角的敌人，若被敌人击中则损失血量，血量为 0 时则游戏停止，游戏停止时可以通过控制电路按钮使游戏重启，角色复活，另外可通过电路开关控制游戏背景音乐的开关。

按模块功能分类主要有以下 14 部分模块：

1) 各类分频模块：根据输入的 50MHz 频率分为各个部分模块所需要的频率；
2) 主角控制模块：通过控制电路按钮可以使主角在最左侧的数码管（第 7 位数码管）上下移动；
3) 攻击模块：通过控制电路按钮可以是主角发射子弹，生成在第 6 位数码管上；
4) 攻击右移模块：从第 5 位数码管开始，一旦上一位数码（级联方式为 6-5-4-3-2-1-0）管出现了攻击信号，便复制上一位的信号并输出并同时输出一个清零信号给上一位，以形成右移效果，第 0 位数码管将自己的清零信号通过 D 触发器使信号延长一倍，以使自己在下一帧被清零。通过输入的频率以确认右移速度；
5) 敌人位置生成模块：敌人的位置可能在上可能在下，要尽可能做到随机生成，或是使有序循环的周期变大，以尽可能随机地生成敌人的上或下的位置信号。
6) 敌人生成模块：根据敌人位置生成模块生成的敌人位置信号，在第 0 位数码管生成敌人；
7) 敌人左移模块：从第 1 位数码管开始，一旦上一位数码管（级联方式为 0-1-2-3-4-5-6-7）出现敌人信号，便复制上一位信号的信号输出并同输出一个清零信号给上一位，以达成左移的效果，第 7 位数码管清零方式参考攻击右移模块。通过输入的频率可确认左移速度；
8) 敌人消除模块：根据伤害判定模块数据输出值与原敌人左移数据值相与，可以将比较模块中为 0 的值清空并阻止其继续左移；
9) 血量及重置模块：主角每当与敌人在数码管同上或同下时，收到伤害，血量指示灯减 1，血量指示灯总共为 3，全部熄灭则游戏停止；
10) 伤害判定模块：每当攻击与敌人相遇则将当前数码管的段选信号置为 0；（此方案具有 bug，在第七板块将会详细说明）
11) 扫描模块：功能同基础时钟；
12) 数据选择模块：功能同基础时钟；
13) 位选模块：功能同基础时钟；
14) 译码模块：对主角/敌人/子弹可能存在的各种情况进行编码，其他模块根据编码方式进

行生成位宽为 4 的段选信号发送给译码模块将，译码模块再将其译码为所需要的图案对应的在数码管上的码值。

5.3.2 设计思路

由于板块较多设计思路将逐个讨论：

1) 各类分频模块：

设计同基础时钟当中的分频模块，以及模 100 模块；

2) 主角控制模块：

模块（Actor）每收到一次来自电路按钮的上升沿信号，便将上下状态反转，参照表 5 可知为段选信号在 0001 与 0010 间来回切换，并将该信号输出；

3) 攻击模块：

攻击模块（Atack）获取主角控制模块的段选信号并通过其反解出主角的位置在上或是在下，若对应的输入电路按钮为高电平便在第 6 位数码管对应主角位置生成输出攻击信号，在上方则是 0011，在下方则是 0100，每当收到一次复位清零信号则输出为 0000；

4) 攻击右移模块：

攻击右移模块（AtackToNext）输入上一位数码管（级联方式 6-5-4-3-2-1-0）的信号，以及下一位数码管传来的的清零信号。每当右移频率的上升沿且清零信号为 0，则将上一位的信号赋值到当前数码管并将其输出，同时输出一个清零上一位数码管的清零信号。若右移信号到达第 0 位数码管，则将其本身的清零信号通过一个 D 触发器进行延迟，在下一帧清零；

5) 敌人位置生成模块：

敌人位置生成模块（SummonEnemy）定义一个位宽为 16 的状态机，并将最低位输出，每当时钟（生成频率）上升沿便使状态机右移 1 位，当状态机为 0 时重新赋值；（该部分代码存在部分问题将在第七部分详细说明）

6) 敌人生成模块：

敌人位置生成模块（Enemy）若清零信号为 0，每当时钟（生成频率）输入，敌人位置生成模块生成的信号（1 代表在上方，0 代表在下方），在第 0 位数码管上生成敌人；

7) 敌人左移模块：

敌人左移模块（EnemyToNext）输入上一位数码管（级联方式 0-1-2-3-4-5-6-7）的信号，若下一位数码管传来的清零信号为 0，则正常工作不清零，将上一位数码管的信号复制并传输给下一位数码管，同时输出一个清零信号给上一位数码管，第 7 为数码管敌人清空方式参考攻击右移模块第 0 位数码管清空方式；

8) 敌人消除模块：

敌人消除模块（Combine）将敌人左移信号和进入比较模块后输出的敌人左移信号相与，作

为新的左移信号输出给左移模块，以将在比较模块中被清 0 的信号阻断（防止其信号继续向左传输），若要消除子弹可参考此方式进行设计；

9） 血量及重置模块：

血量模块（Heal_LED）使用 74390 达成模 3 的功能，将伤害判定模块输出的 HIT 作为受伤信号，并内置一个译码模块（Heal）8421BCD 码译为控制三个 LED 亮灭的高低电平，若三个 LED 均为低电平则判定为角色死亡输出 Dead 信号为 1。当 Dead 为 1，且对应的电路按钮输入为 1 则将其相与并和 HIT 信号相或作为模 3 的触发信号，则将触发 74390 其状态变为 0100 清零，则三盏 LED 重新亮起，Dead 信号也自动变为 0，同时为了防止复活按钮（Reset）在角色为死亡时产生影响，加入一个保险模块（Enab），将 Dead 信号与 Reset 信号合并为位宽为 2 的信号，当且仅当信号为 10 时，将游戏使能端口关闭；

10）伤害判定模块：

每当同一数码管上同时输入攻击在上且敌人在上，或攻击在下且敌人在下的信号时，将此数码管信号置为 0 并输出。若攻击与敌人错开则将对应的信号输出，主角与敌人错开同理。若敌人与敌人在同一数码管上的同一位置，则输出一个 HIT 信号；（设计思路问题在第七部分详细说明）

11）扫描模块：

设计同基础时钟，此处省略；

12）数据选择模块：

设计同基础时钟，此处省略；

13）位选模块：

设计同基础时钟，此处省略；

14）译码模块： 首先设计主角/敌人/子弹在数码管上的显示方式，再将各类情况进行编码如表 5 所示，所有的段选信号以表 5 为基础进行设计，并通过译码模块译码为在数码管上显示图案所对应的码值。

表 5 各类情况编码对照表

情况	编码	数码管码（g-a）
空白	0000	0000000
主角/敌人在上方	0001	1100011
主角/敌人在下方	0010	1011100
攻击在上方	0011	1000001
攻击在下方	0100	1001000
主角/敌人在上、攻击在下	0101	1101011
主角/敌人在下、攻击在上	0110	1011101
主角与敌人错开	0111	1111111

5.3.3 设计结果

由于板块较多设计结果将逐个展示：

1） 游戏顶层设计：

顶层设计较为复杂，将拆分为 5 个部分：左上图 25-1、右上图 25-2、左中图 25-3、右中图 25-4、下图 25-5。

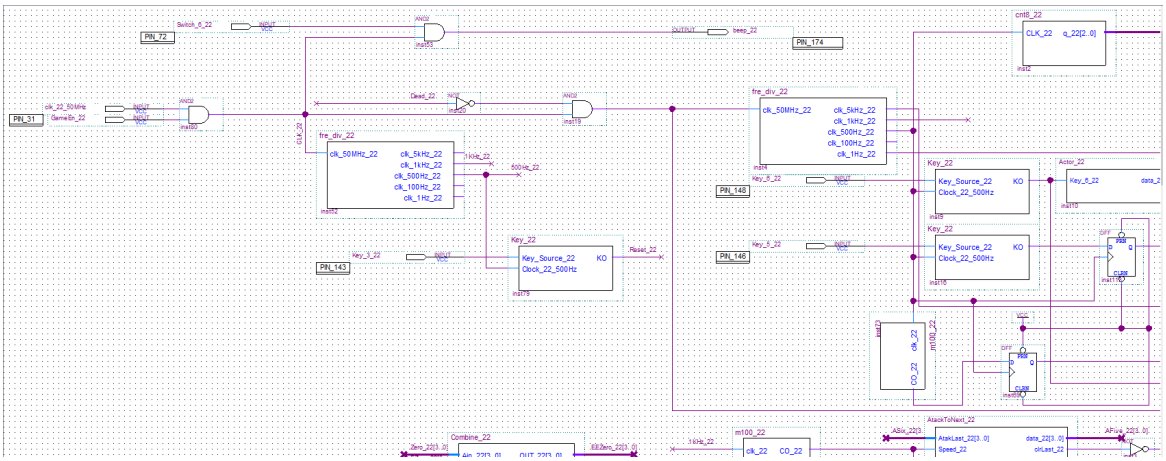

图 25-1 游戏顶层设计左上

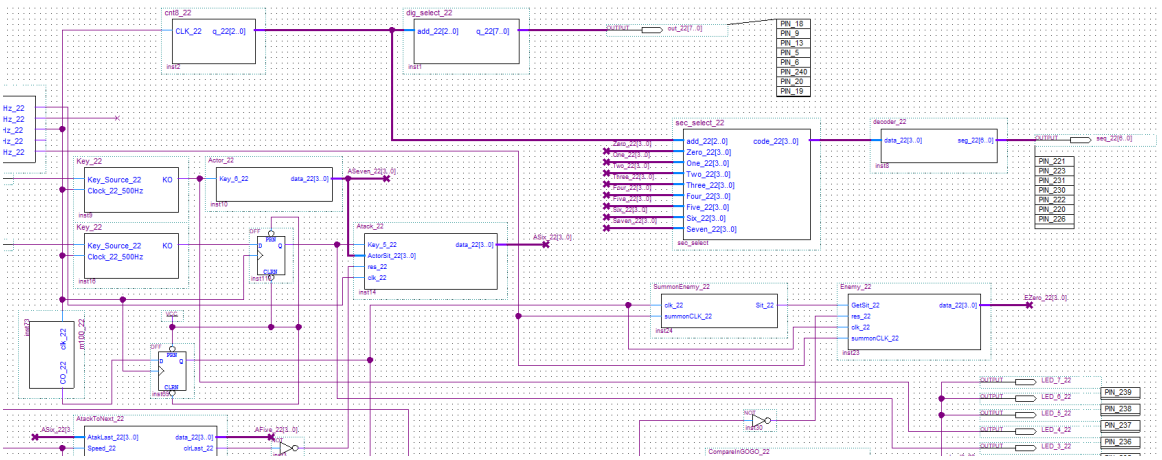

图 25-2 游戏顶层设计右上

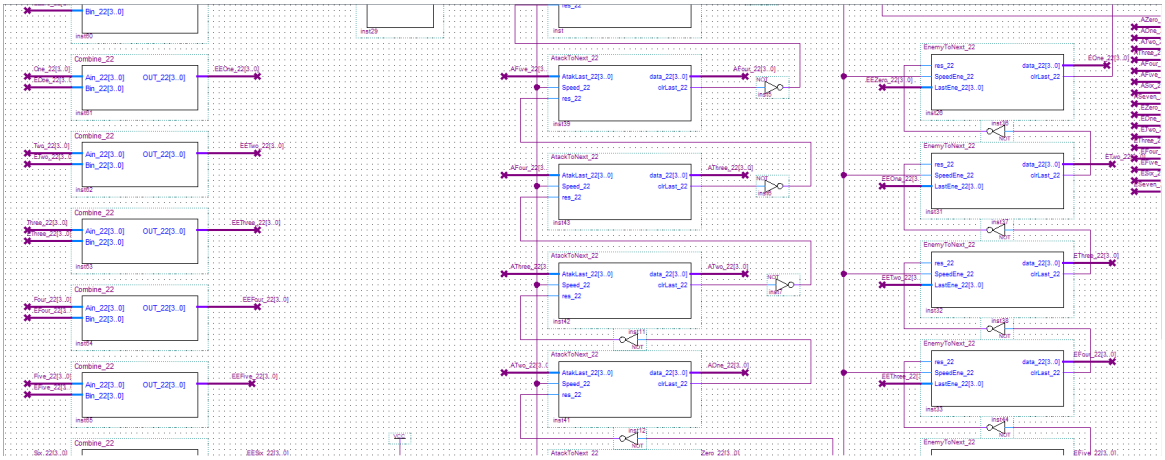

图 25-3 游戏顶层设计左中

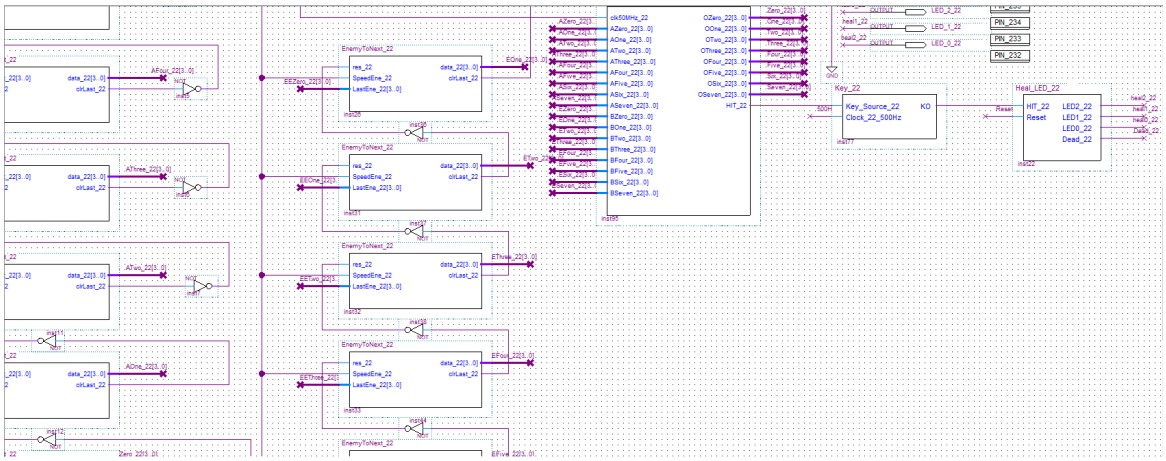

图 25-4 游戏顶层设计右中

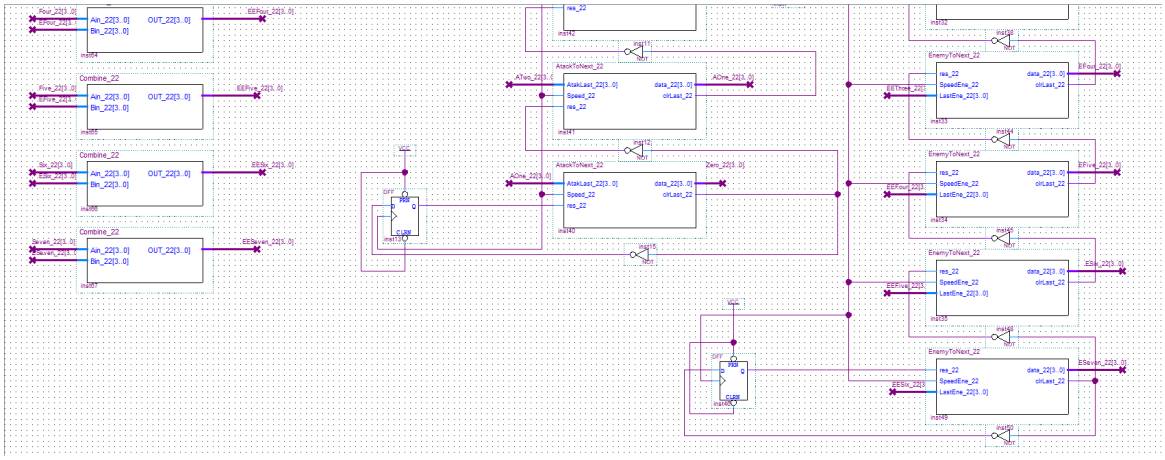

图 25-5 游戏顶层设计下

2) 主角控制模块：

主角控制模块设计代码如代码 3 所示。

代码 3 主角控制模块代码设计

```
module Actor_22(Key_6_22,data_22);//
input Key_6_22;
output[3:0] data_22;
reg[3:0] state;
assign data_22 = state;
always@(posedge Key_6_22)begin
if(state==4'b0010)
state = 4'b0001;
else
state = 4'b0010;
end
endmodule
```

3) 攻击模块：

攻击模块设计代码如代码 4 所示。

代码 4 攻击模块代码设计

```
module Atack_22(Key_5_22,data_22,ActorSit_22,res_22,clk_22);//
input Key_5_22;
input[3:0] ActorSit_22;
input res_22;
input clk_22;
output[3:0] data_22;
reg[3:0] state;
assign data_22 = state;
always@(posedge clk_22)begin
if(res_22)begin
state = 4'b0000;
end
if(Key_5_22)begin
if(ActorSit_22==4'b0001)
 state = 4'b0011;
else
 state = 4'b0100;
end
end
endmodule
```

4) 攻击右移模块：

攻击右移模块设计代码如代码 5 所示。

代码 5 攻击右移模块代码设计

```
module AtackToNext_22(AtakLast_22,data_22,Speed_22,res_22,clrLast_22);//
/攻击右移
```

```
2.  input[3:0] AtakLast_22;
3.  input Speed_22;
4.  input res_22;
5.  output clrLast_22;
6.  output[3:0] data_22;
7.  reg[3:0] state;
8.  reg clr;
9.  assign data_22 = state;
10. assign clrLast_22 = clr;
11. always@(posedge Speed_22)begin
12. if(res_22)
13.  state = 4'b0000;
14. if(AtakLast_22!=4'b0000)
15.  state = AtakLast_22;
16. if(state!=0)
17.  clr <= !1'b1;
18. end
19. endmodule
```

5) 敌人位置生成模块：

敌人位置生成模块设计代码如代码 6 所示。

代码 6 敌人位置生成模块代码设计

```
1.  module SummonEnemy_22(Sit_22,clk_22,summonCLK_22);//
2.  input summonCLK_22;
3.  input clk_22;
4.  output Sit_22;
5.  reg[15:0] state;
6.  reg Sit;
7.  reg[15:0] count;//count 没有使用，累加的方式容易造成过大的间隔
8.  assign Sit_22 = Sit;
9.  always@(posedge clk_22)begin
10. if(summonCLK_22)
11.  Sit = state[0];
12. if(count == 6'd635535)
13.  count = 0;
14. if(state==0)begin
15.  state = 16'b1011101000110111;//state = count;
16. end
17. else begin
18.  state = state >> 1;
19.  count = count + 255'd1;
20. end
21. end
22. endmodule
```

6） 敌人生成模块：

敌人生成模块设计代码如代码 7 所示。

代码 7 敌人生成模块代码设计

```
module Enemy_22(data_22,GetSit_22,res_22,clk_22,summonCLK_22);//
input summonCLK_22;
input GetSit_22;
input res_22;
input clk_22;
output[3:0] data_22;
reg[3:0] state;
assign data_22 = state;
always@(posedge clk_22)begin
if(res_22)begin
state = 4'b0000;
end
if(summonCLK_22)begin
if(GetSit_22==1'b1)
 state = 4'b0001;//1 为上
Else
 state = 4'b0010;//0 为下
end
end
endmodule
```

7） 敌人左移模块：

敌人左移模块设计代码如代码 8 所示。

代码 8 敌人左移模块代码设计

```
module EnemyToNext_22(data_22,res_22,SpeedEne_22,LastEne_22,clrLast_22)
;//
input[3:0] LastEne_22;
input res_22;
input SpeedEne_22;
output[3:0] data_22;
output clrLast_22;
reg clr;
reg[3:0] state;
assign data_22 = state;
assign clrLast_22 = clr;
always@(posedge SpeedEne_22)begin
if(res_22)
 state = 4'b0000;
if(LastEne_22!=4'b0000)
 state = LastEne_22;
if(state!=0)
```

```
17.  clr <= !1'b1;
18. end
19. endmodule
```

8） 敌人消除模块：

敌人消除设计代码如图 26 所示。

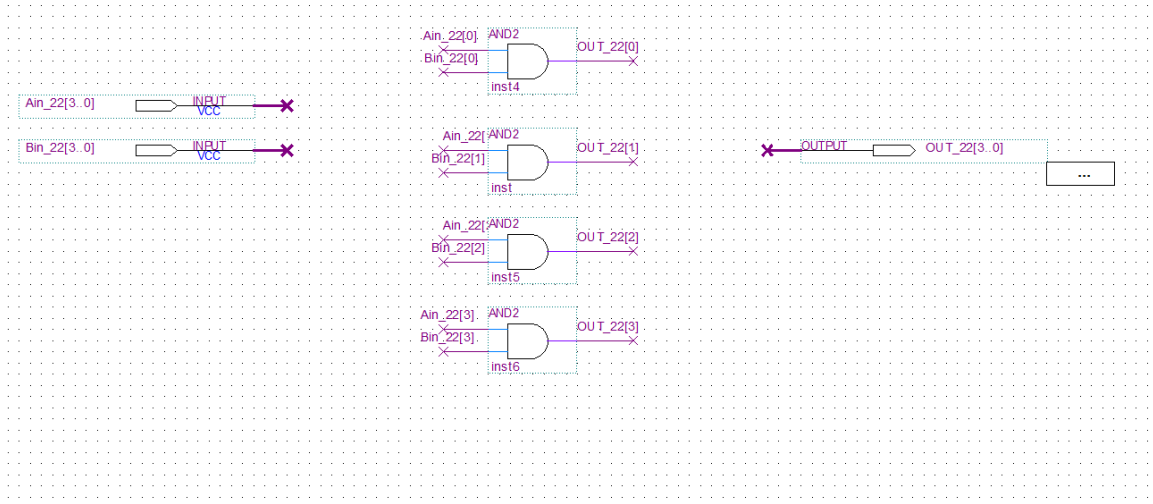

图 26 敌人消除模块电路设计

9） 血量及重置模块：

血量及重置模块顶层设计如图 27 所示，保险模块如代码 9 所示，LED 译码模块如代码 10 所示。

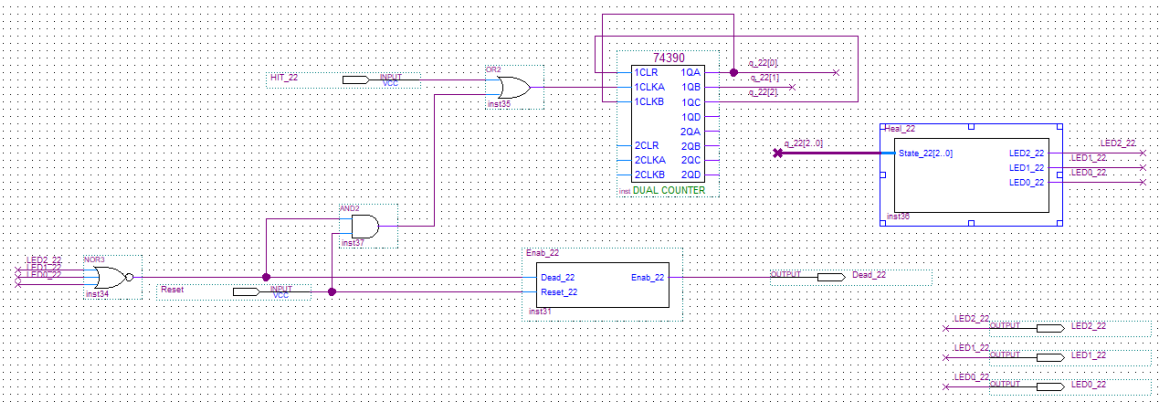

图 27 血量及充值模块顶层电路设计

代码 9 保险模块代码设计

```
1.  module Enab_22(
2.  input Dead_22,
3.  input Reset_22,
4.  output Enab_22);//
5.  reg L;
6.  reg[1:0] state;
7.  assign Enab_22 = L;
8.  always@(*)begin
```

```
state = {Dead_22,Reset_22};
case(state)
 2'b00:begin L=0;end
 2'b01:begin L=0;end
 2'b10:begin L=1;end
 default:begin L=0;end
endcase
end
endmodule
```

代码 10 LED 译码模块代码设计

```
module Heal_22(
input[2:0] State_22,
output LED2_22,
output LED1_22,
output LED0_22);//
reg[2:0] state;
reg L2;
reg L1;
reg L0;
assign LED2_22 = L2;
assign LED1_22 = L1;
assign LED0_22 = L0;
always@(*)begin
state = State_22;
case(state)
 3'b000:begin L0=1;L1=1;L2=1;end
 3'b001:begin L0=1;L1=1;L2=0;end
 3'b010:begin L0=1;L1=0;L2=0;end
 3'b011:begin L0=0;L1=0;L2=0;end
 default:begin L0=0;L1=0;L2=0;end
endcase
end
endmodule
```

10）伤害判定模块：

伤害判定模块设计代码如代码 11 所示。

代码 11 伤害判定模块代码设计

```
module CompareInGOGO_22(clk50MHz_22,AZero_22,AOne_22,ATwo_22,AThree_22,
AFour_22,AFive_22,ASix_22,ASeven_22,BZero_22,BOne_22,BTwo_22,BThree_22,
BFour_22,BFive_22,BSix_22,BSeven_22,OZero_22,OOne_22,OTwo_22,OThree_22,
OFour_22,OFive_22,OSix_22,OSeven_22,HIT_22);
input[3:0] AZero_22;
input[3:0] AOne_22;
input[3:0] ATwo_22;
```

```
5.  input[3:0] AThree_22;
6.  input[3:0] AFour_22;
7.  input[3:0] AFive_22;
8.  input[3:0] ASix_22;
9.  input[3:0] ASeven_22;
10. input[3:0] BZero_22;
11. input[3:0] BOne_22;
12. input[3:0] BTwo_22;
13. input[3:0] BThree_22;
14. input[3:0] BFour_22;
15. input[3:0] BFive_22;
16. input[3:0] BSix_22;
17. input[3:0] BSeven_22;
18. input clk50MHz_22;
19. output HIT_22;
20. output[3:0] OZero_22;
21. output[3:0] OOne_22;
22. output[3:0] OTwo_22;
23. output[3:0] OThree_22;
24. output[3:0] OFour_22;
25. output[3:0] OFive_22;
26. output[3:0] OSix_22;
27. output[3:0] OSeven_22;
28. reg[3:0] Zero;
29. reg[3:0] One;
30. reg[3:0] Two;
31. reg[3:0] Three;
32. reg[3:0] Four;
33. reg[3:0] Five;
34. reg[3:0] Six;
35. reg[3:0] Seven;
36. reg[15:0] count ;
37. reg HIT;
38. assign HIT_22 = HIT;
39. assign OZero_22 = Zero;
40. assign OOne_22 = One;
41. assign OTwo_22 = Two;
42. assign OThree_22 = Three;
43. assign OFour_22 = Four;
44. assign OFive_22 = Five;
45. assign OSix_22 = Six;
46. assign OSeven_22 = Seven;
47. always@(posedge clk50MHz_22)begin
```

```
48. if((ASeven_22 == 4'b0001 && BSeven_22 ==4'b0001)||(ASeven_22 == 4'b0010
     && BSeven_22 ==4'b0010))//主角被敌人击中,HIT = 1
49. HIT = 1;
50. else begin//若未被击中,HIT = 0
51. HIT =0;
52. if(ASeven_22 == 4'b0001 && BSeven_22 ==4'b0010)//若此时在上，敌人则在下
    0111//若此时在下，敌人则在上0111(两种情况输出相同)
53.  Seven = 4'b0111;
54. else if(ASeven_22 == 4'b0010 && BSeven_22 ==4'b0001)
55.  Seven = 4'b0111;
56. else
57.  Seven = ASeven_22;
58. end
59. if(ASix_22 - BSix_22 ==2)begin//攻击命中敌人,敌人与攻击同时清零
60. Six = 4'b0000;
61. count = count + 1;
62. end
63. else begin//若未命中敌人
64. if(ASix_22 == 4'b0011 && BSix_22 == 4'b0010)//若此时攻击在上，敌人在下
    0110
65.  Six = 4'b0011;
66. if(ASix_22 == 4'b0100 && BSix_22 == 4'b0001)//若此时攻击在下，敌人在上
    0101
67.  Six = 4'b0101;
68. else if(ASix_22 != 0)
69.  Six = ASix_22;
70. else
71.  Six = BSix_22;
72. end
73. if(AFive_22 - BFive_22 ==2)begin//攻击命中敌人,敌人与攻击同时清零
74. Five = 4'b0000;
75. count = count + 1;
76. end
77. else begin//若未命中敌人
78. if(AFive_22 == 4'b0011 && BFive_22 == 4'b0010)//若此时攻击在上，敌人在下
    0110
79.  Five = 4'b0011;
80. if(AFive_22 == 4'b0100 && BFive_22 == 4'b0001)//若此时攻击在下，敌人在上
    0101
81.  Five = 4'b0101;
82. else if(AFive_22 != 0)
83.  Five = AFive_22;
84. else
85.  Five = BFive_22;
```

```
86. end
87. if(AFour_22 - BFour_22 ==2)//攻击命中敌人,敌人与攻击同时清零
88. Four = 4'b0000;
89. else if(AFour_22 - BThree_22 == 2)begin
90. Four = 4'b0000;
91. Three = 4'b0000;
92. end
93. else begin//若未命中敌人
94. if(AFour_22 == 4'b0011 && BFour_22 == 4'b0010)//若此时攻击在上，敌人在下
    0110
95.  Four = 4'b0011;
96. if(AFour_22 == 4'b0100 && BFour_22 == 4'b0001)//若此时攻击在下，敌人在上
    0101
97.  Four = 4'b0101;
98. else if(AFour_22 != 0)
99.  Four = AFour_22;
100.    else
101.     Four = BFour_22;
102.    end
103.    if(AThree_22 - BThree_22 ==2)begin//攻击命中敌人,敌人与攻击同时清零
104.    Three = 4'b0000;
105.    count = count + 1;
106.    end
107.    else begin//若未命中敌人
108.    if(AThree_22 == 4'b0011 && BThree_22 == 4'b0010)//若此时攻击在上，敌
    人在下0110
109.     Three = 4'b0011;
110.    if(AThree_22 == 4'b0100 && BThree_22 == 4'b0001)//若此时攻击在下，敌
    人在上0101
111.     Three = 4'b0101;
112.    else if(AThree_22 != 0)
113.     Three = AThree_22;
114.    else
115.     Three = BThree_22;
116.    end
117.    if(ATwo_22 - BTwo_22 ==2)begin//攻击命中敌人,敌人与攻击同时清零
118.    Two = 4'b0000;
119.    count = count + 1;
120.    end
121.    else begin//若未命中敌人
122.    if(ATwo_22 == 4'b0011 && BTwo_22 == 4'b0010)//若此时攻击在上，敌人在下
    0110
123.     Two = 4'b0011;
```

```
124.    if(ATwo_22 == 4'b0100 && BTwo_22 == 4'b0001)//若此时攻击在下，敌人在上
    0101
125.     Two = 4'b0101;
126.    else if(ATwo_22 != 0)
127.     Two = ATwo_22;
128.    else
129.     Two = BTwo_22;
130.    end
131.    if AOne_22 - BOne_22 ==2)begin//攻击命中敌人,敌人与攻击同时清零
132.    One = 4'b0000;
133.    count = count + 1;
134.    end
135.    else begin//若未命中敌人
136.    if(AOne_22 == 4'b0011 &&BOne_22 == 4'b0010)//若此时攻击在上，敌人在下
    0110
137.     One = 4'b0011;
138.    if(AOne_22 == 4'b0100 && BOne_22 == 4'b0001)//若此时攻击在下，敌人在上
    0101
139.     One = 4'b0101;
140.    else if(AOne_22 != 0)
141.     One = AOne_22;
142.    else
143.     One = BOne_22;
144.    end
145.    if(AZero_22 - BZero_22 ==2)begin//攻击命中敌人,敌人与攻击同时清零
146.    Zero = 4'b0000;
147.    count = count + 1;
148.    end
149.    else begin//若未命中敌人
150.    if(AZero_22 == 4'b0011 && BZero_22 == 4'b0010)//若此时攻击在上，敌人
    在下011
151.     Zero = 4'b0011;
152.    if(AZero_22 == 4'b0100 && BZero_22 == 4'b0001)//若此时攻击在下，敌人
    在上0101
153.     Zero = 4'b0101;
154.    else if(AZero_22 != 0)
155.     Zero = AZero_22;
156.    else
157.     Zero = BZero_22;
158.    end
159.    end
160.    endmodule
```

11）译码模块：

译码模块设计代码如代码 12 所示。

代码 12 译码模块代码设计

```
1.  module decoder_22(data_22,seg_22);//
2.  input[3:0] data_22;
3.  output[6:0] seg_22;
4.  reg[6:0] state;
5.  assign seg_22 = state;
6.  always@(*)begin
7.  case(data_22)
8.  ////////////////////gfedcba;
9.  4'b0000:state = 7'b0000000; //空白
10. 4'b0001:state = 7'b1100011;//主角/敌人 上
11. 4'b0010:state = 7'b1011100;//主角/敌人 下
12. 4'b0011:state = 7'b1000001;//攻击上
13. 4'b0100:state = 7'b1001000;//攻击下
14. 4'b0101:state = 7'b1101011;//主角/敌人 上 攻击 下 复合态
15. 4'b0110:state = 7'b1011101;//主角/敌人 下 攻击 上 复合态
16. 4'b0111:state = 7'b1111111;//主角避开敌人
17. default:state = 7'b0000000;//空白
18. endcase
19. end
20. endmodule
```

5.3.4 仿真测试

模块过于繁多且复杂通过观察实物的方式进行验证。

5.4 模式切换

5.4.1 模块功能

模式切换功能实现控制电路按钮，切换显示模式为：基础时钟模式（下文简称 Mode1）、闹钟设定模式（下文简称 Mode2）、游戏模式（下文简称 Mode3）。并自动循环切换，故需要一个模 3 电路，并选择 Mode1、Mode2 输出给译码器的段选信号，由于游戏模块译码器独立，所以还需选择使用 Mode1 或 Mode2 情况下的译码器（下文简称译码器 1）或是 Mode3 情况下的译码器（下文简称译码器 2）。

5.4.2 设计思路

由于 Mode1 和 Mode2 共用同一个译码模块，所以可以让两种模式下的位宽为四的段选信号输入并作选择，而切换到游戏模式时，则需要选用译码器 2，可在此状态使模块输出一个游戏模

式使能（GameEn），并将译码器 2 的段选信号、及其对应的位选信号以及译码器 1 的段选信号、及其对应的位选信号输入一个选择器，并由 GameEn 来控制。

所以模式切换功能需要两个模块配合完成，一个是 Mode1、Mode2 模式选择模块（Mode_select，下文简称模式选择模块 1），一个是译码器 1、译码器 2 选择模块（GameChoose，下文简称模式选择模块 2）。

5.4.3 设计结果

模式选择模块 1（Mode_select）代码设计如代码 13 所示，模式选择模块 2（GameChoose）代码设计如代码 14 所示。

代码 13 模式选择模块 1（Mode_select）代码设计

```
1. module Mode_select_22(Key_7_22,clk50MHz_22,Alarm_22,Clock_22,code_22,Alarm_
   EN_22,GameEn_22);//模式选择
2. input   Key_7_22;
3. input   clk50MHz_22;
4. input[3:0]  Alarm_22;
5. input[3:0]  Clock_22;
6. output GameEn_22;
7. output[3:0] code_22;
8. output  Alarm_EN_22;
9. reg[3:0]  state;//状态
10.reg[3:0]   mode;//模式
11.reg    SetAlarm;
12.reg    GameEn;
13.assign code_22 = mode;
14.assign Alarm_EN_22 = SetAlarm;
15.assign GameEn_22 = GameEn;
16.always@(posedge Key_7_22)begin
17.if(state == 4'b0010)
18.state = 4'b0000;
19.else
20.state = state + 1'b1;
21.end
22.always@(posedge clk50MHz_22)begin
23.case(state)
24.4'b0000:mode = Clock_22;
25.4'b0001:mode = Alarm_22;
26.4'b0010:mode = mode;
27.default:mode = Clock_22;
28.endcase
```

```
29.if(state == 4'b0001)
30.SetAlarm = 1'b1;
31.else
32.SetAlarm = 1'b0;
33.if(state == 4'b0010)
34.GameEn = 1'b1;
35.else
36.GameEn = 1'b0;
37.end
38.endmodule
```

代码 14 模式选择 2（GameChoose）代码设计

```
1. module GameChoose_22(
2. input GameEn_22,
3. input[7:0] Cout_22,
4. input[6:0] Cseq_22,
5. input[7:0] Gout_22,
6. input[6:0] Gseq_22,
7. output[7:0] out_22,
8. output[6:0] seq_22);//
9. reg[7:0] Mode;
10.reg[7:0] out;
11.reg[6:0] seq;
12.assign out_22 = out;
13.assign seq_22 = seq;
14.always@(*)begin
15.if(GameEn_22 == 1)begin
16.out = Gout_22;
17.seq = Gseq_22;
18.end
19.else begin
20.out = Cout_22;
21.seq = Cseq_22;
22.end
23.end
24.endmodule
```

5.4.4 仿真测试

设置 Mode1 与 Mode2 的输入初始值，设置 Key_7 输入信号，观察输出 code 以及 GameEn，波形仿真如图 28 所示。设置译码器 1 对应的段选、位选信号的初始值以及译码器 2 的段选、位选

信号的初始值，当 GameEn 为 1 时，选择输出译码器 2 的信号（Gout，Gseq），波形仿真如图 29 所示。

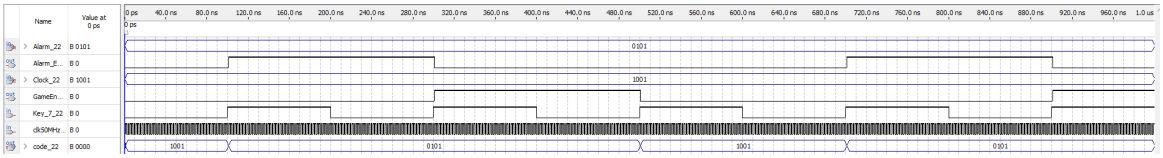

图 28 模式选择模块 1（Mode_select）波形仿真

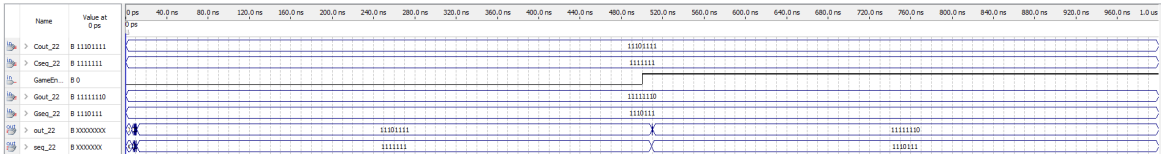

图 29 模式选择模块 2（GameChoose）波形仿真

6. 系统总体测试

系统总体测试结果如“附件一　评分细则及测试原始数据记录”。

7. 系统设计实现过程中遇到的主要问题、解决思路和解决方案

7.1 伤害判定模块

7.1.1 主要问题

伤害判定在判断敌人与子弹位置时具有较大的风险，即若子弹与敌人所在数码管贴近（无间隔），则在下一次上升沿，两者将会错过，造成子弹穿过敌人的现象。

7.1.2 解决思路

将子弹的移动频率尽可能与敌人移动频率错开，并尽可能避免两者移动频率重合。

7.3.3 解决方案

将子弹速度改为 10 格/秒，敌人改为 5 格/秒，但仍存在一定问题，可以进一步增加频率差值或是更改伤害判定模块的判定机制，将两者错开的情况并入判定（由于时间及硬件资源原因未能进一步修改验证）。

7.2 敌人位置生成模块

7.2.1 主要问题

count 在设计中未被使用，原因是将其启用赋值将会造成较大的间隙间隔而看不到敌人生成。

7.2.2 解决思路

拓展 state 的位宽，并使其循环，以尽可能造成一种伪随机。或是将位宽减小与 count 同步，而 count 累次加数增大，再将 count 的值赋给 state，以获得更大的循环周期。

7.2.3 解决方案

代码中采用了第一种解决思路，将 state 位宽拓展为 16，并随机赋予了一组值。

7.3 敌人消除模块

7.3.1 主要问题

在设计过程中，敌人遇到子弹后会消失，但信号仍被正常传输继续左移，传输信号未被中断。

7.3.2 解决思路

将伤害判定前与伤害判定后的信号相与可以将 0000 信号阻断。

7.3.3 解决方案

敌人左移模块的输入值变为相与以后的信号（即被消除模块阻断传输后的信号），并成功解决。