数电综合实验实验报告

任务要求

实验名称

数字记忆力游戏的设计与实现

实验简述

实现一个数字记忆力游戏，从欢迎界面进入游戏后，要求玩家能够记忆在数码管上显示的多位数码并重新输入，若正确则进入下一关，否则游戏重新开始。当游戏胜利时，能够显示胜利图案并使蜂鸣器发声。

实际实现的要求

注意其中提高要求显示为为**粗体。**

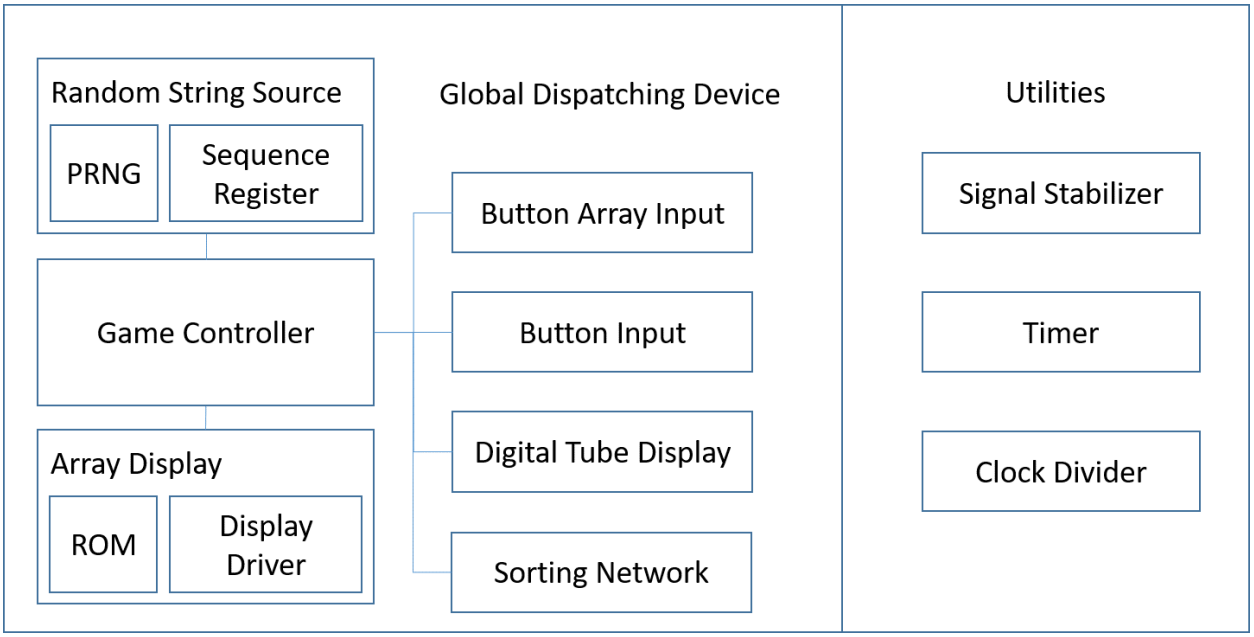
- 点阵显示欢迎界面，并实现**双色灯呼吸**。
- 游戏共三关。第一关五位数字，显示五秒后消失；第二关六位数字，显示五秒后消失；第三关六位数字，显示三秒后消失。所有数字均为随机生成。
- 一关进行中，点阵显示倒计时，一个数码管显示关数。
- 数字显示消失后，可以通过键盘阵列输入数字，并能**通过一个按键清空输入内容，实现重新输入**。
- 当一关失败时，回到第一关。
- 当一关成功通过时，蜂鸣器发声。
- 胜利时显示胜利图案并**播放胜利音乐**。
- **可以通过一个拨码开关开启高级模式：要求记忆所有数字，并从小到大输入**。
- 游戏可以随时退出（回到欢迎界面）。

系统设计

设计思路

由一个模块控制整体游戏逻辑，需求进行细分后各自对应一个模块，并通过顶层模块进行整体的调度。

总体框图



分块设计

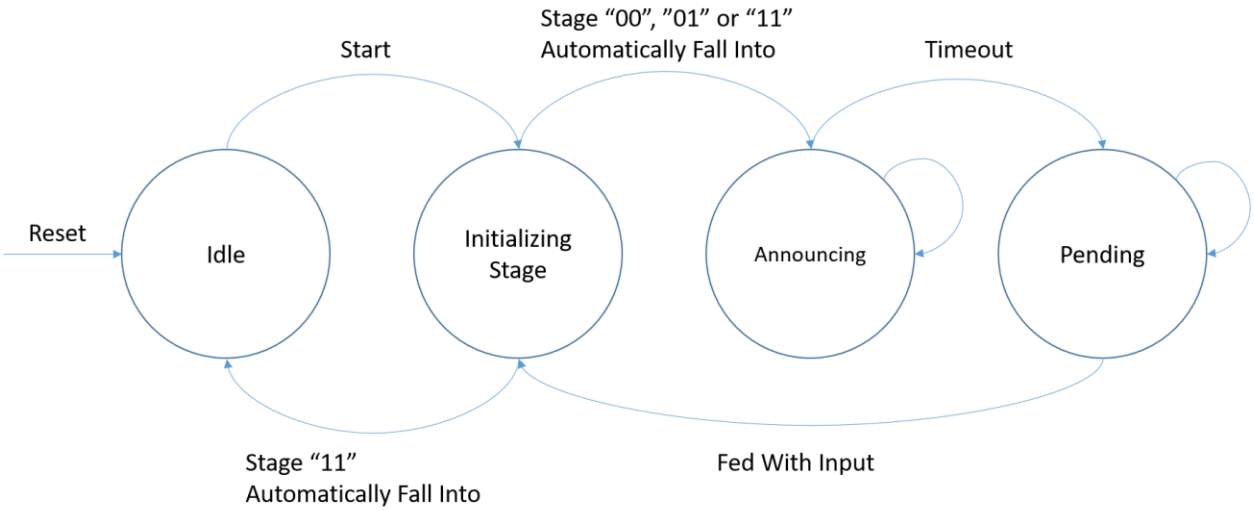
下面主要介绍整体设计中较为重要的模块，诸如单个按键输入、信号防抖、计时器、分频器等较为琐细而简单的部分将被忽略。

另外，由于部分模块的难以进行仿真（包括输入扫描波形难以在仿真中重现或仿真输出正确性难以辨别），因此其仿真图将被略去。

游戏逻辑控制模块

源文件：GameController.vhd

游戏逻辑可以被抽象成如下状态机：



四个状态分别解读为：

1. Idle：空闲状态，显示欢迎界面或胜利界面。
2. StageInit：缓冲状态，用于通知随机序列刷新或蜂鸣器响应，根据上下文会自动在下一个时钟沿掉入 Idle 或 Announcing 状态。
3. Announcing：显示状态，计时并输出倒计时大小，当超时后自动转移至 Pending 状态。
4. Pending：等待输入状态，用户输入抽象为 feed 和 match 两个信号，分别代表是否存在用户输入以及用户输入是否正确。

另外，当前关卡由另一 Stage 信号描述，该信号也会影响该状态机的状态转移选择。其中”00”、”01”、”10”分别对应游戏的三关，”11”对应游戏胜利。Stage 信号仅在由 Pending 状态到 StageInit 状态及状态机重置时更新。

整个游戏的逻辑由游戏逻辑控制模块所控制，其接收由全局翻译的抽象游戏操作，进行游戏状态的转移，实现了游戏逻辑和实际实现的部分剥离。

随机排列生成模块

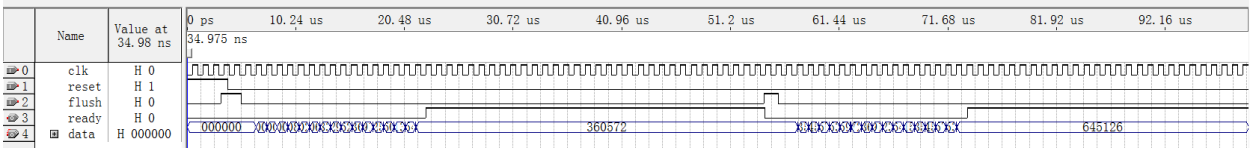
源文件：RandGenerator.vhd、PermBuffer.vhd

本模块共有两个子模块：16-bit LFSR 伪随机生成器负责生成随机比特流、排列寄存器负责锁存住一个排列以待后续使用。

当外部输入的矩形波结束时，寄存器将丢弃当前锁存的数据，并重新从比特流中获得足够的熵组成新的输出（需要几个时钟周期完成）。

值得注意的是，本模块总是生成六个有效的 BCD 码，若不需要这么多的数据，应由外部使用者负责过滤。

其仿真波形如下：



可以看到 flush 信号被设置时，ready 信号输出低电平，直到新的随机序列被锁存，ready 信号回到高电平。另外，所有的输出均为合法的 BCD 码。

点阵显示模块

源文件：LedArrayDriver.vhd、LedArrayRom.vhd

由于本模块只需要显示固定的几个图案，因此我将这几个图案经过一定的编码后储存在一个独立的子模块中，某一行的数据可以通过一定的地址访问，从而简化的显示，也方便了对待显示图案进行修改或扩充。

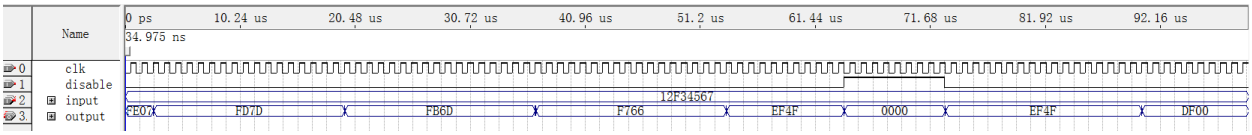
点阵显示主要由时钟驱动两个嵌套的循环，其一控制当前扫描的行，另一控制是否对当前行进行显示（用于调整占空比实现呼吸效果）。

数码管显示模块

源文件：DigitArrayDriver.vhd

本模块接收固定的八个 BCD 码，并将其解码后循环扫描输出至对应的数码管上，对于非法的 BCD 码将不予显示。

其仿真波形如下：



可以发现，output 信号的高八位为扫描信号，低八位为解码后的阴极数码管显示信号。当 disable 信号被设置时，扫描停止，所有输出接地。另外，不合法的 BCD 码不予显示（但会占用扫描周期）。

蜂鸣器模块

源文件：AudioDriver.vhd

由于需求简易，本模块仅提供了单一的接口，当使能端被设置时，蜂鸣器会播放硬编码的音乐。其内部实现为对蜂鸣器驱动方波的频率控制，从而改变其发声的频率来模拟音符。

键盘阵列输入模块

源文件：ButtonArrayDriver.vhd

本模块采用了事件驱动的方式，当其检测到存在按键按下时，会自动进入扫描状态并找到按下的按键后输出其对应编码，输出时会有一个单个时钟周期的脉冲通知外部模块进行处理。当所有按键被松开后，本模块重新回到监听状态。

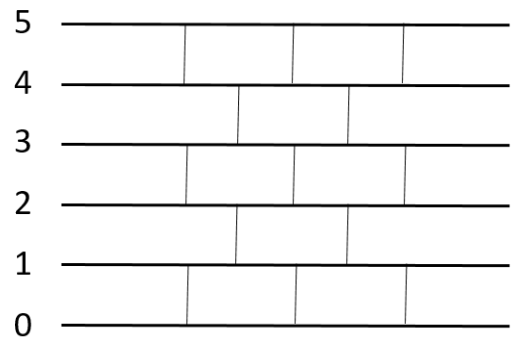
当脉冲不存在时，任何输出都是无效的。

若同一时刻有多个按键被按下，将输出随机一个按键的编码。

排序模块

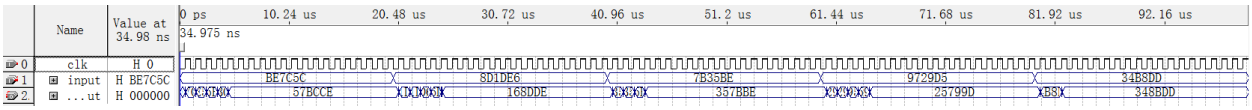
源文件：MessageRearranger.vhd

本模块通过如下图所示的排序网络实现了一个并行的冒泡排序，横线为数据输入，竖线为比较器。



值得注意的是，由于计算延迟的缘故，本模块从输入改变到输出正确的已排序数列存在若干个时钟周期的延时。

其仿真波形如下：



可以发现，当输入序列变化后的若干时钟周期后，输出信号固定为其排序后的状态。排序所需的周期数与输入信号的排列有关，但不超过六个时钟周期。

另外，本模块对所有的十六进制数码进行排序，由于未使用的数码会在外部被设为 FF，因此总能得到正确的结果。

全局调度模块

源文件：Device.vhd

全局调度模块是整个游戏到设备的桥梁，其负责将各个模块连接起来，并对一些信号进行翻译和过滤，从而满足各个模块对其输入和输出的假设，大大简化了单个模块的设计和调试。

功能说明及资源利用情况

编译报告如下图所示：

Flow Status	Successful - Mon Nov 13 00:01:24 2017
Quartus II Version	9.1 Build 350 03/24/2010 SP 2 SJ Web Edition
Revision Name	game
Top-level Entity Name	game
Family	MAX II
Device	EPM1270T144C5
Timing Models	Final
Met timing requirements	Yes
Total logic elements	778 / 1,270 (61 %)
Total pins	54 / 116 (47 %)
Total virtual pins	0
UFM blocks	0 / 1 (0 %)

经过优化后，共使用了 61%的逻辑单元。

综合资源利用率如下图所示：

Analysis & Synthesis Resource Utilization by Entity

	Compilation Hierarchy Node	Logic Cells	LC Registers	UFM Blocks	Pins	Virtual Pins	LUT-Only LCs	Register-Only LCs	LUT/Register LCs	Carry Chain LCs	Packed LCs
1	⊟ |game	826 (84)	519	0	54	0	307 (33)	79 (24)	440 (27)	352 (0)	0 (0)
2	⊟ |AudioDriver:U8|	203 (41)	173	0	0	0	30 (30)	2 (1)	171 (10)	159 (6)	0 (0)
3	|ClockDivider:U0|	18 (18)	18	0	0	0	0 (0)	0 (0)	18 (18)	17 (17)	0 (0)
4	|ClockDivider:U1|	18 (18)	18	0	0	0	0 (0)	0 (0)	18 (18)	17 (17)	0 (0)
5	|ClockDivider:U2|	18 (18)	18	0	0	0	0 (0)	0 (0)	18 (18)	17 (17)	0 (0)
6	|ClockDivider:U3|	18 (18)	18	0	0	0	0 (0)	0 (0)	18 (18)	17 (17)	0 (0)
7	|ClockDivider:U4|	18 (18)	18	0	0	0	0 (0)	0 (0)	18 (18)	17 (17)	0 (0)
8	|ClockDivider:U5|	18 (18)	18	0	0	0	0 (0)	0 (0)	18 (18)	17 (17)	0 (0)
9	|ClockDivider:U6|	18 (18)	18	0	0	0	0 (0)	0 (0)	18 (18)	17 (17)	0 (0)
10	|ClockDivider:U7|	18 (18)	18	0	0	0	0 (0)	0 (0)	18 (18)	17 (17)	0 (0)
11	|Timer:T0|	18 (18)	18	0	0	0	0 (0)	1 (1)	17 (17)	17 (17)	0 (0)
12	⊟ |ButtonArrayDriver:U9|	57 (19)	49	0	0	0	8 (6)	4 (3)	45 (10)	34 (0)	0 (0)
13	|SignalStablizer:U|	20 (20)	18	0	0	0	2 (2)	0 (0)	18 (18)	17 (17)	0 (0)
14	|Timer:T1|	18 (18)	18	0	0	0	0 (0)	1 (1)	17 (17)	17 (17)	0 (0)
15	|ClockDivider:U0|	18 (18)	18	0	0	0	0 (0)	0 (0)	18 (18)	17 (17)	0 (0)
16	⊟ |DigitArrayDriver:U7|	76 (58)	21	0	0	0	55 (55)	1 (0)	20 (3)	17 (0)	0 (0)
17	|Timer:T1|	18 (18)	18	0	0	0	0 (0)	1 (1)	17 (17)	17 (17)	0 (0)
18	⊟ |GameController:U4|	69 (51)	36	0	0	0	33 (33)	1 (0)	35 (18)	32 (15)	0 (0)
19	|Timer:T1|	18 (18)	18	0	0	0	0 (0)	1 (1)	17 (17)	17 (17)	0 (0)
20	⊟ |LedArrayDriver:U6|	116 (87)	30	0	0	0	86 (57)	13 (13)	17 (17)	42 (42)	0 (0)
21	|LedArrayRom:U|	29 (29)	0	0	0	0	29 (29)	0 (0)	0 (0)	0 (0)	0 (0)
22	|MessageRearranger:U10|	88 (88)	49	0	0	0	39 (39)	0 (0)	49 (49)	0 (0)	0 (0)
23	⊟ |PermBuffer:U5|	55 (39)	38	0	0	0	17 (17)	31 (16)	7 (6)	0 (0)	0 (0)
24	|RandGenerator:U|	16 (16)	16	0	0	0	0 (0)	15 (15)	1 (1)	0 (0)	0 (0)
25	|SignalStablizer:U1|	20 (20)	18	0	0	0	2 (2)	1 (1)	17 (17)	17 (17)	0 (0)
26	|SignalStablizer:U2|	20 (20)	18	0	0	0	2 (2)	1 (1)	17 (17)	17 (17)	0 (0)
27	|SignalStablizer:U3|	20 (20)	18	0	0	0	2 (2)	1 (1)	17 (17)	17 (17)	0 (0)

可以发现，音频播放器占用了最多的资源（因为其内部存在大量的分频器对输入时钟进行调制，生成对应的音符），有待进一步的优化，而其他的元件的资源占用较为均匀。

故障及问题分析

- **信号更新时存在的毛刺**
  信号中的毛刺往往来源于电路中的冒险。对于静态冒险，我们可以通过调整实现来消除；对于动态冒险，我们可以通过状态机等待一个时钟以待信号稳定后读取（对于部分信号中的毛刺甚至时可以无视的，因为我们并不会在信号更新后的短时间内用到这个信号）。
- **时序逻辑与组合逻辑的选择**
  对于许多控制信号（包括计数、按钮按下等），我们需要仅在时钟沿进行响应，从而实现电路状态的同步，否则电路则会出现一些跑飞的情况。
  对于计算信号，我们大可以通过组合逻辑进行实现，因为其内部是无状态的，稳定后的输出仅仅与当前输入有关。
- **由于计算延时过长导致的时钟紊乱**
  在实现排序网络时，由于网络复杂，串联元件多，计算延迟长，导致实际综合的电路中整体的时钟受到了影响（虽然整个计算并不关于时钟敏感）。
  最终，我通过将排序的计算分离到多个时钟周期上解决了这个问题。

总结和结论

通过本次实验，我对硬件描述语言与程序设计语言有了更加深刻的理解。在编写 VHDL 代码时，应时刻关心其背后的电路翻译，从而才能够理解一些其与软件设计语言在行为上的不同。比如 VHDL 中的执行顺序对应着电路上的串并联逻辑，条件选择对应着电路中的译码器及数据选择器等。只有理解了其背后的电路原理，才能明白自己正在描述电路而非编写一串程序指令，从而才能保证所得电路的正确性。

但是，软件的一些核心思想通过一定的适配后也能够应用到硬件设计上，如本项目中的排序网络的设计思路来自于冒泡排序、随机排列生成后在进行过滤的设计思路来自于 IP 地址的子网掩码等。组合逻辑的背后是纯粹的数值计算，而这与程序设计中的函数式编程非常相像，但值得注意的是，由于计算存在延时，我们也不能直接对程序设计中的方法原封不动地照搬照抄，而必须得对其进行一定地适配，从而达到有效计算的目的。