多输出电路：编码器，译码器和存储器

介绍

布尔表达式可以用以输出多变量布尔函数值，像 assign 这样的数据流构造可以用来模拟这样的布尔函数。存在多输入和多输出的电路，在这个实验中要求你参照关于如何使用 Vivado 工具创建工程和验证数字电路的教程，设计编码器，译码器和只读存储器。

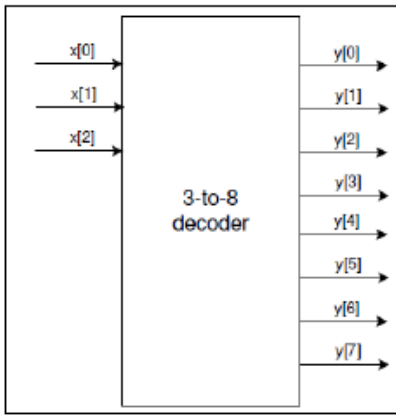
目标

在完成本次实验后，你将具备以下能力：

- 利用行为建模设计多输出译码器
- 利用行为建模设计编码器
- 利用 Verilog 提供的 reg 数据类型和$readmemb 系统函数设计只读存储器

多输出译码电路 Part 1

译码器是拥有多个输出的组合逻辑电路。它们被广泛用于存储芯片中，用来选择输入地址所寻址的一个字。例如一个 8 字宽的内存会拥有 3 位地址输入。译码器解码 3 位的输入地址产生一个选择信号，从 8 个字中选择地址相应的字。3-8 译码器的符号以及真值表如下：



x_0 x_1 x_2	y_7 y_6 y_5 y_4 y_3 y_2 y_1 y_0
0 0 0	0 0 0 0 0 0 0 1
0 0 1	0 0 0 0 0 0 1 0
0 1 0	0 0 0 0 0 1 0 0
0 1 1	0 0 0 0 1 0 0 0
1 0 0	0 0 0 1 0 0 0 0
1 0 1	0 0 1 0 0 0 0 0
1 1 0	0 1 0 0 0 0 0 0
1 1 1	1 0 0 0 0 0 0 0

这样的电路也称为二进制译码器，并且由于每个输出都有唯一的输入组合，所以可以使用数据流语句建模。

1-1. 设计一个 3-8 译码器。以 SW2-SW0 为输入，LED7-LED0 为输出；使用数据流建模结构。

1-1-1. 打开 Vivado 并创建一个空工程并命名为 *lab3_1_1*。

1-1-2. 创建并添加一个 Verilog 模块，命名为 decoder_3to8_dataflow.v,然后定义译码器的三位输入 x 和 8 位输出 y。使用数据流建模。

1-1-3. 将提供的测试文件（decoder_3to8_dataflow_tb.v）加入工程。

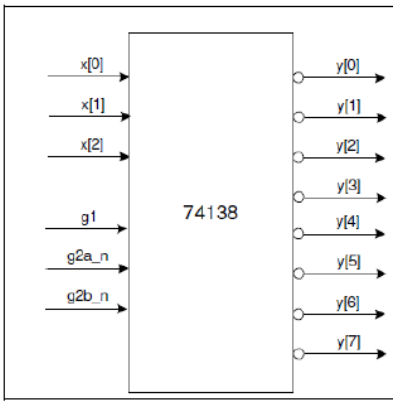
1-1-4. 将设计的工程模拟 50 ns 验证是否正确。

1-1-5. 在工程中添加适当的管脚约束的 XDC 文件，并加入相关联的管脚，将 x 分配给 ***SW2-SW0***，将 y 分配给 ***LED7-LED0***。注意对于一个给定的输入组合，只有一个 LED 会亮。

1-1-6. 综合，实现设计。

1-1-7. 生成比特流文件，下载到 Nexys4 开发板上，验证功能。

1-2. 设计实现一个常用集成电路 74138，使用数据流建模和你在 1-1 中使用的译码器，74138 的符号和真值表如下：



g_1 g_{2a_n} g_{2b_n}	x_0 x_1 x_2	y_0 y_1 y_2 y_3 y_4 y_5 y_6 y_7
0 x x	x x x	1 1 1 1 1 1 1 1
x 1 x	x x x	1 1 1 1 1 1 1 1
x x 1	x x x	1 1 1 1 1 1 1 1
1 0 0	0 0 0	0 1 1 1 1 1 1 1
1 0 0	0 0 1	1 0 1 1 1 1 1 1
1 0 0	0 1 0	1 1 0 1 1 1 1 1
1 0 0	0 1 1	1 1 1 0 1 1 1 1
1 0 0	1 0 0	1 1 1 1 0 1 1 1
1 0 0	1 0 1	1 1 1 1 1 0 1 1
1 0 0	1 1 0	1 1 1 1 1 1 0 1
1 0 0	1 1 1	1 1 1 1 1 1 1 0

X = don't care

这和你在 1-1 中创建的非常相似，它只是增加了控制（使能）信号 G1，/G2A，/G2B。这使得在有些系统中的解码更加简单。

1-2-1. 打开 Vivado 创建一个空工程，命名为 *lab3_1_2*.

1-2-2. 创建并添加 Verilog 模块，命名为 decoder_74138_dataflow，实例化你在 1-1 中开发的模块。添加额外的逻辑，使用数据流建模结构建模所设计的功能。

1-2-3. 将提供的测试文件（decoder_74138_dataflow_tb.v）加入工程。

1-2-4. 将设计的工程模拟 200 ns 验证是否正确。

1-2-5. 将你在 1-1 中使用的 XDC 文件添加到工程。修改 XDC 文件，将 *g1* 分配给 **SW7**，*g2a_n* 分配给 **SW6**，*g2b_n* 分配给 **SW5**。
1-2-6. 综合实现你的设计。

1-2-7. 生成比特流文件，下载到 Nexys4 开发板上，验证功能。

多输出编码电路 Part 2

编码器电路是出于对标准化，速度，保密性，安全性或者通过缩小尺寸来节省空间的考虑，将信息从一种形式（编码）转换为另一种形式（编码）的电路。在数字电路中，编码信息可以减小信息存储所用的空间，确定功能的优先级。广泛使用的编码电路的例子有，优先编码器，哈弗曼编码器等。

2-1. 设计一个 8-3 编码器，其真值表如下，使用行为建模。

Inputs									Outputs				
E1	0	1	2	3	4	5	6	7	A2	A1	A0	GS	E0
1	X	X	X	X	X	X	X	X	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	X	X	X	X	X	X	X	0	0	0	0	0	1
0	X	X	X	X	X	X	0	1	0	0	1	0	1
0	X	X	X	X	X	0	1	1	0	1	0	0	1
0	X	X	X	X	0	1	1	1	0	1	1	0	1
0	X	X	X	0	1	1	1	1	1	0	0	0	1
0	X	X	0	1	1	1	1	1	1	0	1	0	1
0	X	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1

X = don't care

2-1-1. 打开 Vivado 创建一个名为 *lab3_2_1* 的空工程。

2-1-2. 创建并添加以 *v* 和 *en_in_n* 为输入，以 *y*，*en_out* 和 *gs* 为输出的 Verilog 模块。v 是一个 8 位的输入（表中标为 0 到 7），输入 *en_in* 只有一位（E1），输出 y 为 3 位（A2，A1，A0），*en_out* 为一位（E0），*gs* 为一位（GS）。

2-1-3. 在工程中添加适当的管脚约束的 XDC 文件，并加入相关联的管脚约束。将输入 x 分配给 **SW7-SW0**，*en_in_n* 分配给 **SW15**，y 分配给 **LED2-LED0**，*en_out* 分配给 **LED7**，*gs* 分配给 **LED6**。

2-1-4. 综合实现此设计。

2-1-5. 生成比特流文件，下载到 Nexys4 开发板上，验证功能。

只读存储器 Part 3

只读存储器（ROM）由互联阵列组成，由于存储二进制信息数组。它一旦存储了二进制信息，可以随时读取，但不能更改。大型的 ROM 通常用于存储不能被系统中其他电路更改的程序或者数据，小型的 ROM 可以用来实现组合电路。ROM 使用类似于 1-1 中的译码器来寻址特定的存储位置。

一个拥有 m 个地址输入引脚和 n 个数据输出引脚的 ROM 可以存储2^m个字，每个字为 n 位。当给出一个地址访问这一存储器时，地址相应位置的字通过输出引脚读出。

在 Verilog 语言中，存储器可以使用 reg 数据类型的二维数组定义，如下：

reg [3:0] MY_ROM [15:0];

上面代码中 reg 是数据类型，MY_ROM 是一个 16×4 的内存，拥有 16 个地址，每个地址的宽度为 4bit。如果满足下面两个条件，这块内存就是只读的：(i) 这块内存只能被读，不能被写入；(ii) 内存应该以期望的数据初始化。Verilog 语言提供一个系统函数**$readmemb** 可以使用特定内容初始化内存。下面是定义并使用一个 4×2 的 ROM 的例子。

```
module ROM_4x2 (ROM_data, ROM_addr);
output [1:0] ROM_data;
input [1:0] ROM_addr;
reg [1:0] ROM [3:0]; // defining 4x2 ROM
assign ROM_data = ROM[ROM_addr];
    // reading ROM content at the address ROM_addr
initial $readmemb ("ROM_data.txt", ROM, 0, 3);
        // load ROM content from ROM_data.txt file
endmodule
```

在此例中，ROM_data.txt 文件应该与 verilog 模块放在同一目录下（因为引用时没有使用绝对路径），并且文件中可以有 8 行或者更少，比如：

```
10
0x
11
00
```

注意，如果行数少于 ROM 的大小，则未指定的位置将会用 0 初始化，另外还有另一个系统函数$readmembh 允许数据文件使用十六进制编写。

3-1.设计一个 2 位比较器，用于比较两个 2-bit 的数字，并输出字 A 的十进制值大于、小于或者等于 B。你要模拟 ROM 并会用到$readmemb 函数。

3-1-1. 打开 Vivado 并创建一个名为 *lab3_3_1* 的工程。

3-1-2. 使用 ROM 和$readmemb 系统函数创建并添加一个 Verilog 模块，该模块拥有两个输入（a，b）和三个输出（lt，gt 和 eq）。

3-1-3. 在工程中添加适当的管脚约束的 XDC 文件，并加入相关联的管脚约束。将 a 分配给 **SW3** 到 **SW2**，b 分配给 **SW1** 到 **SW0**，*lt* 分配给 **LED2**，*gt* 分配给 **LED1**，*eq* 分配给 **LED0**。

3-1-4. 创建并添加描述设计输出的文本文件

b	a	Lt	Gt	Eq
00	00	0	0	1
00	01	1	0	0

上表列出了 b 与 a 比较前两项，继续这样的比较直到 b 和 a 都达到 11 为止。

将上表的比较结果保存在一个.txt 文件中，然后点击位于 *Flow Navigator* 下的 *Add Sources* 按钮。选择 *Add or create design sources* 并点击 *next*。点击绿色的 *plus* 按钮然后点击 *add file*。添加你所创建的.txt 文件然后点击 *finish*.

3-1-5. 综合实现此设计。

3-1-6. 生成比特流文件，下载到 Nexys4 开发板上，验证功能。

3-2.使用 ROM 实现一个 2 位乘 2 位的乘法器，将结果以二进制形式输出到四个 LED 灯上。

3-2-1. 打开 Vivado 并创建一个名为 *lab3_3_2* 的空工程。

3-2-2. 使用 ROM 和$readmemb 系统函数创建并添加一个拥有两个两位输入（a，b）和一个四位输出（product）的 Verilog 模块。

3-2-3. 在工程中添加适当的管脚约束的 XDC 文件，并加入相关联的管脚约束。将 a 分配给 **SW3-SW2**，b 分配给 **SW1-SW0**，*product* 分配给 **LED3-LED0**。

3-2-4. 创建并添加描述设计输出的文本文件。

3-2-5. 综合实现此设计。

3-2-6. 生成比特流文件，下载到 Nexys4 开发板上验证功能。

总结

在本次试验中，你学到了如何对多输出电路如译码器、编码器和 ROM 建模；你也学到了如何使用系统函数$readmemb 对 ROM 进行初始化。Verilog 还支持很多系统函数，你将在下一个实验中学习另外一些。