数制系统

简介

Verilog HDL 语言允许用不同的进制表示数字。底层的电路使用二进制处理数字，但是电路的输入输出常常是使用十进制的数字完成的。在本次实验中，你会学习到不同的表示方式以及将数字从一种表示方式转换到另一种。*请参阅 Vivado 教程来了解如何使用 Vivado 工具来创建工程并验证数字电路。*

目标

完成本次实验后，你将能做到：

- 定义不同进制的数
- 设计能将数据从一个进制转化到另一个的组合电路
- 设计能实现简单的加法操作的组合电路
- 学到一个能提高加法操作速度的技巧

数字表示 第一部分

在 Verilog HDL 中一个信号可能有如下四种基本的值：

i. 0：逻辑 0 或假
ii. 1：逻辑 1 或真
iii. x：未知
iv. z：高阻态（三态）

在门电路的输入端或是表达式中的 z 值通常会被看作一个 x 值。通常情况下，Verilog HDL 是区分大小写的，但是作为值来表示时是不区分大小写的。

在 Verilog HDL 中有三种常量类型：(i)整型，(ii)实型，和(iii)字符串。在整型常量和实型常量中可以使用下划线（_）来增加可读性。但它不能出现在第一个和最后一个字符的位置上。

整数可以写成 (i) 简单的十进制数或者是 (ii) 带进制格式。一个写成简单十进制形式的整数由一个可省略的+或是一个-和一串数字组成。比如，

15

-32

其中 15 可以被写成 5 比特格式的二进制数 01111，-32 可以写成 6 比特格式的 100000。简单的十进制数在硬件中最终会占用 32 比特。

一个数也可以被表示为进制格式，语法格式如下：

[size]'base value

其中 size 代表数的比特数，base（基）是 o 或者 O（代表八进制 octal），b 或者 B（代表二进制 Binary），d 或者 D（代表十进制 Decimal），h 或者 H（代表十六进制 hexadecimal）中的一个。Value 是一个在该进制下有效的数字的序列。Value 必须是无符号的。比如，

```
wire [4:0] 5'O37        //5 比特八进制表示
reg [3:0] 4'B1x_01      //4 比特二进制
wire [3:0] 4'd-4        //不合法，value 不能是负数
wire [11:0] 7'Hx        //7 比特 x 扩展到 xxxxxxx
```

如果 size 设定得比设定的常数的值的大小要大，这个数会被用 0 扩展到左边，除非最左边的比特是 x 或 z，此时也会相应地用 x 或 z 扩展。如果 size 设定得更小，那左边的多余的位会被忽略。如果 size 未设定，那么就会使用 32 比特。

1-1. 在模型中定义一个 4 比特的数并在最右边的七位数码管中显示出来

1-1-1. 打开 Vivado，创建一个名为 *lab2_1_1* 的空白工程（project）。

1-1-2. 创建并添加一个 Verilog module，在其中定义一个四比特的二进制格式的数并在最右边的七段数码管中将同一个数显示出来。考虑到你定义的数会在 0 到 9 之间，你可能会用到在 Lab1 的 4-2 中开发的模型。

1-1-3. 在工程中添加与相应的开发板对应的 XDC 文件，编辑文件，添加相应的接口。

1-1-4. 综合（Synthesize）并实现（implement）设计。

1-1-5. 生成比特流（bitstream），将它下载到 Basys3 或是 Nexys4 DDR 开发板上，并验证它的功能。

二进制编码 第二部分

虽然大多数处理器使用二进制格式来处理数据，但是输入输出一般是用一些编码过的格式完成的。一般来说，我们使用十进制来交换信息。因此十进制数必须被编码成二进制的表示方式。在最简单的二进制编码中，每个十进制数字会被替代为与之相等的二进制数。这种表示方式被叫做 Binary Coded Decimal (BCD 码) 或是 8-4-2-1 码（8，4，2，1 是表示每个比特位置的权重）。因为只有 10 个十进制数字，因此 1010 到 1111 是不正确的 BCD 码。下表展示了一些被广泛使用的十进制数字的二进制编码。这些编码是为了通信的可靠性和错误检查而被设计和使用的。

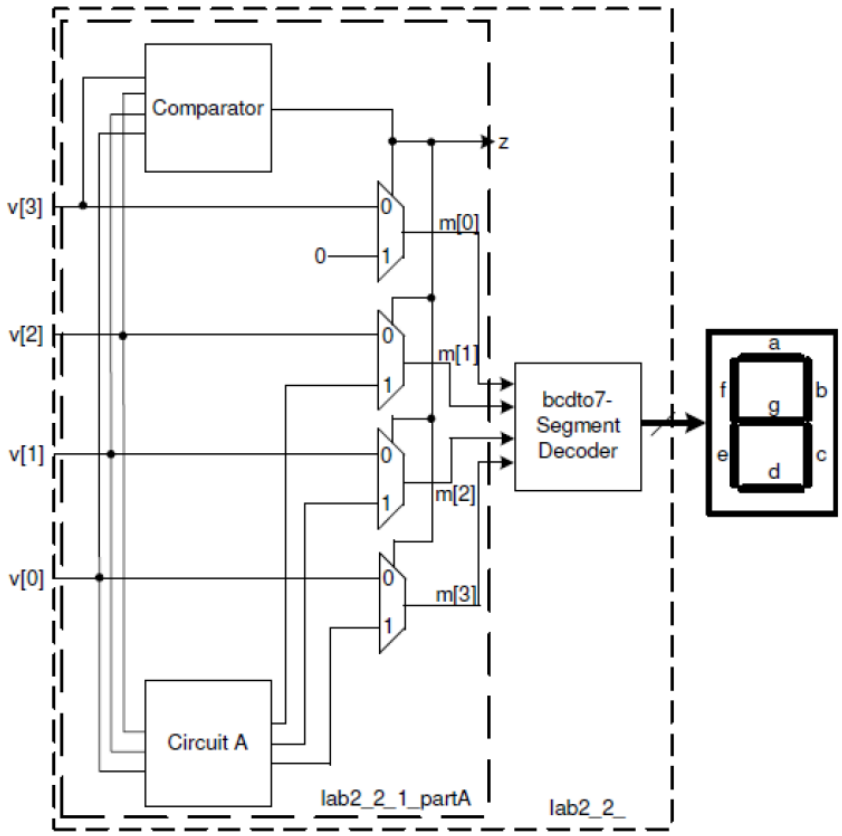
十进制数字	BCD 码(8-4-2-1 码)	6-3-1-1 码	余三码	五中选二码 (2-out-of-5 code)	格雷码
0	0000	0000	0011	00011	0000
1	0001	0001	0100	00101	0001
2	0010	0011	0101	00110	0011
3	0011	0100	0110	01001	0010
4	0100	0101	0111	01010	0110
5	0101	0111	1000	01100	1110
6	0110	1000	1001	10001	1010
7	0111	1001	1010	10010	1011
8	1000	1011	1011	10100	1001
9	1001	1100	1100	11000	1000

2-1. 将 4 比特的输入转化为相应的 2 位十进制数（BCD 码）后输出到 LED 灯（最高位）和最右边的七段数码管（更低位）上。仅使用数据流级建模。

设计一个能将 4 比特的二进制数 v 转换到它相应的 2 位十进制数，z 和 m 的电路。由于正确的输入范围是 0 到 15, 最高位可以只使用一个 LED 灯来显示。下表显示了要求的输出值。下图中给出了设计框图。它包含一个检查 v 值是否大于 9 的比较器（comparator），并使用比较器的输出来控制七段数码管的显示。提示：只要 v 大于二进制的 1001，m3 就会是 0。

v[3:0]	z	m[3:0]
0000	0	0000
0001	0	0001
0010	0	0010
0011	0	0011
0100	0	0100
0101	0	0101
0110	0	0110
0111	0	0111
1000	0	1000
1001	0	1001
1010	1	0000
1011	1	0001
1100	1	0010
1101	1	0011
1110	1	0100

1111	1	0101



2-1-1. 打开 Vivado，创建一个名为 *lab2_2_1* 的空白工程。

2-1-2. 创建并添加一个 Verilog 模块（命名为 lab2_2_1_partA）以 v[3:0]为输入，z 和 m[3:0]作为输出。创建并添加模块，使用数据流结构（dataflow constructs）来完成比较器数据流（comparator_dataflow），lab2 的电路 A 的数据流（lab2_circuitA_dataflow），实例化 2-to-1 多路选择器（Lab1 的），并将它们按上图连接起来。

2-1-3. 将提供的测试文件（lab2_2_1_partA_tb.v）添加到工程中。

2-1-4. 对设计进行仿真（行为仿真）200ns 并验证功能。

2-1-5. 拓展设计，创建一个包含 bcdto7segment_dataflow decoder（你在 Lab1 中开发的）的顶层模块（lab2_2_1）并提供一个 7 比特的输出 *seg0* 而不是输出 m。

2-1-6. 将与开发板对应的 XDF 文件添加到工程中，编辑文件添加相关的端口。将输入 v 赋给 **SW3 到 SW0**，z 赋给 **LED0**，seg0 赋给七段数码管显示阴极 **CA 到 CG**，将 an 赋给端口 **J17，J18，T9，J14，P14，T14，K2，U13**（对于 Nexys4 DDR）或是 **U2，U4，V4，W4**（对于 Basys3）。

2-1-7. 综合（Synthesize）并实现（implement）设计。

2-1-8. 生成比特流（bitstream），将它下载到 Basys3 或是 Nexys4 DDR 开发板上，并验证它的功能。

2-2. 建立一个五中选二码(2-out-of-5)编码模型并将一个编码自十进制输入数的 4 比特二进制数显示到 5 个 LED 灯上。仅使用数据流级建模。

2-2-1. 打开 Vivado 并创建一个名为 *lab2_2_2* 的空白工程。

2-2-2. 创建并添加一个有 4 比特的输入（*x[3:0]*）和 5 比特的输出（*y[4:0]*）的分层设计。只使用数据流级建模描述。

2-2-3. 将与开发板对应的 XDF 文件添加到工程中，编辑文件添加相关的端口。将 **SW3 到 SW0** 赋给 x，**LED4 到 LED0** 赋给 y。

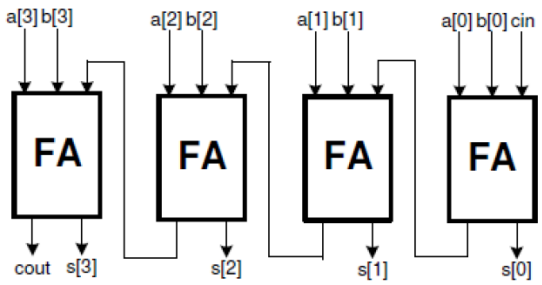
2-2-4. 综合（Synthesize）并实现（implement）设计。

2-2-5. 生成比特流（bitstream），将它下载到 Basys3 或是 Nexys4 DDR 开发板上，并验证它的功能。

执行加法 第三部分

当两个 1 比特的数相加时，它们可能会产生一个 2 比特的输出。比如，1 + 1 = 10（都是二进制）。当你将三个 1 比特的数相加时，结果也会是 2 比特，比如，1 + 1 = 11。这种简单的操作可以被视为将两个比特与一个更低位的操作的进位输入相加，结果产生一个和与一个进位输出——左边的比特是进位输出，右边的比特是和。下图显示了一个 4 比特的加法器。由于进位是像波纹一样从最低比特位(cin)向最高比特位传递的，这样的加法器也叫波纹进位加法器。

b	a	c_i	c_o	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



3-1. 仅使用数据流级建模创建一个 4 比特波纹进位加法器。

3-1-1. 打开 Vivado 并创建一个名为 *lab2_3_1* 的空白工程。

3-1-2. 创建并使用数据流级建模添加一个有三个输入（*a, b, cin*）和两个输出（*s* 和 *cout*）的名为 fulladder_dataflow 的 Verilog module。所有的输入和输出都应该是 1 比特位宽的。

3-1-3. 将提供的测试文件（fulladder_dataflow_tb.v）添加到工程中。

3-1-4. 对设计进行仿真（行为仿真）80ns 并验证功能。

3-1-5. 在工程中创建并添加一个有三个输入（*a, b, cin*）和两个输出（*s* 和 *cout*）的 Verilog module（名为 rca_dataflow），实例化全加器（FA）四次，并将它们按需连接起来。这里的 *a*，*b*，和 *s* 应该是一个 4 比特的向量，*cin* 和 *cout* 应该是 1 比特位宽。

3-1-6. 将与开发板对应的 XDF 文件添加到工程中，编辑文件添加相关的端口。将 ***SW4 到 SW7*** 赋给 *a*，将 ***SW0 到 SW3*** 赋给 *b*，***LED3 到 LED0*** 赋给 *s*，***SW15*** 赋给 *cin*，***LED15*** 赋给 *cout*。

3-1-7. 综合（Synthesize）并实现（implement）设计。

3-1-8. 生成比特流（bitstream），将它下载到 Basys3 或是 Nexys4 DDR 开发板上，并验证它的功能。

3-2. 修改 3-1 中的工程，将 4 比特的输入看作 BCD 码，完成加法，生成 BCD 码的结果并在 LED0 和最右边的七段数码管上显示出来。使用开关来输入两个 4 比特 BCD 数并用 SW15 输入进位。根据需求复用 2-1 和 3-1 中开发的模型。使用数据流级建模。

3-2-1. 打开 Vivado 并创建一个名为 *lab2_3_2* 的空白工程。

3-2-2. 按需求修改 3-1 中的工程，使之能完成需求的功能并将结果输出到 LED0 和最右边的七段数码管上显示。

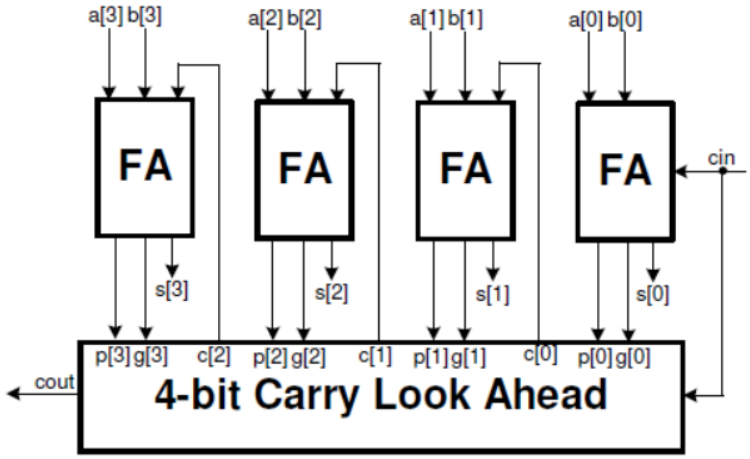
3-2-3. 将与开发板对应的 XDF 文件添加到工程中，编辑文件添加相关的端口。使用开关 ***SW72-SW0*** 来给两个 4 比特的 BCD 提供输入，***SW15*** 给进位。

3-2-4. 综合（Synthesize）并实现（implement）设计。

3-2-5. 生成比特流（bitstream），将它下载到 Basys3 或是 Nexys4 DDR 开发板上，并验证它的功能。

加快加法速度 第四部分

波纹进位加法器在处理两个很大的数（比如 8 比特，16 比特，32 比特）的加法时会花很长时间运算。为了降低运算时间，可以使用另一个结构，超前进位加法器。它的工作方式是，基于以下信息为每个比特位分别创建两个信号（*P* 和 *G*）：进位信号是否会从较低比特位传播过来（即至少有一个输入是 1），进位信号是否由此比特位生成（即两个输入均为 1），或者进位信号是否会被此比特位消灭（两个输入均为 0）。P 和 G 生成后，每个比特位的进位信号也生成了。



此处$P_i = A_i + B_i$，$G_i = A_iB_i$。在超前进位单元内，$C_{i+1} = G_i + P_iC_i$。这个加速是通过使C_i和相应的第 i 位一起生成而达到的。

4-1. 通过修改 3-1 中的工程并使用数据流级建模创建一个超前进位加法器电路

4-1-1. 打开 Vivado 并打开你在 3-1 中创建的工程。

4-1-2. 按需修改 3-1 中的工程，使用超前进位结构完成两个 4 比特的数的加法并将结果输出到 LED 灯上。由 SW15 提供进位。提示：你只需要将全加器修改为输出P_i和G_i，然后创建并添加另一个超前进位（CLA）模块来完成超前进位功能（即输入c_0和$p_i g_i$（i = 0 到 3）并输出c_4、p_g和g_g。

4-1-3. 修改 XDF 文件来通过 ***SW3-SW0*** 提供输入 *b*，通过 ***SW7-SW4*** 提供输入 *a*，***SW15*** 输入 *cin*。通过 ***LED15*** 输出 *cout*，***LED3-LED0*** 输出 *sum*。

4-1-4. 综合（Synthesize）并实现（implement）设计。

4-1-5. 生成比特流（bitstream），将它下载到 Basys3 或是 Nexys4 DDR 开发板上，并验证它的功能。

结论

在这个实验中，你会学到如何定义不同进制的数。你还会使用数据流设计不同的转换数字的电路。你还会学习到一个提高加法速度的技能。