计算机系统贯通课程-计算机系统 I

——Lab1-2: 数码管译码器

洪奕迅 3230102930@zju.edu.cn

史璐欣 3220104390@zju.edu.cn

计算机学院
浙江大学

2025 年 3 月 13 日

通知

- 本次代码不允许使用行为描述法．且本部分 Lab 代码有更新，提供了整体框架和部分代码，请进入 `src/lab1-2` 下执行 `git pull` 指令拉取更新，fork 仓库和已作修改发生冲突的同学可以手动复制新的 `SegDecoder` 代码/自行解决．已经使用结构描述/数据描述完成的同学可以不用更改．

关于向量和数组的语法补充

在 Verilog 中可以通过类似 `[63:0]test` 的方式定义一个向量，需要注意其与 `test[0:63]` 的区别：

- 向量 (Packed array) 是一个位宽为 n 的器件（体现为其在内存上连续），虽然我们有时会将每位拆开进行运算，但是其是一个整体. 在后续的实验中，我们会经常使用单个向量作为一个数据，再使用多个这种数据组成数组的方式来实现多位数据的输入. 并且向量的组成单位只能是 1bit 的各种类型或者其他向量.
- 数组 (unpacked array) 是多个位宽为其单位的器件，例如 test[63:0] 和 `[63:0]test[63:0]` 分别是一个组成单位向量位宽为 1 和 64 的长度为 64 的数组.
- 向量可以作为整体使用，但是需要注意实例化模块时`.O[0](test[0])`这种将向量拆开赋值的行为是不合法的. 数组不能直接作为整体使用
- 为了解决位宽不匹配的问题，可以使用索引来访问部分向量，`test[0:31]` 代表从 0-31 共 32 位数据，该方式与 `test[0+:32]` 和 `test[31-:32]` 等价，还可以通过大括号将多个信号组合成一个向量，如 `{a,b,c,d}` 就可以将四个向量拼接，也可以通过这个方式为数组赋值.

Sample from asic-world

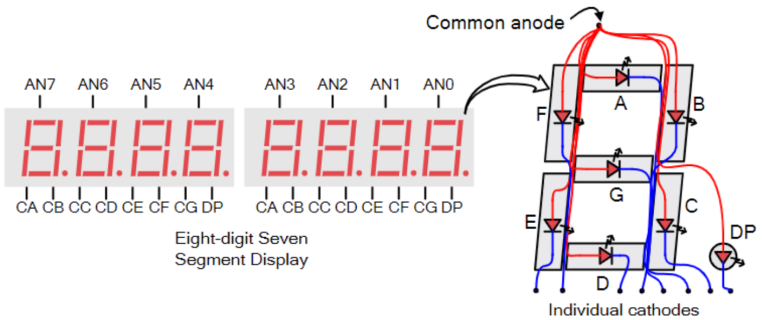
```
// packed array
reg [7:0] packed_array = 8'hAA;
// unpacked array
reg unpacked_array [7:0] = {1,0,1,0,1,0,1,0};

initial begin
    $display ("packed array[0]   = %b", packed_array[0]);
    $display ("unpacked array[0] = %b", unpacked_array[0]);
    $display ("packed array      = %b", packed_array);
    // Below one is wrong syntax
    $display("unpacked array = %b",unpacked_array);
    #1 $finish;
end
```

```
packed array[0]   = 0
unpacked array[0] = 0
packed array      = 10101010
unpacked array = 0
```

数码管译码器

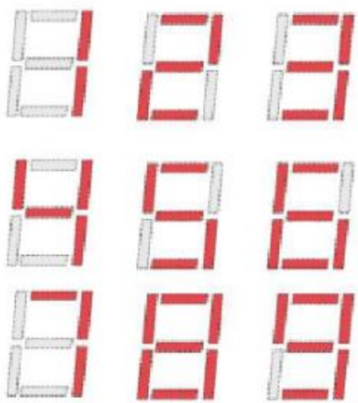
简而言之译码器就是把少输入变成多输出，再利用多输出来控制的部件，在数字电路里很重要．Lab1-1 中提到的 2-4Decoder(n-2^n Decoder) 就是一个实例．我们需要实现的数码管译码器本质就是把数字按照一些逻辑译码到数码管的各个发光管上，从而控制亮灭．

- 7 根发光管对应输出 a-g，以及一个小数点输出 P
- 共阳极发光管，输入 0 时发光



真值表

X	a	b	c	d	e	f	g
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0
A	0	0	0	1	0	0	0
B	1	1	0	0	0	0	0
C	0	1	1	0	0	0	1
D	1	0	0	0	0	1	0
E	0	1	1	0	0	0	0
F	0	1	1	1	0	0	0



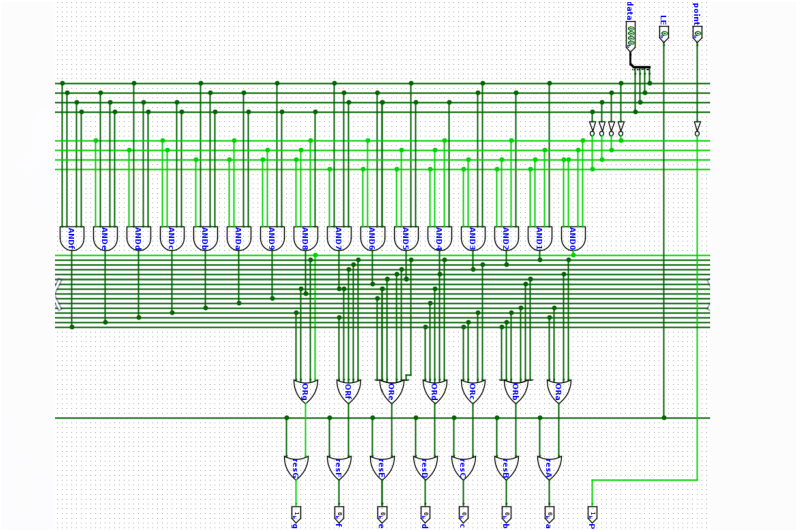
从真值表到电路图

- 发现 a 管在输入 1,4,b,d 时输出 1（不亮），事实上每一个数都对应一个特定情况，如 $\overline{0001}_2$（1 的二进制）输出 1，对应 $F = \overline{A}\ \overline{B}\ \overline{C}\ D$（假设分别对应 data 的四位输入）
- 以此类推，我们不难发现 a 的发光情况可以写为

$$F_a(A,B,C,D) = \overline{A}\ \overline{B}\ \overline{C}\ D + \overline{A}\ B\ \overline{C}\ \overline{D} + A\ \overline{B}\ C\ D + A\ B\ \overline{C}\ D$$

- 以同样的方式不难写出所有发光管的逻辑表达式，但我们会发现，其实很多发光管有共用的最小项，为了节省电路我们选择先画 16 个最小项对应的与门，让每个输出选择自己需要的最小项，最后再加上使能信号.

电路图



从电路图到代码

- 使用结构描述/数据描述法描述整个电路结构
- 我们已经提供了整体框架和部分代码

上板注意事项

```
DIR_REPO                ?= $(CURDIR)/../../repo
DIR_UPSTREAM    ?= $(DIR_REPO)/sys-project/lab1-2
DIR_BUILD               ?= $(CURDIR)/build

DIR_SIM                 ?= $(DIR_UPSTREAM)/sim
DIR_SRC                 ?= $(CURDIR)/submit
DIR_INCLUDE             ?= $(CURDIR)/include
DIR_SYN                 ?= $(DIR_UPSTREAM)/syn
DIR_TCL                 ?= $(DIR_UPSTREAM)/tcl



read_verilog [glob [file join $SRC_DIR *.v]]
read_verilog [glob [file join $SYN_DIR *.v]]
read_verilog [glob [file join $INCLUDE_DIR *.vh]]
read_xdc [glob [file join $SYN_DIR *.xdc]]
```

- `make bitstream`：对于使用 Makfile 的同学，还是可以一键生成 bitstream，但还是推荐看一下 `repo/sys-project/lab-2/tcl/vivado.tcl`，了解在这个过程中到底发生了什么.
- 对于使用 Vivado 图形化界面的同学，通过 `vivado.tcl` 可以知道要加入哪些资源文件，例如这次实验你需要加入 `src/submit,src/include,sys-project/lab1-2/syn` 下的所有源文件和约束文件.