Verilog 三段式状态机(FSM)

网上收集整理……

时序电路的状态是一个状态变量集合，这些状态变量在任意时刻的值都包含了为确定电路的未来行为而必需考虑的所有历史信息。

状态机采用 VerilogHDL 语言编码，建议分为三个 always 段完成。这是为什么呢？

设计 FSM 的方法和技巧多种多样，但是总结起来有两大类：第一种，将状态转移和状态的操作和判断等写到一个模块（process、block）中。另一种是将状态转移单独写成一个模块，将状态的操作和判断等写到另一个模块中（在 Verilog 代码中，相当于使用两个“always”block）。其中较好的方式是后者。其原因如下。

首先 FSM 和其他设计一样，最好使用同步时序方式设计，好处不再累述。而状态机实现后，状态转移是用寄存器实现的，是同步时序部分。状态的转移条件的判断是通过组合逻辑判断实现的，之所以第二种比第一种编码方式合理，就在于第二种编码将同步时序和组合逻辑分别放到不同的程序块（process，block）中实现。这样做的好处不仅仅是便于阅读、理解、维护，更重要的是利于综合器优化代码，利于用户添加合适的时序约束条件，利于布局布线器实现设计。

三段式建模描述 FSM 的状态机输出时，只需指定 case 敏感表为次态寄存器，然后直接在每个次态的 case 分支中描述该状态的输出即可，不用考虑状态转移条件。

三段式描述方法虽然代码结构复杂了一些，但是换来的优势是使 FSM 做到了同步寄存器输出，消除了组合逻辑输出的不稳定与毛刺的隐患，而且更利于时序路径分组，一般来说在 FPGA/CPLD 等可编程逻辑器件上的综合与布局布线效果更佳。

示例如下：

```
//第一个进程，同步时序 always 模块，格式化描述次态寄存器迁移到现态寄存器

always @ (posedge clk or negedge rst_n)     //异步复位
if(!rst_n)
      current_state <= IDLE;
else
      current_state <= next_state;//注意，使用的是非阻塞赋值

//第二个进程，组合逻辑 always 模块，描述状态转移条件判断
always @ (current_state)       //电平触发
    begin
       next_state = x;     //要初始化，使得系统复位后能进入正确的状态
       case(current_state)
       S1: if(...)
            next_state = S2;     //阻塞赋值
       ...
       endcase
end
```

//第三个进程，同步时序 always 模块，格式化描述次态寄存器输出

```
always @ (posedge clk or negedge rst_n)
...//初始化
case(next_state)
S1:
    out1 <= 1'b1;    //注意是非阻塞逻辑
S2:
    out2 <= 1'b1;
default:...      //default 的作用是免除综合工具综合出锁存器。
endcase
end
```

三段式并不是一定要写为 3 个 always 块，如果状态机更复杂，就不止 3 段了。

//注：==========

1. 三段 always 模块中，第一个和第三个 always 模块是同步时序 always 模块，用非阻塞赋值(“ <= ”)；第二个 always 模块是组合逻辑 always 模块，用阻塞赋值(“ = ”)。
2. 第二部分为组合逻辑 always 模块，为了抑制 warning 信息，对于 always 的敏感列表建议采用 always@（*）的方式。
3. 第二部分，组合逻辑 always 模块，里面判断条件一定要包含所有情况！可以用 else 保证包含完全。
4. 第二部分，组合逻辑电平要维持超过一个 clock，仿真时注意。
5. 需要注意：第二部分 case 中的条件应该为当前态（current_state），第三部分 case 中的条件应该为次态(next_state)。
6. 编码原则，binary 和 gray-code 适用于触发器资源较少，组合电路资源丰富的情况（CPLD），对于 FPGA，适用 one-hot code。这样不但充分利用 FPGA 丰富的触发器资源，还因为只需比较一个 bit，速度快，组合电路简单。
7. 初始化状态和默认状态。

一个完备的状态机（健壮性强）应该具备初始化状态和默认状态。当芯片加电或者复位后，状态机应该能够自动将所有判断条件复位，并进入初始化状态。需要注明的一点是，大多数 FPGA 有 GSR（Global Set/Reset）信号，当 FPGA 加电后，GSR 信号拉高，对所有的寄存器，RAM 等单元复位/置位，这时配置于 FPGA 的逻辑并未生效，所以不能保证正确的进入初始化状态。所以使用 GSR 企图进入 FPGA 的初始化状态，常常会产生种种不必一定的麻烦。一般的方法是采用异步复位信号，当然也可以使用同步复位，但是要注意同步复位的逻辑设计。解决这个问题的另一种方法是将默认的初始状态的编码设为全零，这样 GSR 复位后，状态机自动进入初始状态。

令一方面状态机也应该有一个默认（default）状态，当转移条件不满足，或者状态发生了突变时，要能保证逻辑不会陷入“死循环”。这是对状态机健壮性的一个重要要求，也就是常说的要具备“自恢复”功能。对应于编码就是对 case，if－else 语句要特别注意，要写完备的条件判断语句。VHDL 中，当使用 CASE 语句的时候，要使用“When Others”建立默认状态。使用“IF...THEN...ELSE”语句的时候，要用在“ELSE”指定默认状态。Verilog 中，使用“case”语句的时候要用“default”建立默认状态，使用“if...else”语句的注意事项相似。

8. 另外提一个技巧：大多数综合器都支持 Verilog 编码状态机的完备状态属性－－“full case”。这个属性用于指定将状态机综合成完备的状态，如 Synplicity 的综合工具（Synplify/Synplify Pro,Amplify，etc）支持的命令格式如下：

```
case (current_state) // synthesis full_case
2’b00 : next_state <= 2’b01;
2’b01 : next_state <= 2’b11;
2’b11 : next_state <= 2’b00;
//这两段代码等效
case (current_state)
2’b00 : next_state <= 2’b01;
2’b01 : next_state <= 2’b11;
2’b11 : next_state <= 2’b00;
default : next_state <= 2bx;
```

9. Synplicity 还有一个关于状态机的综合属性，叫“synthesis parallel_case”，其功能是检查所有的状态是“并行的”（parallel），也就是说在同一时间只有一个状态能够成立。

10.状态机的定义可以用 parameter 定义，但是不推荐使用`define 宏定义的方式，因为`define 宏定义在编译时自动替换整个设计中所定义的宏，而 parameter 仅仅定义模块内部的参数，定义的参数不会与模块外的其他状态机混淆。

11.对于状态比较多的状态机，可以将所有状态分为几个大状态，然后再使用小状态，可以减少状态译码的时间。

12.在代码中添加综合器的综合约束属性或者在图形界面下设置综合约束属性可以比较方便的改变状态的编码。

如 VHDL 的示例：

Synplicity:

```
attribute syn_encoding : string;
attribute syn_encoding of <signal_name> : type is "value ";
-- The syn_encoding attribute has 4 values : sequential, onehot, gray and safe.
Exemplar:
-- Declare TYPE_ENCODING_STYLE. attribute
-- Not needed if the exemplar_1164 package is used
type encoding_style. is (BINARY, ONEHOT, GRAY, RANDOM, AUTO);
attribute TYPE_ENCODING_STYLE. encoding_style;
...
attribute TYPE_ENCODING_STYLE. of <typename> : type is ONEHOT;
```

Verilog 示例：

```
Synplicity:
Reg[2:0] state; /* synthesis syn_encoding = "value" */;
// The syn_encoding attribute has 4 values : sequential, onehot, gray and safe.
   Exemplar:
Parameter /* exemplar enum <type_name> */ s0 = 0, s1 = 1, s2 = 2, s3 = 3, S4 = 4;
Reg [2:0] /* exemplar enum <type_name> */ present_state, next_state ;
```

13.小技巧：仔细检查综合器的综合报告，目前大多数的综合器对所综合出的 latch 都会报

“warning”，通过综合报告可以较为方便地找出无意中生成的 latch。

//==========