从电路设计的角度入门VerilogHDL

邸志雄，西南交通大学

www.dizhixiong.cn

CONTENTS

目录

知识点列表

变量

- wire型：表示电路模块中的连线， 仿真波形中不可见。
- reg型：占用仿真环境的物理内存，会显示在仿真波形中。

凡是在 always initial 语句中赋值的变量， 一定是reg型

凡是在 assign语句中赋值的变量，一定是wire型

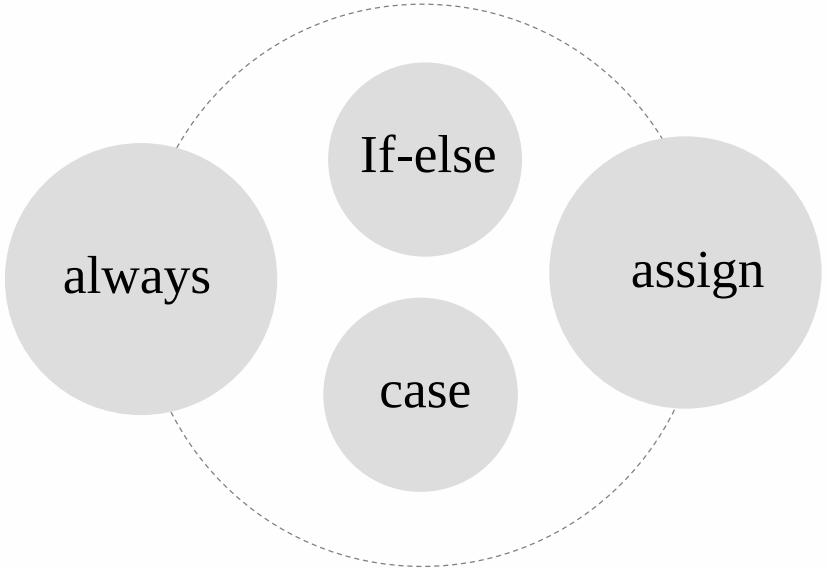
一定要注意：

reg变量仅仅是语法定义，**不等于**电路中寄存器；**只有时序电路**中的reg变量才会被逻辑综合工具认为是寄存器

使用时，需要定义wire与reg的位宽，否则默认位宽为1bit

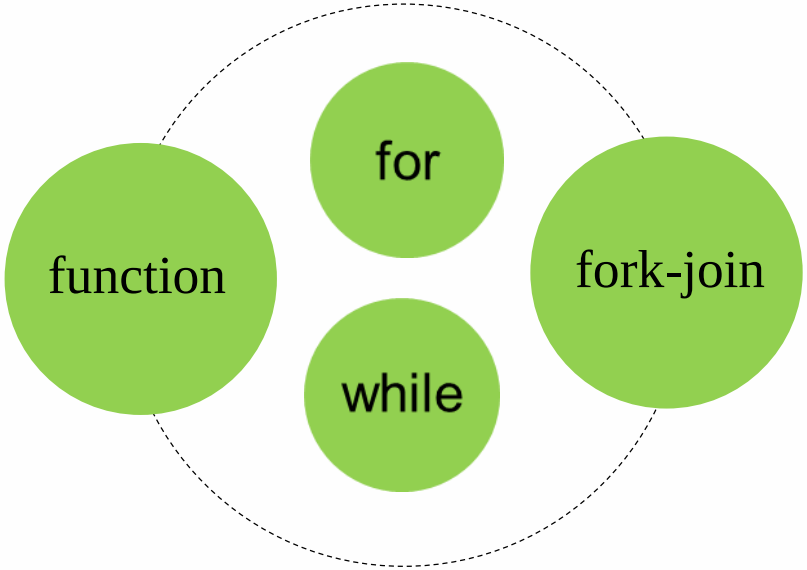
知识点列表

可综合描述必须使用的关键词



可综合“四大法宝”

可综合风格禁止出现



可综合描述必须使用的关键词

assign

- 连续驱动赋值：assign a = b | c ；
- 赋值的对象应该是wire类型
- “=”右边的任何变化都会被立即计算并驱动给等号左边
- 用于对信号连接，重命名，简单组合逻辑

```
assign  a = b | c;
assign  a = b ? c : d;
assign  a = d;
assign  a = e[3:2];
assign  a = (b | c) & (d ^ e);
```

可综合描述必须使用的关键词

always语句块

- 后接敏感列表，用@表示；
- always@(a or b or c)表示只要a,b,c中有一个产生变换，则执行该always块；
- 一般包含begin…end语句组；
- always@(*)表示自动将该always块中所有引用的信号都自动添加到敏感列表中；
- always@(posedge clk or negedge rst_n)代表只在clk上升沿或rst_n下降沿上执行该always块 ——寄存器，时序逻辑
- always@(posedge clk or posedge rst)
- always@(posedge clk)
- always@(negedge clk)

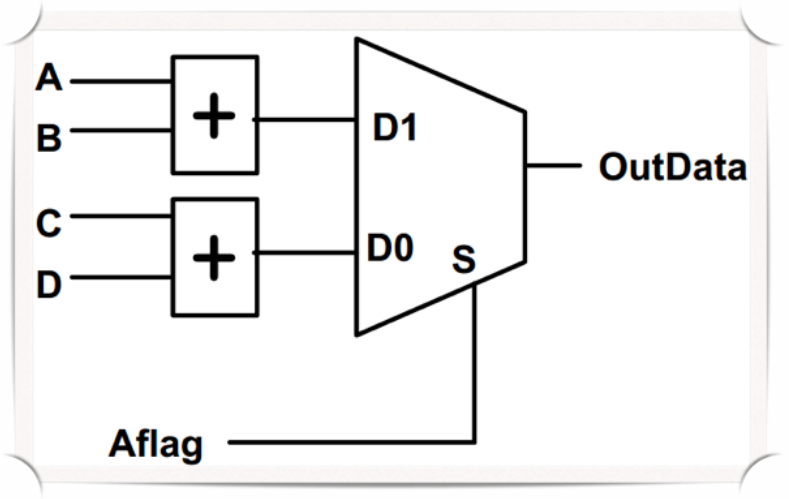
知识点列表

MUX、触发器与锁存器、组合逻辑、时序逻辑、存储器

1.电路单元描述：MUX

映射的硬件结构：Multiplexing Hardware （多路选择器）输出结果由输入的选择条件决定

```
always  @ ( * )
begin
    if (aflag == 1'b 1)
        outdata = A+B;
    else
        outdata = C+D;
end
```

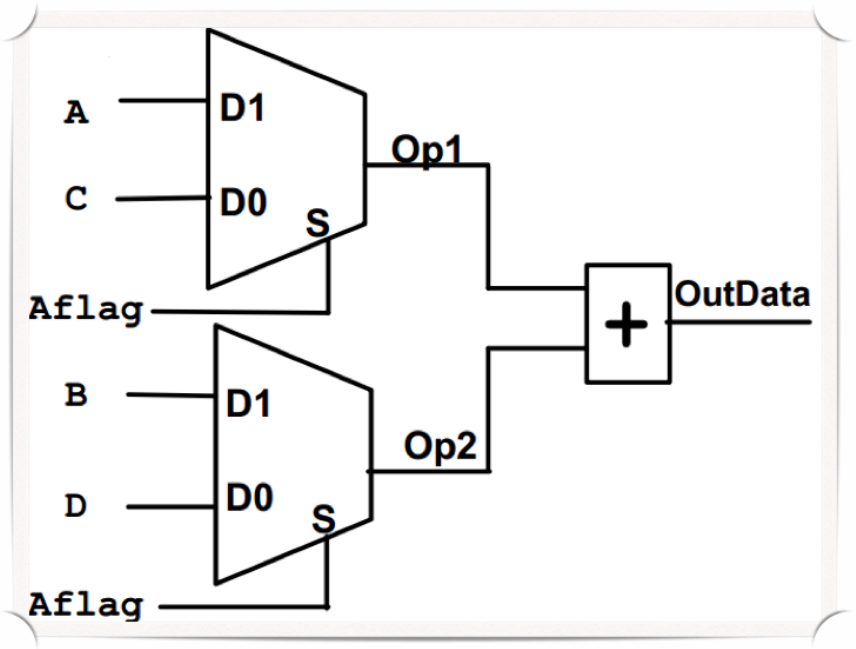


1.电路单元描述：MUX

映射的硬件结构：Multiplexing Hardware （多路选择器）输出结果由输入的选择条件决定

```
always  @ ( * )
begin
    if (aflag == 1'b 1)
    beign
        OP1= A ;
        OP2= B ;
    end
    else  begin
        OP1= C ;
        OP2= D ;
    end
end

assign outdata = OP1 + OP2
```

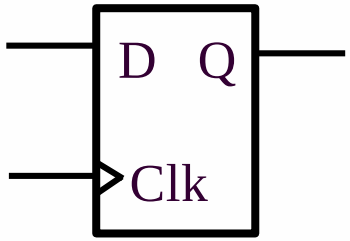


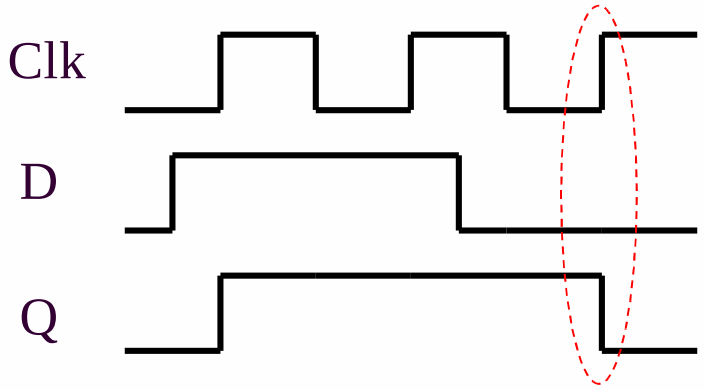
if-else，有优先级

case，无优先级

2.电路单元描述：触发器与锁存器

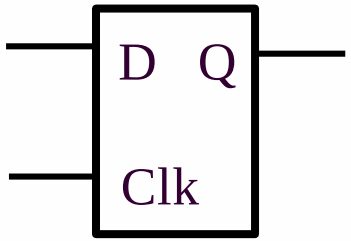
- 寄存器
  - Register （flip-flops）
  - 边沿触发
  - 输入 - 输出不透明

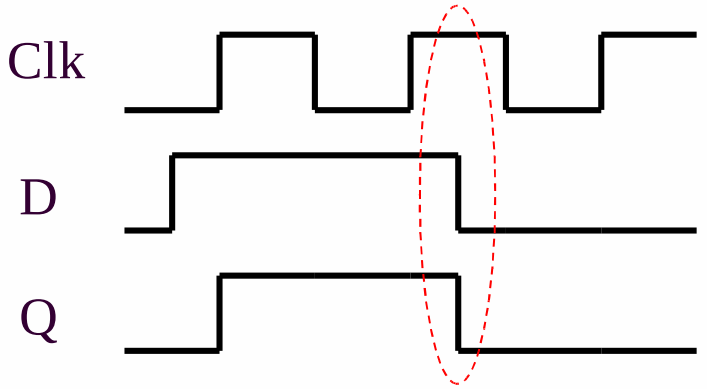




时钟处于上升沿（或下降沿）时存入数据（D→Q），其它时间保持数据

- 锁存器
  - Latch
  - 电平敏感
  - 输入 - 输出透明





时钟处于高电平（或低电平）时存入数据（D→Q），其保持据

Latch易传播毛刺，且难分析时序，因此，仅用于异步电路和低功耗电路，除此以外，禁止出现latch

2.电路单元描述：触发器与锁存器

- 易引入latch的途径：使用不完备的条件判断语句

缺少else

```
always @(cond1 or data in)
begin
   if(cond1==1 )
      data_out=data_in;
end
```

缺少default

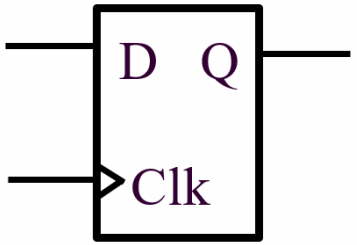
```
casex ({sel0,  sel1,  se12})
   2'd0: z=d;
   2'd1: z=c;
   2'd2: z=b;
endcase
```

防止产生非目的性Latch的措施：

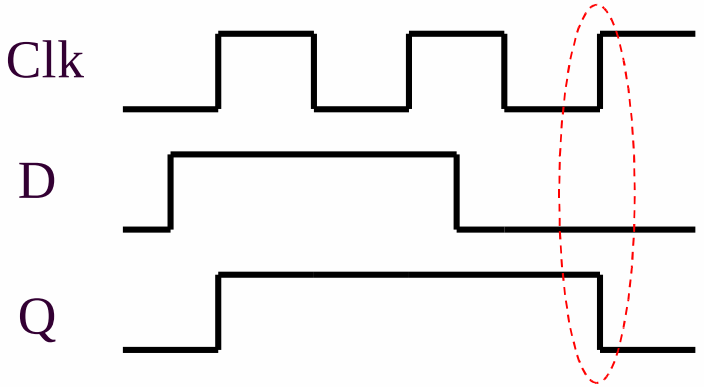
- 使用完备的if...else语句
- 为每个输入条件设计输出操作，为case语句设置default操作
- 仔细检查综合器生成的报告，latch会以warning的形式报告

2.电路单元描述：触发器与锁存器

触发器

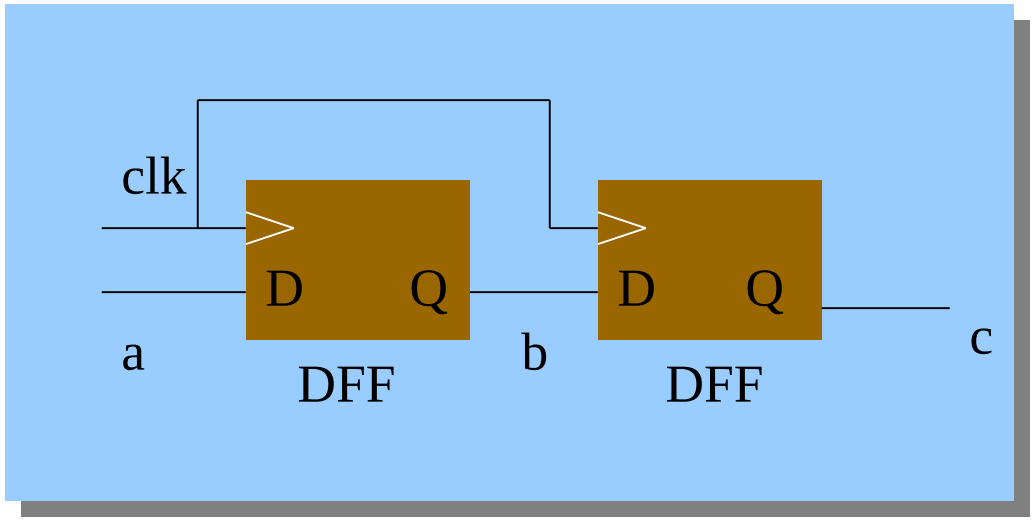



```
always  @ (posedge clk  or  negedge rst_n )
begin
    if (rst_n == 1'b 0)
        Q  <= 1'b 0 ;
    else
        Q  <= D ;
end
```



2.电路单元描述：触发器与锁存器

```
always @(posedge clk)
        begin
            b <=  a ;
            c <= b;
        end
```



非阻塞赋值在块结束时才完成赋值操作!

注：c的值比b的值落后一个时钟周期!

3.电路单元描述：组合逻辑

使用 assign 或者 always 块 描述，二者在描述方式上是等价的

简单的组合逻辑使用“**assign 和 : ?”语句**描述比较清晰，但是如果所描述的组合逻辑过于复杂，则需要很多条 assign 语句或者嵌套?语句来描述，不易解读，此时推荐使用 **always块** 描述。

```
always @ (CS or addr)
   if (CS)
      {cs1, cs2, cs3, cs4} = 4'b 1111;
   else
      begin
         case (addr[7:6])
            chip1_decode: {cs1, cs2, cs3, cs4} = 4'b 0111;
            chip2_decode: {cs1, cs2, cs3, cs4} = 4'b 1011;
            chip3_decode: {cs1, cs2, cs3, cs4} = 4'b 1101;
            chip4_decode: {cs1, cs2, cs3, cs4} = 4'b 1110;
         endcase
      end
```

3.电路单元描述：组合逻辑

使用 assign 或者 always 块 描述，二者在描述方式上是等价的

简单的组合逻辑使用**“assign 和 : ?”语句**描述比较清晰，但是如果所描述的组合逻辑过于复杂，则需要很多条 assign 语句或者嵌套?语句来描述，不易解读，此时推荐使用 **always块** 描述。

```
wire    cs1, cs2, cs3, cs4;
assign cs1 = (!CS && (addr[7:6] == chip1_decode))? 0 : 1 ;
assign cs2 = (!CS && (addr[7:6] == chip2_decode))? 0 : 1 ;
assign cs3 = (!CS && (addr[7:6] == chip3_decode))? 0 : 1 ;
assign cs4 = (!CS && (addr[7:6] == chip4_decode))? 0 : 1 ;
```

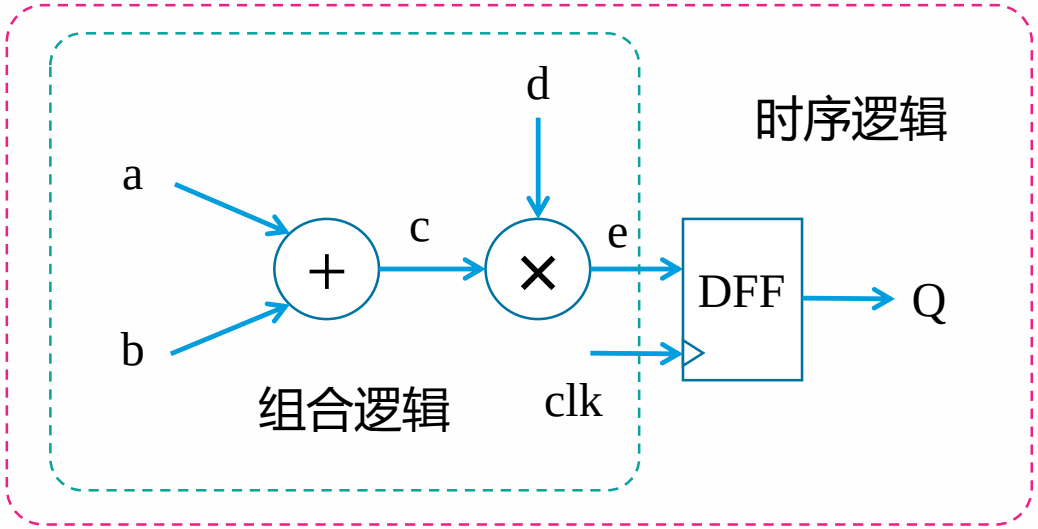
3.电路单元描述：组合逻辑

使用 assign 或者 always 块 描述，二者在描述方式上是等价的

注意事项

- 组合逻辑的赋值方式，必须使用“=”， 即 阻塞赋值
- 组合逻辑无保存或者锁存功能，因此，没有复位信号与相关的复位逻辑

4.电路单元描述：时序逻辑



- 组合逻辑 必须使用 “=”
- 时序逻辑 必须使用 “<=”

- 时序逻辑必须要复位
- 每加一个DFF，输出就推迟1个时钟周期

```
always  @ ( * )
begin
   c  = a+b  ;
   e  = c * d ;
end
```

```
always  @ (posedge clk  or  negedge rst_n )
begin
   if (rst_n == 1'b 0)
      Q  <= 0 ;
   else
      Q  <= e  ;
end
```

等价

```
always  @ (posedge clk  or  negedge rst_n )
begin
   if (rst_n == 1'b 0)
      Q  <= 0 ;
   else
      Q  <= (a+b) * d   ;
end
```

5.存储器

逻辑电路设计会经常使用一些单口RAM、双口RAM 和 ROM 等类型的存储器。

Verilog 语法中基本的存储单元定义格式如下：

reg [datawidth] MemoryName [addresswidth];

例如定义一个数据位宽为8bit，地址为 64 位宽 的 RAM8x64 ，则可定义为:

reg [7:0] RAM8x64 [0:63];

在使用存储单元时，不能直接引用存储器某地址的某比特位值，如想对地址为 "32 "的第 2bit 和高 2bit 的值进行操作，则下面这两种描述都是错误的。

RAM8x64 [32] [2]

RAM8x64 [32] [6:7]

正确的操作方法是，先将存储单元赋值给某个寄存器，然后在对该寄存器的某位进行相关操作

5.存储器

不推荐使用Verilog 直接建模 RAM。

FPGA 内嵌的 RAM 资源大致分为两类： 块RAM (Block RAM) 资源和分布式RAM 资源(Distributed RAM ， 是一种基于特殊底层逻辑单元，通过查找表和触发器实现的 RAM 结构)

推荐：

通过器件商的开发平台中内嵌的IP 生成器，在图形化界面中直接选择存储器类型(如双口 RAM、单RAM 、 ROM 和分布式RAM)，配置存储器参数，生成相应 IP ， 然后在用户逻辑中直接调用该IP。

参考资料

轻松成为设计高手:VerilogHDL实用精解. EDA先锋工作室. 北京航空航天大学出版社. 2012年.

中国大学MOOC平台：芯动力——硬件加速设计方法

https://www.icourse163.org/course/SWJTU-1207492806

谢谢！

邸志雄，西南交通大学

www.dizhixiong.cn