信号命名和定义应该明确

在设计中，我们不断的给目录、源代码、文件、函数、变量、参数、类、封包进行命名与定义。当一件工作需要进行的次数非常之多，足以证明它是不可或缺的基本工作。我们一定要知道一点，基础工作是整个项目的基石。忽视抑或是轻视基础工作是一件非常错误的工作理念。我们需要用最严谨认真的态度去对待，同时作为回报，它将令你的作品显得专业而优雅。

我们以信号的定义为例来说明这个问题。先来看这么一组代码：

```
1  always @(posedge clk or negedge rst_n)begin
2      if(!rst_n)begin
3          cnt <= 0;
4      end
5      else if(add_cnt)begin
6          if(end_cnt)
7              cnt <= 0;
8          else
9              cnt <= cnt + 1;
10     end
11 end
12
13 assign add_cnt = flag1||flag2 ;
14 assign end_cnt = add_cnt && cnt==x-1 ;
15
16 always  @(posedge clk or negedge rst_n)begin
17     if(rst_n==1'b0)begin
18         flag1 <= 1'b0;
19     end
20     else if(en1)begin
21         flag1 <= 1'b1;
22     end
23     else if(end_cnt)begin
24         flag1 <= 1'b0;
25     end
26 end
27
28 always  @(posedge clk or negedge rst_n)begin
29     if(rst_n==1'b0)begin
30         flag2 <= 1'b1;
31     end
32     else if(en2)begin
```

```
33          flag2 <= 1'b1;
34      end
35      else if(end_cnt)begin
36          flag2 <= 1'b0;
37      end
38  end
39
40  always  @(*)begin
41      if(flag1)
42          x = 5;
43      else if(flag2)
44          x = 7;
45      else begin
46          x = 0;
47      end
48  end
```

这组代码的功能是当 en1 时计数 5 下；en2 计数 7 下。在这组代码中，en1 时 flag1 拉高；end-cnt 时 flag1 变低；en2 时 flag2 拉高；end-cnt 时 flag2 变低；也就是在 flag1 或者 flag2 时加一，然后用 flag1 和 flag2 分别区分计数 5 下和 7 下。

尽管能够实现功能，但是在这组代码中，存在信号定义不明确得现象。 flag1 和 flag2 到底是什么意思？是表示 flag1（flag2）时 en1 产生，还是 en1（en2）时的计数状态？为说明这一点就得用到 XXXXX（写加一条件时需要用到 add_cnt = flag1||flag2)语句。

这里重申一下我们很重要的那条简单原则，一个代码（信号）只做一件事且做好这件事！按照这个规则，思路就是这样了：用一个信号 flag1 来表示计数状态，另外一个信号 flag2 表示是由 en1 还是 en2 所产生。那么，加一与否的条件非常简单，就是是否处于工作状态（flag1）；同理，计数 5 或者 7 下只需要使用 flag2 一个信号。那么代码就会是这样：

```
always @(posedge clk or negedge rst_n)begin
    if(!rst_n)begin
        cnt <= 0;
    end
    else if(add_cnt)begin
        if(end_cnt)
            cnt <= 0;
        else
            cnt <= cnt + 1;
    end
end

assign add_cnt = flag1 ;
assign end_cnt = add_cnt && cnt==x-1 ;

always  @(posedge clk or negedge rst_n)begin
    if(rst_n==1'b0)begin
        flag1 <= 1'b0;
    end
    else if(en1||en2 )begin
        flag1 <= 1'b1;
    end
    else if(end_cnt)begin
        flag1 <= 1'b0;
    end
end

always  @(posedge clk or negedge rst_n)begin
    if(rst_n==1'b0)begin
        flag2 <= 1'b1;
    end
    else if(en1)begin
        flag2 <= 1'b0;
    end
    else if(en2)begin
        flag2 <= 1'b1;
    end
end

always  @(*)begin
    if(flag==0)
        x = 5;
    else
        x = 7;
end
```

看到这里，也许有些朋友会觉得：好像区别没那么大啊？ok，我们假设一下，如果程序中不仅是是 en1,en2，而是有 en3,en4……enX，又或者将来需要维护和优化，这两者的区别将会天壤之别。

关于信号定义方面，《至简设计法》的作者潘文明给出了一个近乎完美的答案。例如在计数器代码设计中的“架构八步法”，第一步就是明确定义信号，用具体、清晰且无疑异的语句，定义每个信号所要实现的功能，以及重点描述信号的变化情况。如下图中的信号列表。

信号列表。(4)（用文字版）

信号名	I/O	位宽	说明
clk	I	1	系统工作时钟
rst_n	I	1	系统复位信号
Din_sop	I	1	当 vld=1 时才有效，输入报文头指示信号
Din_eop	I	1	当 vld=1 时才有效，输入报文尾指示信号
Din_vld	I	1	输入数据有效标志，高电平有效
Din_err	I	1	输入报文错误标志，在 eop 有效时才有效
din	I	8	输入数据总线
Dout_sop	O	1	当 vld=1 时才有效，输出报文头指示信号
Dout_eop	O	1	当 vld=1 时才有效，输出报文尾指示信号
Dout_vld	O	1	输出数据有效标志，高电平有效
dout	O	8	输出数据总线
Dout_err	O	1	输出报文错误标志，在 eop 有效时才有效

从中可以看出，优秀的 FPGA 设计师一开始就从顶层结构明确定义信号，将可能出现的混乱从根源上解决。这样的思路和方法实在非常值得我们每一位从业者学习和借鉴。

FPGA 培训定位非常明确，我们坚持帮助学员实现两个目标：

具备独立项目开发能力+FPGA 就业。

我们认为，只要具备了独立开发项目能力，完成工作项目需求、导师项目要求、高薪就业这些就都不成问题。经过明德扬专业的培训，您将完全掌握到一种科学规范的 FPGA 设计方法，运用这套方法可以完成所有 FPGA 项目设计。完全具备 FPGA 工程师的能力，足以满足企业或实际项目的要求。