在 FPGA 中何时用组合逻辑或时序逻辑

数字逻辑电路分为组合逻辑电路和时序逻辑电路。时序逻辑电路是由组合逻辑电路和时序逻辑器件构成（触发器），即数字逻辑电路是由组合逻辑和时序逻辑器件构成。所以 FPGA 的最小单元往往是由 LUT（等效为组合逻辑）和触发器构成。

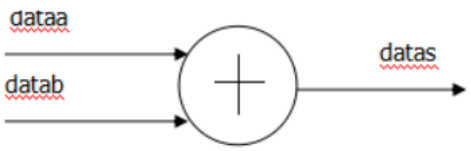
在进行 FPGA 设计时，应该采用组合逻辑设计还是时序逻辑？这个问题是很多初学者不可避免的一个问题。

设计两个无符号的 8bit 数据相加的电路。

组合逻辑设计代码：

```
module adder_8bit (

  input   wire        [7:0]   dataa,
  input   wire        [7:0]   datab,

  output  wire        [8:0]   datas
);

  assign datas = dataa + datab;

endmodule
```

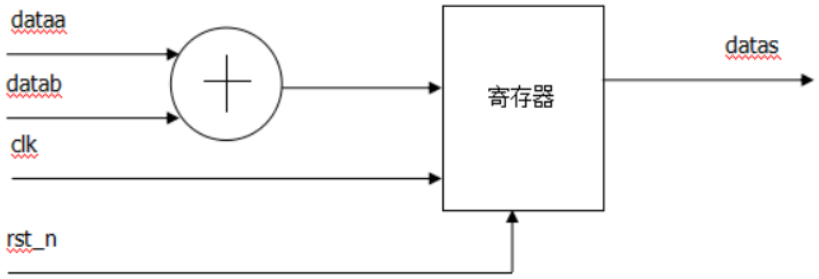
对应的电路图：



时序逻辑对应代码为：

```
module adder_8bit (

  input   wire                clk,
  input   wire                rst_n,

  input   wire        [7:0]   dataa,
  input   wire        [7:0]   datab,

  output  reg         [8:0]   datas
);

  always @ (posedge clk, negedge rst_n) begin
    if (rst_n == 1'b0)
      datas <= 9'd0;
    else
      datas <= dataa + datab;
  end

endmodule
```

对应的电路为：



可以思考一下，这个两种设计方法都没有任何错误。那么在设计时应该用哪一种呢？在设计时，有没有什么规定必须要用组合逻辑或者时序逻辑？

归纳如下：

1. 带有反馈的必须用时序逻辑

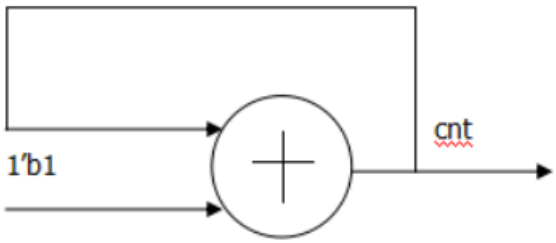
何为带有有反馈？即输出结果拉回到输入。

如：自加一计数器。

代码为：

```
//  assign cnt  = cnt + 1'b1;

  always @ (*) begin
    cnt = cnt + 1'b1;
  end
```

对应的电路图：



这种电路在工作时，就会出现无限反馈，不受任何控制，一般情况下，我们认为结果没有任何意义。

和上面的情况类似的还有取反。

```
assign flag = ~flag;
```

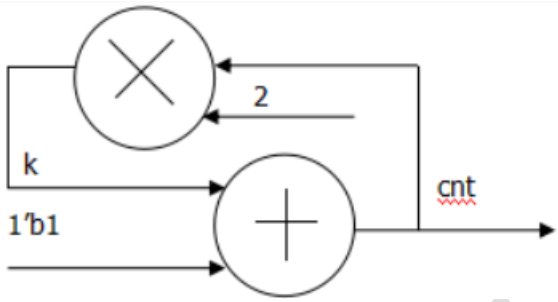
类似情况还有很多就不在一一列举。

上述说的情况都是直接带有反馈，下面说明间接反馈。

代码为：

```
assign k = cnt * 2;

 always @ (*) begin
     cnt = k + 1'b1;
 end
```

从代码上来看，没有什么明确反馈，下面看实际对应的电路。



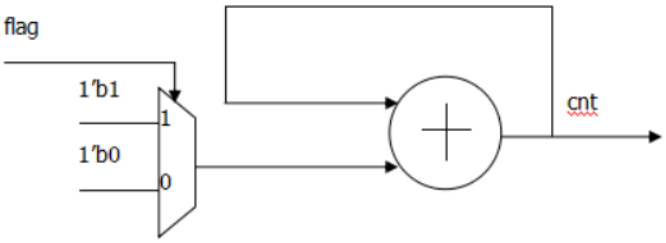
从实际的电路上来看，一旦运行起来，还是会出现无限反馈，不受任何控制。

还有一种情况是带有控制的反馈。

设计代码为：

```
always @ (flag) begin
  if (flag == 1'b1)
    cnt = cnt + 1'b1;
  else
    cnt = cnt;
end
```

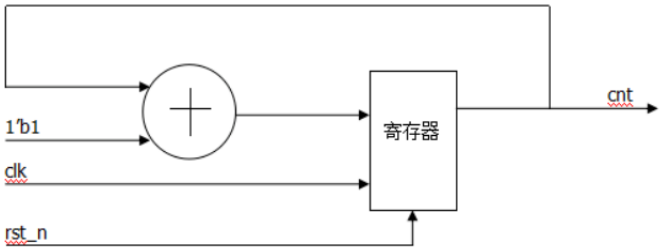
这个电路可以等效为：



在 flag 等于 1 期间，此电路依然会无限制的反馈，无法确定在此期间进行了多少次反馈。

从代码的角度理解是 flag 变化一次，加一次。可是对应于电路后，和预想的是不相同的。

说了这么多的这么多不对的情况，下面考虑正确的情况。



设计代码为：

```
always @ (posedge clk, negedge rst_n) begin
  if (rst_n == 1'b0)
    cnt <= 4'd0;
  else
    cnt <= cnt + 1'b1;
end
```

在上述的电路中，clk 每来一个上升沿，cnt 的数值增加一。可以用作计时使用。

利用寄存器将反馈路径切换即可。此时的反馈是可控制，并且此时的结果就有了意义。

其他的反馈中，加入寄存器即可。而加入寄存器后，就变为时序逻辑。

2.根据时序对齐关系进行选择

在很多的设计时，没有反馈，那么应该如何选择呢？

举例说明：输入一个八位的数据（idata），然后将此八位数据进行平方后，扩大 2 倍，作为输出。要求输出结果（result）时，将原数据同步输出（odata），即数据和结果在时序上是对齐的。

设计代码为：

```
assign odata = idata;
assign result = 2 * (idata * idata);
```

这种设计方法是可以的，因为都采用组合逻辑设计，odata 和 result 都是和 idata 同步的，只有逻辑上的延迟，没有任何时钟的延迟。

另外一种设计代码为：

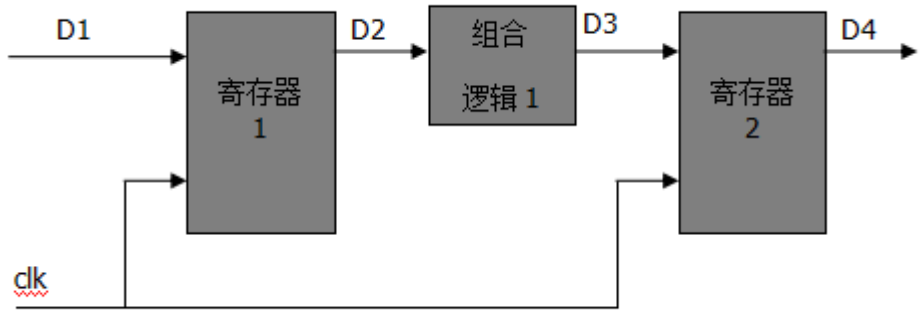
```
assign odata = idata;

always @ (posedge clk, negedge rst_n) begin
  if (rst_n == 1'b0)
    odata <= 17'd0;
  else
    result <= 2 * (idata * idata);
end
```

这种设计方法为错误，odata 的输出是和 idata 同步的，而 result 的输出将会比 idata 晚一拍，最终导致 result 要比 odata 晚一拍，此时结果为不同步，设计错误。

修改方案为：将 result 的寄存器去掉，修改为组合逻辑，那就是第一种设计方案。第二种为将 odata 也进行时序逻辑输出，那么此时 odata 也将会比 idata 延迟一拍，最终结果为 result 和 odata 同步输出。

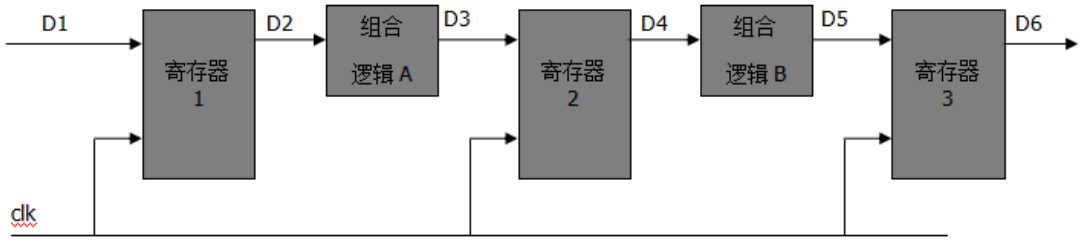
3 根据运行速度进行选择

在数字逻辑电路中，中间某一部分为组合逻辑，两侧的输入或者输出也会对延迟或者输入的数据速率有一定的要求。



组合逻辑 1 越复杂延迟越大，而导致的结果就是 clk 的时钟速率只能降低，进而导致设计结果失败。

当组合逻辑 1 无法进行优化时，还想要达到自己想要的速度时，我们可以进行逻辑拆分，增加数据的输出潜伏期，增加数据的运行速度。



将组合逻辑 1 的功能拆分为组合逻辑 A 和组合逻辑 B，此时，输入的数据得到结果虽然会多延迟一拍，但是数据的流速会变快。

那么这个和选用组合逻辑和时序逻辑有什么关系呢？

举例说明：目前要设计模块 A，不涉及反馈，不涉及时序对齐等，可以采取组合逻辑设计也可以采用时序逻辑设计。

模块 A 的输出连接到模块 B，经过一些变换（组合逻辑 N）连接到某个寄存器 K 上。如果模块 A 采用组合逻辑，那么模块 A 的组合逻辑和模块 B 到达寄存器 K 之前的组合逻辑 N 会合并到一起。那么此时组合逻辑的延迟就会变得很大，导致整体设计的时钟速率上不去。

当运行速率比较快时，建议对于复杂的组合逻辑进行拆分，有利于时序分析的通过。

在上述的三个规则中，第一个和第二个用的是最多的，第三个在设计时，有时不一定能够注意到，当出现时序违例时，知道拆分能够解决问题就可以。

上述参考资料链接：https://bbs.elecfans.com/jishu_2339061_1_1.html