数字系统高级设计技术

主 讲：何宾

Email：hebin@mail.buct.edu.cn

2019.06

并行和流水线
--并行设计

串行设计是最常见的一种设计

- 当一个功能模块对输入的处理是分步骤进行的，而且后一步骤依赖于前一步骤的结果时，功能模块的设计就需要采用串行设计的思想。

并行处理就是用几个处理流程同时处理到达的负载

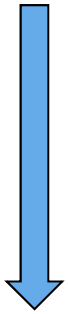
- 并行处理要求这些处理任务之间是不相关的，彼此之间是不互相依赖的，如果存在相互依赖就很难用并行处理的方法。

并行和流水线
--并行设计（乘法运算）

首先给出该运算的数学表达式

$$y = a_0 \times b_0 + a_1 \times b_1 + a_2 \times b_2 + a_3 \times b_3$$

```
module top(
    input [7:0] a0,
    input [7:0] b0,
    input [7:0] a1,
    input [7:0] b1,
    input [7:0] a2,
    input [7:0] b2,
    input [7:0] a3,
    input [7:0] b3,
    output [17:0] y
    );
assign y=a0*b0+a1*b1+a2*b2+a3*b3;
endmodule
```

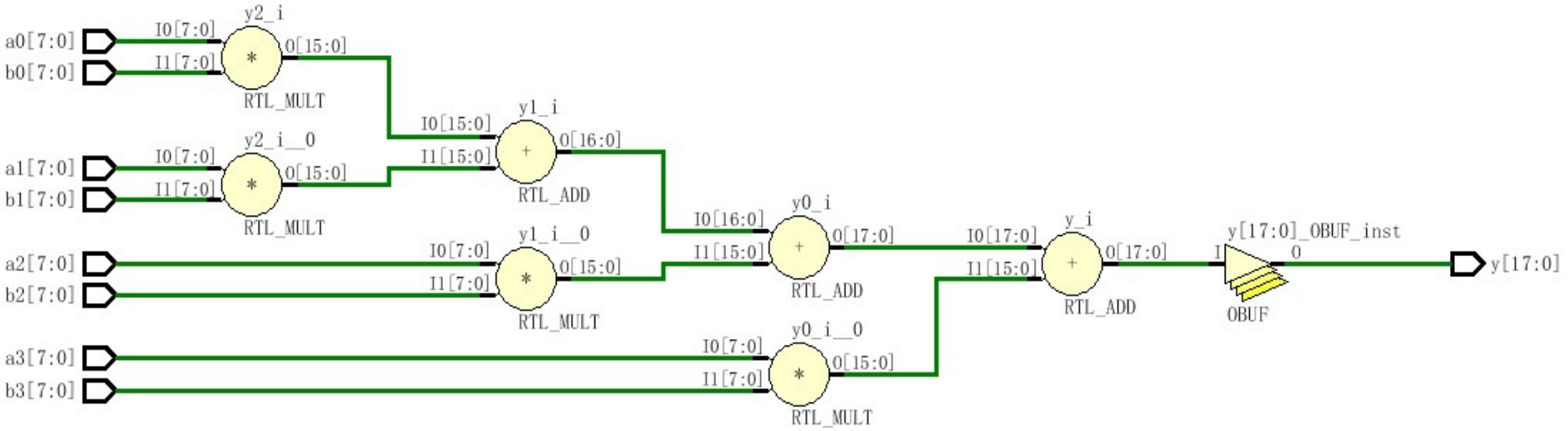


对应的并行乘法器结构

并行和流水线
--并行设计

下图给出了实现该功能的并行结构。

- 通过使用多个乘法器，使得四个乘法运算可以同时进行。
- 但是这种速度的提高是以面积为代价的。

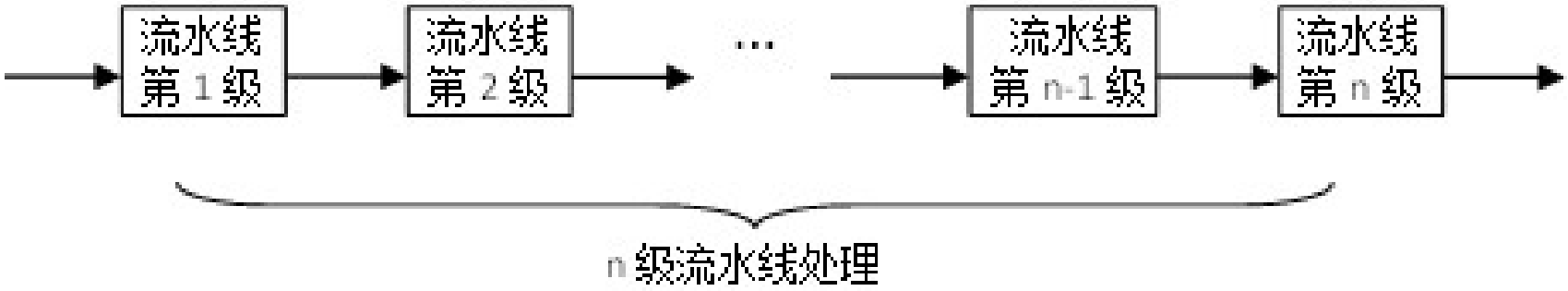


并行乘法的实现结构

并行和流水线
--流水线设计

流水线设计从宏观上来看平均每一个事件的处理时间为一个单位时间。

- 流水线的设计要求事件所分成的这n个步骤的处理时间是相同的，以保证流水线不会因为处理时间的不同而发生阻塞。



并行和流水线
--流水线设计

- 如果在串行设计中系统处理效能正比于系统运行的频率，那么对于流水线设计，在不提高系统运行频率的情况下，n级流水线的处理效能可以用下面公式来描述：

处理效能=系统运行频率×流水线级数

并行和流水线
--流水线设计

- 采用流水线设计的好处实在不提高系统运行频率的情况下，能够获得更好的处理效率。
- 受制造工艺的影响，提高系统频率将会增加后端制造的难度。
- 同时由于需要使用更先进的制造工艺，以及产品频率的下降，将会带来产品成本的提高。

并行和流水线
--流水线设计

由此可见，在不提高系统运行频率的情况下，提高流水线的级数将成倍地提高系统处理的效能。但是流水线的设计也是有一定的限制的：

- 只有对那些能分成n个步骤完成，并且对每个步骤都需要固定相同处理时间的操作来说才能采用流水线设计；
- 受硬件资源的限制，流水线的级数是有限制的；
- 对于存在处理分支预测流水线的设计（广泛应用于微处理器的设计中），流水线的处理效能还要取决于分支预测算法的设计。

并行和流水线
--流水线设计

流水线能动态地提升器件性能

- 它的基本思想是对经过多级逻辑的长数据通路进行重新构造，把原来必须在一个时钟周期内完成的操作分成在多个周期内完成。
- 这种方法允许更高的工作频率，因而提高了数据吞吐量。因为FPGA的寄存器资源非常丰富，所以对FPGA而言，流水线是一种先进的而又不耗费过多器件资源的结构。
- 采用流水线后，数据通道将会变成多时钟周期，所以要特别考虑设计的其余部分，解决增加通路带来的延迟。

并行和流水线
--流水线设计

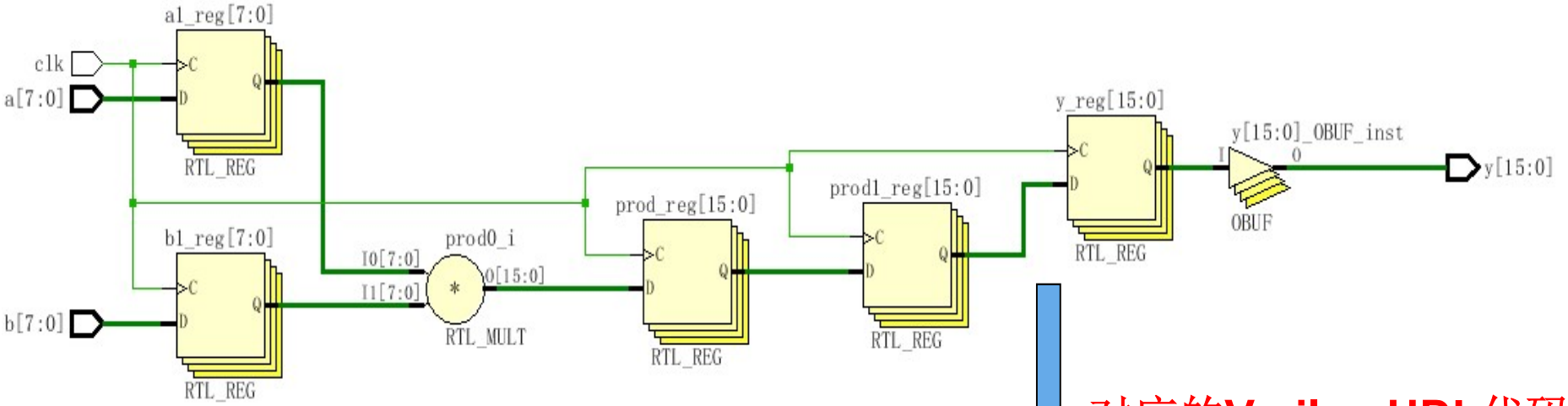
流水线基本结构是将适当划分的N个操作步骤串连起来。

- 流水线操作的最大特点是数据流在各个步骤的处理，从时间上看是连续的；
- 其操作的关键在于时序设计的合理安排、前后级接口间数据的匹配。如果前级操作的时间等于后级操作的时间，直接输入即可；
- 如果前级操作时间小于后级操作时间，则需要对前级数据进行缓存，才能输入到后级；如果前级操作时间大于后者，则需要串并转换等方法进行数据分流，然后再输入到下一级。

并行和流水线
--流水线设计

流水线乘法器结构

- 从图中可以看出，该流水线乘法器，在每个时钟节拍下，均可以得到一个乘法结果的输出，乘法器的效率大大增加。



流水线乘法器的结构图

对应的Verilog HDL代码

并行和流水线
--流水线设计

```
module top(
    input [7:0] a,
    input [7:0] b,
    input clk,
    output reg [15:0] y
    );
 reg [7:0] a1,b1;
 reg [15:0] prod,prod1;
```

```
always @(posedge clk)
begin
  a1<=a;
  b1<=b;
  prod<=a1*b1;
  prod1<=prod;
  y<=prod1;
end
endmodule
```