我心狂野
——文字和影像比我们的活得更长久

< 2011年5月 >

日	一	二	三	四	五	六
24	25	26	27	28	29	30
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4

搜索

谷歌搜索

随笔分类(60)

编程语言(7)

电路设计(10)

软件相关(11)

数字逻辑(24)

算法相关(5)

随便写写(3)

阅读排行榜

1. DDR工作原理(28118)

2. 第三层交换机和路由器的区别(23279)

3. 乘法器的Verilog HDL实现(19261)

4. 千兆以太网芯片88E1111 RGMII模式的驱动(15997)

5. 位图文件（BMP）格式以及Linux下C程序实现(7863)

6. 以太网PHY和MAC(7835)

7. 关于建立时间和保持时间(7499)

8. Pacman 命令详解(6328)

9. 交换机和路由器各自的实现原理(5703)

10. 流水线技术原理和Verilog HDL实现(4928)

11. [转]TimeQuest约束外设之诡异的

流水线技术原理和Verilog HDL实现

所谓流水线处理，如同生产装配线一样，将操作执行工作量分成若干个时间上均衡的操作段，从流水线的起点连续地输入，流水线的各操作段以重叠方式执行。这使得操作执行速度只与流水线输入的速度有关，而与处理所需的时间无关。这样，在理想的流水操作状态下，其运行效率很高。

如果某个设计的处理流程分为若干步骤，而且整个数据处理是单流向的，即没有反馈或者迭代运算，前一个步骤的输出是下一个步骤的输入，则可以采用流水线设计方法来提高系统的工作频率。

下面用8位全加器作为实例，分别列举了非流水线方法、2级流水线方法和4级流水线方法。

（1）非流水线实现方式

```
module adder_8bits(din_1, clk, cin, dout, din_2, cout);
    input [7:0] din_1;
    input clk;
    input cin;
    output [7:0] dout;
    input [7:0] din_2;
    output cout;

        reg [7:0] dout;
        reg       cout;

        always @(posedge clk) begin
               {cout,dout} <= din_1 + din_2 + cin;
        end

endmodule
```

（2）2级流水线实现方式：

```
module adder_4bits_2steps(cin_a, cin_b, cin, clk, cout, sum);
    input [7:0] cin_a;
    input [7:0] cin_b;
    input cin;
    input clk;
    output cout;
    output [7:0] sum;

        reg cout;
        reg cout_temp;
        reg [7:0] sum;
        reg [3:0] sum_temp;

        always @(posedge clk) begin
               {cout_temp,sum_temp} = cin_a[3:0] + cin_b[3:0] + cin;
        end

        always @(posedge clk) begin
               {cout,sum} = {{1'b0,cin_a[7:4]} + {1'b0,cin_b[7:4]} + cout_temp,
sum_temp};
        end
endmodule
```

注意：这里在always块内只能用阻塞赋值方式，否则会出现逻辑上的错误！

（3）4级流水线实现方式：

```
module adder_8bits_4steps(cin_a, cin_b, c_in, clk, c_out, sum_out);
    input [7:0] cin_a;
    input [7:0] cin_b;
    input c_in;
```

Create Generated Clocks用法(4667)

12. 路由器换大Flash(3465)

13. [转]Verilog 中 define parameter localparam的区别(3296)

14. [转]换位思考多周期约束(3116)

15. 关于C/C++中的点操作符和箭头操作符(3025)

```
    input clk;
    output c_out;
    output [7:0] sum_out;

        reg c_out;
        reg c_out_t1, c_out_t2, c_out_t3;

        reg [7:0] sum_out;
        reg [1:0] sum_out_t1;
        reg [3:0] sum_out_t2;
        reg [5:0] sum_out_t3;

        always @(posedge clk) begin
            {c_out_t1, sum_out_t1} = {1'b0, cin_a[1:0]} + {1'b0, cin_b[1:0]} + c_in;
        end

        always @(posedge clk) begin
            {c_out_t2, sum_out_t2} = {{1'b0, cin_a[3:2]} + {1'b0, cin_b[3:2]} +
c_out_t1, sum_out_t1};
        end

        always @(posedge clk) begin
            {c_out_t3, sum_out_t3} = {{1'b0, cin_a[5:4]} + {1'b0, cin_b[5:4]} +
c_out_t2, sum_out_t2};
        end

        always @(posedge clk) begin
            {c_out, sum_out} = {{1'b0, cin_a[7:6]} + {1'b0, cin_b[7:6]} + c_out_t3,
sum_out_t3};
        end


endmodule
```

总结：利用流水线的设计方法，可大大提高系统的工作速度。这种方法可广泛运用于各种设计，特别是大型的、对速度要求较高的系统设计。虽然采用流水线会增大资源的使用，但是它可降低寄存器间的传播延时，保证系统维持高的系统时钟速度。在实际应用中，考虑到资源的使用和速度的要求，可以根据实际情况来选择流水线的级数以满足设计需要。

这是一种典型的以面积换速度的设计方法。这里的"面积"主要是指设计所占用的FPGA逻辑资源数目，即利用所消耗的触发器（FF）和查找表（LUT）来衡量。"速度"是指在芯片上稳定运行时所能达到的最高频率。面积和速度这两个指标始终贯穿着FPGA的设计，是设计质量评价的最终标准。

分类：数字逻辑

我心狂野
关注 - 3
粉丝 - 61
+加关注

« 上一篇：乘法器的Verilog HDL实现
» 下一篇：深入浅出之正则表达式

posted @ 2011-05-23 16:22 我心狂野 阅读(4928) 评论(3) 编辑 收藏

评论列表

#1楼 2012-02-15 17:39 空度

试用

支持(0) 反对(0)

#2楼 2012-09-04 17:55 猪一头

牛逼！受用了！在我这个初学者 看来多么神圣的一个问题，竟然让你用这么简单一个例子清楚的呈现出来，佩服，看来对于复杂晦涩难懂之体用实际例子来解释最好不过咯。

支持(0) 反对(0)

#3楼 2015-09-04 11:10 zl138764

请问下，“这里在always块内只能用阻塞赋值方式，否则会出现逻辑上的错误”，这是为什么呢

支持(0) 反对(0)

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。