架构实验室

热身

Chenlu Miao 09/2022

概述

- 实验课
- Vivado（工具）
- Verilog热身
- Verilog事件驱动仿真

实验课程

实验课程

- **实验01**
  - 实现一个具有转发和预测不跳转功能的五级流水线CPU，以支持RV32I指令集。
- 实验02
  - 在实验01的流水线CPU上实现中断和异常处理。
- 实验03
  - 通过仿真实现一个两路组相联缓存。
- 实验04
  - 将实验03中的两路组相联缓存集成到实验02的流水线CPU中。
- 实验05
  - 扩展实验02的流水线CPU，以支持多周期操作、乱序执行和冒险检测
- 实验06
  - 扩展实验05的流水线CPU，以支持计分板或托马苏洛算法等动态调度。

维维杜（Vivado一般音译为"维维杜"，在FPGA开发领域是赛灵思公司的设计套件名称）

维维尔多（注：Vivado是赛灵思公司的一款软件，一般不做翻译，保留英文）

- 新的维维杜用户？→ 点击此处
- 维维杜2020.2及更高版本
- 我们的开发板 → xc7k325tffg676 - 2L

Verilog热身

Verilog热身

- 变量
- 向量
- 赋值
- 始终块
- if - else语句
- case语句

- ROM & RAM

HDLBits中的Verilog实践：https://hdlbits.01xz.net/wiki/Main Page

Verilog - 变量

•线网（wire）：物理上是一根导线。不能“保存”值。必须持续赋值。 - 寄存器（reg）：不一定是寄存器。可以“保存”值。可以有条件地赋值。

Verilog：线网（wire）与寄存器（reg） →

https://inst.eecs.berkeley.edu/~cs150/Documents/Nets.pdf

Verilog – 向量

•拼接

输入 [15:0] in; 输出
[23:0] out; // 左侧

assign {out[7:0], out[15:8]} = in; // 右侧
assign out[15:0] = {in[7:0], in[15:8]};
assign out = {in[7:0], in[15:8]};

Verilog - 向量

•拼接
•反转

```
模块顶层模块(
    输入 [7:0] in,
    输出 [7:0] out
);
    // 赋值 out[7:0] = in[0:7]; 错误
    赋值 out = {in[0], in[1], in[2],
in[3], in[4], in[5], in[6], in[7]};
模块结束
```

Verilog – 向量

•拼接
•反转
•复制

{5{1'b1}} // 5'b11111（或 5'd31 或 5'h1f）
$\{2\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}\}$ // 与 {a,b,c,a,b,c} 相同
{3'd5, {2{3'd6}}} // 9'b101_110_110

{num{向量}}

这将向量复制 num 次。num 必须是一个常量

Verilog - 向量

•拼接
•反转
•复制
•归约

& a[3:0] // 与运算：a[3]&a[2]&a[1]&a[0]。等同于 (a[3:0] == 4'hf)
$|\ \mathrm{b}\,[3:0]$ // 或：b[3]/b[2]/b[1]/b[0]。等同于 (b[3:0] != 4'h0)
^ c[2:0] // 异或：c[2]^c[1]^c[0]

Verilog - 赋值

•连续赋值 (assign x = y;)
•过程阻塞赋值 $(\mathrm{x} = \mathrm{y};)$
•过程非阻塞赋值 $(\mathrm{x} <= \mathrm{y};)$

Verilog - 赋值

•连续赋值

示例：

线网 x;
赋值 x = y;

要求：

•只能在不在过程块（“always块”）内时使用
•赋值语句的左侧必须是网络类型（例如，wire）

Verilog – 赋值

•过程阻塞赋值

示例：

寄存器 x;
始终 $a\,(*)x = y\,;$

要求：

•只能在过程中使用。
•过程赋值（在always块中）的左侧必须是变量类型（例如，reg）

Verilog - 赋值

•过程性非阻塞赋值

示例：

寄存器 x;
始终在时钟信号的上升沿触发，将 y 的值赋给 x;

要求：

- 只能在程序块内使用。
- 过程赋值语句（在 always 块中）的左侧必须是变量类型（例如，寄存器）

Verilog - 始终块

•组合逻辑：always a(*)

•时钟触发：always @(posedge clk)

Verilog - 始终块

•组合逻辑：

始终使用 a(*)

•等同于赋值语句

•过程块拥有更丰富的

语句集（例如，if - then、case）...

```
reg [3:0] out;
始终 @(*) 开始
    如果(sel1)
        输出 = 输入1[3:0];
    否则如果(选择信号2)
        输出 = 输入2[3:0];
    否则如果(sel3)
        out = in3[3:0];
    否则
        输出 = 4位二进制0;
结束
```

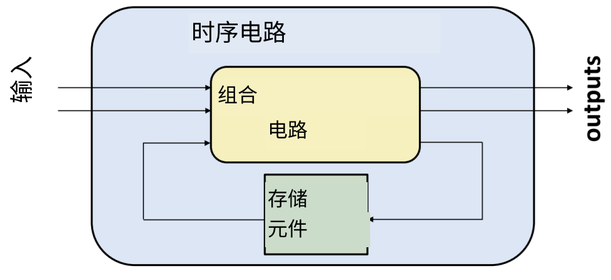
北

```
4位线网输出信号 out;
赋值 out = （{4{sel1}} 与 in1[3:0] 按位与运算）
        或 （{4{sel2}} 与 in2[3:0] 按位与运算）
        或 （{4{sel3}} 与 in3[3:0] 按位与运算）;
```

Verilog – 始终块

•组合逻辑：始终使用 @(*)

•时钟逻辑：始终使用 @(上升沿时钟信号)




```
3位寄存器变量 q;

始终 @(时钟信号上升沿)
开始
    如果(复位) 输出q 赋值为 0;
    否则 开始
        q <= q + 1;
    结束
结束
```

Verilog - 始终块

•组合逻辑：always $\omega(*)$

•时钟控制：always @(posedge clk)

1 始终使用

组合逻辑 always $\omega(*)$ 中使用阻塞赋值

在时序逻辑中，始终在时钟上升沿使用非阻塞赋值。

Verilog - if - else

首选

```
如果（出售）
    out = in1[3:0];
否则如果(sel2)
    输出 = in2[3:0];
否则如果(sel3)
    输出 = in3[3:0];
否则
    out = 4'b0;
```

```
赋值 out = sel1 ? in1[3:0] :
                sel2 ? in2[3:0] :
                sel3 ?  in3[3:0] :
                        4位二进制0;
```

```
赋值 out = ({4{sel1}} 与 in1[3:0])
        或 ({4{sel2}} 与 in2[3:0])
        或 ({4{sel3}} 与 in3[3:0]);
```

Verilog - 条件语句

```
always (*) begin // 这是一个组合电路
    case (in)
        1'b1: begin
                out = 1'b1; // 如果有多个语句，使用begin-end
            结束
        1'b0: 输出 = 1'b0;
        默认情况: 输出 = 1'bx;
    结束条件判断
结束
```

Verilog – 只读存储器与随机存取存储器

```
模块 ROM (
    输入 [ 6:0] 地址
    输出 [31:0] 读取数据
);
    寄存器 [31:0] 内存[127:0];

    初始开始
        $以十六进制读取内存文件("rom.hex", 内存);
    结束

    将 rd_data 赋值为 mem[addr];
模块结束
```

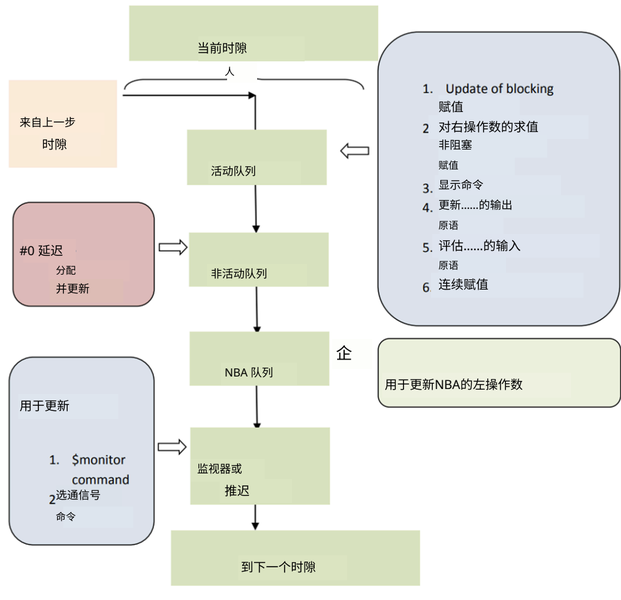
1 将 rom.hex 放入 xxx\sim_1\behav\xsim 目录
或使用绝对路径

Verilog – Event-Driven Simulation

Verilog - 分层事件队列

•活动队列：大多数Verilog事件安排在活动事件队列。这些事件可以按任意顺序调度，并按任意顺序进行评估或更新。活动队列用于更新阻塞赋值、连续赋值、评估非阻塞赋值的右侧（非阻塞赋值的左侧不在活动队列中更新），$ 显示命令并更新原语。

•非活动队列：#Odelay赋值在非活动队列中更新。在Verilog中使用 $\#0$ 延迟不是好的做法，它会不必要地使事件调度和排序复杂化。大多数情况下，设计人员使用 $\#0$ 延迟赋值来欺骗模拟器，以避免竞争条件。



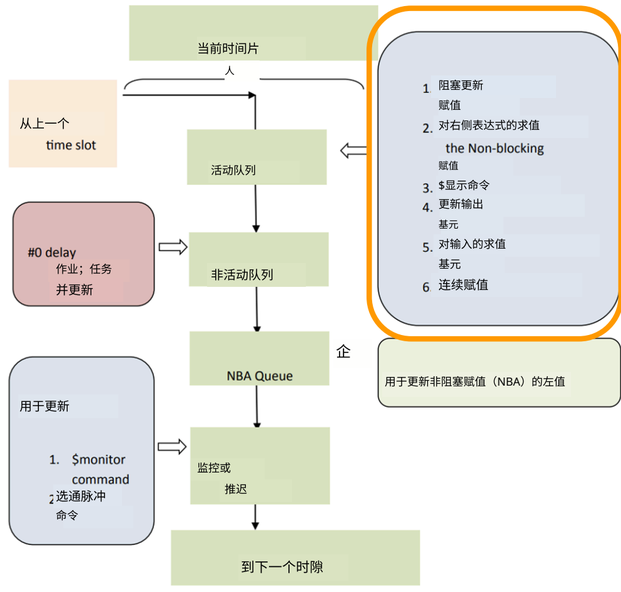
Verilog - 分层事件队列

```
当（存在事件）时 {
    如果（没有活跃事件）{
        如果（存在非活跃事件）{
            激活所有非活跃事件;
        } 否则如果（存在非阻塞赋值更新事件）{
            激活所有非阻塞赋值更新事件;
        } 否则，如果存在监控事件 {
        激活所有监控事件;
        } 否则 {
            将 T 推进到下一个事件时间;
            激活时间 T 的所有非活动事件;
        }
    }
    E = 任何活动事件;
    如果 (E 是一个更新事件) {
        更新修改后的对象;
        将敏感进程的评估事件添加到事件队列;
    } 否则 { /* 应为评估事件 */
        评估该进程;
        将更新事件添加到事件队列;
    }
}
```

Verilog - 分层事件队列

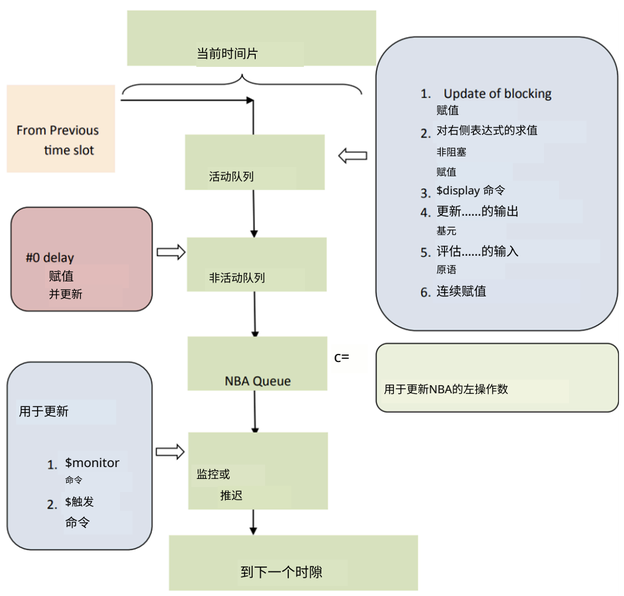
•活动队列：大多数Verilog事件被调度在活动事件队列。这些事件可以按任意顺序调度，并按任意顺序进行评估或更新。活动队列用于更新阻塞赋值、连续赋值、评估非阻塞赋值的右侧（非阻塞赋值的左侧不在活动队列中更新），$ 显示命令并更新原语。

•非活动队列：#0延迟赋值在非活动队列中更新。在Verilog中使用 $\#0$ 延迟不是好的做法，它会不必要地使事件调度和排序复杂化。大多数情况下，设计人员使用 $\#0$ 延迟赋值来欺骗模拟器，以避免竞争条件。



Verilog - 分层事件队列

•NBA（非阻塞赋值更新）队列：非阻塞赋值的左值在该队列中更新。- 监控事件：用于评估和更新$monitor和$strobe命令。所有变量的更新都在当前仿真时间内进行。



Verilog - 分层事件队列

•活动队列：

始终 ①(*)

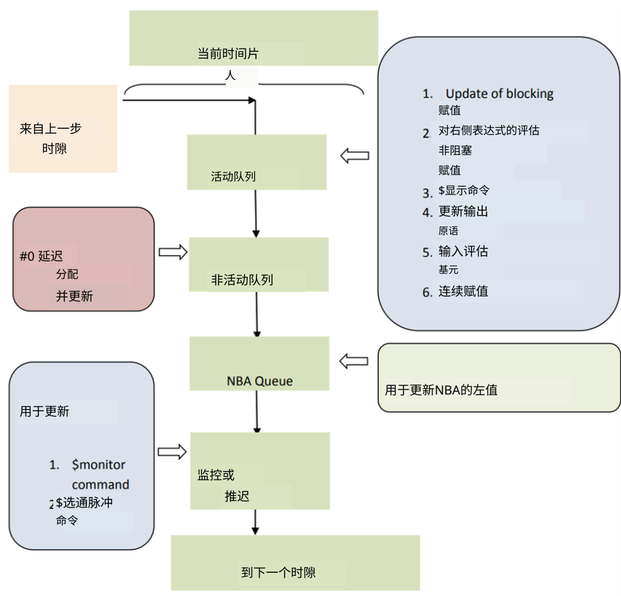
f = a & b;

always ①(*)

以任意顺序执行！

f = ?

f = a | b;



Verilog - 分层事件队列

•活动队列：

始终 ①(*)

开始

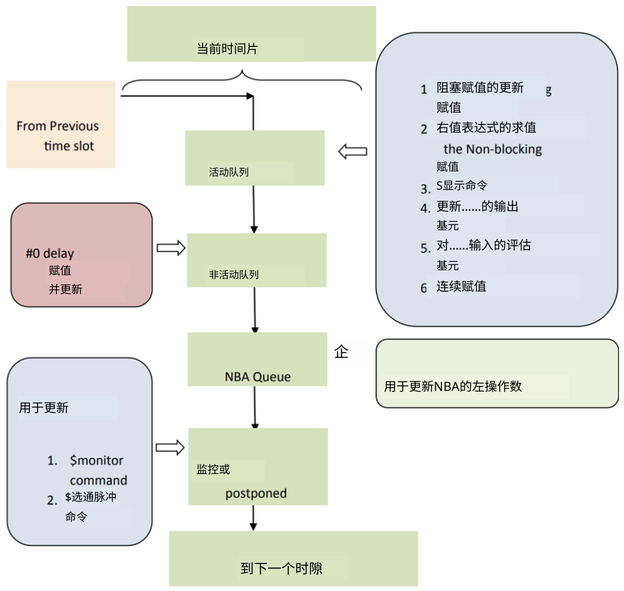
f = a 与 b;

f = a 或 b;

结束

按顺序执行！

1. $f = a \& b$
2. $f = a \mid b$

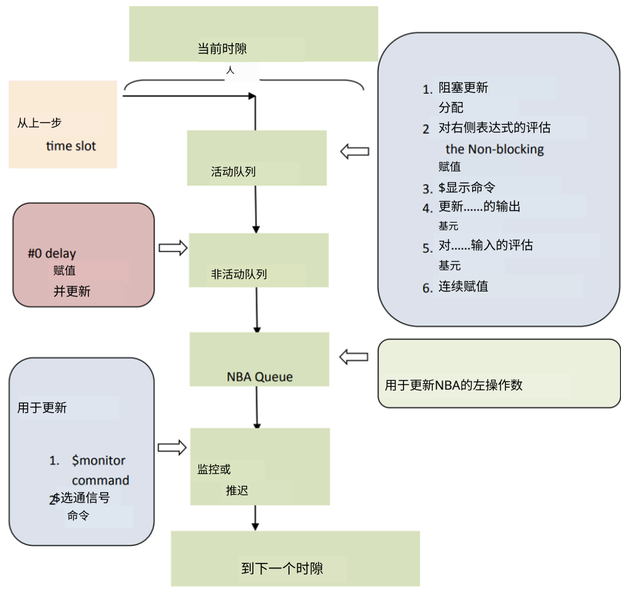


Verilog - 分层事件队列

始终 @(时钟信号上升沿)

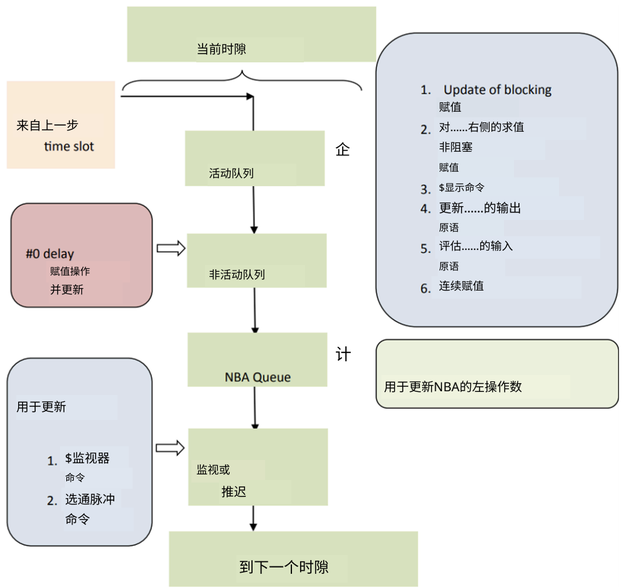
```
f <= a & b;
```

活动队列：右侧表达式求值



Verilog - 分层事件队列

始终 @(时钟信号上升沿) f <= a 与 b;非阻塞赋值队列：左值更新



Verilog - 分层事件队列

连线 A_in, B_in, C_in;
寄存器 A_out, B_out, C_out;

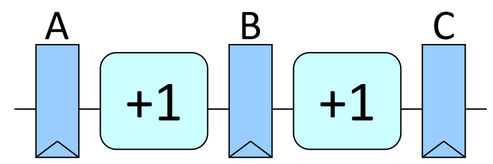
始终 @(时钟信号上升沿)
A输出 = A输入;

将 B输入 赋值为 A输出 + 1;

始终 @(时钟信号上升沿)
B输出 = B输入;

将 C输入 赋值为 B输出 + 1;

始终 @(时钟信号上升沿)
输出进位 = 输入进位;



Verilog - 分层事件队列

线网 A 输入, B 输入, C 输入;
寄存器 A 输出, B 输出, C 输出;

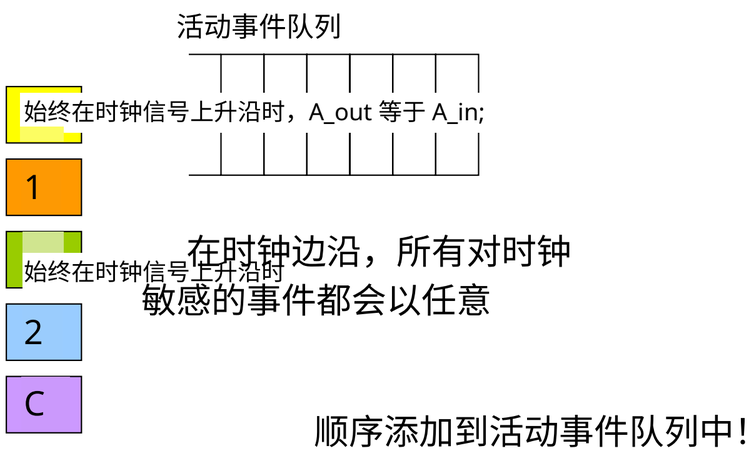
始终 @(时钟信号上升沿)
A输出 = A输入;

将 B输入 赋值为 A输出 + 1;

始终 @(时钟信号上升沿)
B输出 = B输入;

将 C输入 赋值为 B输出 + 1;

始终 @(时钟信号上升沿)
进位输出 = 进位输入;



在时钟边沿，所有对时钟敏感的事件都会以任意

顺序添加到活动事件队列中！

Verilog - 分层事件队列

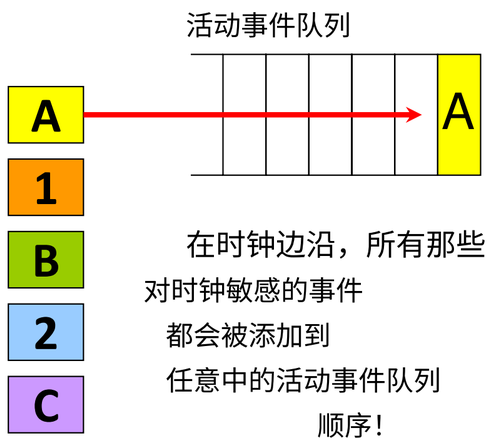
线网 A_in、B_in、C_in；寄存器 A_out、B_out、C_out;

将 B_in 赋值为 A_out 加 1;

B_out 等于 B_in;

将 C_in 赋值为 B_out 加 1;

始终在时钟信号上升沿时，C_out 等于 C_in;



在时钟边沿，所有那些对时钟敏感的事件都会被添加到任意中的活动事件队列

顺序！

Verilog - 分层事件队列

线网 A_in、B_in、C_in;
寄存器 A_out、B_out、C_out;

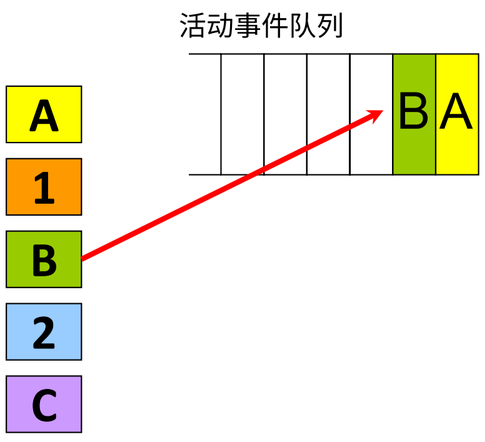
始终 @(时钟信号的上升沿)
A_out = A_in;

赋值 B_in = A_out + 1;

始终 @(时钟信号上升沿)
B输出 = B输入;

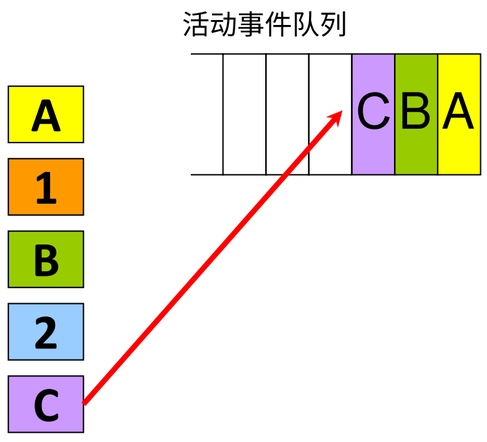
将 C输入 赋值为 B输出 + 1;

始终 @(时钟信号上升沿)
C输出 = C输入;



Verilog - 分层事件队列

线网 A_in、B_in、C_in；寄存器 A_out、B_out、C_out；始终在时钟上升沿触发时，A_out 赋值为 A_in；将 A_out 加 1 赋值给 B_in；始终在时钟上升沿触发时，B_out 赋值为 B_in；将 B_out 加 1 赋值给 C_in；始终在时钟上升沿触发时，C_out 赋值为 C_in；



Verilog - 分层事件队列

线网 A_in、B_in、C_in;
寄存器 A_out、B_out、C_out;

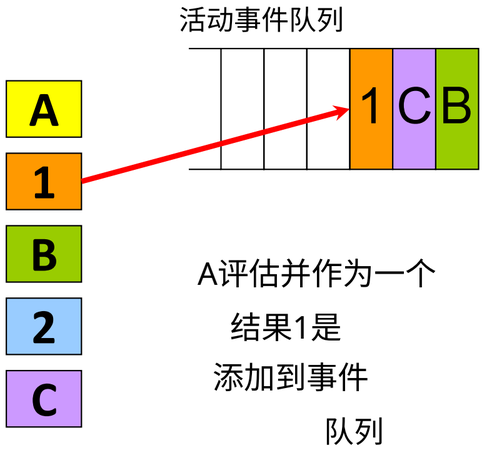
始终在时钟上升沿触发时
A_out 赋值为 A_in;

将 B_in 赋值为 A_out + 1;

始终在时钟上升沿触发
B_out 等于 B_in;

将 C_in 赋值为 B_out + 1;

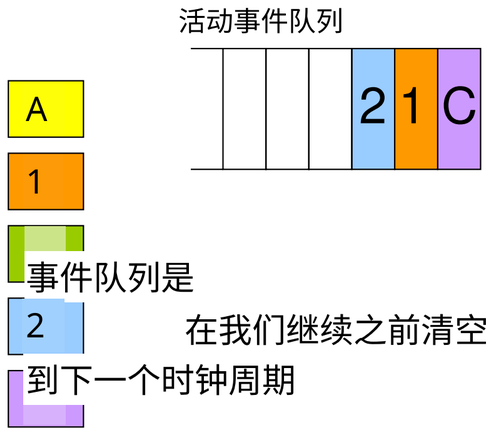
始终在时钟上升沿触发
C_out 等于 C_in;



A评估并作为一个结果1是添加到事件队列

Verilog - 分层事件队列

线A_in、B_in、C_in；寄存器A_out、B_out、C_out；始终@(时钟上升沿) A_out = A_in；赋值B_in = A_out + 1；始终@(时钟上升沿) B_out = B_in；赋值C_in = B_out + 1；始终@(时钟上升沿) C_out = C_in;



事件队列是在我们继续之前清空到下一个时钟周期

Verilog - 分层事件队列

线网 A_in、B_in、C_in;
寄存器 A_out、B_out、C_out;

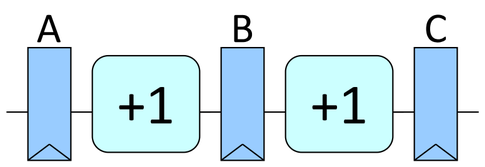
始终 @(时钟信号的上升沿)
A_out = A_in;

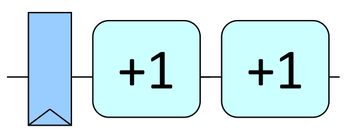
赋值 B_in = A_out + 1;

始终 @(时钟信号上升沿)
B输出 = B输入;

将 C输入 赋值为 B输出 + 1;

始终 @(时钟信号上升沿)
C输出 = C输入;





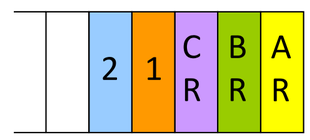
Verilog - 分层事件队列

线网 A输入、B输入、C输入；寄存器 A输出、B输出、C输出；始终 @(时钟信号上升沿)A输出 ≤ A输入；将 B输入 赋值为 A输出 + 1；始终 @(时钟信号上升沿)

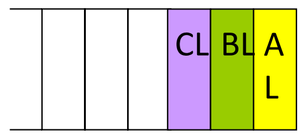
B_out ≤ B_in;（B_out ≤ B_in;可理解为"B输出 ≤ B输入"，因占位符含义不明，保留原样）

assign C_in = B_out + 1;always @(posedge clk)C_out ≤ C_in;（赋值C_in = B_out + 1；始终在时钟上升沿触发时，C_out ≤ C_in;）

活动事件队列



非阻塞队列



Verilog - 分层事件队列

线网型变量 A_in, B_in, C_in;（可理解为"连线 A输入、B输入、C输入"）

寄存器型变量 A_out, B_out, C_out;（可理解为"寄存器 A输出、B输出、C输出"）

始终在时钟上升沿触发时
A_out 小于等于 A_in;

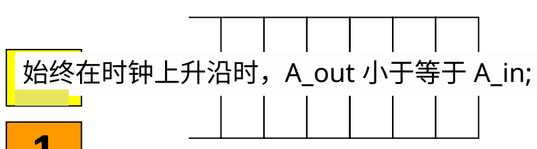
赋值 B_in 等于 A_out 加 1;

始终在时钟上升沿触发
B_out 小于等于 B_in;

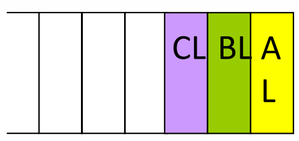
赋值 C_in 等于 B_out 加 1;

始终在时钟上升沿触发
C_out 小于等于 C_in;

活动事件队列



非阻塞队列



Verilog - 分层事件队列

线网 A_in、B_in、C_in；寄存器 A_out、B_out、C_out;

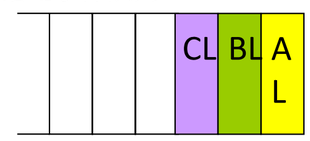
赋值 B_in = A_out + 1;

B_out ≤ B_in;

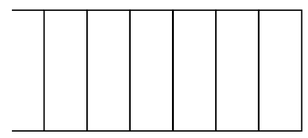
赋值 C_in = B_out + 1;

始终 @(时钟上升沿)C_out ≤ C_in;

活动事件队列



非阻塞队列



Verilog - 分层事件队列

线网 A_in、B_in、C_in;
寄存器 A_out、B_out、C_out;

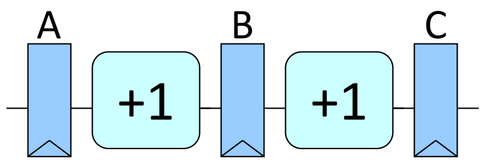
始终在时钟上升沿触发
A_out 小于等于 A_in;

将 B_in 赋值为 A_out 加 1;

始终在时钟上升沿触发
B输出 <= B输入;

赋值 C输入 = B输出 + 1;

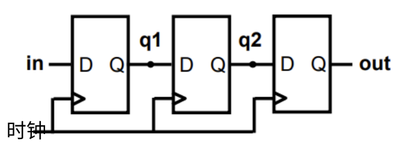
始终在(时钟信号上升沿)
C输出 <= C输入;



Verilog - 分层事件队列

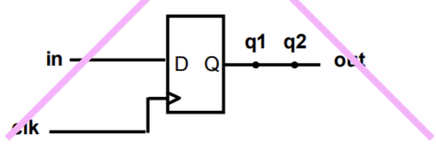
始终在(时钟信号上升沿)
开始
q1 <= 输入;
q2 <= q1;
输出 <= q2 ;
结束

"在每个时钟上升沿， $q1, q2$ ，并且 out 同时接收 in、q1 和 q2 的旧值。"



始终在(时钟信号上升沿)
开始
$r1 = \text{in}$;
$\text{q2} = \text{q1}$;
输出 = q2;
结束

"在每个时钟上升沿， $q1 =$ 输入。在那之后， $q2 = q\ i = in$; 在那之后，输出 $= q2 = r, i =$ 输入；最终输出 $=$ 输入。



•阻塞赋值无法反映
多级时序逻辑的内在行为

https://courses.csail.mit.edu/6.111/f2007/handouts/L06.pdf

参考文献

Verilog通用指南
https://safari.ethz.ch/digitaltechnik/spring2021/lib/exe/fetch.php?media=onur-digitaldesign_comparch-2021-lecture7-hdl-verilog-afterlecture.pdf
- https://safari.ethz.ch/digitaltechnik/spring2021/lib/exe/fetch.php?media=onur-digitaldesign_comparch-2021-lecture6-sequential-logic-updated-beforelecture.pdf
HDLBits
•从仿真器的角度理解Verilog语言
•RISC - V CPU处理器设计
• 使用Verilog编码和RTL综合的数字逻辑设计（第二版）
https://cseweb.ucsd.edu/classes/sp09/cse141L/Slides/02-Verilog2.pdf
- https://electronics.stackexchange.com/questions/443641/why-we-need-non-blocking-assignments-in-verilog

- https://courses.csail.mit.edu/6.111/f2007/handouts/L06.pdf
•IEEE Verilog硬件描述语言标准