Understanding Verilog Blocking and Non-blocking Assignments

International Cadence
User Group Conference

September 11, 1996

presented by

Stuart Sutherland

Sutherland HDL Consulting

About the Presenter

Stuart Sutherland has over 8 years of experience using Verilog with a variety of software tools. He holds a BS degree in Computer Science, with an emphasis on Electronic Engineering, and has worked as a design engineer in the defense industry, and as an Applications Engineer for Gateway Design Automation (the originator of Verilog) and Cadence Design Systems. Mr. Sutherland has been providing Verilog HDL consulting services since 1991. As a consultant, he has been actively involved in using the Verilog langiage with a many different of software tools for the design of ASICs and systems. He is a member of the IEEE 1364 standards committee and has been involved in the specification and testing of Verilog simulation products from several EDA vendors, including the Intergraph-VeriBest *VeriBest* simulator, the Mentor *QuickHDL* simulator, and the Frontline *CycleDrive* cycle based simulator. In addition to Verilog design consutlting, Mr. Sutherland provides expert on-site Verilog training on the Verilog HDL language and Programing Language Interface. Mr. Sutherland is the author and publisher of the popular *"Verilog IEEE 1364 Quick Reference Guide"* and the *"Verilog IEEE 1364 PLI Quick Reference Guide"*.

Please conact Mr. Sutherland with any questions about this material!

Sutherland HDL Consulting

Verilog Consulting and Training Services
22805 SW 92nd Place
Tualatin, OR 97062 USA

phone: (503) 692-0898
fax: (503) 692-1512
e-mail: stuart@sutherland.com

copyright notice

©1996

The material in this presentation is copyrighted by Sutherland HDL Consulting, Tualatin, Oregon. The presentation is printed with permission as part of the proceedings of the 1996 International Cadence User Group Conference. All rights are reserved. No material from this presentation may be duplicated or transmitted by any means or in any form without the express written permission of Sutherland HDL Consulting.

Sutherland HDL Consulting
22805 SW 92nd Place
Tualatin, OR 97062 USA

phone: (503) 692-0898
fax: (503) 692-1512
e-mail: info@sutherland.com

Objectives

- The primary objective is to understand:

What type of hardware is represented by blocking and non-blocking assignments?

- The material presented is a subset of an advanced Verilog HDL training course

Procedural Assignments

- Procedural assignment evaluation can be modeled as:
  - Blocking
  - Non-blocking
- Procedural assignment execution can be modeled as:
  - Sequential
  - Concurrent
- Procedural assignment timing controls can be modeled as:
  - Delayed evaluations
  - Delayed assignments

Blocking Procedural Assignments

- The **=** token represents a ***blocking*** procedural assignment
  - Evaluated and assigned in a single step
  - Execution flow within the procedure is blocked until the assignment is completed
  - Evaluations of concurrent statements in the same time step are blocked until the assignment is completed

These examples will *not* work — Why not?

```
//swap bytes in word
always @(posedge clk)
  begin
    word[15:8] = word[ 7:0];
    word[ 7:0] = word[15:8];
  end
```

```
//swap bytes in word
always @(posedge clk)
  fork
    word[15:8] = word[ 7:0];
    word[ 7:0] = word[15:8];
  join
```

Non-Blocking Procedural Assignments

- The **<=** token represents a ***non-blocking*** assignment
  - Evaluated and assigned in two steps:
  - ① The right-hand side is evaluated immediately
  - ② The assignment to the left-hand side is postponed until other evaluations in the current time step are completed
    - Execution flow within the procedure continues until a timing control is encountered (flow is not blocked)

These examples will work — Why?

```
//swap bytes in word
always @(posedge clk)
 begin
  word[15:8] <= word[ 7:0];
  word[ 7:0] <= word[15:8];
 end
```

```
//swap bytes in word
always @(posedge clk)
 fork
  word[15:8] <= word[ 7:0];
  word[ 7:0] <= word[15:8];
 join
```

Representing Simulation Time as Queues

Sutherland HDL

- Each Verilog simulation time step is divided into 4 queues

Time 0:

- Q1 — *(in any order)* :
  - Evaluate RHS of all non-blocking assignments
  - Evaluate RHS and change LHS of all blocking assignments
  - Evaluate RHS and change LHS of all continuous assignments
  - Evaluate inputs and change outputs of all primitives
  - Evaluate and print output from $display and $write
- Q2 — *(in any order)* :
  - Change LHS of all non-blocking assignments
- Q3 — *(in any order)* :
  - Evaluate and print output from $monitor and $strobe
  - Call PLI with reason_synchronize
- Q4 :
  - Call PLI with reason_rosynchronize

Time 1:

...

Note: this is an abstract view, not how simulation algorithms are implemented

Sequential Procedural Assignments

- The order of evaluation is **determinate**
  - A sequential blocking assignment evaluates and assigns before continuing on in the procedure

```
always @(posedge clk)
begin
    A = 1;              ← evaluate and assign A immediately
  #5 B = A + 1;         ← delay 5 time units, then evaluate and assign
end
```

  - A sequential non-blocking assignment evaluates, then continues on to the next timing control before assigning

```
always @(posedge clk)
begin
    A <= 1;             ← evaluate A immediately; assign at end of time step
  #5 B <= A + 1;        ← delay 5 time units, then evaluate; then assign at
end                       end of time step (clock + 5)
```

Concurrent Procedural Assignments

Sutherland HDL

The order of concurrent evaluation is **indeterminate**

- Concurrent blocking assignments have unpredictable results

```
always @(posedge clk)
  #5 A = A + 1;

always @(posedge clk)
  #5 B = A + 1;
```

Unpredictable Result:
(new value of B could be evaluated before or after A changes)

- Concurrent non-blocking assignments have predictable results

```
always @(posedge clk)
  #5 A <= A + 1;

always @(posedge clk)
  #5 B <= A + 1;
```

Predictable Result:
(new value of B will always be evaluated before A changes)

Delayed Evaluation Procedural Assignments

Sutherland HDL

- A timing control before an assignment statement will postpone when the next assignment is evaluated
  - Evaluation is delayed for the amount of time specified

```
begin
  #5 A = 1;        ← delay for 5, then evaluate and assign
  #5 A = A + 1;    ← delay 5 more, then evaluate and assign
     B = A + 1;    ← no delay; evaluate and assign
end
```

What values do A and B contain after 10 time units?

Delayed Assignment Procedural Assignments

- An *intra-assignment delay* places the timing control after the assignment token
  - The right-hand side is evaluated before the delay
  - The left-hand side is assigned after the delay

```
always @(A)
  B = #5 A;
```

A is evaluated at the time it changes, but is not assigned to B until after 5 time units

```
always @(negedge clk)
  Q <= @(posedge clk) D;
```

D is evaluated at the negative edge of CLK, Q is changed on the positive edge of CLK

```
always @(instructor_input)
  if (morning)
    understand = instructor_input;
  else if (afternoon)
    understand = #5 instructor_input;
  else if (lunch_time)
    understand = wait (!lunch_time) instructor_input;
```

Intra-Assignment Delays With Repeat Loops

Sutherland HDL

- An edge-sensitive intra-assignment timing control permits a special use of the repeat loop
  - The edge sensitive time control may be repeated several times before the delay is completed
  - Either the blocking or the non-blocking assignment may be used

```
always @(IN)
  OUT <= repeat (8) @(posedge clk) IN;
```

The value of IN is evaluated when it changes, but is not assigned to OUT until after 8 clock cycles

Choosing the Correct Procedural Assignment

- Which procedural assignment should be used to model a combinatorial logic buffer?

```
always @(in)
  #5 out = in;

always @(in)
  out = #5 in;

always @(in)
  #5 out <= in;

always @(in)
  out <= #5 in;
```

- Which procedural assignment should be used to model a sequential logic flip-flop?

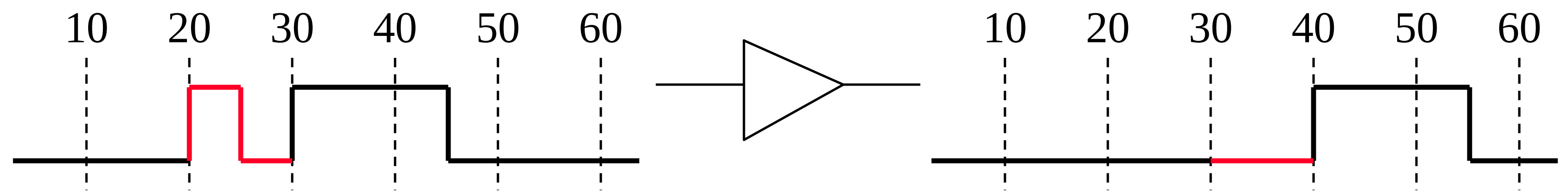
```
always @(posedge clk)
  #5 q = d;

always @(posedge clk)
  q = #5 d;

always @(posedge clk)
  #5 q <= d;

always @(posedge clk)
  q <= #5 d;
```

- The following pages will answer these questions

Transition Propagation Methods

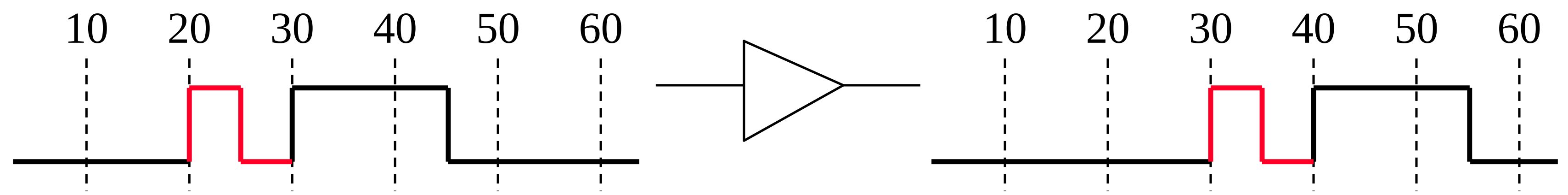
Sutherland HDL

- Hardware has two primary propagation delay methods:
  - **Inertial delay** models devices with finite switching speeds; input glitches do not propagate to the output



Buffer with a 10 nanosecond propagation delay

  - **Transport delay** models devices with near infinite switching speeds; input glitches propagate to the output

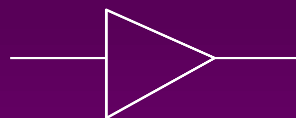


Buffer with a 10 nanosecond propagation delay

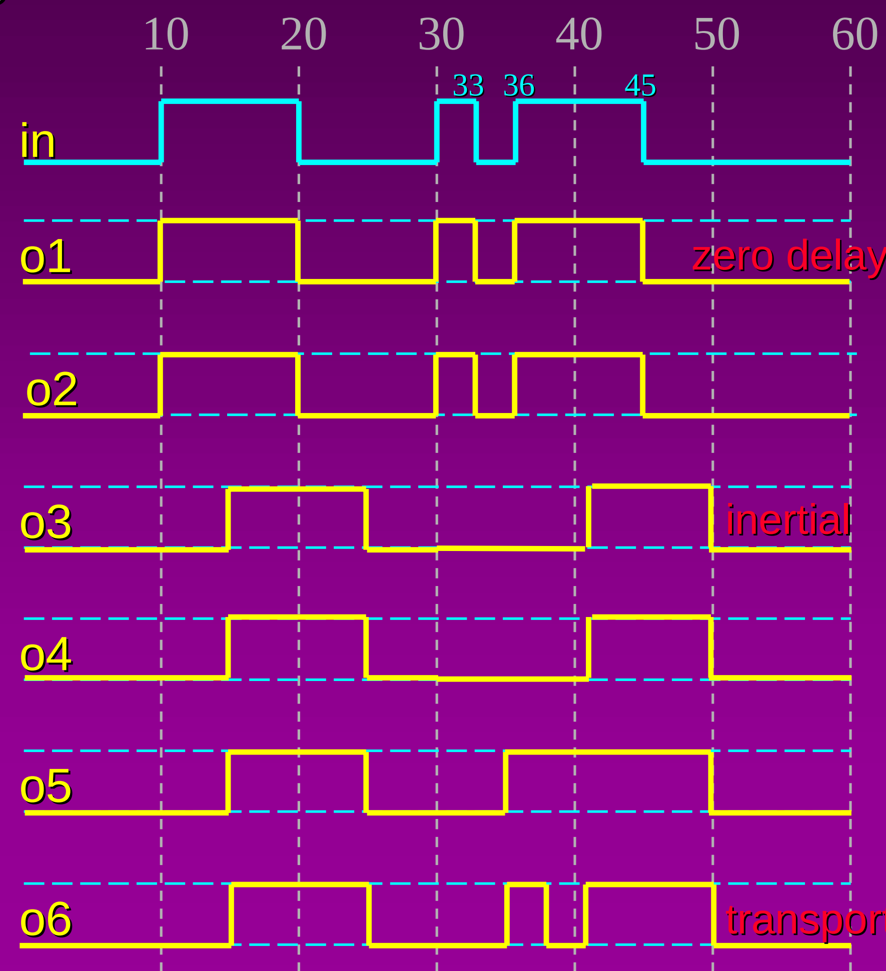
Combinational Logic Procedural Assignments

Sutherland HDL

- How will these procedural assignments behave?

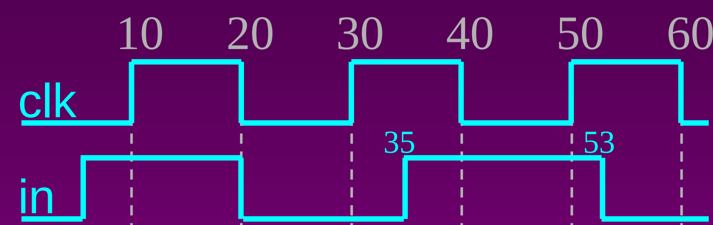


Blocking, No delay	`always @(in)` `o1 = in;`
Non-blocking, No delay	`always @(in)` `o2 <= in;`
Blocking, Delayed evaluation	`always @(in)` `#5 o3 = in;`
Non-blocking, Delayed evaluation	`always @(in)` `#5 o4 <= in;`
Blocking, Delayed assignment	`always @(in)` `o5 = #5 in;`
Non-blocking, Delayed assignment	`always @(in)` `o6 <= #5 in;`

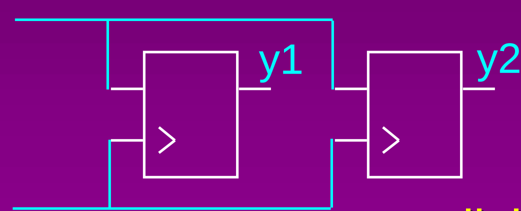


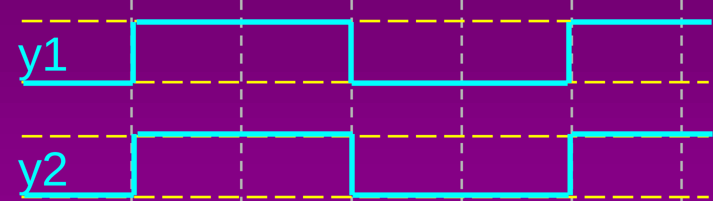
Sequential Logic Procedural Assignments

- How will these procedural assignments behave?
  - Sequential assignments
  - No delays



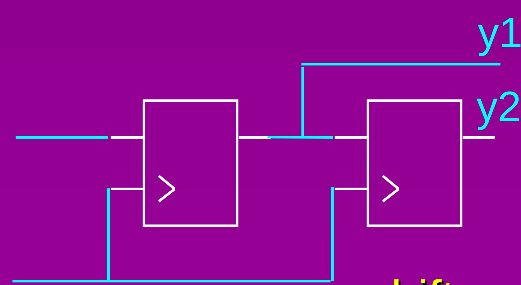

```
always @(posedge clk)
  begin
    y1 = in;
    y2 = y1;
  end
```

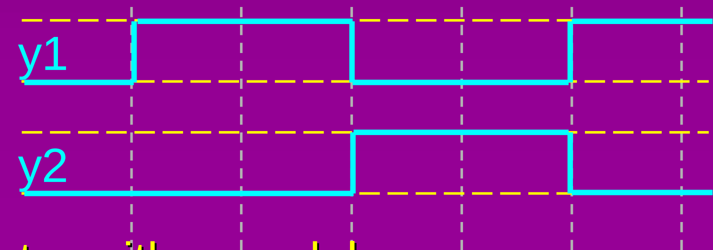




parallel flip-flops

```
always @(posedge clk)
  begin
    y1 <= in;
    y2 <= y1;
  end
```



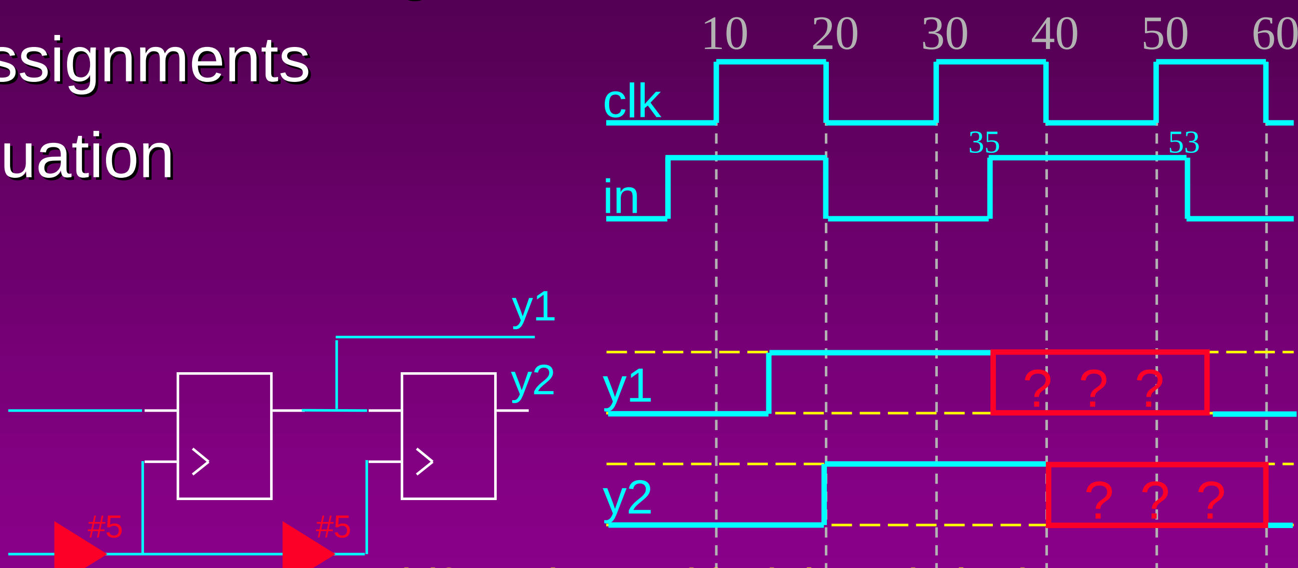


shift-register with zero delays

Sequential Logic Procedural Assignments

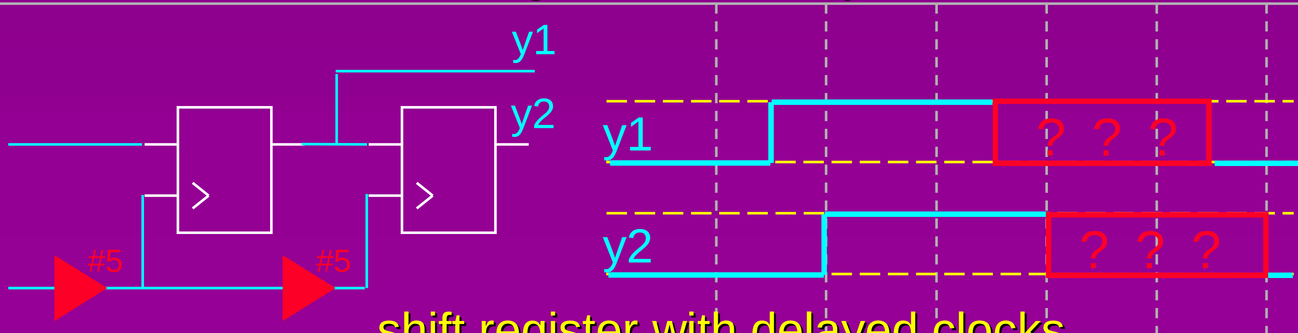
- How will these procedural assignments behave?
  - Sequential assignments
  - Delayed evaluation

```
always @(posedge clk)
  begin
    #5 y1 = in;
    #5 y2 = y1;
  end
```



shift register with delayed clocks

```
always @(posedge clk)
  begin
    #5 y1 <= in;
    #5 y2 <= y1;
  end
```

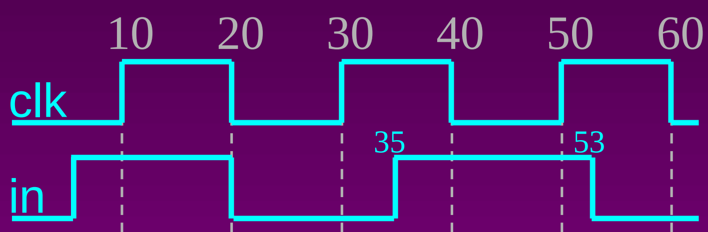


shift register with delayed clocks

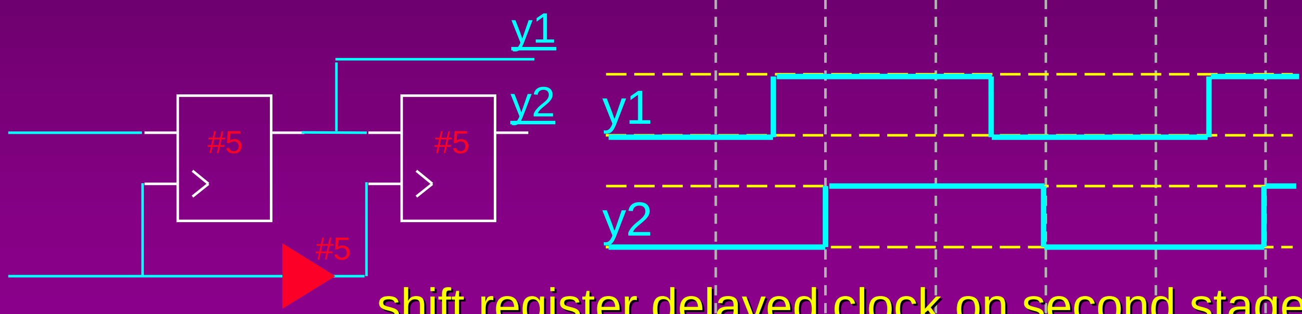
Sequential Logic Procedural Assignments

Sutherland HDL

- How will these procedural assignments behave?
  - Sequential assignments
  - Delayed assignment

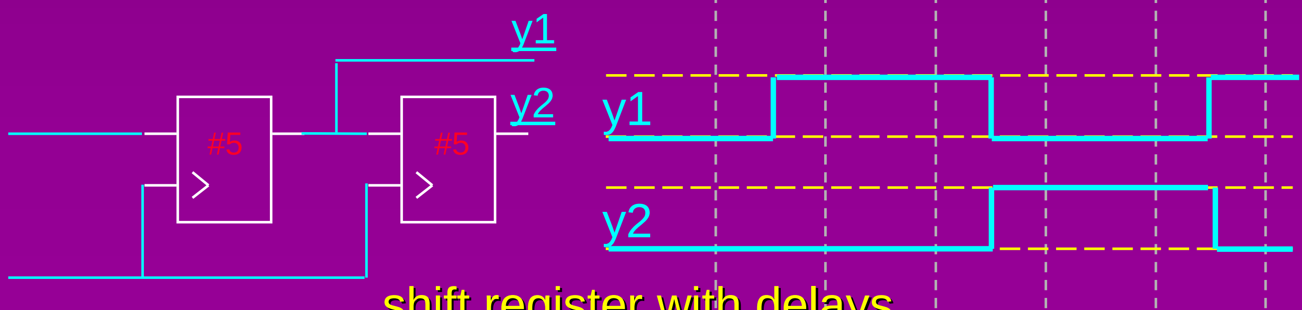



```
always @(posedge clk)
  begin
    y1 = #5 in;
    y2 = #5 y1;
  end
```



shift register delayed clock on second stage

```
always @(posedge clk)
  begin
    y1 <= #5 in;
    y2 <= #5 y1;
  end
```

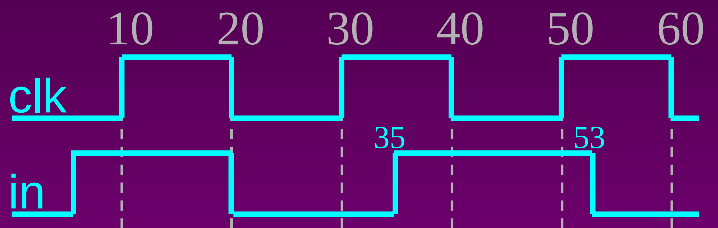


shift register with delays

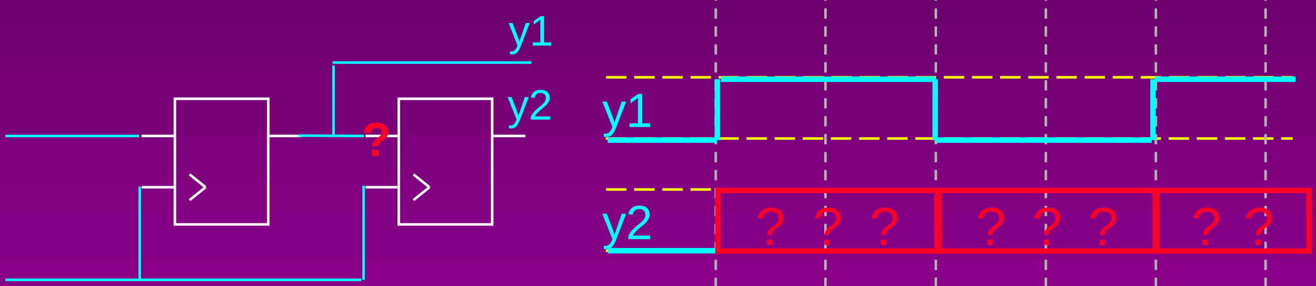
Sequential Logic Procedural Assignments

Sutherland HDL

- How will these procedural assignments behave?
  - Concurrent assignments
  - No delays

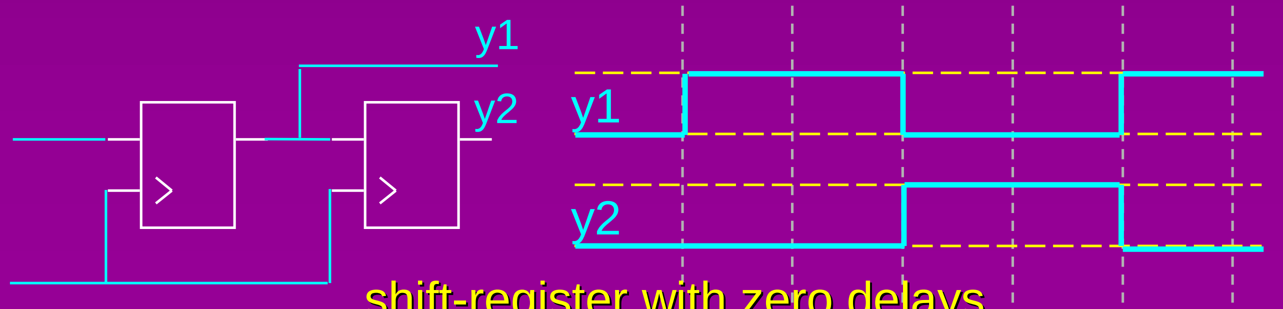



```
always @(posedge clk)
  y1 = in;

always @(posedge clk)
  y2 = y1;
```



shift register with race condition

```
always @(posedge clk)
  y1 <= in;

always @(posedge clk)
  y2 <= y1;
```

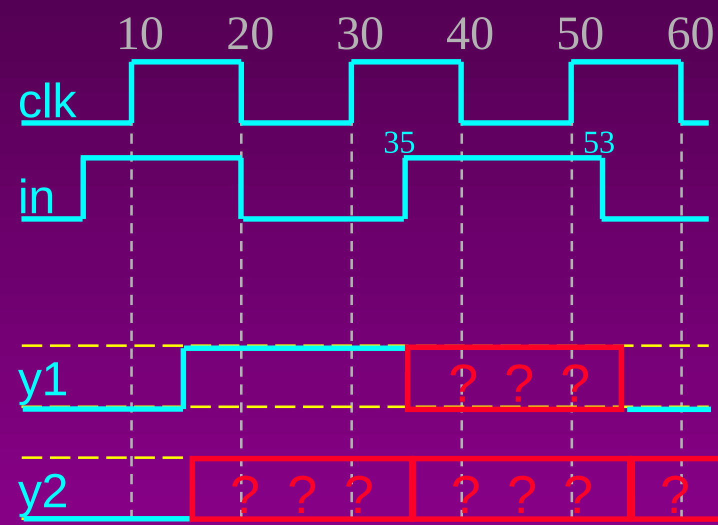


shift-register with zero delays

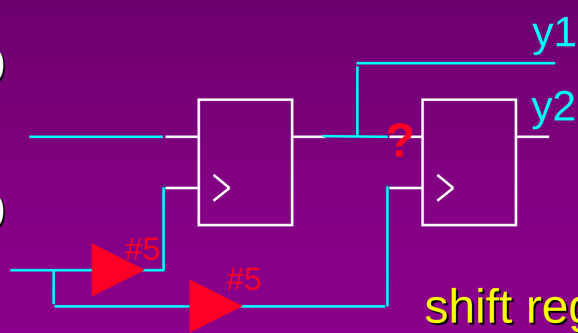
Sequential Logic Procedural Assignments

Sutherland HDL

- How will these procedural assignments behave?
  - Concurrent assignments
  - Delayed evaluation

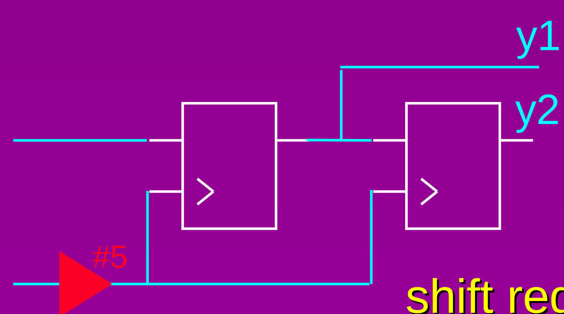


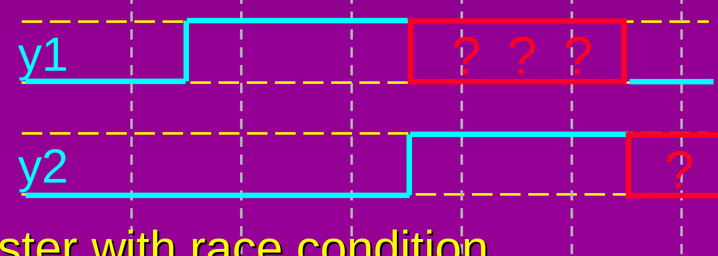

```
always @(posedge clk)
  #5 y1 = in;

always @(posedge clk)
  #5 y2 = y1;
```



shift register with race condition

```
always @(posedge clk)
  #5 y1 <= in;

always @(posedge clk)
  #5 y2 <= y1;
```



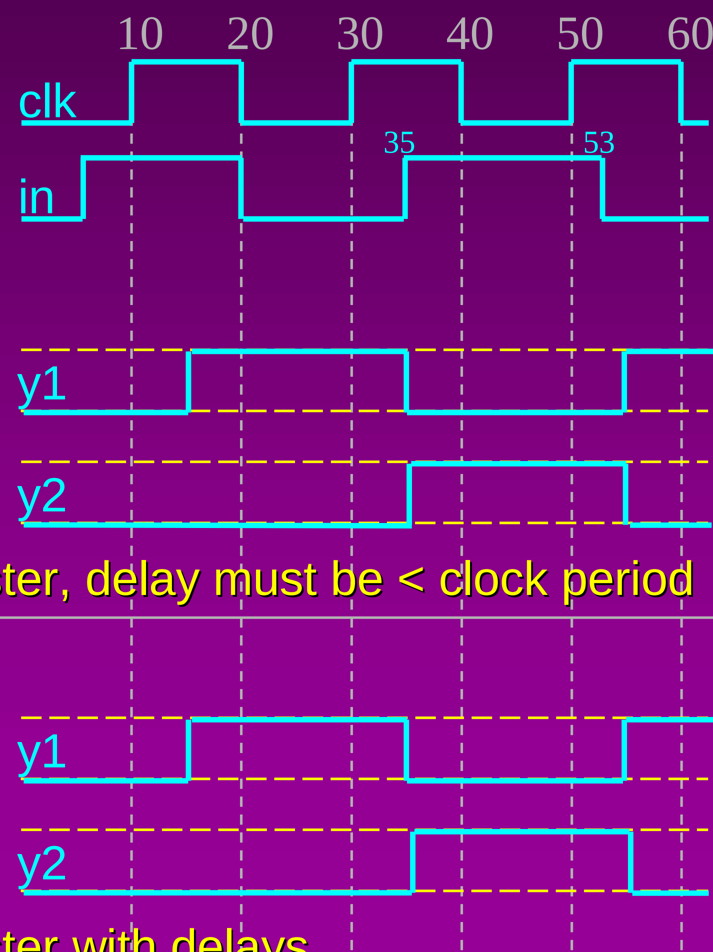


shift register with race condition

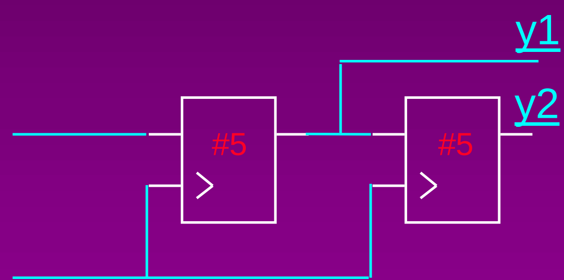
Sequential Logic Procedural Assignments

Sutherland HDL

- How will these procedural assignments behave?
  - Concurrent assignments
  - Delayed assignment

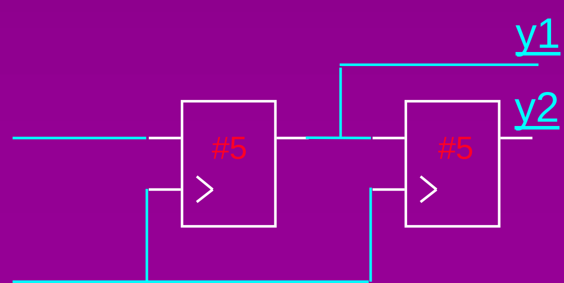



```
always @(posedge clk)
  y1 = #5 in;

always @(posedge clk)
  y2 = #5 y1;
```



shift register, delay must be < clock period

```
always @(posedge clk)
  y1 <= #5 in;

always @(posedge clk)
  y2 <= #5 y1;
```



shift register with delays

Rules of Thumb for Procedural Assignments

- Combinational Logic:
  - No delays: Use blocking assignments (a = b;)
  - Inertial delays: Use delayed evaluation blocking assignments (#5 a = b;)
  - Transport delays: Use delayed assignment non-blocking assignments (a <= #5 b;)
- Sequential Logic:
  - No delays: Use non-blocking assignments (q <= d;)
  - With delays: Use delayed assignment non-blocking assignments (q <= #5 d;)

An Exception to Non-blocking Assignments in Sequential Logic

- Do not use a non-blocking assignment if another statement in the procedure requires the new value in the same time step

```
begin
  #5 A <= 1;
  #5 A <= A + 1;
     B <= A + 1;
end
```

What values do A and B contain after 10 time units? **A is 2 B is 2**

```
always @(posedge clk)
  begin
    case (state)
     `STOP: next_state <= `GO;
     `GO:   next_state <= `STOP;
    endcase
    state <= next_state;
  end
```

Assume state and next_state are `STOP at the first clock, what is state:

- At the 2nd clock? **`STOP**
- At the 3rd clock? **`GO**
- At the 4th clock? **`GO**
- At the 5th clock? **`STOP**

Exercise 3: Procedural Assignments

Sutherland HDL

- Write a procedure for an adder (combinational logic) that assigns C the sum of A plus B with a 7ns propagation delay.

```
always @(A or B)
 #7 C = A + B;
```

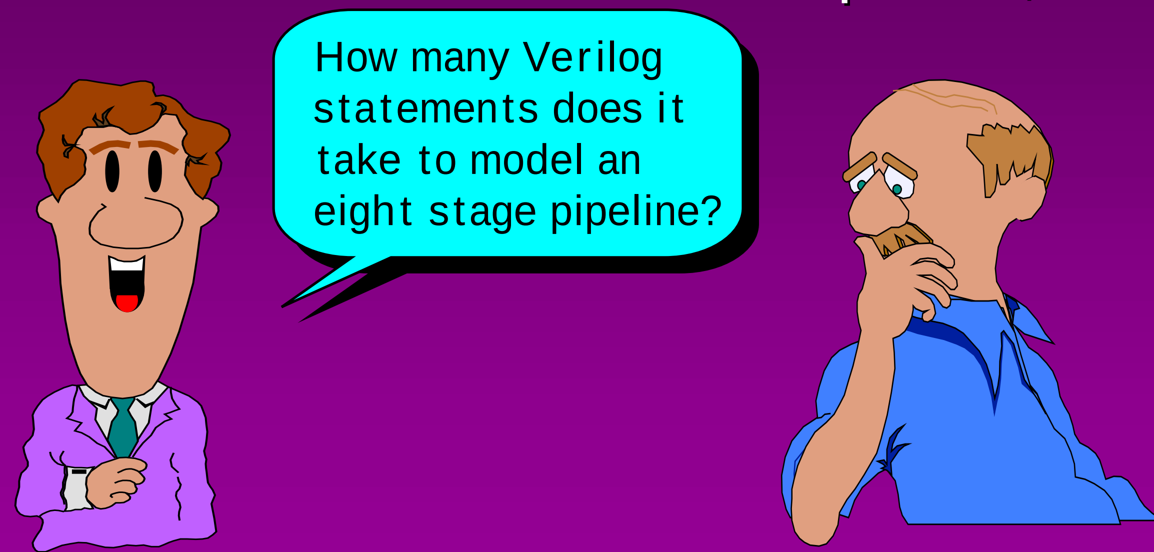
- Write the procedure(s) for a 4-bit wide shift register (positive edge triggered) of clock and has a 4ns propagation delay.

```
always @(posedge clk)
 begin
  y1  <= #4 in;
  y2  <= #4 y1;
  y3  <= #4 y2;
  out <= #4 y3;
 end
```

```
always @(posedge clk)
 y1  <= #4 in;
always @(posedge clk)
 y2  <= #4 y1;
always @(posedge clk)
 y3  <= #4 y2;
always @(posedge clk)
 out <= #4 y3;
```

Exercise 3 (continued): Procedural Assignments

- Write a Verilog procedure for a “black box” ALU pipeline that takes 8 clock cycles to execute an instruction. The pipeline triggers on the positive edge of clock. The “black box” is represented as call to a function named ALU with inputs A, B and OPCODE.




```
always @(posedge clk)
  alu_out <= repeat(7) @(posedge clk) ALU(A,B,OPCODE);
```