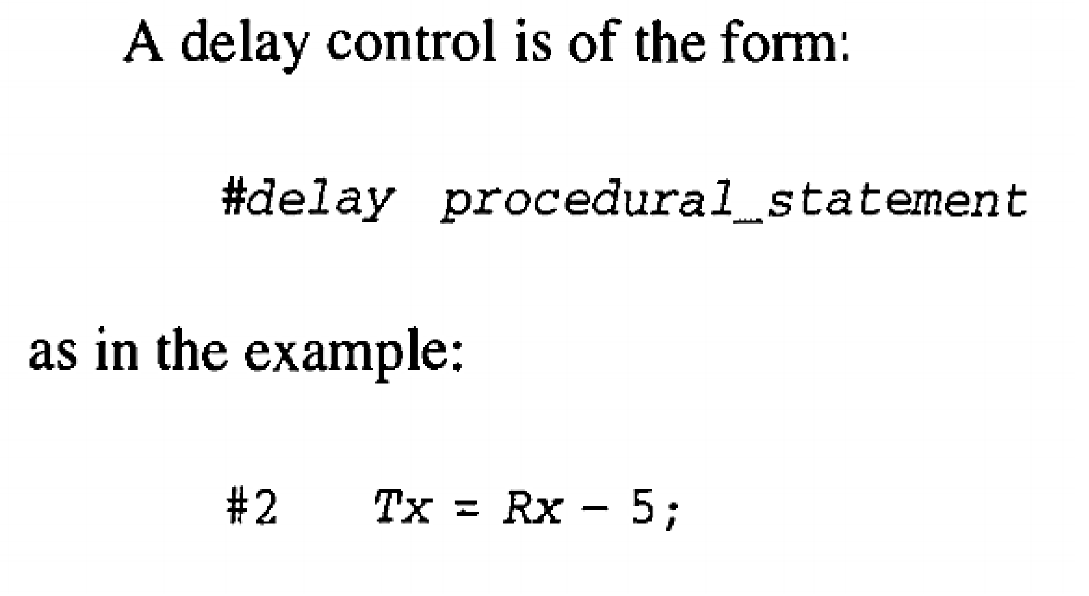
TIMING CONTROLS IN BEHAVIORAL MODEL

Timing Controls

- A timing control may be associated with a procedural statement.
- Various behavioral timing control constructs are available in Verilog.In Verilog, if there are no timing control statements, the simulation time does not advance.
- Timing controls provide a way to specify the simulation time at which procedural statements will execute.
- There are three methods of timing control:
  1. delay-based timing control
  2. event-based timing control,
  3. level-sensitive timing control.

1. Delay Control

- Delay-based timing control in an expression specifies the time duration between when the statement is encountered and when it is executed.
- Basically it means \"wait for delay\" before executing the statement.
- Delays are specified by the symbol #.



A delay control is of the form:

```
#delay  procedural_statement
```

as in the example:

```
#2    Tx = Rx - 5;
```

Here, procedural assignment statement is executed two time units after the statement is reached; it is equivalent to wait for 2 time units and then execute the assignment.

- A delay control can also be specified in the form:

#delay ;

This statement causes a wait for the specified delay before the next statement is executed.

E.g:

```
parameter ON_DELAY = 3, OFF_DELAY = 5;
always
  begin
    # ON_DELAY;           // wait for ON_DELAY.
    RefClk = 0;
    # OFF_DELAY;          // wait for OFF_DELAY.
    RefClk = 1;
  end
```

- The delay in a delay control can be an arbitrary expression, that is, it need not be restricted to a constant

```
#Strobe
  Compare = Tx ^ Mask;


#(PERIOD / 2)
  Clock = ~ Clock;
```

- If the value of the delay expression is 0, then it is called an explicit zero delay.

#0;   // Explicit zero delay.

- An explicit zero delay causes a wait until all other events that are waiting to be executed at the current simulation time are completed, before it resumes; the simulation time does not advance.
- If the value of a delay expression is an x or a z, it is the same as zero delay. If the delay expression evaluates to a negative value, the two's complement signed integer value is used as the delay

Zero delay control

- Procedural statements in different always-initial blocks may be evaluated at the same simulation time.
- The order of execution of these statements in different always-initial blocks is nondeterministic.
- Zero delay control is a method to ensure that a statement is executed last, after all other statements in that simulation time are executed.
- This is used to eliminate race conditions. However, if there are multiple zero delay statements, the order between them is nondeterministic.

Zero Delay Control

```
initial
begin
    x = 0;
    y = 0;
end

initial
begin
    #0 x = 1; //zero delay control
    #0 y = 1;
end
```

- The four statements " x = 0, y = 0, x = 1, y = 1" are to be executed at simulation time 0. However, since x = 1 and y = 1 have #0, they will be executed last.
- Thus, at the end of time 0, x will have value 1 and y will have value 1. The order in which x = 1 and y = 1 are executed is not deterministic.
- However, using #0 is not a recommended practice.

2. Event Control

- An event is the change in the value on a register or a net. Events can be utilized to trigger execution of a statement or a block of statements.
- With an event control, a statement executes based on events.
- An event control with a procedural statement delays the execution of the statement until the occurrence of the specified event.
- There are two kinds of event control.

  i. Edge-triggered event control

  ii. Level-sensitive event control

i) Edge-triggered Event Control

- An edge-triggered event control is of the form:

```
@ event  procedural_statement
```

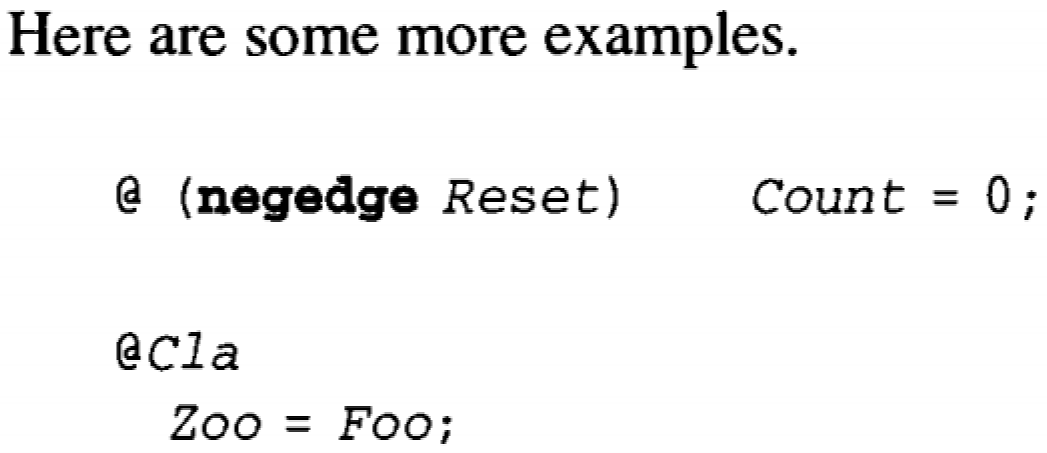
as in the example:

```
@ (posedge Clock)
  Curr_State = Next_State;
```

- In the above example, if a positive edge occurs on Clock the assignment statement executes, otherwise execution is suspended until a positive edge occurs on Clock.

Regular event control

The @ symbol is used to specify an event control. Statements can be executed on changes in signal value or at a positive or negative transition of the signal value.


Here are some more examples.

```
@ (negedge Reset)     Count = 0;

@Cla
  Zoo = Foo;
```

1. *The assignment statement executes only when a negative edge occurs on Reset.*
2. *Foo is assigned to Zoo when an event occurs on Cla, that is, wait for an event to occur on Cla, when it does occur, assign Foo to Zoo*

- Event control can also be of the form:

  @ event ;

- This statement causes a wait until the specified event occurs.
- Here is an example of such an usage in an initial statement that determines the on-period of a clock.

```
time RiseEdge, OnDelay;

initial
  begin
    // Wait until positive edge on clock occurs:
    @ (posedge ClockA);
    RiseEdge = $time;
    // Wait until negative edge on clock occurs:
    @ (negedge ClockA);
    OnDelay = $time - RiseEdge;
    $display ("The on-period of clock is %t.", OnDelay);
  end
```

Event OR Control

- Sometimes a transition on any one of multiple signals or events can trigger the execution of a statement or a block of statements.
- This is expressed as an OR of events or signals.
- The list of events or signals expressed as an OR is also known as a sensitivity list.
- The keyword or is used to specify multiple triggers
- Events or'ed to indicate **\"if any of the events occur\".**

```
@ (posedge Clear or negedge Reset)
  Q = 0;


@ (Ctrl_A or Ctrl_B)
  Dbus = 'bz;
```

Note that the keyword or does not imply a logical-or such as in an expression.

Event OR Control (Sensitivity List)

```
//A level-sensitive latch with asynchronous reset
always @( reset or clock or d)
                                //Wait for reset or clock or d to
change
begin
        if (reset)               //if reset signal is high, set q to 0.
                q = 1'b0;
        else    if(clock)       //if clock is high, latch input
                q = d;
end
```

Sensitivity List with Comma Operator

```
//A level-sensitive latch with asynchronous reset
always @( reset, clock, d)
                                //Wait for reset or clock or d to
change
begin
        if (reset)               //if reset signal is high, set q to 0.
                q = 1'b0;
        else    if(clock)       //if clock is high, latch input
                q = d;
end


//A positive edge triggered D flipflop with asynchronous falling
//reset can be modeled as shown below
always @(posedge clk, negedge reset) //Note use of comma operator
if(!reset)
   q <=0;
else
   q <=d;
```

posedge and negedge

- posedge and negedge are keywords in Verilog HDL that indicate a positive edge and a negative edge respectively.
- A negative edge is one of the following transitions:

```
1 -> x
1 -> z
1 -> 0
x -> 0
z -> 0
```

- A positive edge is oneof the following transitions

```
0 -> x
0 -> z
0 -> 1
x -> 1
z -> 1
```

ii) Level-sensitive Event Control

- In a level-sensitive event control, the procedural statement is delayed until a condition becomes true. This event control is written in the form:

```
wait (condition)
  procedural_statement
```

- The procedural statement executes only if the condition is true, else it waits until the condition becomes true.
- If the condition is already true when the statement is reached, then the procedural statement is executed immediately.
- The procedural statement is optional.

e.g.

```
wait (Sum > 22)
  Sum = 0;

wait (DataReady)
  Data = Bus;

wait (Preset);
```

- In the first statement, only when Sum becomes greater than 22 will the assignment of 0 to Sum occur.
- In the second example, Bus is assigned to Data only if DataReady is true, that is, DataReady has the value 1.
- In the last example, the execution is simply delayed until Preset becomes true.

THANK YOU