Bridge of Life Education

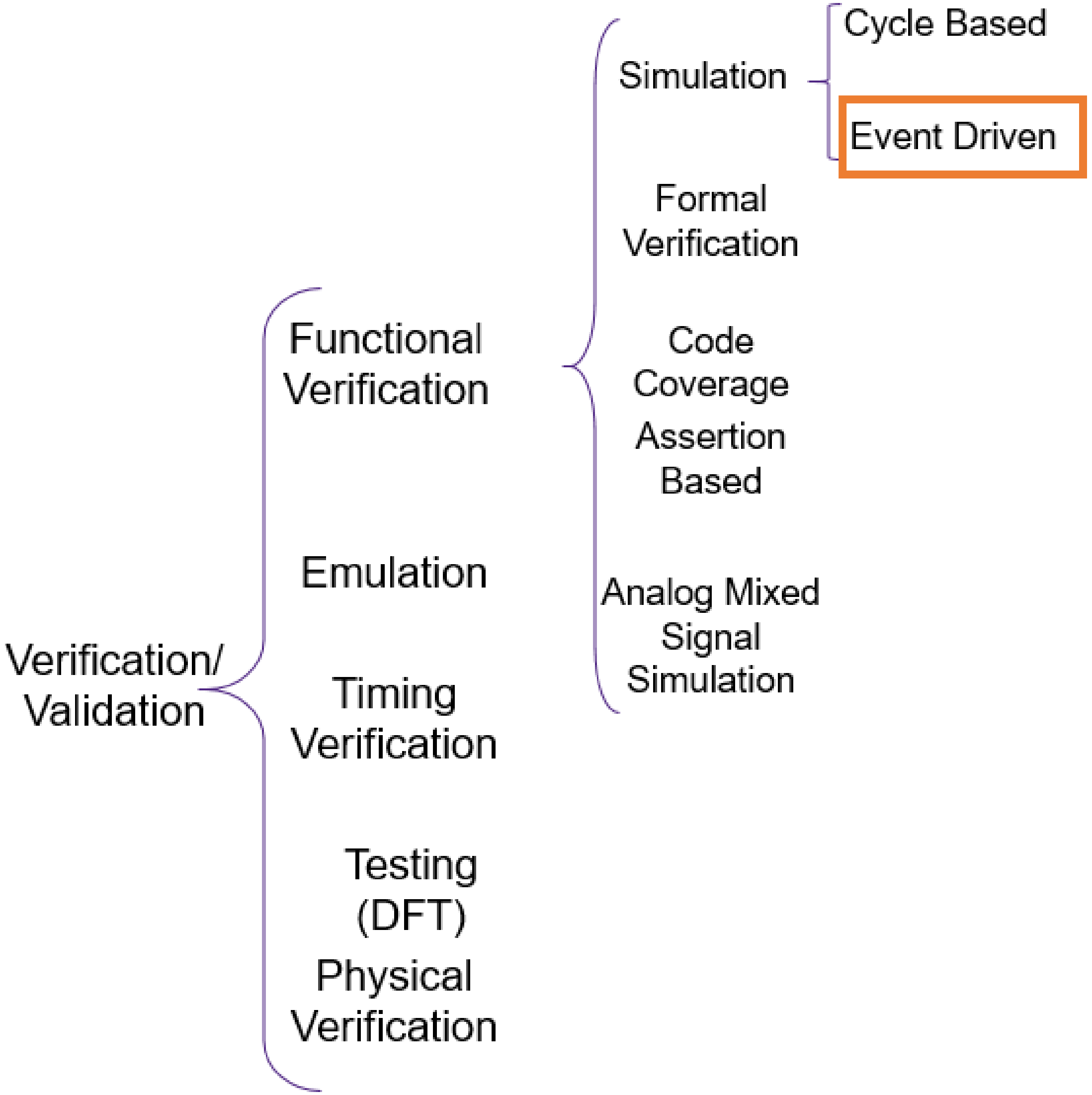
SOC Design
Verification - Testbench

Jiin Lai

Agenda

1. Scope of Verification
2. Testbench Basics

Scope of Verification



Verification/ Validation

Functional Verification

Emulation

Timing Verification

Testing (DFT)

Physical Verification

Simulation

Cycle Based

Event Driven

Formal Verification

Code Coverage

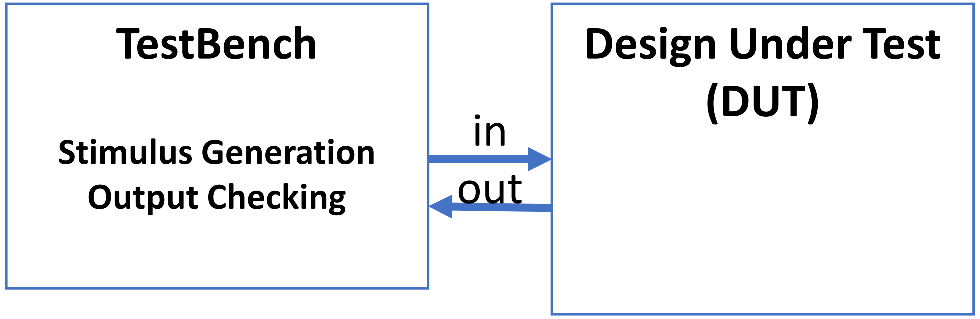
Assertion Based

Analog Mixed Signal Simulation

Testbench & DUT Structure

```
module top;

  wire  in;
  wire out;
```



TestBench

Stimulus Generation
Output Checking

in

out

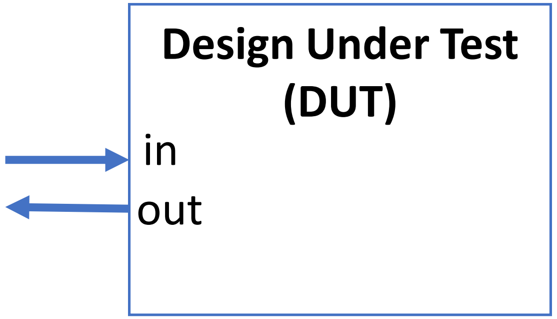
Design Under Test
(DUT)

```
endmodule
```

```
module top;
  reg  in;
  wire out;

    initial begin
    #d1  in = a;
    #d11
      if(out != exp) begin
        $display(“ERROR” ..);

    #d2  in = b;
      for ( i  ) begin
        #d3 in = c[i];
      end
    end
endmodule
```



Design Under Test
(DUT)

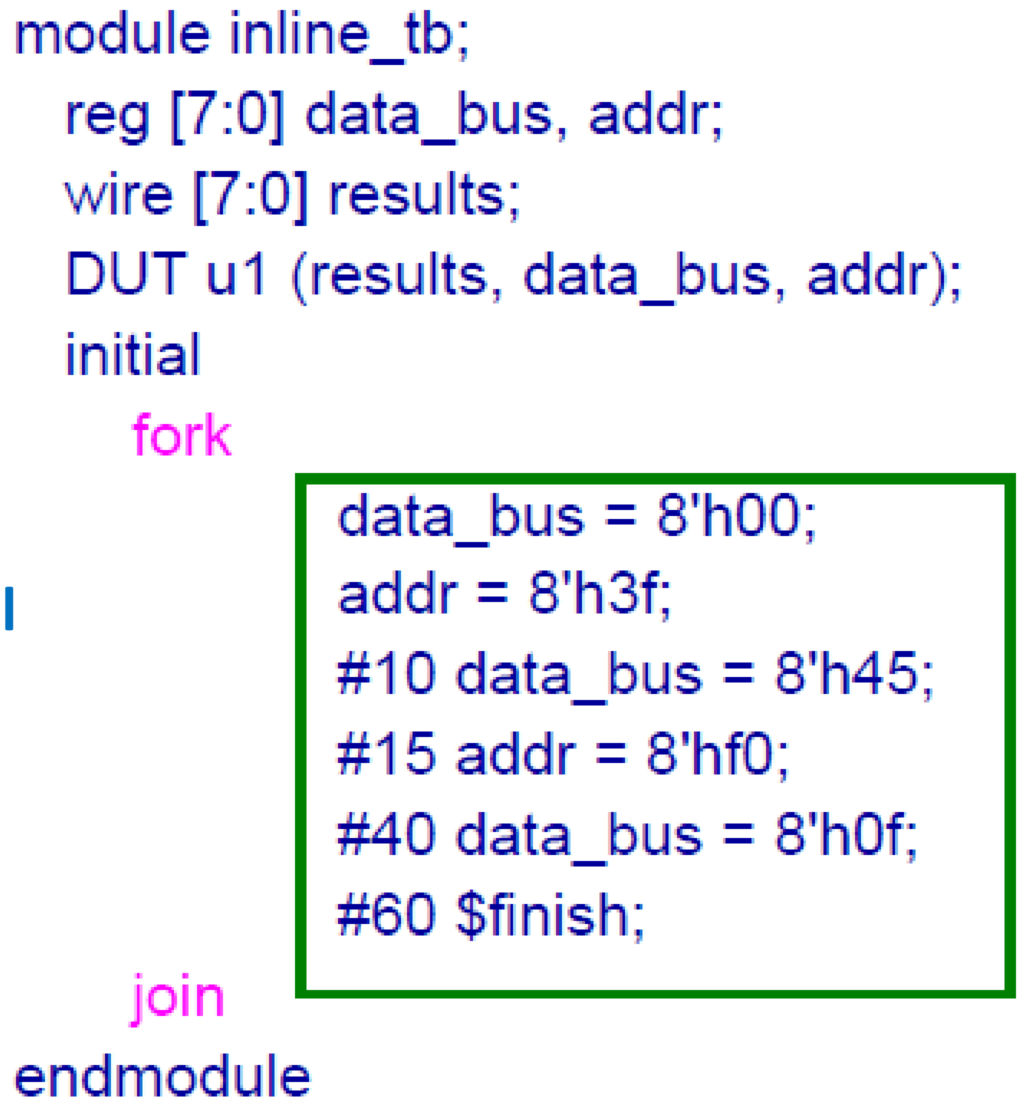
in

out

Applying Stimulus

- Inline initial block
- From a loop or always block
- From an array of vectors or integer

Inline



```
module inline_tb;
  reg [7:0] data_bus, addr;
  wire [7:0] results;
  DUT u1 (results, data_bus, addr);
  initial
     fork
            data_bus = 8'h00;
            addr = 8'h3f;
            #10 data_bus = 8'h45;
            #15 addr = 8'hf0;
            #40 data_bus = 8'h0f;
            #60 $finish;
     join
endmodule
```

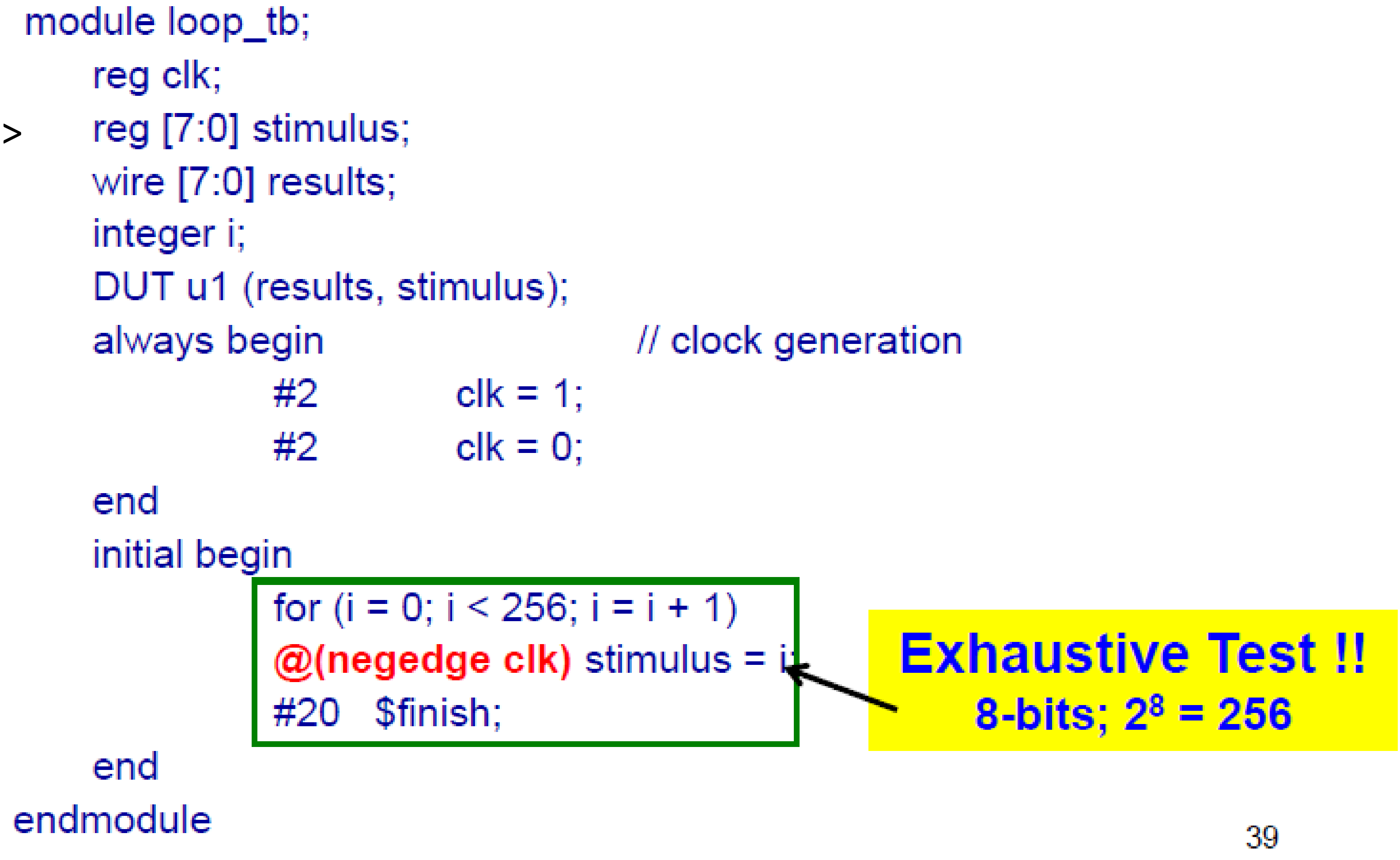
Better to use @ clock event control
Instead of # delay

From Loops with event control

Event Control
@<event_expression><statements>

Initial begin
- @(ee) rega = regb;
- @(posedge clk)

end



```
module loop_tb;
   reg clk;
   reg [7:0] stimulus;
   wire [7:0] results;
   integer i;
   DUT u1 (results, stimulus);
   always begin                    // clock generation
          #2      clk = 1;
          #2      clk = 0;
   end
   initial begin
          for (i = 0; i < 256; i = i + 1)
          @(negedge clk) stimulus = i;
          #20   $finish;
   end
endmodule
```

Exhaustive Test !!
8-bits; 2^8 = 256

39

From Arrays (from a file and read into an array)

```
module array_tb;
reg [7:0] data_bus, stim_array[0:15];   // array
integer i;
DUT u1 (results, stimulus);
initial begin
//load array with values
          #20 stimulus = stim_array[0];
          #30 stimulus = stim_array[15];    // in line
          #20 stimulus = stim_array[1];
          for (i = 14; i > 1; i = i - 1)          // from loop
                    #50 stimulus = stim_array[i] ;
          #30 $finish;
          end
endmodule
```

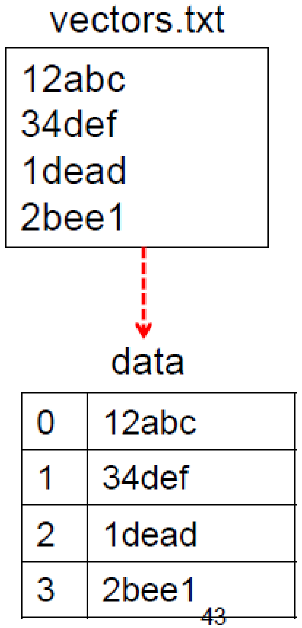
File Format for $readmemb and $readmemh

- **syntax:** *$readmemb("mem_file.txt",mema);*

```
module file_readmemh;
// Declare a array 4*20
 reg [19:0] data [0:3];
// initialize the hexadecimal reads from the vectors.txt file
 initial $readmemh("vectors.txt", data);

integer i;
 initial begin
     for (i=0; i < 4; i=i+1)
   $display("%d:%h",i,data[i]);
 end
endmodule
```

```
address 0=12abc
address 1=34def
address 2=1dead
address 3=2bee1
```



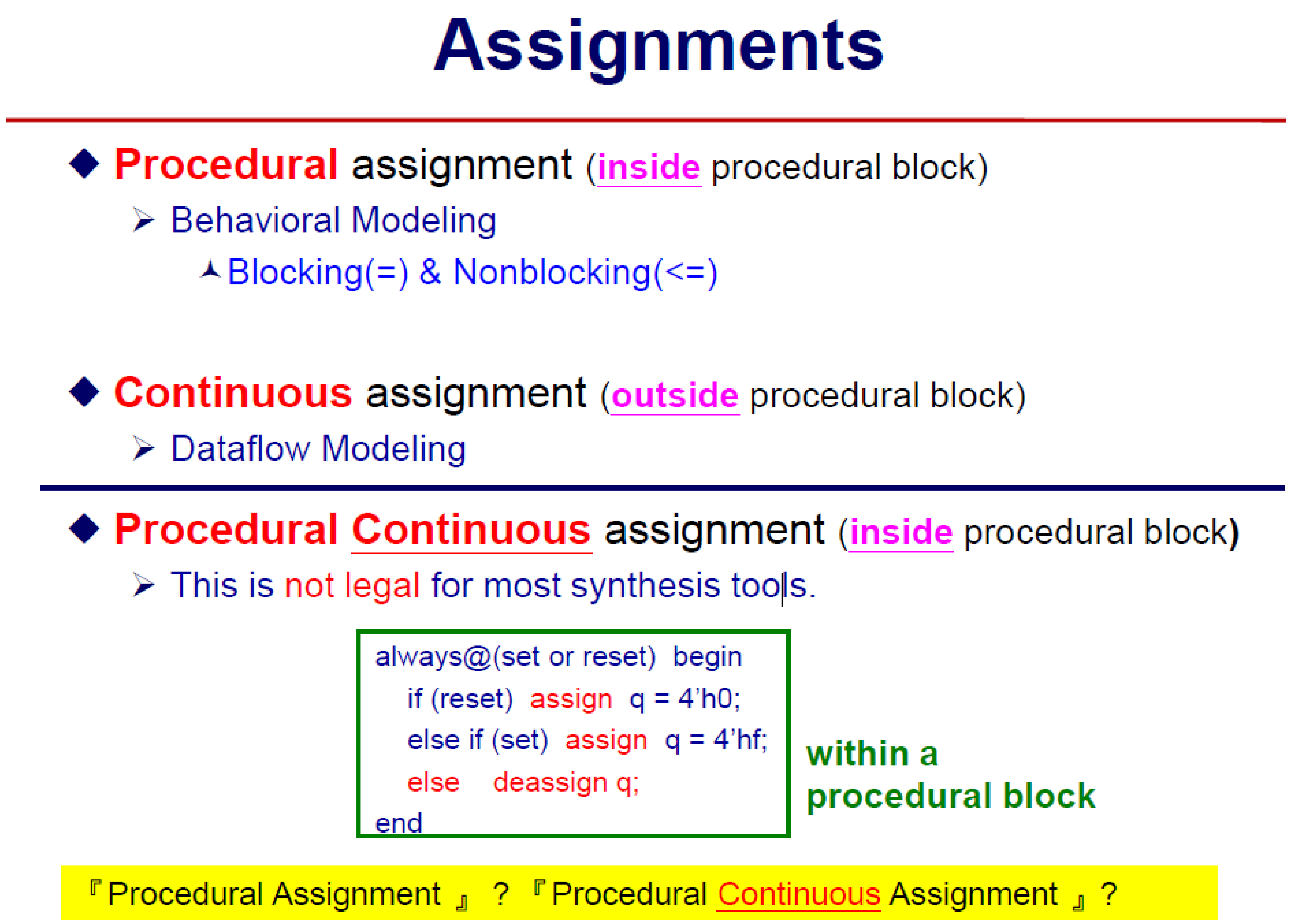
vectors.txt

12abc
34def
1dead
2bee1

data

0	12abc
1	34def
2	1dead
3	2bee1

Assignments



Assignments

- **Procedural** assignment (inside procedural block)
  - Behavioral Modeling
    - Blocking(=) & Nonblocking(<=)
- **Continuous** assignment (outside procedural block)
  - Dataflow Modeling
- **Procedural Continuous** assignment (inside procedural block)
  - This is not legal for most synthesis tools.

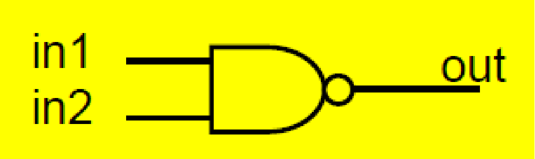
```
always@(set or reset)  begin
   if (reset)  assign  q = 4'h0;
   else if (set)  assign  q = 4'hf;
   else    deassign q;
end
```

within a procedural block

『Procedural Assignment 』 ? 『Procedural Continuous Assignment 』 ?

Task and Function

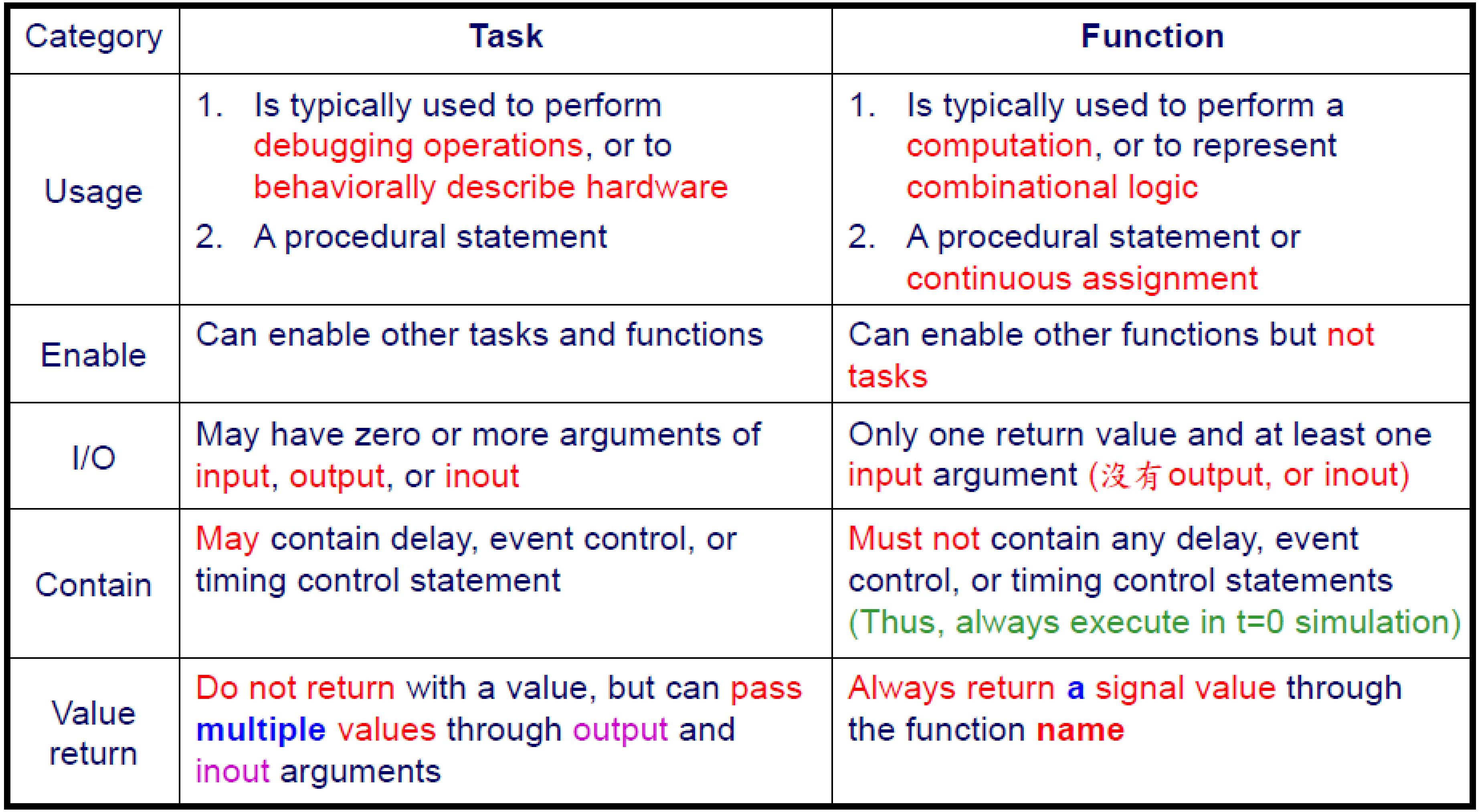
```
module t5(in1, in2, out);
    input in1, in2;
    output out;
    reg out;
    always @(in1 or in2)
    begin
            NOT(in1 & in2, out);
    end
    task NOT; //definition of
                  task NOT
            input a;
            output a_;
            a_ = ~a;
    endtask
endmodule
```



```
module t5(in1, in2, out);
    input in1, in2;
    output out;
    reg out;
    always @(in1 or in2)
    begin
            out = NOT(in1 & in2);
    end
    function NOT; //definition of
                       function NOT
            input a;
            NOT = ~a;
    endfunction
endmodule
```

Return signal value through the function name

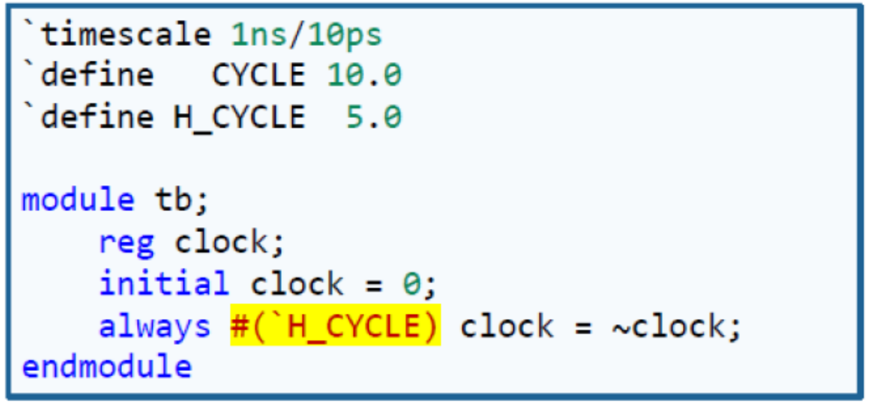
Function v.s. Task



Category	Task	Function
Usage	1. Is typically used to perform debugging operations, or to behaviorally describe hardware 2. A procedural statement	1. Is typically used to perform a computation, or to represent combinational logic 2. A procedural statement or continuous assignment
Enable	Can enable other tasks and functions	Can enable other functions but not tasks
I/O	May have zero or more arguments of input, output, or inout	Only one return value and at least one input argument (沒有output, or inout)
Contain	May contain delay, event control, or timing control statement	Must not contain any delay, event control, or timing control statements (Thus, always execute in t=0 simulation)
Value return	Do not return with a value, but can pass **multiple** values through output and inout arguments	Always return **a** signal value through the function **name**

Creating Clock

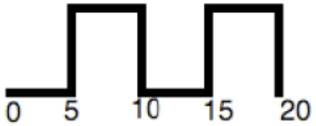
- **Creating clocks using** `always`



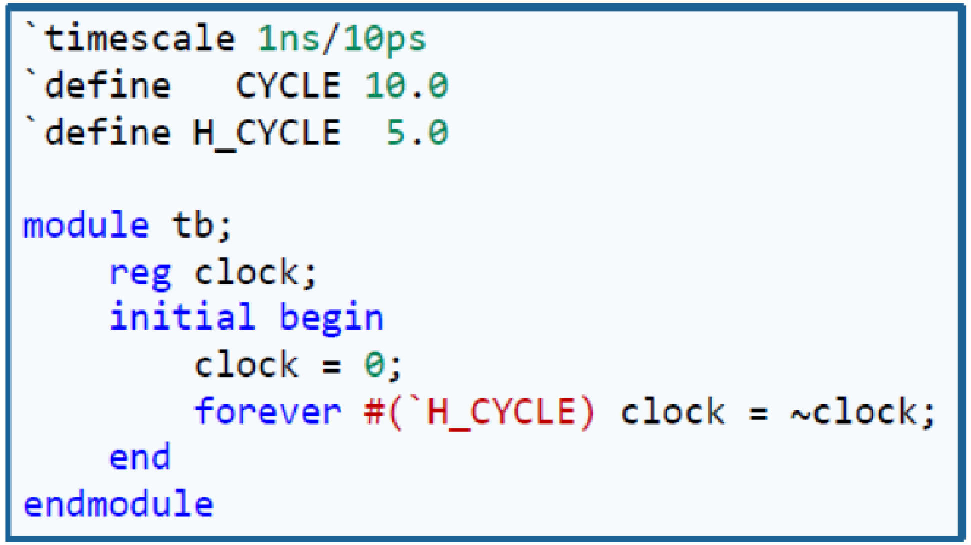
```
`timescale 1ns/10ps
`define   CYCLE 10.0
`define H_CYCLE  5.0

module tb;
    reg clock;
    initial clock = 0;
    always #(`H_CYCLE) clock = ~clock;
endmodule
```

Generated clock:



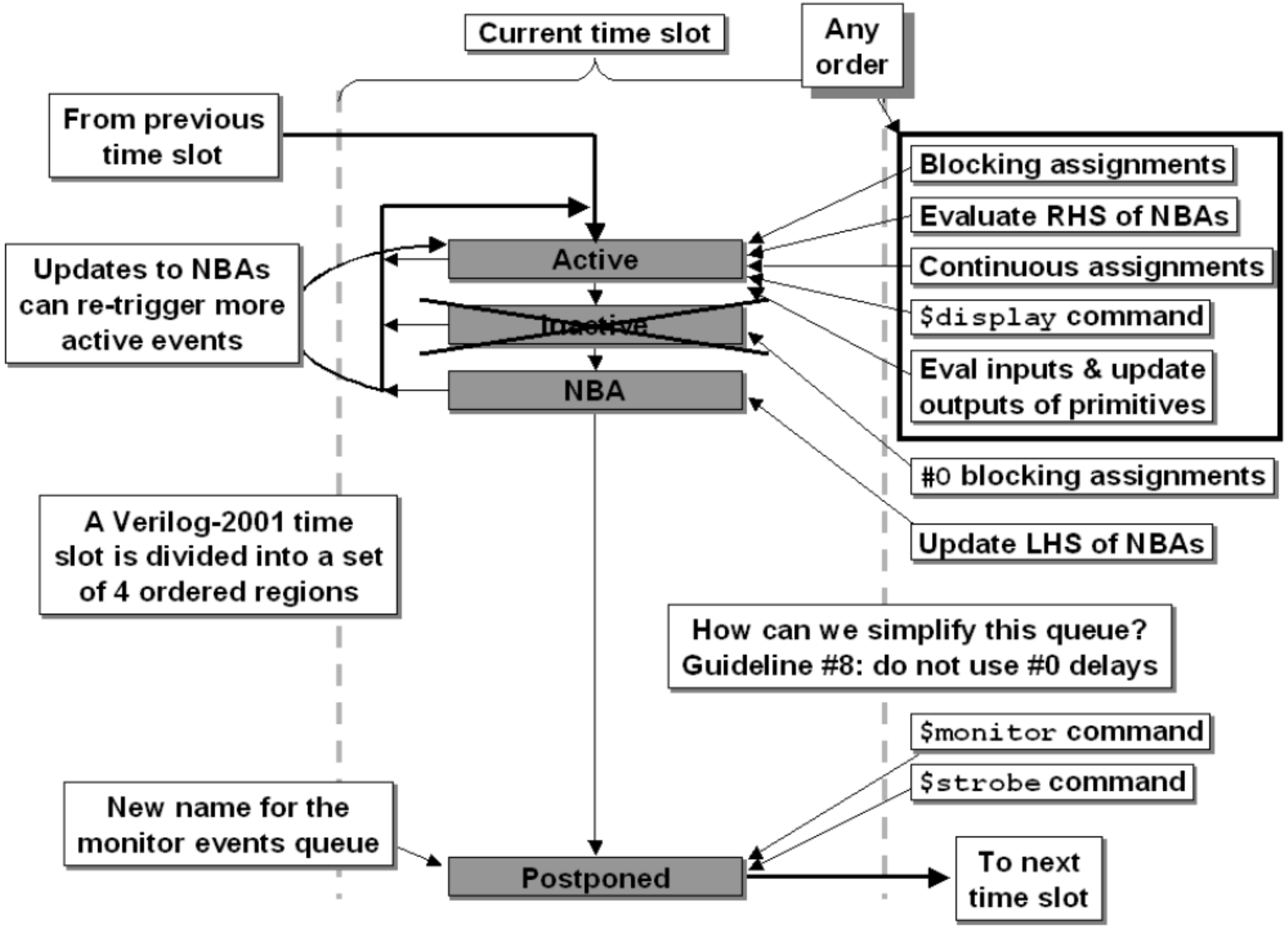
- **Creating clocks using** `forever`



```
`timescale 1ns/10ps
`define   CYCLE 10.0
`define H_CYCLE  5.0

module tb;
    reg clock;
    initial begin
        clock = 0;
        forever #(`H_CYCLE) clock = ~clock;
    end
endmodule
```

Stratified Event Queue - Understand simulator scheduling scheme



Current time slot
Any order
From previous time slot
Updates to NBAs can re-trigger more active events
Active
~~Inactive~~
NBA
Blocking assignments
Evaluate RHS of NBAs
Continuous assignments
$display command
Eval inputs & update outputs of primitives
#0 blocking assignments
Update LHS of NBAs
A Verilog-2001 time slot is divided into a set of 4 ordered regions
How can we simplify this queue? Guideline #8: do not use #0 delays
$monitor command
$strobe command
New name for the monitor events queue
Postponed
To next time slot

Figure 1 - Verilog "stratified event queue"

Refer to ppt: **verilog-critical-concepts**

Testbench Coding – System tasks

- System tasks and Functions: $
- $random (32-bit random number)
- File Input: [$readmemb | $readmemh](“filename”, array);
- $time (64-bit integer), $stime (32-bit integer), $realtime (64-bit real)
- $monitor, $strobe, $display, $write
- $finish, $stop
- $dumpfile(“file.dump); $dumpvars(); $dumpflush; $dumpoff; $dumpon; $dumpall;
- $fsdbDumpfile(“file”); $fsdbDumpvars;