CS2074: Computer Organization Laboratory

A.Y. 2021-2022 (Spring)

Faculty: Dr. Bidyut Kumar Patra

Behavioral Modeling

- It is primarily used to model sequential circuits, but can also be used to model pure combinatorial circuits.
- The mechanisms (statements) for modeling the behavior of a design are:
 - initial Statements
 - always Statements
- A module may contain an arbitrary number of initial or always statements which are executed concurrently with respect to each other.
- They may contain one or more procedural statements within them enclosed between begin ... end block and are executed sequentially.
- Both initial and always statements are executed at time=0 and then only always statements are executed during the rest of the time. The syntax is as follows:

initial [timing_control] procedural_statements;
always [timing_control] procedural_statements;

always Block

- Statements inside an always block are executed sequentially.
- An always block always executes, unlike initial blocks that execute only once at the beginning of the simulation.
- The always block have a sensitive list or a delay associated with it.
- Syntax

- The *symbol* @ after reserved word always, indicates that the block will be triggered at the condition in parenthesis after symbol @.
- It can drive reg and integer data types but cannot drive wire data types.

always Block (Example)

2:1 mux is described below with input x and y. The sel is the select input, and f is the mux output.

```
module mux(f, a, b, sel);
      input [3:0] x, y;
      input sel;
      output [3:0] f;
      reg[3:0] f;
      always @ (x or y or sel)
      begin
             if (sel == 0)
                    f = x;
             else
                    f = y;
             end
endmodule
```

Event Control

- An event controls the execution of a statement or a block of a statement.
- Value changes on variables and nets can be used as a synchronization event to trigger the execution of other procedural statements and is an implicit event.
- A negedge is a transition from 1 to X, Z or o and from X or Z to o.
- A posedge is a transition from 0 to X, Z or 1 and from X or Z to 1.
- Example

```
always @(posedge clock) out = in;
always @(negedge clock) z = n \ll 2;
```

 Here clock is of type reg, posedge and negedge is an event

initial Block

- The *initial* block indicates a process executes exactly once.
- Multiple initial blocks, execute in parallel
 - All start at time o
 - Each finishes independently
- Syntax

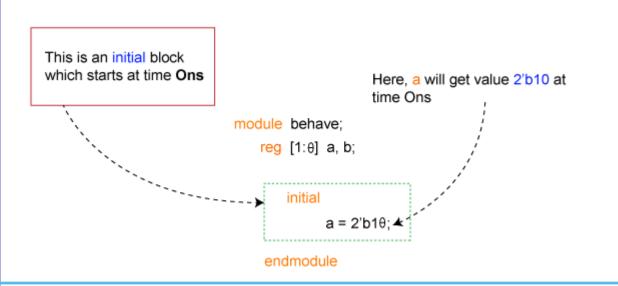
initial
[statement]

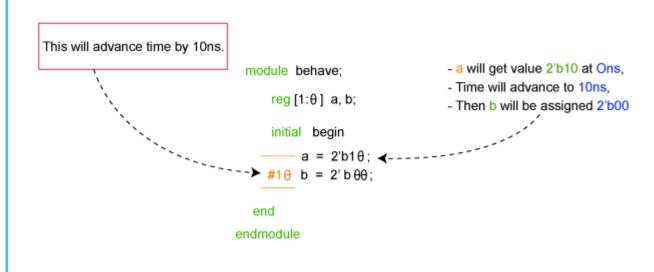
"OR"

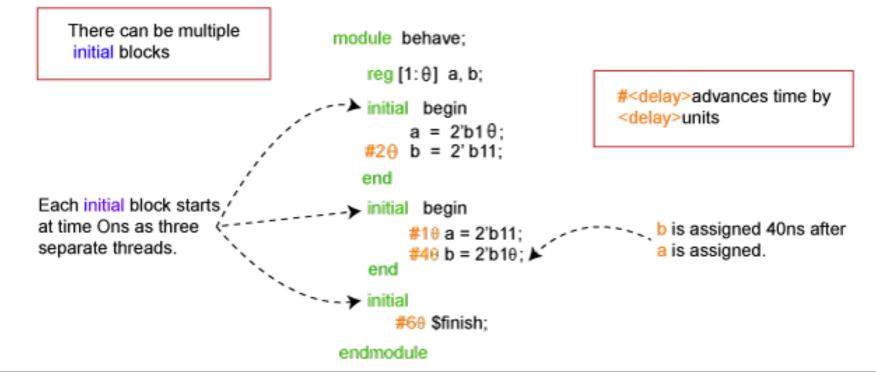
initial
begin
[multiple statements]
end

They are commonly used in test benches.

initial Block (Example)







Conditional Statements

- Just the same as if-else in C
- Syntax:

```
if (<expression>) true_statement;
```

```
if (<expression>) true_statement;
else false_statement;
```

```
if (<expression>) true_statement1;
else if (<expression>) true_statement2;
else if (<expression>) true_statement3;
else default_statement;
```

- True is 1 or non-zero, False is 0 or ambiguous (x or z)
- More than one statement: begin end

for loop

- Just the same as in C
- Syntax:

```
for( init_expr; cond_expr; change_expr)
begin
```

end

- Two important differences in verilog and C
 - In for loop we have begin and end in place of { and }.

statement;

- Statements like i++ are not allowed, we have to write instead as i = i+1;
- Make use of for loop freely in test benches.
- Instead of linearly specifying the stimulus, use for loop to go through a set of values.

while loop

- Just the same as in C
- Syntax:
 while (<expression>)
 statement;

Branching Statement

- Similar to switch-case statement in C
- Syntax:

```
case (<expression>)
     alternative1: statement1;
     alternative2: statement2;
```

default: default_statement; // optional endcase

Behavioral Statements (Procedural assignment statements)

- Blocking assignment (=)
 - Blocking assignment statements are assigned using (=) operator and are executed one after the other in a procedural block.
 - It will not prevent the execution of statements that run in a parallel block.
- Non-Blocking assignment (<=)
 - Non-blocking assignment statements are allowed to be scheduled without blocking the execution of the following statements and is specified by a (<=) symbol.
 - The same symbol is used as a relational operator in expressions, and as an assignment operator in the context of a non-blocking assignment.

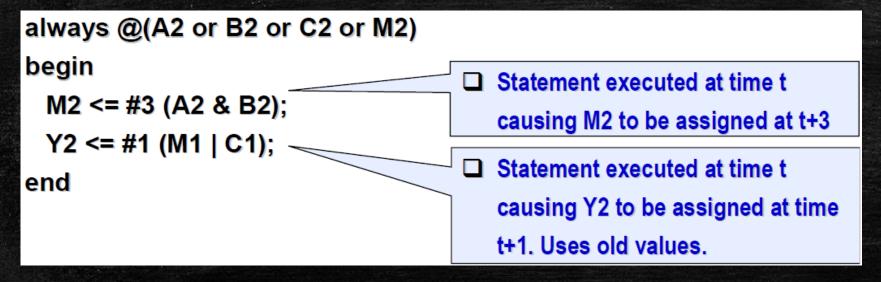
Blocking Assignment

```
always @(A1 or B1 or C1 or M1)
begin
M1 = #3 (A1 & B1);
Y1 = #1 (M1 | C1);
end

Statement executed at time t
causing M1 to be assigned at t+3

Causing Y1 to be assigned at time
t+4
```

Non-Blocking Assignment



Thank You.