UM TAGUM COLLEGE

DEPARTMENT OF ENGINEERING EDUCATION

COMPUTER ENGINEERING PROGRAM

Behavioral Modeling

DRILL 5

NAME:

STUDENT NUMBER:

TERMINAL NUMBER:

DATE OF PERFORMANCE:

DATE OF SUBMISSION:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PROFESSOR

I. DISCUSSION

Behavioural modelling represents digital circuits at a functional and algorithmic level. It is used mostly to describe sequential circuits, but can also be used to describe combinational circuits.

The primary mechanisms for modeling the behavior of a design are the following two statements:

* Initial statement
* Always statement

An initial statement executes only once and begins its execution at start of simulation which is at time 0.

***initial*** *[timing\_control] procedural\_statement*

An always statement executes repeatedly and also begins its execution at start of simulation which is at time 0.

**always** [timing\_control] procedural\_statement

A procedural\_statement is one of the following:

* procedural\_assignment (blocking or non\_blocking)
* procedural\_continuous\_assignment
* conditional\_statement
* case\_statement
* loop\_statement
* wait\_statement
* disable\_statement
* event\_trigger
* sequential\_block
* parallel\_block
* task\_enable (user or system)

Behavioral descriptions use the keyword **always**,followed by an optional event control expression specifies when the statements will execute. The target output of procedural assignment statements must be of the reg data type.

A block statement provides a mechanism to group two or more statements to act syntactically like a single statement. There are two kinds of blocks in Verilog HDL. These are:

* Sequential block (**begin…end**): Statements are executed sequentially in the given order.
* Parallel block (**fork…join**): Statements in this block execute concurrently.

A procedural assignment is an assignment within an initial statement or an always statement. It is used only to assign to a register data type. There are two kinds of procedural assignments:

* **Blocking procedural assignment**

A procedural assignment in which the assignment operator is an “=” is a blocking procedural assignment.

* **Non-blocking procedural assignment**

A procedural assignment in which the assignment operator is an “<=” is a non-blocking procedural assignment.

The Conditional Statement if-else

The *if - else* statement controls the execution of other statements. In programming language like C, *if - else* controls the flow of program. When more than one statement needs to be executed for an *if* condition, then we need to use begin and end as seen in earlier examples.

One-Way Selection:

***if*** *(condition)*

*statements;*

Two-Way Selection (if-else):

**i*f*** *(condition)*

*statements;*

***else***

*statements;*

Nested if (if-else-if):

***if*** *(condition)*

*statements;*

***else******if*** *(condition)*

*statements;*

*else*

*statements;*

Case construct

A case statement is a multi-way conditional branch. It has the following syntax:

**case** ( case\_expr )

case\_item\_expr {,case\_item\_expr } : procedural\_statement

…

…

[ **default** : procedural\_statement ]

**endcase**

The **case** construct has two important variations: casex and casez

There are four kinds of loop statements. These are:

* + Forever-loop
    - This loop continuously executes the procedural statement.

**f*orever*** *procedural\_statement*

* + Repeat-loop
    - Executes the procedural statement the specified number of times.

***repeat*** *(loop\_count)**procedural\_statement*

* + While-loop
    - Executes the procedural statement until the specified condition becomes false.

***while*** *(condition)*

*procedural\_statement*

* + For-loop
    - Repeats the execution of the procedural assignment a certain number of times.

***for*** *(initial\_assignment ; condition; step-assignment)*

*procedural\_statement*

II. Drill Exercises

1. Design a Verilog behavioural model of a sequence detector using D flip-flops.

The state table for a sequence detector is as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Present State | | Input | Next State | | Output |
| A | B | x | A | B | y |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

The program below describes a behavioural model of the circuit obtained from the state table above. Once compiled, save the file as drill5\_1.vl

*module flip\_flop (clk, reset, d, q);*

*input clk, reset, d;*

*output q;*

*reg q;*

*always @ (posedge clk )*

*begin*

*if (reset == 1)*

*begin*

*q <= 0;*

*end*

*else*

*begin*

*q <= d;*

*end*

*end*

*endmodule*

*module circuit\_ff(input clk, reset, x, output y);*

*wire OR1, OR2, AND1, AND2, AND3;*

*wire A, B, Bnot;*

*not (Bnot, B);*

*flip\_flop ff1(clk, reset, OR1, A);*

*flip\_flop ff2(clk, reset, OR2, B);*

*and (AND1,A,x), (AND2,B,x), (AND3,Bnot,x), (AND4,A,B);*

*or (OR1, AND1,AND2), (OR2,AND1,AND3);*

*and (y,B,A);*

*endmodule*

*module testff;*

*reg clk, reset,d;*

*wire q;*

*circuit\_ff cff(clk, reset, d, q);*

*initial begin*

*clk=0; reset=1; d=0;*

*$monitor("clk=%b reset=%b d=%b q=%b",clk,reset,d,q);*

*end*

*initial begin*

*forever #1 clk=~clk;*

*end*

*initial fork*

*#1 reset=0;*

*#2 d=1;*

*#3 reset=1;*

*#4 reset=0;*

*#5 d=0;*

*#8 d=1;*

*#10 $finish;*

*join*

*endmodule*

2. Create a behavioural description of an eight-to-one line multiplexer.

The program below is a behavioural description of an 8-1 MUX. Compile the file then save it as drill5\_2.vl

*module mux\_8\_1 (output reg m\_out, input [7:0]in\_x, input[2:0] select);*

*always@(in\_x[0],in\_x[1],in\_x[2],in\_x[3],in\_x[4],in\_x[5],in\_x[6],in\_x[7],select)*

*case(select)*

*3'b000: m\_out=in\_x[0];*

*3'b001: m\_out=in\_x[1];*

*3'b010: m\_out=in\_x[2];*

*3'b011: m\_out=in\_x[3];*

*3'b100: m\_out=in\_x[4];*

*3'b101: m\_out=in\_x[5];*

*3'b110: m\_out=in\_x[6];*

*3'b111: m\_out=in\_x[7];*

*endcase*

*endmodule*

*module testMux();*

*reg [7:0] x;*

*reg [2:0] select;*

*wire m\_out;*

*mux\_8\_1 MUX1(m\_out, x, select);*

*initial begin*

*select=2'b00; x=8'h4F;*

*$strobe("Select Input Output");*

*$monitorb(select," ", x, " ", m\_out);*

*#1 select=3'b000;*

*#1 select=3'b001;*

*#1 select=3'b010;*

*#1 select=3'b011;*

*#1 select=3'b100;*

*#1 select=3'b101;*

*#1 select=3'b110;*

*#1 select=3'b111;*

*#1 $display("Changing value of input");*

*#1 x=8'h98;*

*#1 select=3'b000;*

*#1 select=3'b001;*

*#1 select=3'b010;*

*#1 select=3'b011;*

*#1 select=3'b100;*

*#1 select=3'b101;*

*#1 select=3'b110;*

*#1 select=3'b111;*

*#100 $finish;*

*end*

*endmodule*

3. Create an HDL model of the operation of a sequential circuit based from the given state diagram:



The program below is formulated based from the given diagram above. Compile the file then save it as drill5\_3.vl

*module state\_diagram(*

*output reg y\_out,*

*input x\_in, clock, reset*

*);*

*reg[1:0] state, next\_state;*

*parameter S0=2'b00,*

*S1=2'b01,*

*S2=2'b10,*

*S3=2'b11;*

*always@(posedge clock, negedge reset)*

*if (reset==0) state <= S0;*

*else state <= next\_state;*

*always@(state, x\_in)*

*case (state)*

*S0: if (x\_in) next\_state=S1; else next\_state=S0;*

*S1: if (x\_in) next\_state=S3; else next\_state=S0;*

*S2: if (~x\_in) next\_state=S0; else next\_state=S2;*

*S3: if (x\_in) next\_state=S2; else next\_state=S0;*

*endcase*

*always@(state, x\_in)*

*case (state)*

*S0: y\_out=0;*

*S1,S2,S3: y\_out=~x\_in;*

*endcase*

*endmodule*

*module sdiag;*

*wire t\_y\_out;*

*reg t\_x\_in, t\_clock, t\_reset;*

*state\_diagram sd(t\_y\_out, t\_x\_in, t\_clock, t\_reset);*

*initial #200 $finish;*

*initial begin*

*t\_clock=0;*

*forever #5 t\_clock=~t\_clock;*

*end*

*initial fork*

*$monitor($time,,"reset=%b clock=%b x=%b y=%b" ,t\_reset, t\_clock, t\_x\_in, t\_y\_out);*

*t\_reset=0;*

*#2 t\_reset=1;*

*#87 t\_reset=0;*

*#89 t\_reset=1;*

*#10 t\_x\_in=1;*

*#30 t\_x\_in=0;*

*#40 t\_x\_in=1;*

*#50 t\_x\_in=0;*

*#52 t\_x\_in=1;*

*#54 t\_x\_in=0;*

*#70 t\_x\_in=1;*

*#80 t\_x\_in=1;*

*#70 t\_x\_in=0;*

*#90 t\_x\_in=1;*

*#100 t\_x\_in=0;*

*#120 t\_x\_in=1;*

*#160 t\_x\_in=0;*

*#170 t\_x\_in=1;*

*join*

*endmodule*

III. Programming Exercise

1. Design a Verilog HDL behavioural model of a 4-bit up-down counter designed using negative edge triggered T flip flops. Save the file as exercise5\_1.vl
2. Code a behavioural description of a sequential circuit with two JK flip-flops F1 and F2 and two inputs X and Y. If X=0 and Y=1, the circuit goes from transitions 01 to 00 to 10 to 11 back to 01. If X=0, and Y=0, the circuit goes through the transition 01 to 11 to 10 to 00 back to 01. The circuit remains the same regardless of the value of Y if X=1.
3. Code a behavioural model of a four-bit shift register with parallel load using positive edge triggered D flip-flops. There are two control inputs: *load* and *shift*. When *shift*=1, the contents of the register are shifted by one position. New data are transferred into the register when *load*=1 and *shift*=0. If both control inputs are equal to 0, the contents of the register do not change.

IV. Review Questions

1. How is race condition experienced in Verilog HDL programming?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Is there a difference between the instructions

*always #1 a=!a;*

and

*forever #1 a=!a;*

If yes, what is/are their difference/s?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Differentiate the selection constructs used in Verilog with those used in high-level languages (C++, C#, etc).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. How are multiple *always@* blocks executed within a given program?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. What are sensitivity lists? How do they affect the entire behavioural model description of a program?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Why does the register operation of state transitions use a non-blocking operator ( <= ) ?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_