Name Solution

# Digital Design using Verilog
# EE 4702-1
# Practice Midterm Examination

31 March 2000 13:40–?

|  |  |  |
|---|---|---|
| Problem 1 |  | (20 pts) |
| Problem 2 |  | (20 pts) |
| Problem 3 |  | (20 pts) |
| Problem 4 |  | (40 pts) |
| Exam Total |  | (100 pts) |

Alias

*Good Luck!*

## Problem 1:

(*a*) Write a Verilog *behavioral* description of a four-bit adder module. The adder should have three inputs, `a`, `b`, and `cin`, and two outputs, `sum` and `cout`. Ports `cin` and `cout` are one bit, the other ports are four bits each. (10 pts)

```
// Solution
module add4(sum,cout, a, b, cin);
   output sum, cout;
   input  a, b, cin;

   reg [3:0] sum;
   reg       cout;

   wire [3:0] a, b;
   wire       cin;
   reg [4:0]  fullsum;

   always @( a or b or cin ) {cout,sum} = a + b + cin;

endmodule // add4
```

(*b*) Write a Verilog explicit structural description of an eight-bit adder that uses two of the four-bit adders above. (That is, instantiate the modules designed above.) (10 pts)

```
// Solution
module add8(sum, cout, a, b, cin );
   output sum, cout;
   input  a, b, cin;

   wire [7:0] sum;
   wire       cout;

   wire [7:0] a, b;
   wire       cin;

   wire       cbetween;

   // Named ports are used for readability.
   add4 a0(.sum(sum[3:0]),.cout(cbetween),.a(a[3:0]),.b(b[3:0]),.cin(cin));
   add4 a1(.sum(sum[7:4]),.cout(cout),    .a(a[7:4]),.b(b[7:4]),.cin(cbetween));

endmodule // add8
```

Problem 2: The code fragment below implements the clock in the testbenches for homeworks 2 and 3 (and the solution to 4).

```
// Clock.
always
  begin:CLOCK
     wait( start === 1 && done === 0 );
     forever # (`timeunit * 0.5/s1.freq ) clk <= ~ clk;
  end
```

(*a*) Symbol `‘timeunit` was set to the number of simulator time units per second. What kind of Verilog thing is `‘timeunit` (register, integer, etc.) and how was it defined? (5 pts)

It is a macro, it is defined using the `define` compiler directive.

(*b*) Expression `s1.freq` is the frequency of the clock provided to the tachometer. What kind of Verilog thing is it and how did it get its value? (5 pts)

Expression `s1.freq` is the hierarchical name of parameter `freq` in instance `s1` of module `tach1`. As with all parameters, the expression is a constant. In a part of the the testbench code that is not shown the parameter value is specified where `s1` is instantiated.

(*c*) Symbol `start` is an input to the testbench module, it is set to `1` when the testbench is to start. Symbol `done` is an output which the testbench sets to one tests are completed. What would happen if `start === 1` were removed from the code fragment above? (5 pts)

The clock would start when the simulator started. This would waste CPU time (of the system running the simulator) if multiple instances were being tested, but should not result in incorrect execution.

(*d*) What would happen if `done === 0` were removed from the code fragment above? (5 pts)

The clock would not stop when the testbench finished. This would waste CPU cycles if multiple instances were being tested and the simulator would not stop when the testbench finished (unless the user or some other code stopped it).

Problem 3: An *accumulator* has three inputs, `amt`, `reset`, and `clk`, and an output, `sum`. The accumulator has an internal 32-bit register which is updated as follows: On a positive edge of `clk` it adds `amt`, a 32-bit integer, to the register; on the negative edge of `clk` it places the new sum on its outputs (until the next negative edge). Whenever reset is high the register is set to zero and the output changes immediately. Write a Verilog behavioral description of this module. (20 pts)

```
// Solution
module acc(sum,amt,reset,clk);
   output sum;
   input  amt, reset, clk;

   reg [31:0] sum;
   wire [31:0] amt;
   wire        reset, clk;
   reg [31:0]  nextsum;

   always @( clk or posedge reset)
     if( reset ) begin
        nextsum = 0; sum = 0;
     end else if( clk === 1 )
       nextsum = amt + sum;
     else sum = nextsum;

endmodule // acc
```

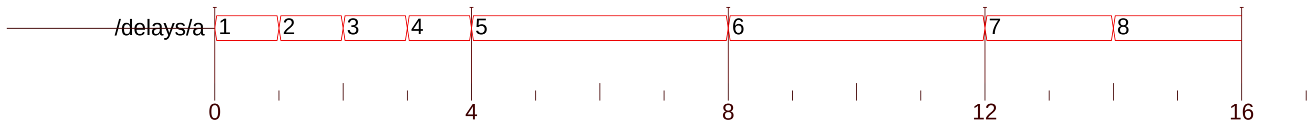
Problem 4: Answer each question below.

($a$) The code below starts executing a $t = 0$. Show all changes in `a`, include the time of the change and the new value. (5 pts)

```
integer a;

initial begin
   a = 1;
   #1;
   a = 2;
   #1;
   a = 3;
   #1;
   a <= 4;
   #1;
   a <= 5;
   #1;
   #3 a = a+1;
   #1;
   a = #3 a+1;
   #1;
   a <= #3 a+1;
   #1;
   a = a+1;
end
```

Timing is shown below:



Entity:delays  Architecture:  Date: Mon Apr 03 18:00:50 CDT 2000   Row: 1 Page: 1

(*b*) The programmer expected execution to exit the loop below when either i was 1000 or a[i] == c, but that's not what happened. What goes wrong and how can it be fixed? The loop must be exited using a disable statement. (5 pts)

```
integer i, c;
integer a[0:999];

// ...

i = 0;

while( i < 1000 ) begin:LOOP

   if( a[i] == c ) disable LOOP;

   i = i + 1;

end
```

Execution of the disable statement forces execution to move to the **end**, but the **begin:LOOP** ··· **end** statement is inside the **while** loop, so after moving to **end** execution will return to the beginning of the while loop and if **i** is less than 1000, another iteration will be performed. The corrected code is shown below.

```
// Solution

      integer i, c;
      integer a[0:1000];

      // ...

      i = 0;

      begin:LOOP
         while( i < 1000 ) begin

            if( a[i] == c ) disable LOOP;

            i = i + 1;

         end
      end
```

(*c*) Describe three uses for Verilog behavioral code. (5 pts)

(1) Testbenches.

(2) Behavioral description for a synthesis program.

(3) Behavioral description of some part of a larger system so the larger system can be simulated. Later the behavioral description may be re-written so that it is synthesizable.

(*d*) What is the difference between the following two declarations? (5 pts)

```
  wire [7:0] w1;
  wire [0:7] w2;
```

In the first (`w1`) bit 7 is the most significant bit, in the second (`w2`) bit 7 is the least significant bit.

(*e*) What do the values `x` and `z` signify? (5 pts)

Value `x` is unknown (there is no way to determine a signal value). Value `z` is high impedance.

(*f*) What is `10'had`? (5 pts)

A 10-bit integer written using a hexadecimal representation.

(*g*) Name two features of primitives that are not available for modules. (5 pts)

(1) Primitives can have any number of inputs (for example and gates).

(2) Primitives can be instantiated anonymously. (There is no need to specify a name.)

(3) Delay can be specified in a primitive instantiation.

(*h*) What is the difference between `case`, `casex`, and `casez`? (5 pts)

In the ordinary `case` statement there must be an exact match between the case expression and the case item. With `casex` an unknown value will match anything. With `casez` a high-impedance value will match anything.