Each question is worth 2 points. The correct answer earns 2 points, the incorrect 0. You have up to three hours to take this test. Answer all of the questions. Off-campus students, please return this test completed as specified in the syllabus.

The exam is open book, open notes. No computers, PDAs, or cell-phones are allowed. **Please do NOT have cell phones on your desk.**

Note that sometimes the answer is INTENTIONALLY E.

Clearly indicate your answers at the question.

Name: ___________________

Student Number: ______________

Question 1
What is wrong with the following code fragment?

```verilog
class @(posedge clock)
begin
    D <= A & C;
    if (E) G <= D | F;
end
```

A. Unintentional latches are being built.
B. A timing loop (or arc) is present.
C. D should not be assigned like this. Unbuildable “wired-or” logic is implied.
D. E is missing from the second sensitivity list.
E. Nothing is wrong with this code fragment.

E

Question 2
What is wrong with the following code fragment?

```verilog
class @(*)
begin
    A = E ^ D;
    if (C) A = ~A;
end
```

A. Unintentional latches are being built.
B. A should not be assigned like this. Unbuildable “wired-or” logic is implied.
C. There is combinational logic feedback, i.e. a “timing loop” or “arc”.
D. The sensitivity list is incomplete.
E. Nothing is wrong with this code fragment.

E
Question 3
What logic function does the following code fragment describe? Sketch the logic. All variables are 1 bit wide.

```verbatim
always@(*)
begin
  A = B ^ D;
  if (E) A = A | E;
  else A = A ^ E;
end
```

Mux with 2 inputs: one two and gates, other xor gate and or gate. (first xor gate shared)

Question 4
Sketch the logic.

```verbatim
integer i;
always@(*)
begin
  N = 0;
  for (i=0; i<=3; i=i+1)
    N = N | A[i];
end
```

Four OR gates, connecting A[i]. There are various optimizations that are OK.

Question 5
Complete the missing phrase. In comparison with standard cell ASICs, FPGAs are limited in all resources but it is particularly important when designing FPGAs to be very aware of the number of ______ being specified in the design.

A. DRAM interfaces.
B. long distance wires
C. flip-flops
D. adder carry chains.
E. None of the above.apply

B
Question 6
You are designing a security chip for use in a high volume application. Highly customized logic structures are required in order to make the chip secure against reverse engineering its logic details. What design style are you like to use?

A. FPGA.
B. Standard Cell ASIC.
C. Gate Array.
D. All are suitable.
E. None are suitable.

B

Question 7
Sketch the logic being described by the following statement group.

wire signed [3:0] A, B, C;
wire [1:0] D;
assign C = {4{B[3]},B} >> D;

Arithmetic shifter

Question 8
What style is the following FSM code fragment?

always@(posedge clock) state <= next_state;
always@(state or A)
begin
  case (state)
    2’b01 : if (A) next_state = 2’b10;
             else next_state = 2’b01;
    2’b10 : next_state = 2’b01;
2'bxx : next_state = 2'b01;
   endcase
   out = state[0];
end

A. Moore Machine
B. Mealy Machine
C. Neither
D. Either
E. Other.

B

**Question 9**
The code fragment in Question 8 is likely to implement unintended latches?
   A. True
   B. False

B

**Question 10**
The main purpose of adding Design For Test (DFT) features is to help debug the chips as they come off the factory floor?
   A. True
   B. False

B

**Question 11**
Sketch the design described by the following code fragment.

``` VERILOG
task foo (A, B);
input A, B;
begin
   foo = A | B;
end

always@(+clock) D = foo(D, E);
```

OR gate with OP of flipflop feeding it. OR gate goes to FF input.
Question 12
If “enable” is high only 1% of the time, and the code below is synthesized, which of the following will have the lowest power consumption.

A.
wire [31:0] A, B, D;
reg [31:0] E;
always@(posedge clock)
    begin
        if (enable) E <= A * B;
    end

B.
wire [31:0] A, B;
reg [31:0] AA, BB, E;
always@(posedge clock)
    begin
        AA <= enable ? A : 0;
        BB <= enable ? B : 0;
        E <= AA * BB;
    end

C.
wire [31:0] A, B, AA, BB;
reg [31:0] E;
always@(posedge clock)
    begin
        E <= AA * BB;
    end
    assign AA = enable ? A : 0;
    assign BB = enable ? B : 0;

D.
wire [31:0] A, B;
reg [31:0] E, AA, BB;
always@(posedge clock)
    begin
        AA <= A;
        BB <= B;
        if (enable) E <= AA * BB;
    end

F. These all consume the same power.

C
**Question 13**
If the following logic is built exactly as described, which test vector sensitizes a stuck-at-0 fault at b and propagates it to the output h.

```verilog
wire a, b, c, d, e, f, g, h;

assign f = a ? b : c;
assign g = d | f;
assign h = e & g;
```

A. \(\{a, b, c, d, e\} = 5'b10100;\)
B. \(\{a, b, c, d, e\} = 5'b01101;\)
C. \(\{a, b, c, d, e\} = 5'b11001;\)
D. \(\{a, b, c, d, e\} = 5'b11011;\)
E. None of the above

C

**Question 14**
What does the following primitive describe:

```verilog
primitive PlanetX (A, B, F);
output F;
reg F;
input A, B;
table
  0 0 : 0
  0 1 : 1
  1 1 : 1
  1 0 : 1
endtable
endprimitive
```

A. An XOR gate.
B. An OR gate.
C. An AND gate
D. A D flip-flop
E. None of the above

B
Question 15
Sketch the logic described by the following code fragment. Note the use of blocking assignment. All variables are one bit wide.

```verilog
always@(posedge clock)
  begin
    E = C & D;
    C = A | B;
    D = C ^ A;
  end

E uses Q outputs. D uses D input of C.
```

Question 16 / 17
Consider the following code fragment:

```verilog
thingy U1 (clock, A, B);
thingy U2 (clock, C, D);
```

The interfacing logic connected to A and B has very different timing than the interfacing logic connected to C and D. Write a short script fragment that ensures this will synthesize correctly. [4 points]

```bash
uniquify -cell U1 -new_name thingy2
characterize -constraints {U1}
current_design thingy2
compile
current_design top
characterize -constraints {U2}
current_design thingy
compile
// minor syntax errors are not penalized
```
**Question 18**
What does the following line of Verilog code do? You are to give ONE answer. Multiple answers will get zero points.

```verilog
\ \ synopsys off
```

It is a synopsus directive specifying that the code after this is not synthesized.

**Question 19, 20**
Both of these questions apply to the following code implemented exactly as described.

```verilog
always@(posedge clock)
  begin
    A <= C;
    B <= F;
    E <= H;
  end
assign C = A;
assign F = B | E;
assign G = F & C;
assign H = G | A;
```

Each gate has a delay from its input to output as given as \{1 : 2 : 3\}, \( t_{Cp_Q} \) is \{2 : 3 : 5\} the clock skew is 1 ns, the flip-flop setup time is 1 ns and the hold time 1 ns. Format above is \{min : typical : max\}.

**Question 19**
What is the fastest possible viable clock period?

16 ns

**Question 20**
Is there potential for a hold violation in this logic? With what margin?

No 0 ns