Quiz 4: Lectures 14 to 18

EEE4084F

2015-05-12

Instructions:

- Answer on a separate page.
- Make sure that your student number is on all your answer pages.
- There are 5 questions, each divided into sub-questions. Answer all questions.
- Total time: 45 minutes.
- Total marks: 35.

Question 1: Reconfigurable Computing **[10 Total]**

#1 Briefly explain what is meant by "reconfigurable computing". **[4]**

#2 Name one example where you would recommend using FPGA-based reconfigurable computing, rather than general-purpose GPU, in order to speed up the application. Explain why you would prefer reconfigurable computing in this example. **[6]**

Question 2: Portability **[6 Total]**

#1 Select the most accurate definition of what is meant by "making a program portable": **[2]**

(a) Any executable of the program can run directly on any architecture.

(b) It can be compiled and run, with little or no alteration, on another computer system.

(c) The program is designed around use for only one architecture.

(d) The platform that the program is designed to run on is designed to be easy to move about.

#2 Name two advantages and two disadvantages of making use of third-party libraries in order to gain portability. **[4]**

Question 3: Parallel and Distributed Systems **[4 Total]**

#1 Consider a parallel computing system with 12 processing nodes. The system is used to implement an algorithm that has 85% parallelisable code. What is the maximum achievable benefit of using this 12-processing node system above a purely serial version. **[2]**

Amdahl's Law: $$P = \frac{1}{S + \frac{1-S}{N}}$$

#2 The following acronyms relate to client / server protocols. Which one is incorrectly defined? **[1]**

(a) RPC = Remote Procedure Call

(b) RMI = Request Method Indirectly

(c) SQL = Standard Query Language

#3 Write out the correct definition for it. **[1]**

#4 **Bonus mark:** The CORBA software architecture defines a distributed-object standard, where objects residing on different machines can pass data between each other. The standard is full of obscure acronyms such as GIOP, IDL, ORB, and the POA. One of these is especially important, as it is the layer that enables objects on different platforms to communicate with each other. Which one of these is it? **[1]**

(a) GIOP (b) IDL (c) ORB (d) POA (e) OBV (f) SSLIOP

Question 4: Verilog **[10 Total]**

Consider the two code extracts below:

```
// Option A                         // Option B
reg  A, B;                          reg  A, B;
wire C, Clk;                        wire C, Clk;

always @(posedge Clk) begin         always @(posedge Clk) begin
 A = B & C;                          A <= B & C;
 B = A | C;                          B <= A | C;
end                                 end
```

#1 Draw the circuit (gates and registers, not logic element internals) produced by each of the options above. **[4]**

#2 Use your circuits to explain the difference between blocking and non-blocking assignments (option A is blocking). **[2]**

#3 Assume that all the registers are initialised to 0. Further assume that the wire C is connected to an external button that goes high some time between positive clock edges 2 and 3; and then low again some time between positive clock edges 6 and 7. Draw the timing diagram, of option B above, for the first 10 clock cycles. Include the signals A, B, C and Clk. You may assume a slow clock (i.e. neglect timing delays and requirements). **[4]**

Question 5: Timing Calculations **[5 Total]**

Consider the Verilog code extract below:

```
wire [17:0]a, b, c;
reg  [17:0]A, B;
reg  [36:0]C, Y;

always @(posedge Clk) begin
 A <=  a;
 B <=  b;
 C <= {c, {19{c[0]}}};

 Y <= A * B + C;
end
```

The registers have a clock-edge to output-valid delay of 3 ns, a setup requirement of 2 ns and a hold requirement of 1 ns. The multiplier has a propagation delay of 22 ns and the adder has a propagation delay of 13 ns. What is the maximum clock frequency that this circuit can run at? You may neglect detail such as clock skew and other parasitic effects. **[5]**

Verilog Reference

Comments

```
// One-liner
/* Multiple
   lines */
```

Numeric Constants

```
// The 8-bit decimal number 106:
8'b_0110_1010 // Binary
8'o_152       // Octal
8'd_106       // Decimal
8'h_6A        // Hexadecimal
"j"           // ASCII

78'bZ         // 78-bit high-impedance
```

Too short constants are padded with zeros on the left. Too long constants are truncated from the left.

Nets and Variables

```
wire [3:0]w; // Assign outside always blocks
reg  [1:7]r; // Assign inside always blocks
reg  [7:0]mem[31:0];

integer j; // Compile-time variable
genvar  k; // Generate variable
```

Parameters

```
parameter  N     = 8;
localparam State = 2'd3;
```

Assignments

```
assign Output = A * B;
assign {C, D} = {D[5:2], C[1:9], E};
```

Operators

```
// These are in order of precedence...
// Select
A[N] A[N:M]
// Reduction
&A ~&A |A ~|A ^A ~^A
// Compliment
!A ~A
// Unary
+A -A
// Concatenate
{A, ..., B}
// Replicate
{N{A}}
// Arithmetic
A*B A/B A%B
A+B A-B
// Shift
A<<B A>>B
// Relational
A>B A<B A>=B A<=B
A==B A!=B
// Bit-wise
A&B
A^B A~^B
A|B
// Logical
A&&B
A||B
// Conditional
A ? B : C
```

Module

```
module MyModule
#(parameter N = 8) // Optional parameter
 (input  Reset, Clk,
  output [N-1:0]Output);
 // Module implementation
endmodule
```

Module Instantiation

```
// Override default parameter: setting N = 13
MyModule #(13) MyModule1(Reset, Clk, Result);
```

Case

```
always @(*) begin
 case(Mux)
  2'd0: A = 8'd9;
  2'd1,
  2'd3: A = 8'd103;
  2'd2: A = 8'd2;
  default:;
 endcase
end

always @(*) begin
 casex(Decoded)
  4'b1xxx: Encoded = 2'd0;
  4'b01xx: Encoded = 2'd1;
  4'b001x: Encoded = 2'd2;
  4'b0001: Encoded = 2'd3;
  default: Encoded = 2'd0;
 endcase
end
```

Synchronous

```
always @(posedge Clk) begin
 if(Reset) B <= 0;
 else      B <= B + 1'b1;
end
```

Loop

```
always @(*) begin
 Count = 0;
 for(j = 0; j < 8; j = j+1)
  Count = Count + Input[j];
end
```

Function

```
function [6:0]F;
 input [3:0]A;
 input [2:0]B;
 begin
  F = {A+1'b1, B+2'd2};
 end
endfunction
```

Generate

```
genvar j;
wire [12:0]Output[19:0];

generate
 for(j = 0; j < 20; j = j+1)
 begin: Gen_Modules
  MyModule #(13) MyModule_Instance(
   Reset, Clk,
   Output[j]
  );
 end
endgenerate
```

State Machine

```
reg   [1:0]State;
localparam Start = 2'b00;
localparam Idle  = 2'b01;
localparam Work  = 2'b11;
localparam Done  = 2'b10;

reg tReset;

always @(posedge Clk) begin
 tReset <= Reset;

 if(tReset) begin
  State <= Start;

 end else begin
  case(State)
   Start: begin
    State <= Idle;
   end
   Idle: begin
    State <= Work;
   end
   Work: begin
    State <= Done;
   end
   Done: begin
    State <= Idle;
   end
   default:;
  endcase
 end
end
```