Test 2: Lectures 6 to 17

EEE4084F

2016-05-10

Instructions:

- Answer on a separate page.
- Make sure that your student number is on all your answer pages.
- There are 4 questions, each divided into sub-questions. Answer all questions.
- Total time: 35 minutes.
- Total marks: 35.

Question 1: CPU Architectures [8 Total]

(a) What are some of the advantages of distributed memory over shared memory? Name at least two benefits. **[2]**

(b) Briefly explain what is meant by instruction level parallelism (ILP). **[2]**

(c) What is meant by the concept of super-pipelining in regards to ILP? Indicate two potential challenges of designing a processor architecture to support super-pipelining. **[4]**

Question 2: Reconfigurable Computing [10 Total]

(a) Briefly define what is meant by a digital accelerator (i.e. the type of thing we are trying to prototype in the YODA project). **[2]**

(b) Discuss at least two advantages and two disadvantages of using an FPGA-based reconfigurable computing approach to implement a digital accelerator, instead of a more standardised multi-processor CPU-based approach (eg. using a cluster of Intel PC's running MPI)? **[5]**

(c) Briefly explain the difference between a PLA, a CPLD and an FPGA. Don't go into too much detail regarding the internal circuitry – a brief overview of the main differences, in three sentences, is sufficient. **[3]**

(d) **Bonus mark:** Which company manufactures the small-package and low-power IGLOO FPGA. **[1]**

Question 3: Cell Processors [4 Total]

(a) What is the difference between the PPE and the SPE in a cell processor? **[1]**

(b) Are there more of the one than the other? **[1]**

(c) Would you say the interconnection bus is statically configured in connecting the SPE's, or more controllable? Briefly elaborate. **[2]**

Question 4: Verilog [13 Total]

(a) Draw the circuit described by the Verilog code below. Don't draw the full gate-level circuit – rather make use of multi-bit registers, multi-bit multiplexers and high-level blocks such as `Add one` or `Shift right`. **[8]**

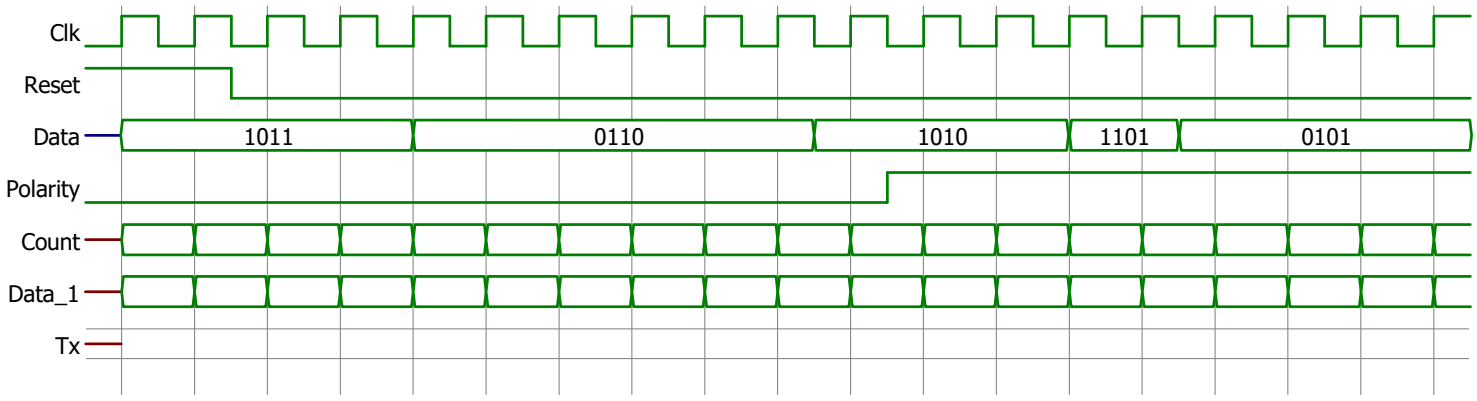
```
module Shifter(input Clk, input Reset, input [3:0]Data, input Polarity, output reg Tx);

reg [3:0]Data_1; reg [1:0]Count;

always @(*) Tx <= Data_1[0] ^ Polarity;

always @(posedge Clk) begin
 if(Reset) begin
  Count <= 0;
 end else begin
  Count <= Count + 1'b1;
  if(|Count) Data_1 <= {1'b0, Data_1[3:1]};
  else       Data_1 <= Data;
 end
end
endmodule
```

(b) Draw the timing diagram of all the signals in the Verilog above. Use the timing diagram skeleton below (i.e. draw on the question paper). Use `X` to mean "unknown". **[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
```