Computer-Aided VLSI System Design
Midterm Examination
Nov. 12, 2010

Your Name _____

Student ID Number _____

Instructions

Exams: Consultation during the exam is not permitted. This is not an open book exam. The exam is to be completed in one and half hours. If you need scratch paper, just use the blank parts of these pages; show all of your work on these pages. Before you start writing, please check if you have all 16 pages of the exam.

Regrading Policy: Exams will be accepted for regrading up to two weeks after you get the graded exam. No regrades after two weeks.

Please sign the following statement upon completing this exam:

I certify that I will follow the above instructions. I have neither received nor given unpermitted aid on this examination.

Your signature _____

********** Score Board (to be filled by graders) **********

	Total Point	Your Point
Problem 1	10	
Problem 2	15	
Problem 3	15	
Problem 4	15	
Problem 5	10	
Problem 6	5	
Problem 7	5	
Problem 8	10	
Problem 9	10	
Total	95	

Problem 1, (10%), <Basic Concept 1>

The questions of (3), (4), and (5) have multiple choices. The points can be given only when all the choices are correct in each question.

B (1) (2%) ___ Which one is used as the non-blocking assignment?

(a) X = A + B; (b) X <= A + B.

A (2) (2%) ___ Which one is the flip-flop with a synchronous reset?

```
module dff_sync_clear(d, clearb,
clock, q);
input d, clearb, clock;
output q;
reg q;
always @ (posedge clock)
begin
  if (!clearb) q <= 1'b0;
  else q <= d;
end
endmodule
```

(a)

```
module dff_async_clear(d, clearb, clock, q);
input d, clearb, clock;
output q;
reg q;

always @ (negedge clearb or posedge clock)
begin
  if (!clearb) q <= 1'b0;
  else q <= d;
end
endmodule
```

(b)

BE (3) (2%) _____ What are the combinational circuits in the follows?

(a) SRAM; (b) AND gate; (c) Registers; (d) Latch; (e) OR gate;

CE (4) (2%) _____ What tools are ***not*** used in the front-end design part of the cell-based design flow?

(a) Design Compiler; (b) NC Verilog; (c) Calibre; (d) Nlint; (e) SoC Encounter; (f) Debussy.

CE (5) (2%) _____ What codes are not synthesizable?

(a) assign Y=10'd5; (b) assign Y=10'h0a3; (c) assign Y=10'dx;

(d) always@(*) begin
Y = 10'd5; end

(e) initial begin
Y = 10'd5; end

Problem 2, (15%), <Basic Concept 2>

(1) (5%) Please give the correct order of the following steps in the cell-based design flow: (a) DRC & LVS; (b) DFT insertion; (c) RTL coding; (d) Place & route; (e) Tape out; (f) Synthesis; (g) Hardware specification.

g -> c -> f -> b -> d -> a -> e

(2) (2%) What is the setup time violation? (2%) Describe at least two possible ways to fix it in the whole cell-based design flow. The way may be the modification on the architecture level or setup different constraints for the design tools.

1. 在 clock edge 之前，register 的 input 應該要穩定一段時間，讓 register 可以抓到正確的值，此時間爲 setup time，若此時間不夠長，register 即無法抓到正確的值，此爲 setup time violation。
2. A.在 critical path 上切 pipeline; B. 把 operation frequency 的設定調慢一點;

(3) (2%) The latch is not usually expected in a basic digital design. What may be the reason if there is an unexpected latch reported during the logic synthesis?

Case 沒寫滿，或是 if 沒加 else。

(4) (2%) What is the meaning of the warning information about the combinational loop when we use the NLint to check the RTL code?

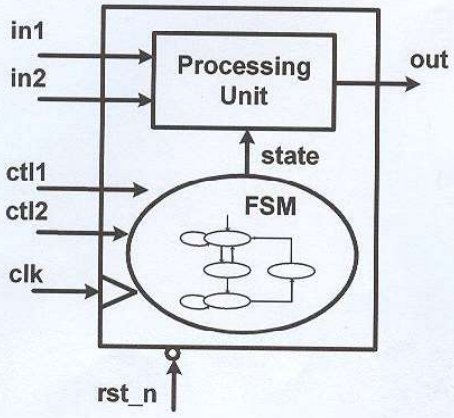
當 combinational 的 logic 串成一個 loop，中間沒有用 non-combinational 的電路隔開的話，即會形成一個 combinational loop。如果 synthesis tool 沒有對這些 loop 做特殊的設定，即無法做 synthesis。

(5) (2%) What is the memory BIST ?

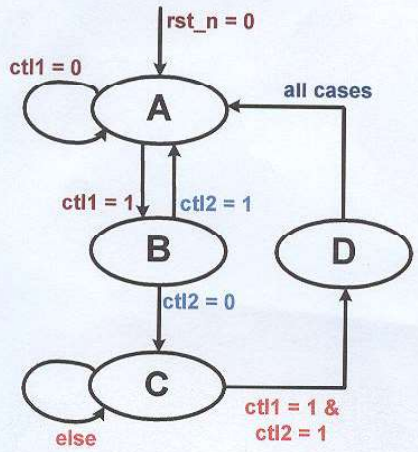
Build in self test for memory。將一般的 memory 包一個 test wrapper，其中會有原本的 SRAM 和一個 test control module (TCM)。在 Test Mode 下，TCM 會產生 test pattern 打入 SRAM，並比對讀出的結果是否正確，來驗證 SRAM 是否有正常工作。

Problem 3, (15%), <Verilog Coding and Finite State Machine>

Please complete the verilog code of the hardware below. The hardware has a finite state machine (FSM) along with a processing unit (PU). The hardware block diagram is shown in (a). The flow chart of the FSM is shown in (b). The behavior of the PU is shown in (c).



(a) Block diagram of the hardware



(b) Flow chart of the finite state machine(FSM)

State	Output
A	in1 + in2
B	in1 - in2
C	in1
D	in2

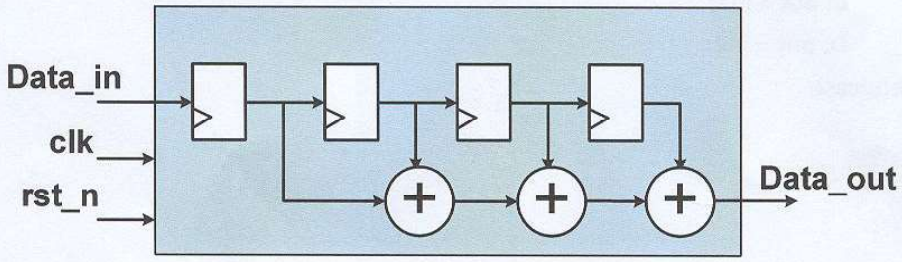
(c) Behavior of the processing unit

```
module MyHardware(clk, rst_n, in1, in2, ctl1, ctl2, out);
input clk, rst_n;
output [3:0] out;
input [3:0] in1, in2;
input ctl1, ctl2;
parameter A=2'b00, B=2'b01; C=2'b10; D=2'b11;

reg [1:0] state; //current state
reg [1:0] next_state; //next state logic output
reg [3:0] out;

//next state logic (combinational circuits)
always@( * )
begin

   case(state)
        A: next_state = (ctl1==1'b1) ? B : A;
        B: next_state = (ctl2==1'b1) ? A : C;
        C: next_state = (ctl1==1'b1 && ctl2==1'b1) ? D : C;
        D: next_state = A;
   endcase

end
```

```
//processing unit and output logic   (combinational circuits)
always@( * )
begin

   case(state)
        A: out = in1 + in2;
        B: out = in1 - in2;
        C: out = in1;
        D: out = in2;
   endcase









      endcase
end

//state   (non-combinational circuits)
always@(posedge clk )
begin
   if(rst_n = 1'b0)
   begin
      state <= 2'd0;
   end
   else
   begin
      state <= next_state;
   end
end
endmodule
```

Problem 4, (15%), <Verilog Debugging and Simulation>

(1) (10%) Here shows the block diagram of the hardware unit and the corresponding verilog code. However, there are **10 types of different errors** in the verilog code. Please find the bugs and try to fix them. You may circle out the bugs and write the correct codes beside them.

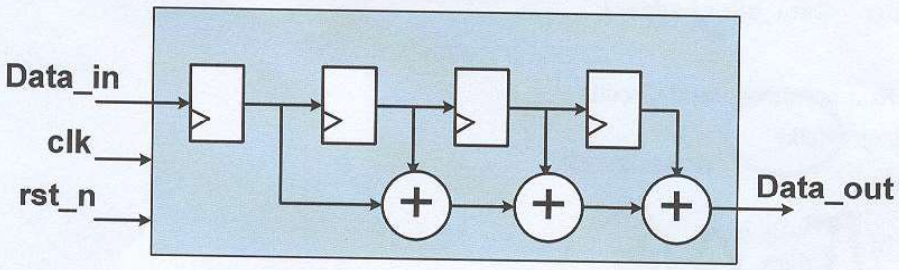


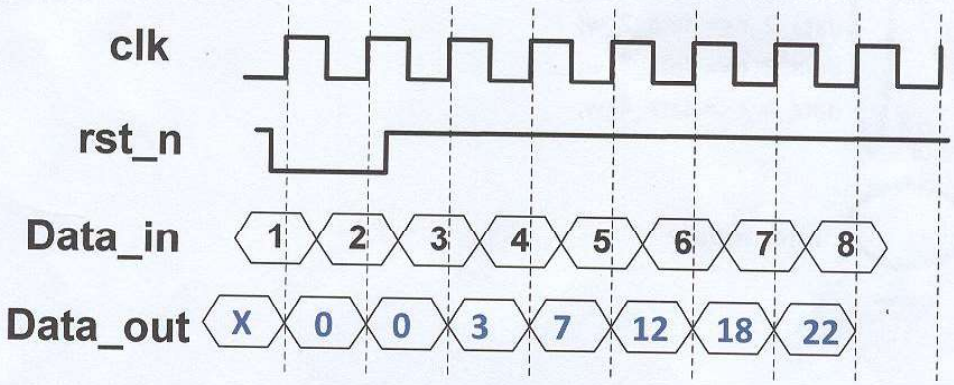
Verilog code with bugs

```
module MyHardware(clk, rst_n, Data_in, Data_out) ;
Input clk, rst_n;
input   [3:0]   Data_in;
output  [6:0]   Data_out;
[6:0]
wire    Data_out;
reg     [3:0]   data_1_r, data_2_r, data_3_r, data_4_r;
reg     [3:0]   data_1_w, data_2_w, data_3_w, data_4_w;
reg     [4:0]   adder1;
reg     [5:0]   adder2;
reg     [5:0]   adder3;
wire    [6:0]
// Shift Register Array
assign  data_1_w <= Data_in;
assign  data_2_w <= data_1_r;
assign  data_3_w <= data_2_r;
assign  data_4_w <= data_3_r;
=
```

```
// Adder Tree

assign   adder1 <= data_1_r + data_2_r;
assign   adder2 <= adder1 + data_3_r;
assign   adder3 <= adder2 + data_4_r;
assign   Data_out <= adder3;

// Non-combinational Circuits
always@(clk)
begin
    if(!rst_n)
begin
        data_1_w <= 3'd0;
        data_2_w <= 3'd0;
        data_3_w <= 3'd0;
        data_4_w <= 3'd0;
end
    else
begin
        data_1_r <= data_1_w;
        data_2_r <= data_2_w;
        data_3_r <= data_3_w;
        data_4_r <= data_4_w;
end
end

endmodule
```

(2) (5%) Here shows the waveform of the above hardware unit. The test pattern of "Data_in" signals are given. Please fill in the result of the "Data_out" signals. Note that "X" can be used if the signal should be unknown. Note that the synchronous reset is used in this design, and the registers are set to zeros after the reset.





d1	0	0	3	4	5	6	7	8
d2	0	0	0	3	4	5	6	7
d3	0	0	0	0	3	4	5	6
d4	0	0	0	0	0	3	4	5
out	0	0	3	7	12	18	22	26

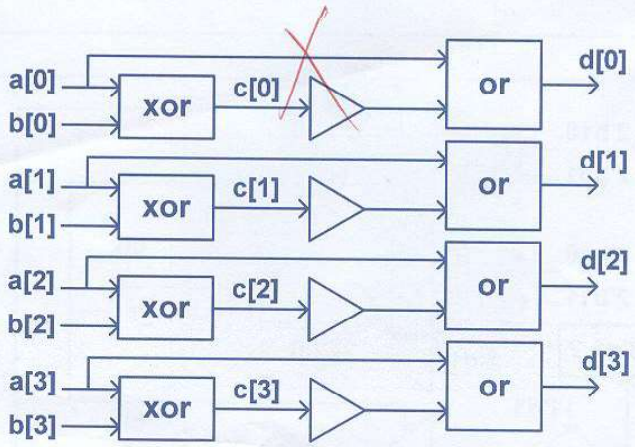
Problem 5, (10%), <Manually Synthesis from Verilog RTL>

Given the Verilog RTL code, please draw the corresponding block diagram. You may use registers, adders, subtraction units, MUX unit, and other logic gates if required in the block diagram.

(a) (5%) Verilog RTL

```
reg [3:0] a;
reg [3:0] b;
reg [3:0] c;
wire [3:0] d;
integer i;
always@(*)
   for(i= 0; i<=3; i=i+1)
      d[i] = (~c[i]) | a[i];
end
always@(*)
   for(i= 0; i<=3; i=i+1)
      c[i] = b[i] ^ a[i];
end
```

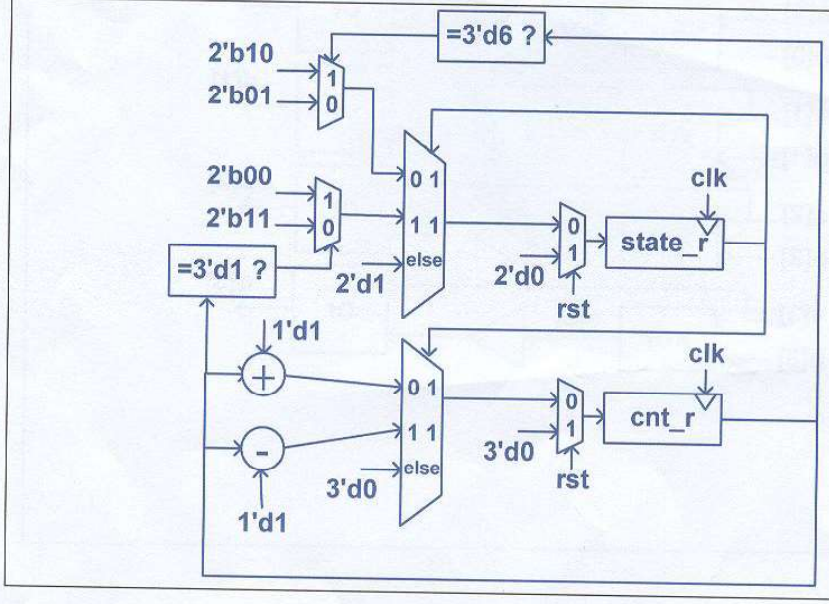
Circuit



(b) 5(%) Verilog RTL

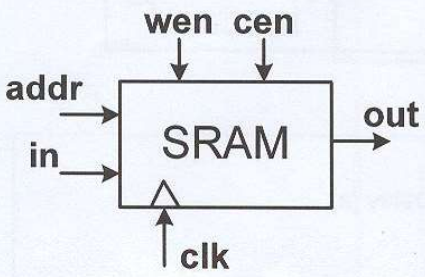
```
always@(*)
begin
 case(state_r)
  2'b01: begin cnt_w = cnt_r + 1'b1; state_w = (cnt_r == 3'd6)? 2'b10 : 2'b01; end
  2'b10: begin cnt_w = cnt_r – 1'b1; state_w = (cnt_r == 3'd1)? 2'b00 : 2'b10; end
  default: begin cnt_w = 3'd0; state_w = 2'b01; end
 endcase
end

always@(posedge clk)
begin
  if(rst)
  begin
    state_r <= 2'b00;
    cnt_r   <= 3'd0;
  end
  else
  begin
    state_r <= state_w;
    cnt_r   <= cnt_w;
  end
end
```

Circuit

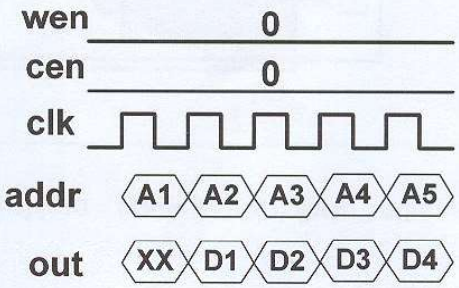


Problem 6, (5%), <Memory Usage>

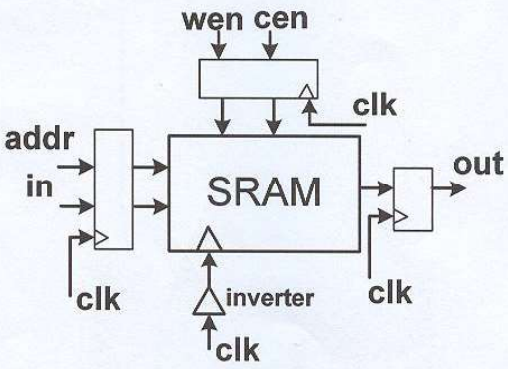
An SRAM module and the corresponding waveform for the reading operation are shown in (a) and (b) respectively. "A5" indicates the address of 5 in the memory. "D5" means the data read out from the address 5. Note that "wen" and "cen" are write-enable-not and chip-enable-not signals, and are set to zeros during the reading operation. For some reasons, an engineer uses this memory module with the configuration shown in (c). Please fill in the output of this memory configuration for the memory reading operation in (d). You may use "D1", "D2",...., or "D9" to fill in.



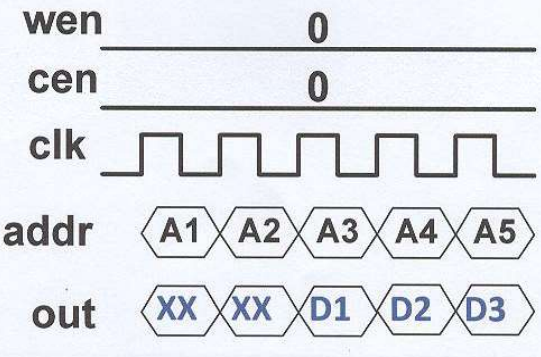
(a)



(b)



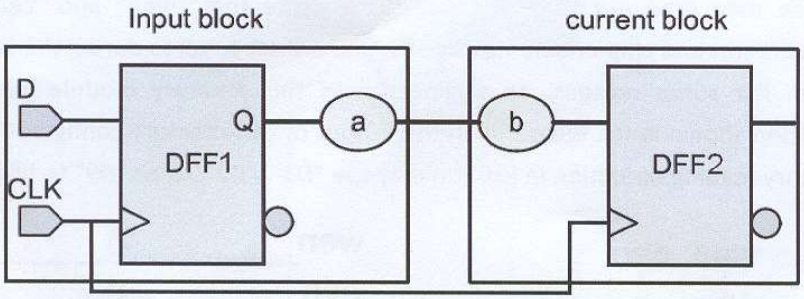
(c)



(d)

Problem 7, (5%), <Input Delay>

Please specify the input delay of the current block in the following circuit.



Input_delay (current block) = DFF_1 (clk->Q) + Delay (a)

Problem 8, (10%), <Logic Synthesis>

Before logic synthesis using design compiler, some files are needed. Some are for the tool, some are for the design, and some are for the library. After logic synthesis, some files are also outputted as the reports or for the following backend processing. In the following, there is a list of files. Please describe which file is used as the input files of logic synthesis, and which file is the output file (3%). In addition, please briefly describe the purposes of these files (7%).

(a) MyHardware_TimeInfo.sdf
(b) MyHardware_GateLevelCode.vg
(c) SRAM64x8_fast_syn.lib
(d) .synopsys_dc.setup
(e) TXMC_CellLib_Typical.db
(f) MyHardware_VerilogCode.v
(g) MyHardware_ TimeConstrain.sdc

1. input: c, d, e, f, g
   output: a, b, g
   Ps: g can be either input or out.

2.
(a) Timing information of the synthesized gate-level code
(b) Synthesized gate-level code
(c) SRAM best-case library, need to be converted into db file for DC
(d) DC initial file, indicate the path and libraries that are needed during the synthesis. The naming rule can be defined here as well.
(e) Standard cell data base file for the typical case.
(f) Verilog code that is needed to be synthesized
(g) Timing constraint file, can be used as the input for synthesis or or the output file for the following P&R step.

Problem 9, (10%), <Design for Testing>

Apply D-algorithm to find out the test pattern for the "stuck-at-1" fault at node N. Show the process of the testing flow using this test pattern. Please fill in the input, output, and middle results in the blank square of the circuit.

Hint for D Algorithm:

1. Target a specific stuck-at fault.
2. Drive fault site to opposite value.
3. Propagate error to primary output.
4. Record pattern; go to next fault.