컴퓨터공학 기초 실험2

Lab #6

Counter & Shifter

Counter

- Counter
  - 펄스신호에 따라 어떤 정해진 순서대로 상태의 변이가 진행되는 레지스터를 counter라 한다. Counter는 어떤 사건이 발생할 때마다 펄스신호를 만들어 그 사건의 발생횟수를 세는 등에 사용된다.
  - 설계할 counter는 8-bit loadable up/down counter이고, 다음과 같은 control signal을 입력으로 받는다.

Signal	Description
reset_n	Active low에 동작하는 reset signal로 register 값을 0으로 초기화시킨다.
load	입력된 데이터를 register 값으로 load한다.
inc	Counter의 증가, 감소를 제어하는 신호로, 1일 경우에는 가산, 0일 경우에는 감산을 수행한다.

  - Control signal 간의 우선 순위는 reset_n, load, inc이다.

Shifter

- Shifter
  - Shifter는 register에 저장되어 있는 정보를 단방향이나 양방향으로 이동시킬 수 있는 하드웨어이다.
  - 설계할 shifter는 8-bit loadable shifter이고, 다음과 같은 control signal을 입력으로 받는다.

Signal	Description
reset_n	Active low에 동작하는 reset signal로 register 값을 0으로 초기화
op	Shift를 시키기 위한 명령어로써 다음의 명령어를 가진다. - NOP : No operation(현재 register의 값을 그대로 출력) - Load : 입력된 data를 출력 - LSL : Logical shift left를 수행 - LSR : Logical shift right를 수행 - ASR : Arithmetic shift right를 수행 (ASL은 LSL과 동작이 같기 때문에 구현하지 않는다.)
shamt	Shift amount로2 bit 값을 가진다.

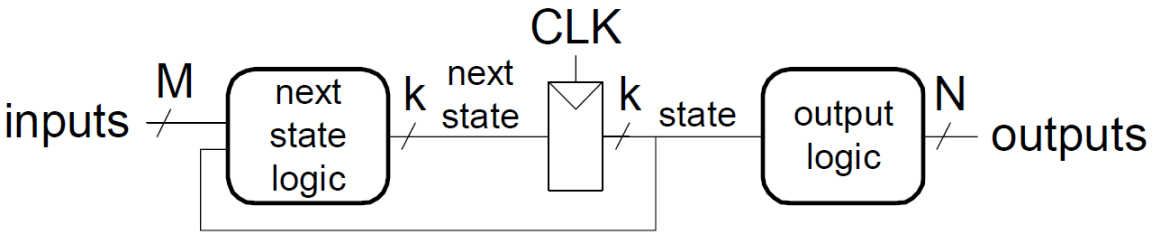
A 5-way Counter

PRACTICE I

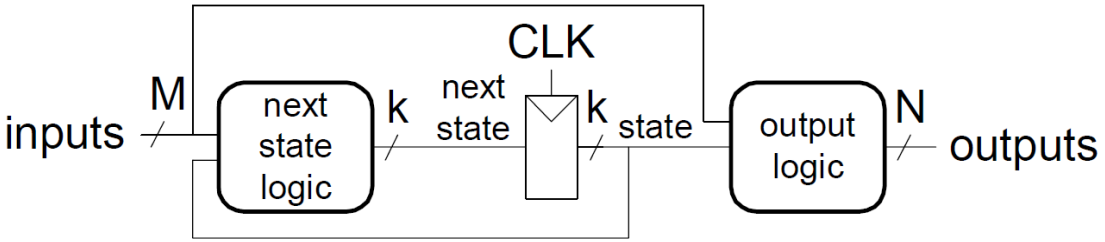
Finite State Machine

- 정의
  - FSM 모델은 시스템의 동작을 상태(state)와 상태간의 천이(transition)로 표현한다. FSM은 동작방식에 따라 Moore machine과 Mealy machine으로 구분된다. Moore machine은 출력이 단지 현재 상태에 의해서 결정되는 회로이며, Mealy machine은 현재 상태와 입력에 의해 출력이 결정되는 회로이다.

- Moore FSM vs. Mealy FSM
  - Moore FSM



  - Mealy FSM



Finite State Machine

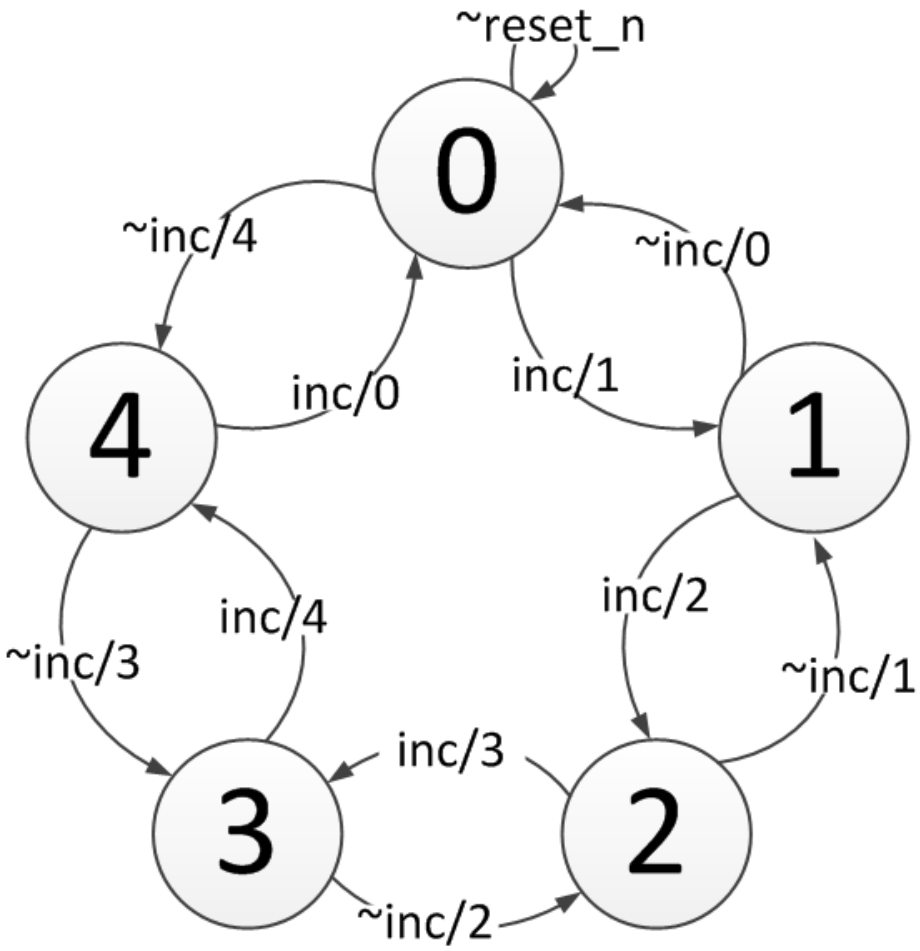
- Design
  1. Drawing the finite state diagram
     - Define states
     - Define inputs
     - Define outputs
     - Draw the diagram
  2. Encoding states
  3. Coding the module header
  4. Coding state registers(flip-flops) – sequential circuits
  5. Coding combinational circuits

A 5-way Counter (1/6)

- New Project Wizard
  - Project name : cnt5
  - Top module name : cnt5
  - Family & Device : Cyclone V 5CSXFC6D6F31C6 (밑에서 6번째)
- Verilog file
  - Add files : -
  - New files : cnt5.v tb_cnt5.v

A 5-way Counter (2/6)

- Design a 5-way counter
  - ✓ Increments when input inc is high and
  - ✓ Decrements when inc is low
- Drawing state diagram
  - ✓ Define states
    - zero, one, two, three, four
  - ✓ Define inputs
    - inc
  - ✓ Define output
    - The current state
  - ✓ Draw the diagram



A 5-way Counter (3/6)

- Encoding states

Binary encoding	One-hot encoding

zero = 3'b000;
one = 3'b001;
two = 3'b010;
three = 3'b011;
four = 3'b100;
	zero = 5'b00001;
	one = 5'b00010;
	two = 5'b00100;
	three = 5'b01000;
	four = 5'b10000;

- Coding module header

```
module cnt5(cnt, clk, reset_n, inc);
  input          clk, reset_n, inc;
  output [2:0]   cnt;

  // Encoded state
  parameter zero       = 3'b000;
  parameter one        = 3'b001;
  parameter two        = 3'b010;
  parameter three      = 3'b011;
  parameter four       = 3'b100;
```

A 5-way Counter (4/6)

- Coding sequential circuits

```
// Sequential circuit part
reg [2:0]       cnt;
reg [2:0]       next_cnt;

always @ (posedge clk or negedge reset_n)
begin
  if (reset_n == 1'b0)          cnt <= zero;
  else                          cnt <= next_cnt;
end
```

A 5-way Counter (5/6)

- Coding combinational circuits

```
  // Combinational circuit part
  always @ (inc or cnt)
  begin
    case({cnt, inc})
    {zero, 1'b0}       :     next_cnt <= four;
    {zero, 1'b1}       :     next_cnt <= one;
    {one, 1'b0}        :     next_cnt <= zero;
    {one, 1'b1}        :     next_cnt <= two;
    {two, 1'b0}        :     next_cnt <= one;
    {two, 1'b1}        :     next_cnt <= three;
    {three, 1'b0}      :     next_cnt <= two;
    {three, 1'b1}      :     next_cnt <= four;
    {four, 1'b0}       :     next_cnt <= three;
    {four, 1'b1}       :     next_cnt <= zero;
    default            :     next_cnt <= 3'bx;
    endcase
  end
endmodule
```

A 5-way Counter (6/6)

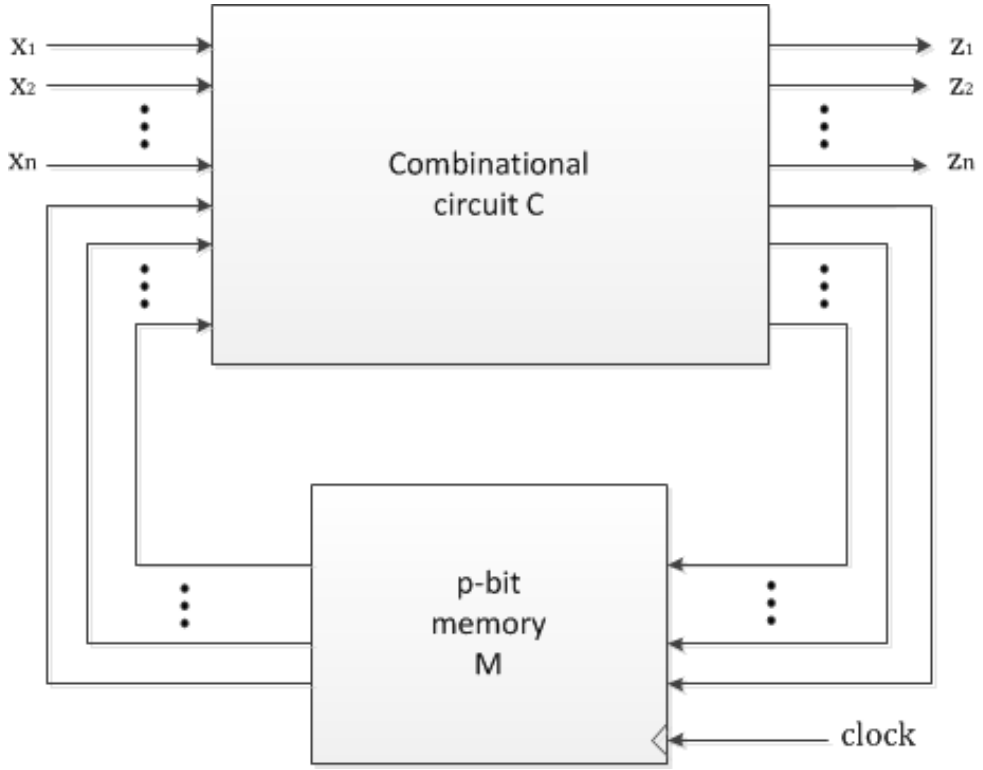
- Verification
  - Testbench를 직접 작성 후 검증 후 보고서에 삽입하기 바랍니다.
- RTL Viewer
  - 확인 후 보고서에 삽입하기 바랍니다.
- Flow Summary
  - 확인 후 보고서에 삽입하기 바랍니다.

An 8-bit loadable shifter

PRACTICE II

Sequential logic

- 조합회로는 게이트들로만 이루어졌지만, 순차회로는 게이트뿐만 아니라, 플립플롭까지 포함하고있어 회로에 저장능력이 있는 회로를 뜻한다. 즉, 순차회로의 출력은 현재의 입력 값과 현재 플립플롭에 저장되어 있는 값에 의해 결정된다.



8-bit Loadable Shifter

- New Project Wizard
  - Project name : shifter8
  - Top module name : shifter8
  - Family & Device : Cyclone V 5CSXFC6D6F31C6 (밑에서 6번째)
- Verilog file
  - Add files : gates.v, mx2.v
  - New files : mx4.v, LSL8.v, LSR8.v, ASR8.v, cc_logic.v,
    _dff_r.v, _register8_r.v, shifter8.v, tb_shifter8.v

4-to-1 Multiplexer

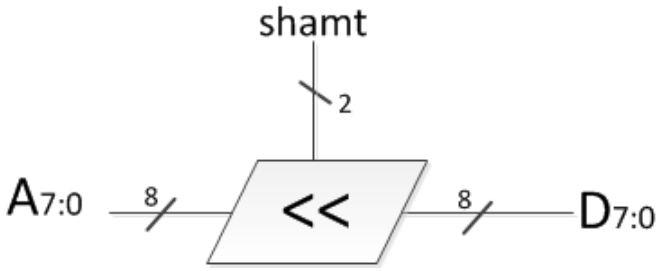
- 1-bit 2-to-1 multiplexer를 사용하여 1-bit 4-to-1 multiplexer를 구현
  - 구현한 multiplexer를 이용하여 logical shifter, arithmetic shifter를 구현

Select(sel)	Output(y)
2'b00	d0
2'b01	d1
2'b10	d2
2'b11	d3

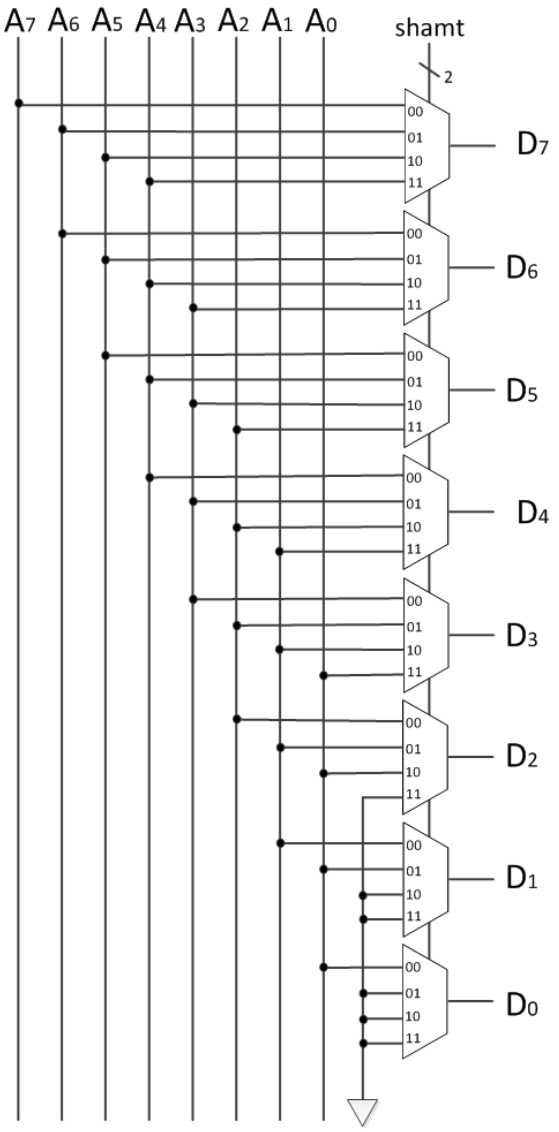
```
module mx4(y, d0, d1, d2, d3, s);
  input          d0, d1, d2, d3;
  input [1:0]    s;
  output         y;

  Instance of mx2

endmodule
```

Logical Shift Left

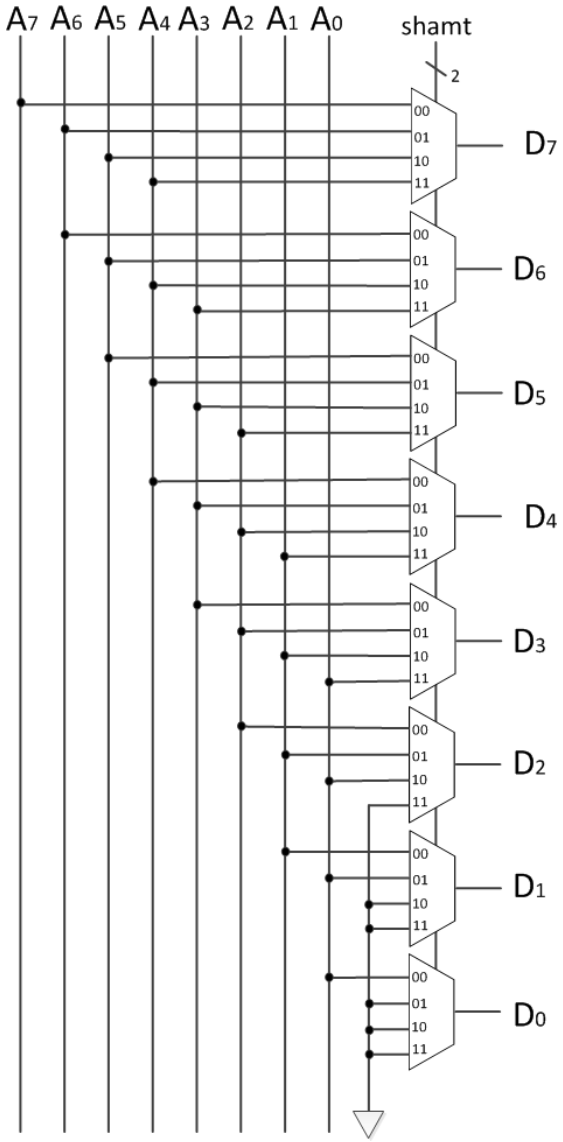
- Logical shift left
  - Register를 shift amount만큼 왼쪽으로 shift시킨 후, 빈 공간을 0으로 채운다.



Symbol



Logical Shift Left - Implementation



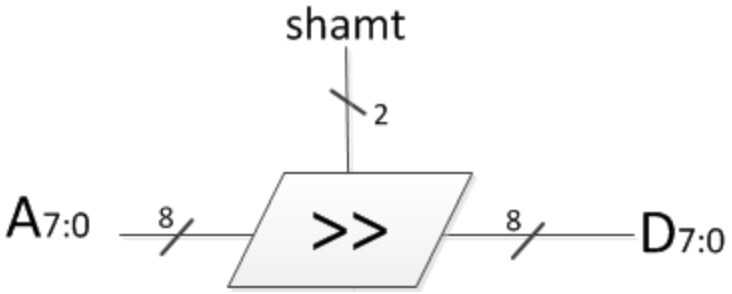

```
module LSL8(d_in, shamt, d_out);
  input   [7:0]   d_in;
  input   [1:0]   shamt;
  output  [7:0]   d_out;
```

Instance of 4-to-1 multiplexers

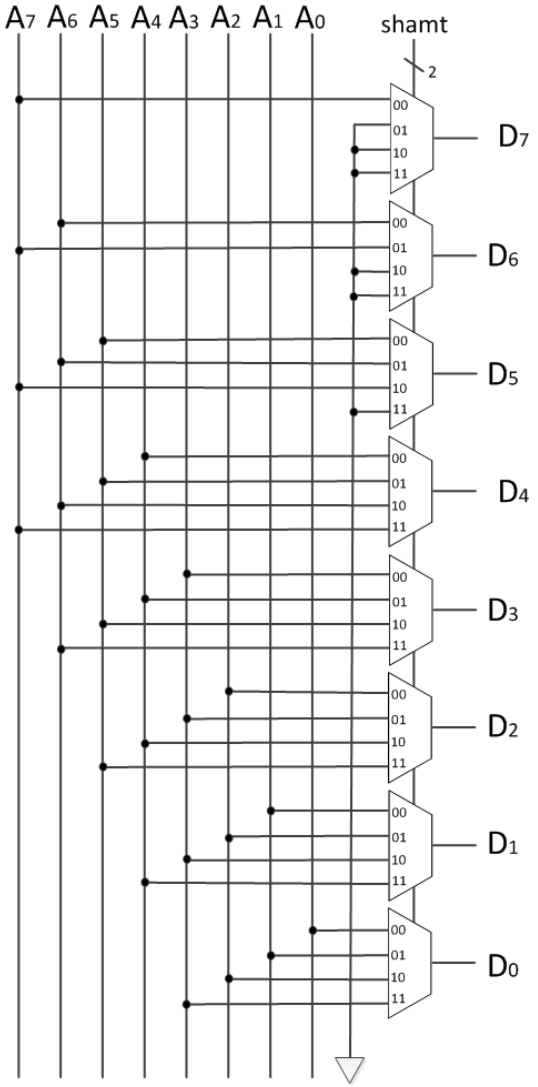
```
endmodule
```

Logical Shift Right

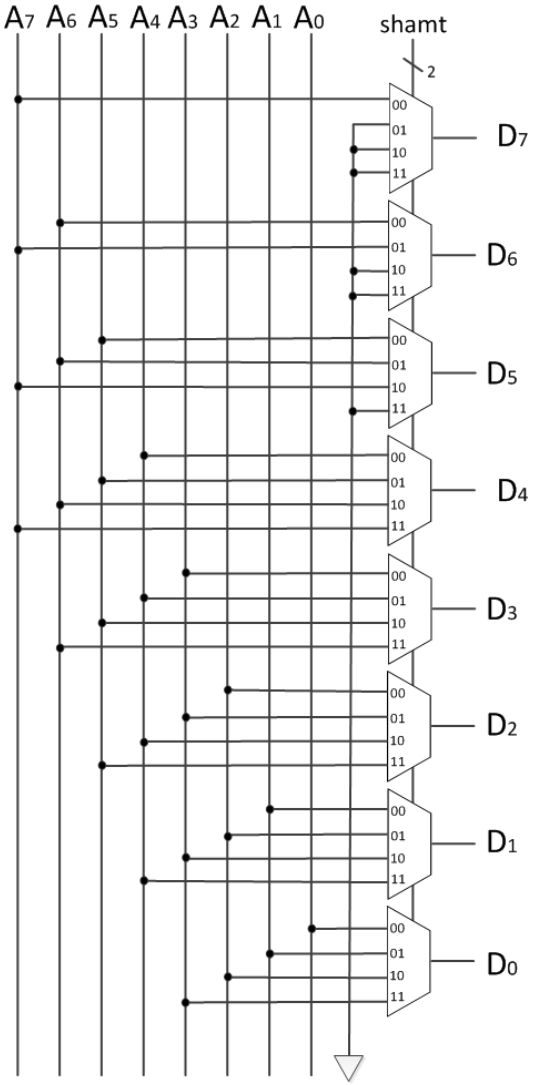
- Logical shift right
  - Register를 shift amount만큼 오른쪽으로 shift시킨 후, 빈 공간을 0으로 채운다.



Symbol



Logical Shift Right - Implementation




```
module LSR8(d_in, shamt, d_out);
  input   [7:0]   d_in;
  input   [1:0]   shamt;
  output  [7:0]   d_out;
```

Instance of 4-to-1 multiplexers

```
endmodule
```

FSM Design – Drawing the finite state diagram

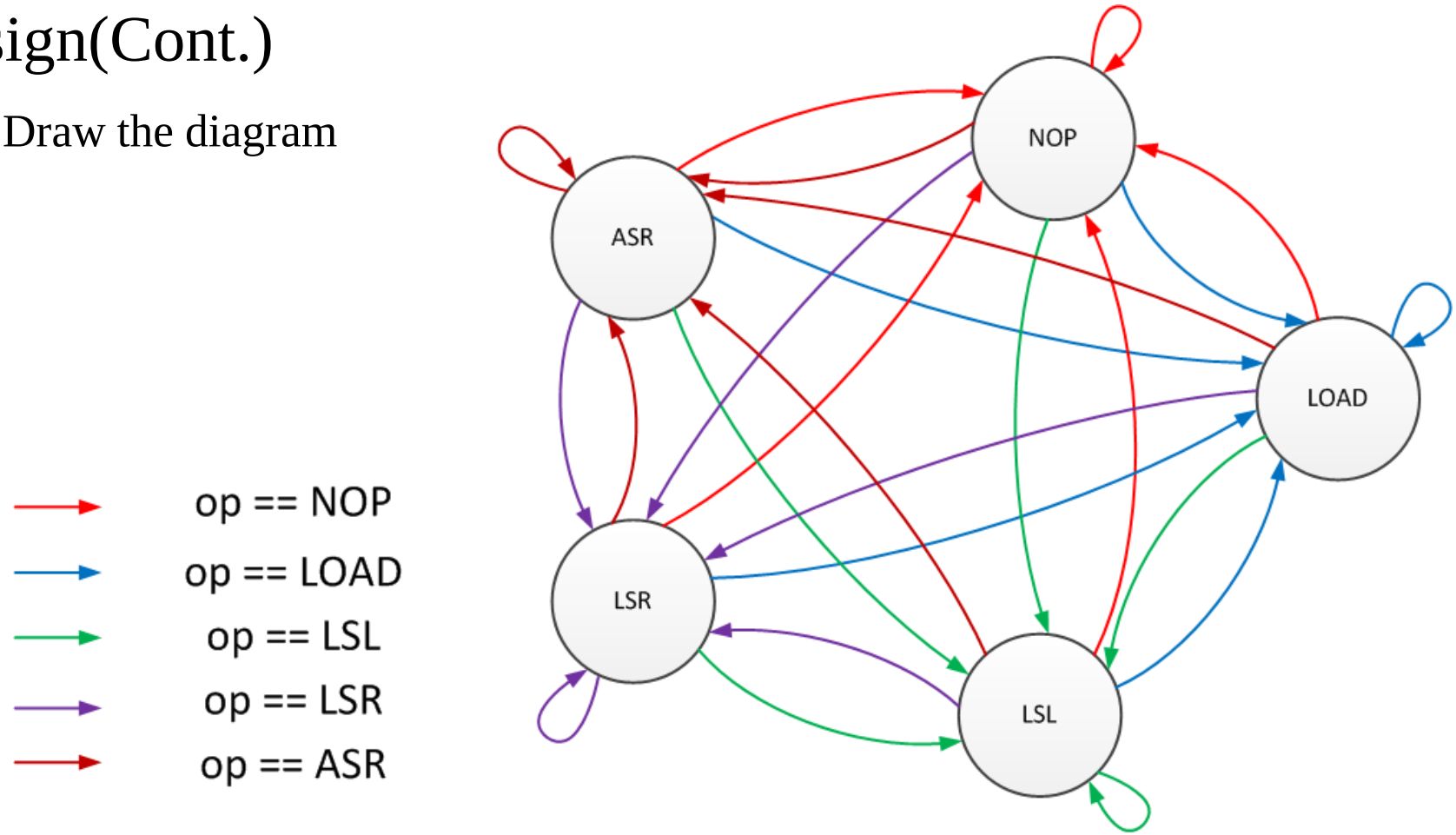
- Design
  - ✓ 앞선 FSM 설계 design에 맞추어 8-bit loadable shifter를 설계
  - ✓ Drawing the finite state diagram
    - Define states
      - NOP – No operation(현재 register의 값을 그대로 출력)
      - LOAD – 입력 값을 register에 할당
      - LSL – Shift amount만큼 logical shift left를 수행
      - LSR – Shift amount만큼 logical shift right를 수행
      - ASR – Shift amount만큼 arithmetic shift right를 수행

FSM Design – Define Inputs/Outputs

- Design(Cont.)
  - Define inputs
    - reset_n – register 값을 0으로 초기화
    - op – operation(3-bit)
    - shamt – shift amount(2-bit)
    - d_in – data in(8-bit)
  - Define outputs
    - d_out – data out(8-bit)

FSM Design – Draw the Diagram

- Design(Cont.)
  - Draw the diagram



  - State 간의 분기는 그린다면 위와 같지만, 실질적으로 입력으로 들어오는 op와 reset_n만으로 결정이 되기 때문에 매우 단순하다.

FSM Design – Encoding States

- Design(Cont.)
  - Encoding states
    - Binary encoding을 사용
    - NOP = 3'b000;
    - LOAD = 3'b001;
    - LSL = 3'b010;
    - LSR = 3'b011;
    - ASR = 3'b100;

FSM Design – Coding the Module Header

- Design(Cont.)
  - Coding the module header
  - shifter8.v로 저장

```
module shifter8(clk, reset_n, op, shamt, d_in, d_out);
  input           clk, reset_n;
  input   [2:0]   op;
  input   [1:0]   shamt;
  input   [7:0]   d_in;
  output  [7:0]   d_out;
```

8-bits register와 combinational circuit instance

```
Endmodule
```

FSM Design – Coding State Registers

- Design(Cont.)
  - Coding state registers(flip-flops) – sequential circuits

```
module _dff_r(clk, reset_n, d, q);
  input          clk, reset_n, d;
  output reg     q;

    always@(posedge clk or negedge reset_n)
    begin
        if(reset_n == 0) q <= 1'b0;
        else q <= d;
    end
endmodule
```

_dff_r.v로 저장

_register8_r.v로 저장

```
module _register8_r(clk, reset_n, d, q);
  input          clk, reset_n;
  input  [7:0]   d;
  output [7:0]   q;
```

Instance of resettable D FF

```
endmodule
```

FSM Design – Coding Combinational Circuits

- Design(Cont.)
  - Coding combinational circuits

```
module cc_logic(op, shamt, d_in, d_out,
d_next);
  input       [2:0]   op;
  input       [1:0]   shamt;
  input       [7:0]   d_in;
  input       [7:0]   d_out;
  output reg  [7:0]   d_next;

  wire        [7:0]   d_lsl;
  wire        [7:0]   d_lsr;
  wire        [7:0]   d_asr;

  parameter NOP     = 3'b000;
  parameter LOAD    = 3'b001;
  parameter LSL     = 3'b010;
  parameter LSR     = 3'b011;
  parameter ASR     = 3'b100;
```

```
  always@ (op, shamt, d_in, d_out,
      d_lsl, d_lsr, d_asr)
    begin
      case(op)
        endcase
    end

  LSL8
  LSR8
  ASR8

endmodule
```

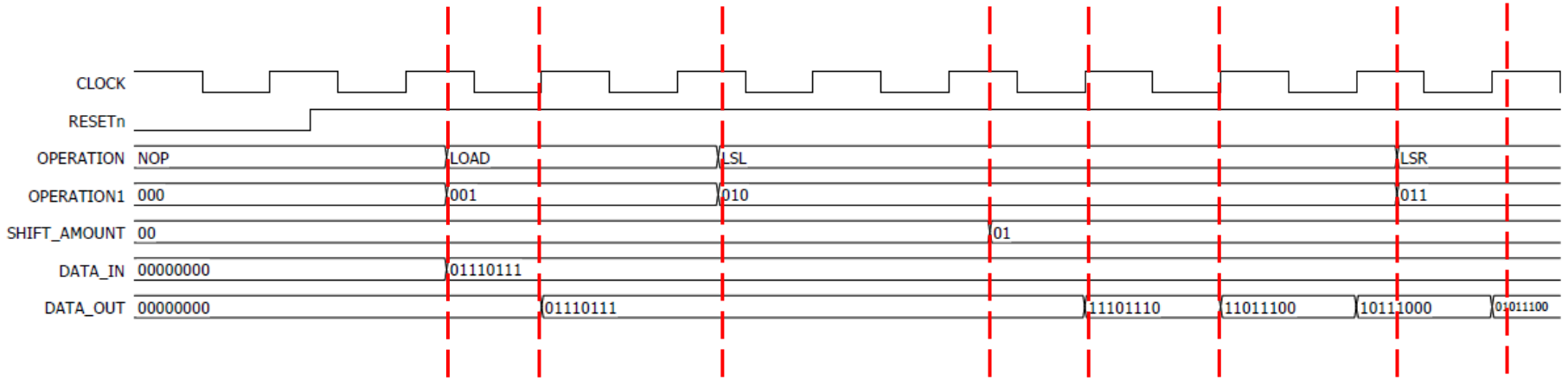
Select do_next

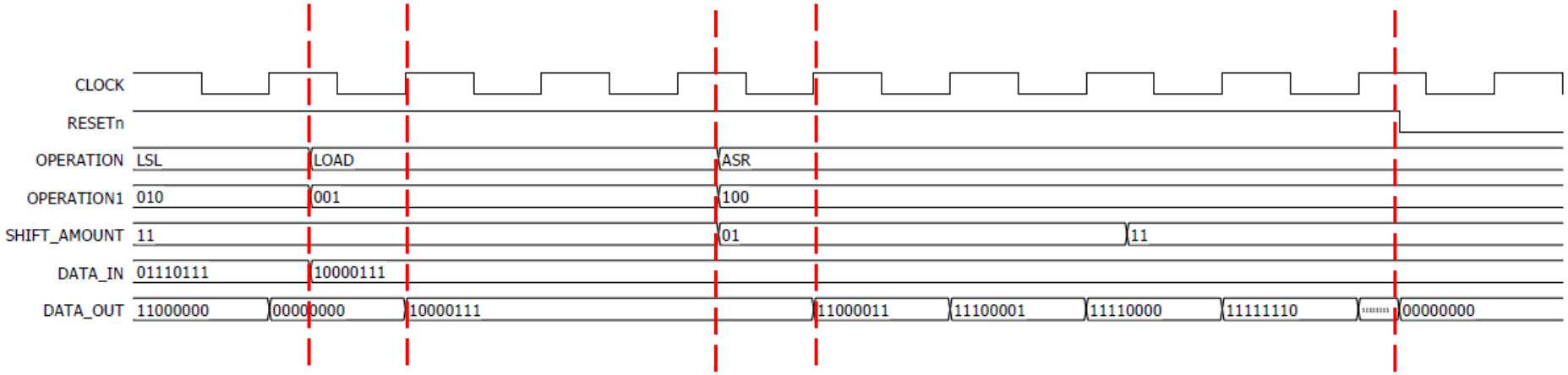
Instance of LSL8, LSR8, ASR8

cc_logic.v로 저장

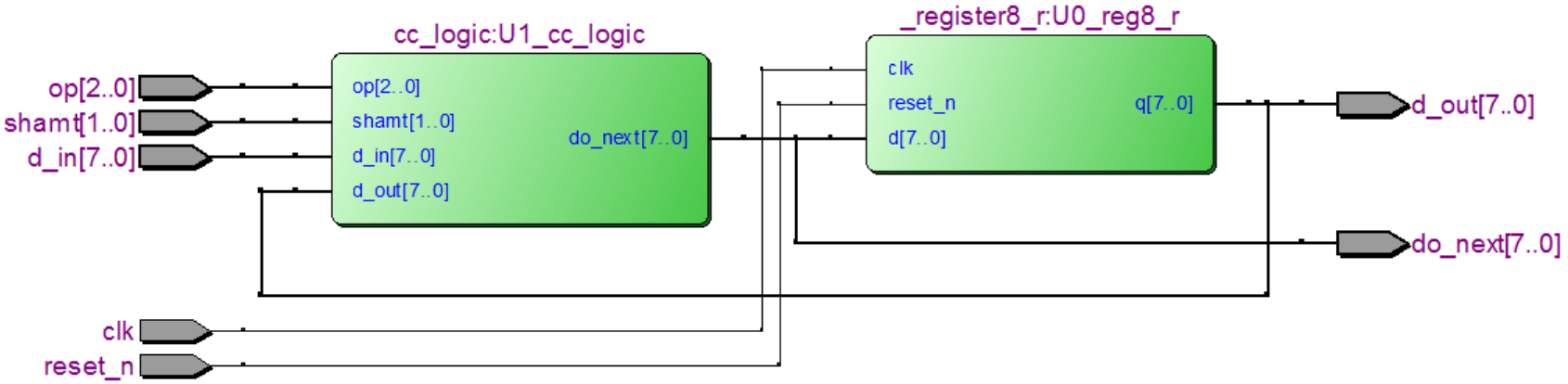
Verification

- Testbench
  - 작성한 module에 대하여 testbench를 작성하여 ModelSim에서 검증 수행

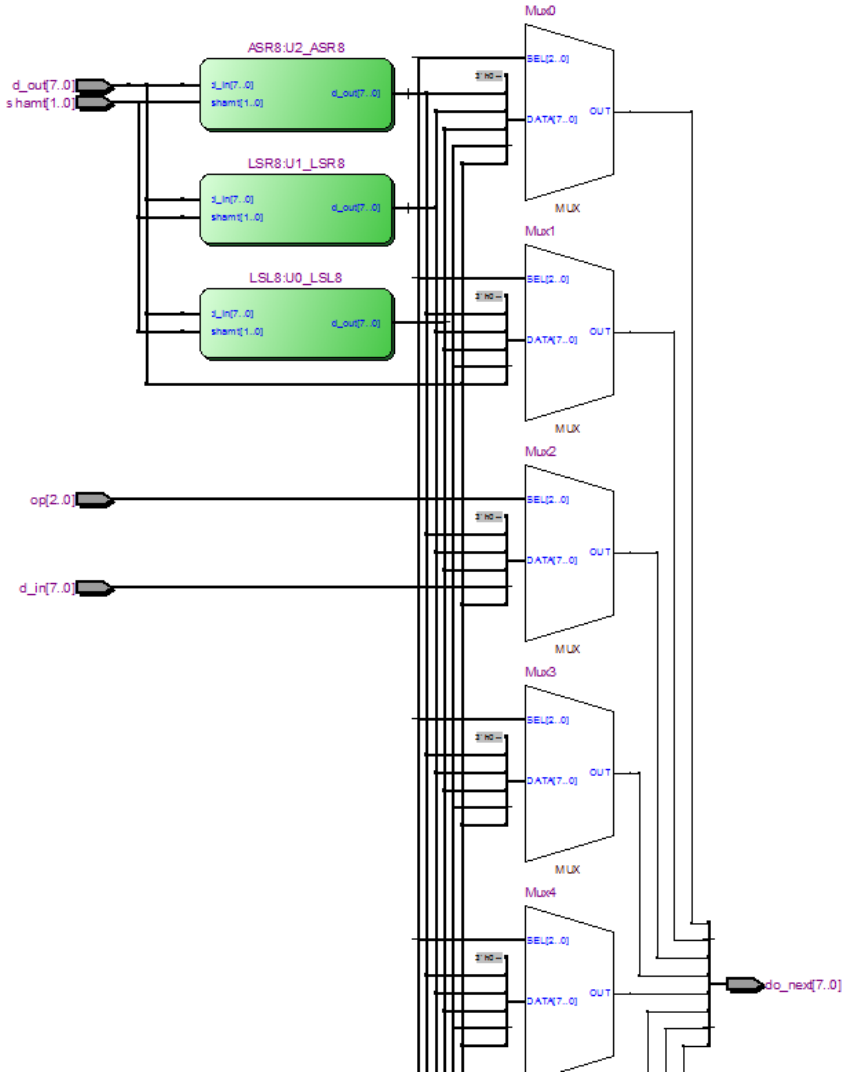


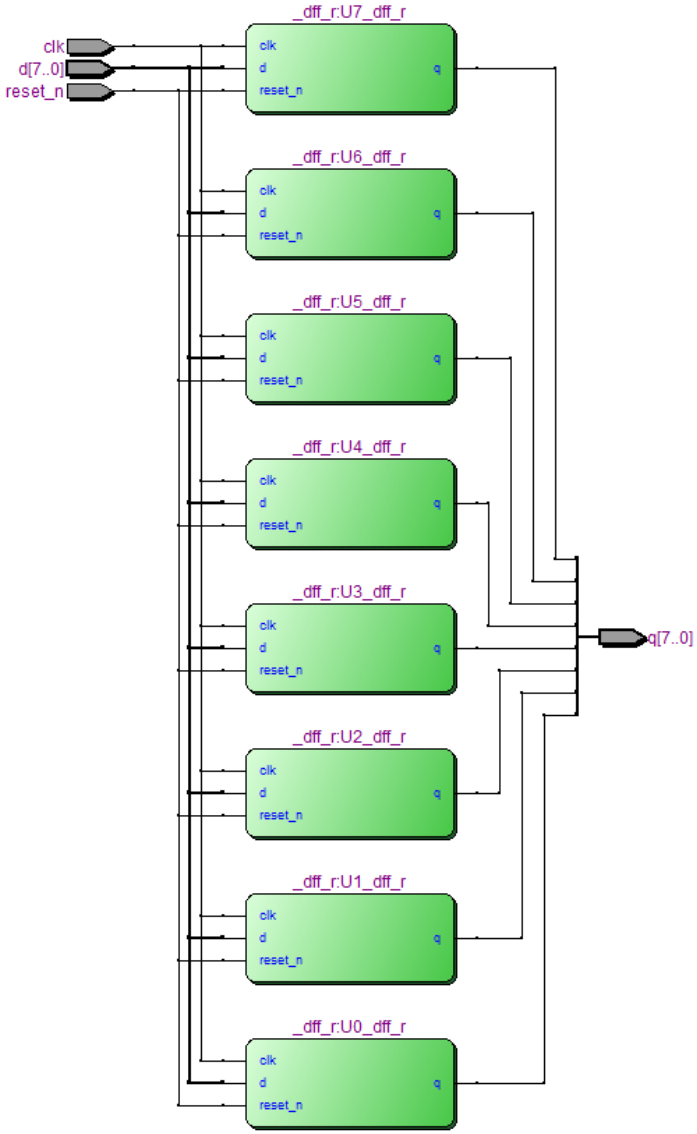


RTL Viewer (1/2)



RTL Viewer (2/2)





An 8-bit loadable up/down counter

PRACTICE III

8-bit Loadable Up/Down Counter

- New Project Wizard
  - Project name : cntr8
  - Top module name : cntr8
  - Family & Device : Cyclone V 5CSXFC6D6F31C6 (밑에서 6번째)
- Verilog file
  - Add files : gates.v, fa_v2.v, clb4.v, cla4.v, _dff_r.v
  - New files : _register3_r.v, cla8.v, os_logic.v, ns_logic.v, cntr8.v, tb_cntr8.v

Carry Look-ahead Adder

- 8-bit CLA
  - ✓ 이전 실습에서 구현하였던 4-bit CLA를 instance하여 8-bit CLA를 구현
  - ✓ 해당 모듈은 counter에서 값을 증가시키거나 감소하는 데 사용

```
module cla8(a, b, ci, s, co);
  input     [7:0]       a,b;
  input                 ci;
  output    [7:0]       s;
  output                co;

  wire c1;

  Instance of resettable D FF

endmodule
```

FSM Design – Drawing the Finite State Diagram

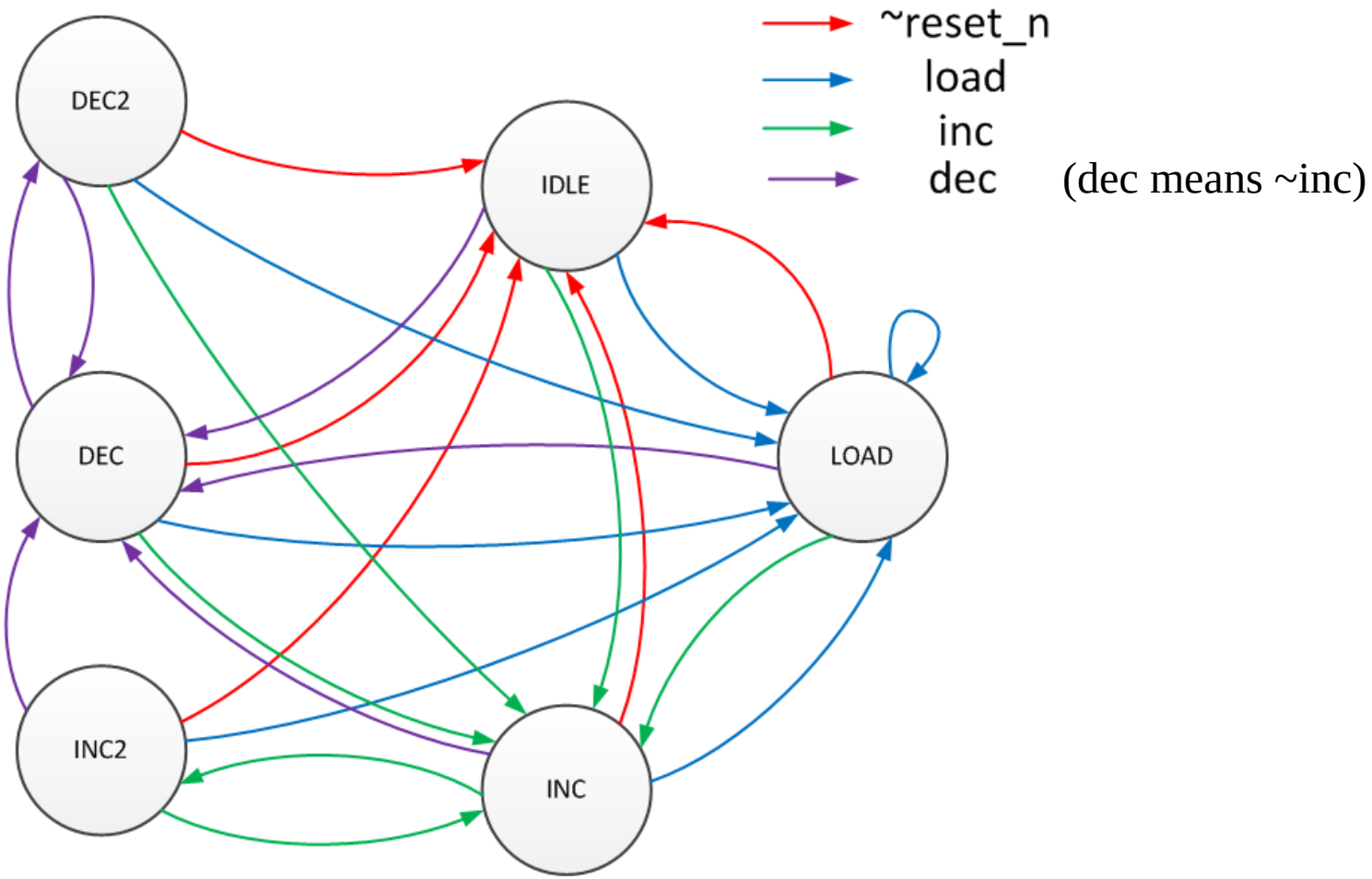
- Design
  - ✓ 앞선 FSM 설계 design에 맞추어 8-bit loadable up/down counter를 설계
  - ✓ Drawing the finite state diagram
    - Define states
      - IDLE – reset되었을 때, count 값을 0으로 하는 state
      - LOAD – 입력 data를 count 값에 할당하는 state
      - INC, INC2 – count 값을 증가하기 위한 state
        - inc가 1인 동안 두 state로 서로 이동하며 값을 증가시킨다.
      - DEC, DEC2 – count 값을 감소시키기 위한 state
        - inc가 0인 동안 두 state로 서로 이동하며 값을 감소시킨다.

FSM Design – Define Inputs/Outputs

- Design(Cont.)
  - Define inputs
    - clk - clock
    - reset_n, load, inc – control signal
    - d_in – data in(8-bit)
  - Define outputs
    - d_out – data out(8-bit)
    - o_state – current state(3-bit, 검증용)
  - **Control signal 우선 순위**
    - **reset_n >> load >> inc**

FSM Design – Draw the Diagram

- Design(Cont.)
  - Draw the diagram



FSM Design – Encoding States

- Design(Cont.)
  - Encoding states
    - Binary encoding을 사용

```
IDLE_STATE= 3'b000;
LOAD_STATE= 3'b001;
INC_STATE = 3'b010;
INC2_STATE= 3'b011;
DEC_STATE = 3'b100;
DEC2_STATE= 3'b101;
```

FSM Design – Coding the Module Header

- Design(Cont.)
  - Coding the module header

```
module cntr8(clk, reset_n, inc, load, d_in, d_out, o_state);
    input           clk, reset_n, inc, load;
    input   [7:0]   d_in;
    output  [7:0]   d_out;
    output  [2:0]   o_state;

    wire        [2:0]   next_state;
    wire        [2:0]   state;

    assign o_state = state;

    Instances of register, next state logic, output logic
endmodule
```

FSM Design – Coding State Registers

- Design(Cont.)
  - Coding state registers(flip-flops) – sequential circuits

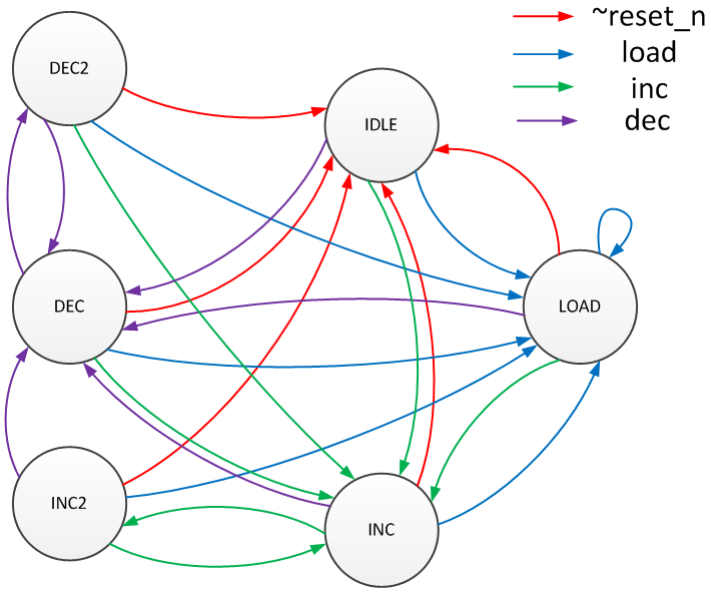
```
module _register3_r(clk, reset_n, d, q);
  input                clk, reset_n;
  input         [2:0]  d;
  output        [2:0]  q;

_dff_r  U0_dff_r (.clk(clk), .reset_n(reset_n), .d(d[0]), .q(q[0]));
_dff_r  U1_dff_r (.clk(clk), .reset_n(reset_n), .d(d[1]), .q(q[1]));
_dff_r  U2_dff_r (.clk(clk), .reset_n(reset_n), .d(d[2]), .q(q[2]));

endmodule
```

FSM Design – Coding Combinational circuits

- Design(Cont.)
  - Coding combinational circuits
    - Next state logic part와 output logic part 두 부분으로 나누어 구현한다.
    - Next state logic part
      - 입력으로 들어오는 load, inc의 값과 현재 state의 값을 통하여 다음 next state를 계산한다.
    - Output logic part
      - 현재 state 값을 통하여서 출력될 값을 계산한다.

FSM Design – Next States Logic Part

- Design(Cont.)
  - Next state logic part



ns_logic.v로 저장

```
module ns_logic(load, inc, state, next_state);
   parameter   IDLE_STATE  = 3'b000;
   parameter   LOAD_STATE  = 3'b001;
   parameter   INC_STATE   = 3'b010;
   parameter   INC2_STATE  = 3'b011;
   parameter   DEC_STATE   = 3'b100;
   parameter   DEC2_STATE  = 3'b101;

   input             load, inc;
   input    [2:0]    state;
   output   [2:0]    next_state;

   reg      [2:0]    next_state;

   always @ (load, inc, state)
   begin
   case(state)

   endcase
   end

endmodule
```

Case 구문

FSM Design – Output Logic Part

- Design(Cont.)
  - Output logic part
    - 각각의 state에 맞춰 결과 값(d_out)을 출력시켜야 한다.
    - 8-bit CLA를 2개 instance하여 받은 결과 값(d_inc, d_dec)를 해당하는 state에 할당하여 주어야 한다.

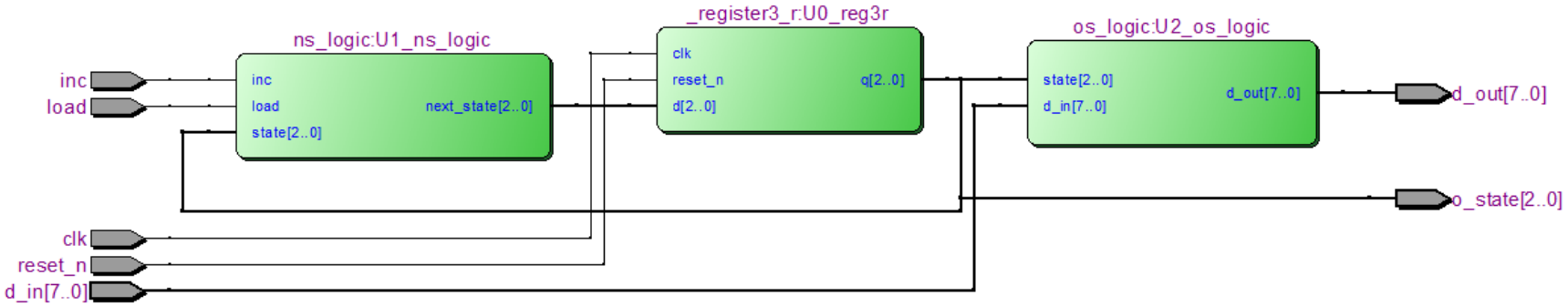
```
module os_logic(state, d_in, d_out);

   parameter   IDLE_STATE  = 3'b000;
   parameter   LOAD_STATE  = 3'b001;
   parameter   INC_STATE   = 3'b010;
   parameter   INC2_STATE  = 3'b011;
   parameter   DEC_STATE   = 3'b100;
   parameter   DEC2_STATE  = 3'b101;

   input    [2:0]    state;
   input    [7:0]    d_in;
   output   [7:0]    d_out;

   reg      [7:0]    d_out;
   wire     [7:0]    d_inc;
   wire     [7:0]    d_dec;

   always @ (state)
   begin
      case(state)
         IDLE_STATE  : d_out = 8'b00000000;
         LOAD_STATE  :
         INC_STATE   :
         INC2_STATE  :
         DEC_STATE   :
         DEC2_STATE  :
         default     :
      endcase
   end

   cla8
   cla8

endmodule
```

????

Instances of CLA

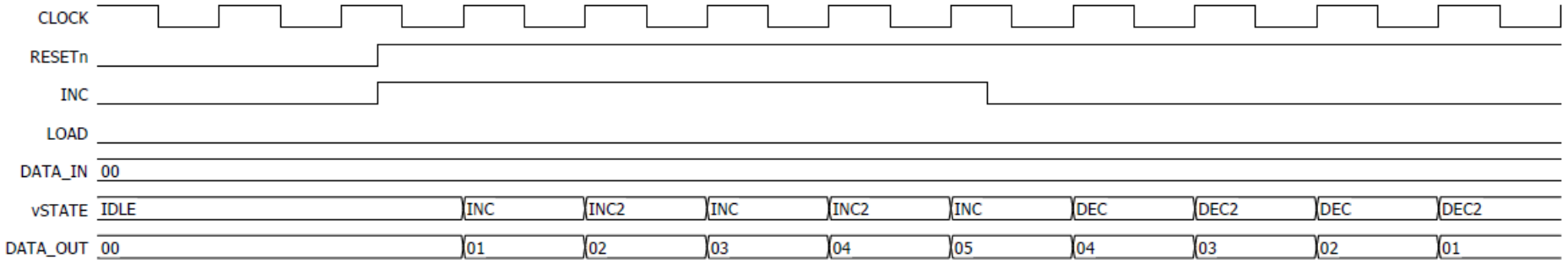
Verification

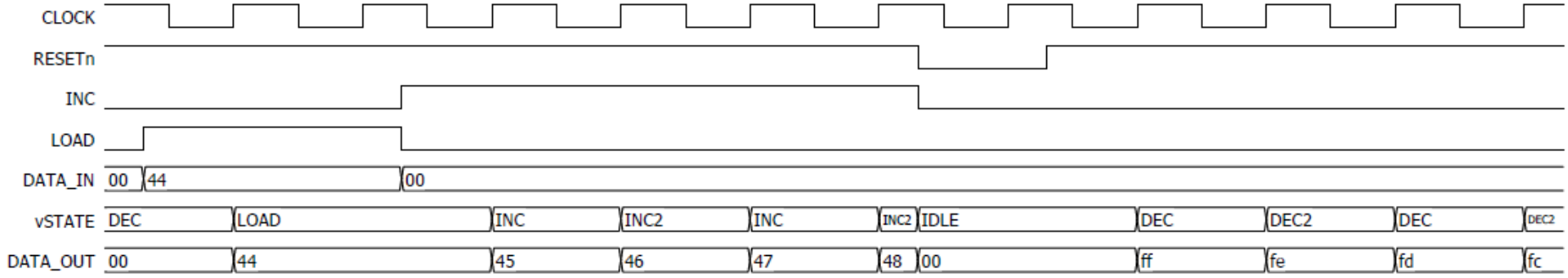
- Testbench
  - ✓ 작성한 module에 대하여 testbench를 작성하여 ModelSim에서 검증 수행
- RTL Viewer
  - ✓ 확인 후 레포트에 이에 대하여 정리한다.



- Flow Summary
  - ✓ 확인 후 레포트에 보고한다.

Waveform


CLOCK
RESETn
INC
LOAD
DATA_IN
vSTATE
DATA_OUT
00
IDLE
INC
INC2
INC
INC2
INC
DEC
DEC2
DEC
DEC2
00
01
02
03
04
05
04
03
02
01


CLOCK
RESETn
INC
LOAD
DATA_IN
vSTATE
DATA_OUT
00
44
00
DEC
LOAD
INC
INC2
INC
INC2
IDLE
DEC
DEC2
DEC
DEC2
00
44
45
46
47
48
00
ff
fe
fd
fc

Assignment 6

- Report
  - 자세한 사항은 lab document 참고
- Submission
  - Soft copy
    - 강의 당일 후 1주까지 (delay 2 days : 20%감점)
    - 실습 미수강은 디지털 논리2 조교 공지에 따름

References

- Altera Co., www.altera.com/
- D. M. Harris and S. L. Harris, Digital Design and Computer Architecture, Morgan Kaufmann, 2007
- 이준환, 디지털논리회로2 강의자료, 광운대학교, 컴퓨터 공학과, 2019

Q&A

THANK YOU