VERILOG 설계언어 중급 1일차

반도체설계교육센터
IC DESIGN EDUCATION CENTER

김두영
doo0@hanyang.ac.kr

2018. Aug. 09

강의 시간표 (1일차)

1일차 (8월 9일)

10:00 ~ 11:20	VERILOG HDL 개요 및 기초 문법
11:40 ~ 13:00	하드웨어 설계 기술
13:00 ~ 14:00	점심시간
14:00 ~ 15:20	기초 설계 실습 – 환경 설정 및 설계 기초
15:40 ~ 17:00	심화 설계 실습 – 교통 신호 시스템 설계

VERILOG HDL 개요 및 기초 문법

반도체설계교육센터
IC DESIGN EDUCATION CENTER

What is Verilog HDL?

1. Origin

* Verilog HDL
  - Originated in 1983 at Gateway Design Automation
  - Accepted as an IEEE standard
  - IEEE 1364-1995
  - IEEE 1364-2001
* VHDL
  - Developed under contract from DARPA later than verilog HDL

=> The advent of logic synthesis in the late 1954s changed the design methodology radically. Finally, digital circuits coud be described at a register transfer level by use of an HDL.

Why Verilog?

1. Digital circuit design

* Trends
  - Faster
  - Larger numbers of gates
  - Physically smaller
  - Packages have many fine-pitch pins

=> The design cannot be described without a language.
Especially, the behavioral design is the most effective way.

Why Verilog?

2. Digital circuit designers

* The underlying design concerns

  - Be understandable to others, especially who will work on the design later

  - Be logically correct. The designer collects user specifications, device parameters, and design entry rules, then creates a design that meets the needs of the end user

  - Perform under worst-case conditions of temperature and process variation. Variations in the device timing specifications, including clock skew, register setup and hold times, propagation delay times, and output rise/fall times must be accounted for.

Why Verilog?

2. Digital circuit designers

* The underlying design concerns

- Be reliable. The end design cannot exceed the package power dissipation limits. Internally generated temperature rises are proportional to the number of gates and the speed at which they are changing states

- Be testable and can be proven to meet the specifications

- Do not exceed the power consumption goals

Real World

1. Example

* *Verilog design in fields*

Listing 1-1 Non- Real World Example

```
// Transfer the content of register b to register a.
     a       <=      b;
```

Listing 1-2 Real World Example

```
/* Signal b must transfer to signal a in less than 7.3 nsec in a
-3 speed grade device as part of a much larger design that must
draw less than 80 uA while in standby and 800 uA while operating.
The whole design must cost less than $1.47, pass CE testing, and
take less than two months to be written, debugged, integrated,
documented, and shipped to the customer. Signal a must be
synchronized to the 75 MHz system clock and reset by the global
system reset. The signal b input should be located at or near pin
79 on the 208-pin package in order to help meet the setup and
hold requirement of register a.*/

     a       <=      b;
```

Basic Concepts

1. Data types

* Values
  - 0, 1, x, z

* Nets
  - Represents connections between hardware elements
  - Nets have values continuously driven
  - declared by the keyword wire

* Registers
  - Do not mean the hardware register
  - The signal assigned in always or initial block
  - declared by the keyword reg

Basic Concepts

1. Data types

* Vectors
  - Nets or regs can be declared as vectors
  - [high#:low#] or [low#:high#]
  - The left number is always the MSB
  - Partial reference(vector part select) is possible
  - Variable vector part select : [<starting_bit>+|-:width]

* Integer
  - General purpose register data type
  - The default width is the host-machine word size, but at-least 32 bits

Basic Concepts

1. Data types

* Arrays
  - Allowed for reg, integer, time, real, realtime, nets, and vector register data types
  - Do not confused with vectors

* Parameters
  - Constants defined in a module
  - Parameters can be overridden at compile time : customize
  - defparam : change the parameter values at module instantiation
  - localparam : cannot be changed

2. Syntax

* Modules
  - A module definition always begins with the keyword 'module'
  - The 'endmodule' must always come last in a definition
  - Components : port declarations, variable declarations, instantiation of lower modules, continuous assignments, procedural assignments in behavioral blocks, tasks and functions

* Module definition
  - Module name
  - Port list

2. Syntax

* Port declarations

  - Ports provide the interface by which a module can communicate with its environment

  - Type : input, output, inout

  - In default, declared as a net type

  - The output and inout ports can be redefined as a reg type, when they are assigned in always block

* Variable declararations

  - Signals inside the module have to be declared as net or reg type

2. Syntax

* Instantiation of lower modules
  - Instantiate the module already defined
  - Port mapping by ordered list

ex> *full_add4 fa (sum, c_out, a, b, c_in);*

  - Port mapping by name

ex> *full_add4 fa (.S(sum), .C_O(c_out), .A(a), .B(b), .C_I(c_in));*

* Continuous assignments for combinational logics
  - Assignment with "assign" keyword

ex> *assign b = a*

: as soon as the value of a changes, the changed value is assigned to b

2. Syntax

* Procedural assignments

- Assignment in procedure block : always or initial
- LVALUE must be a "reg" type
- Blocking : sequential assignment

ex> *always @ (*) begin*

b = a; c = b;

end

- Non-blocking : parallel assignment

ex> *always @ (*) begin*

b <= a; c <= b;

end

2. Syntax

* Sensitivity list

- As soon as any event of signals which are belonged to sensitivity list is occurred, the assignments in always block are executed

- Level-triggered sensitivity list makes combinational circuits, but it sometimes causes latches

ex> *always @ (a or b)*

- Edge-triggered sensitivity list makes sequential circuits. Altough the sequential logic uses asynchronous reset, reset signal have to be edge-triggered because mixed list is not acceptable

ex> *always @ (posedge clk or negedge rstn)*

2. Syntax

* Conditional operator (? :)

  - The action of the operator is similar to a multiplexer

  - Used in continuous assignment

  ex> *assign out = (sel)? i0 : i1;*

  - Can be nested

  ex> *assign out = (A)? (B? x : y) : (C? x-1 : y-1);*

2. Syntax

* Conditional commands

- Only can be used in a procedural assignment
- Not enough description of conditions, some latches may be causes
- if ~ else if ~ else

ex>
```
if ( condition_1 ) statement;
    else if ( condition_2) begin
        statements;
    end
    else statement;
```

2. Syntax

* Conditional commands

- case ~ endcase

ex> *case (condition)*

```
case_1 : begin
                statements;
            end
case_2 : statement;
default : statement;
endcase;
```

Wrap-up

Design structure in Verilog HDL

```
module module1 (port1, port2, port3, port4, port5, clk, reset);
    input port1, clk, reset; output port2, port3, port4;
    output [3:0] port5;

    reg port3; reg [3:0] port5;

    sub_module s1(.sub_port1(port1), .sub_port2(port2));

    assign port3 = port1;

    always @ (port1) port4 = port1 + 1;
    always @ (posedge clk or negedge reset) begin
        if(!reset)port5 <= 4'b0000;
        else port5 <= port5 + port1;
    end
endmodule
```

하드웨어 설계 기술

반도체설계교육센터
IC DESIGN EDUCATION CENTER

Finite State Machines

1. Introduction

* Definition

- Useful computational model for both hardware and certain types of software

- A model of computation consisting of

  a set of states,

  a start state,

  an input alphabet,

  a transition function that maps input symbols and current states to a next state.

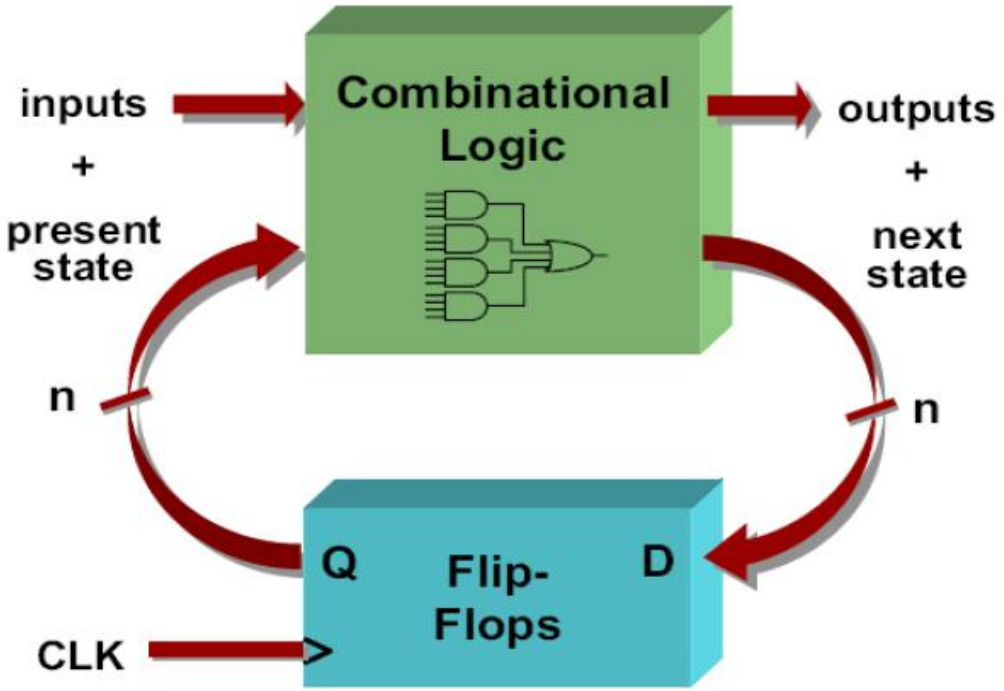
Finite State Machines

1. Introduction

* FSM in hardware

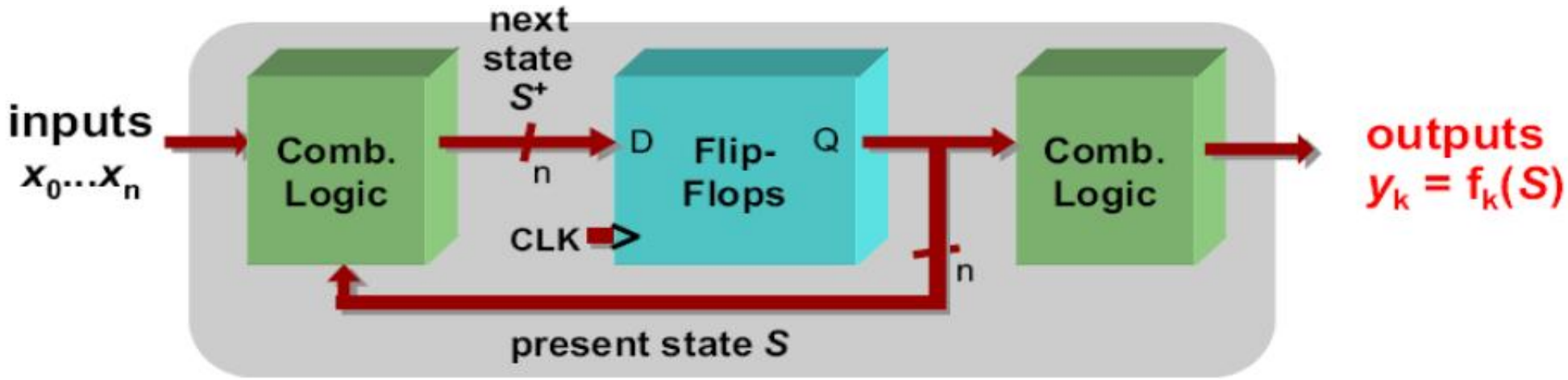
- Finite State Machines (FSMs) are a useful abstraction for sequential circuits with centralized "states" of operation

- At each clock edge, combinational logic computes outputs and next state as a function of inputs and present state



2. Two types of FSM

* Moore machine
  - Distinguished by their output generation mechanism
  - Only current state affects on the output values
  - Moore machine

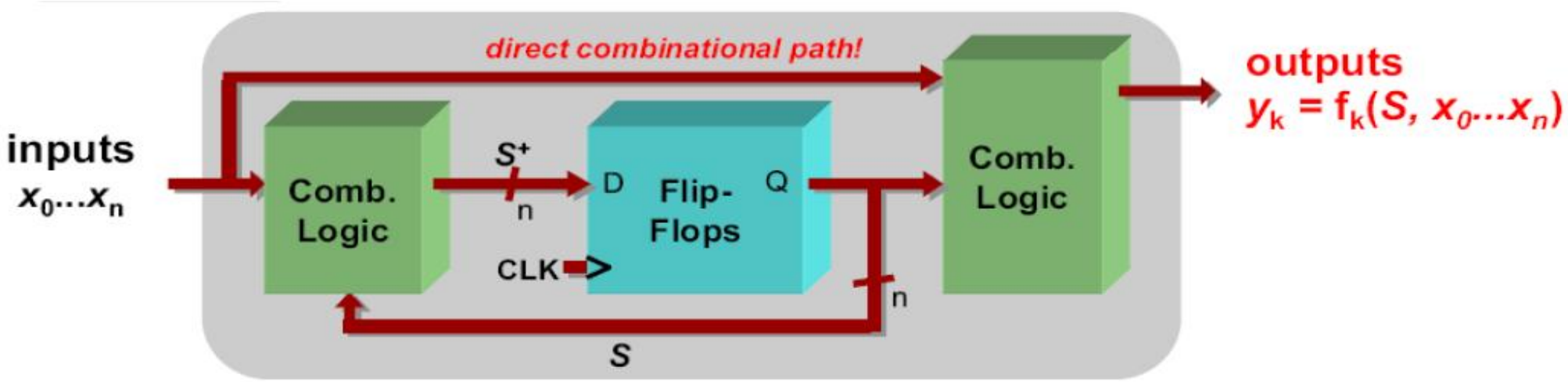


2. Two types of FSM

* Mealy machine

- Current state and input data of FSM determines the output values

- Mealy machine



3. Design example

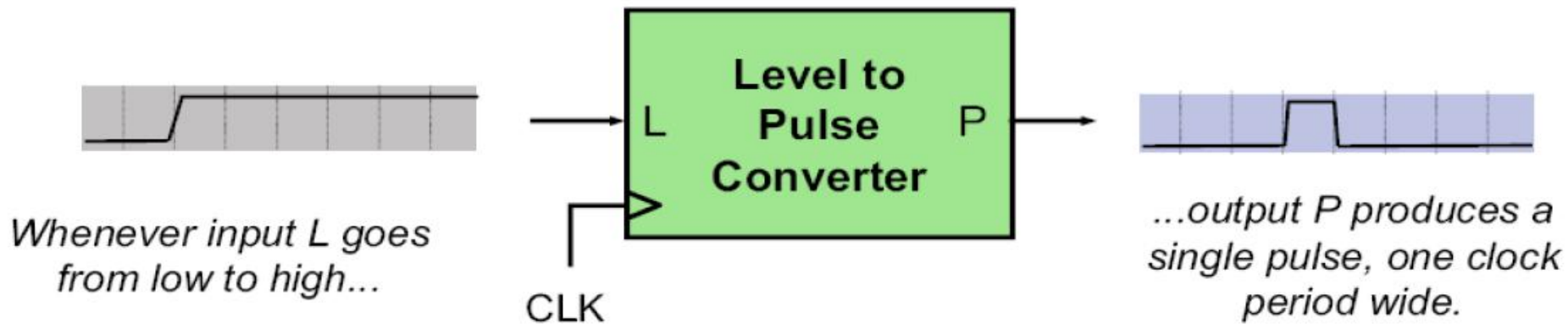
* Level-to-Pulse converter

- A level-to-pulse converter produces a single-cycle pulse each time its input goes high

- In other words, it's a synchronous rising- edge detector

- Sample uses:

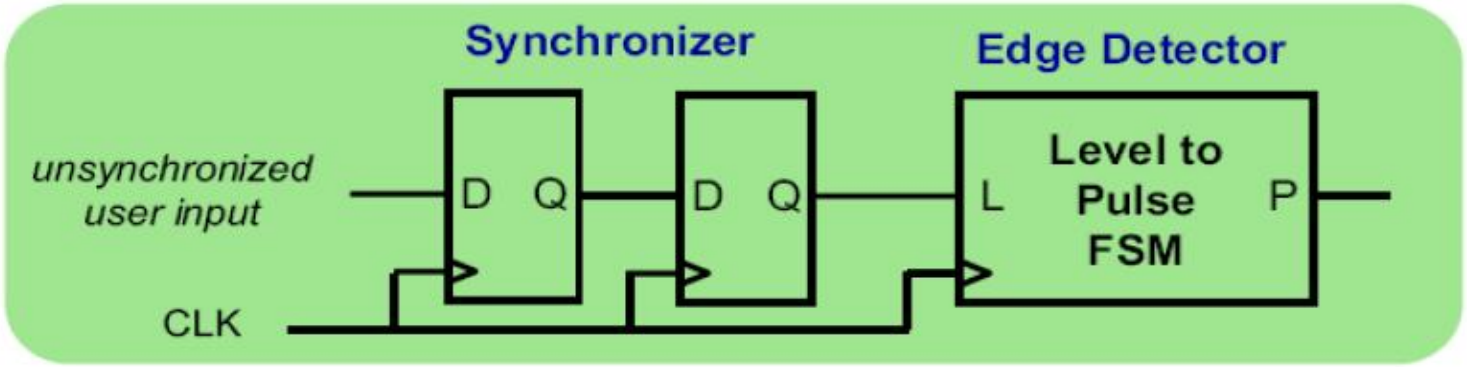
Buttons and switches pressed by humans for arbitrary periods of time => Single-cycle enable signals for counters



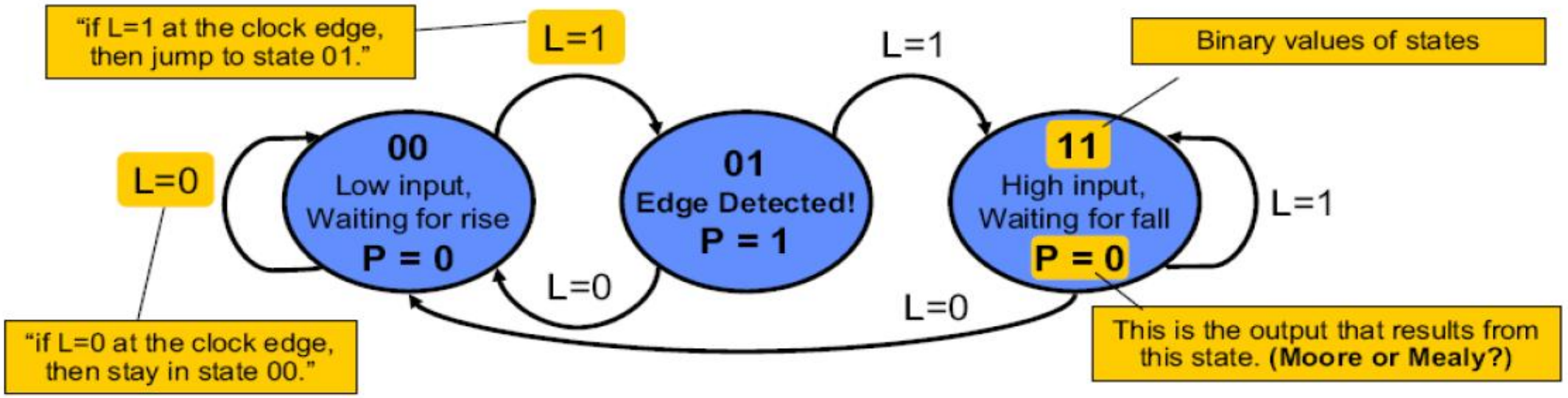
Finite State Machines

4. State transition diagrams

* Block diagram



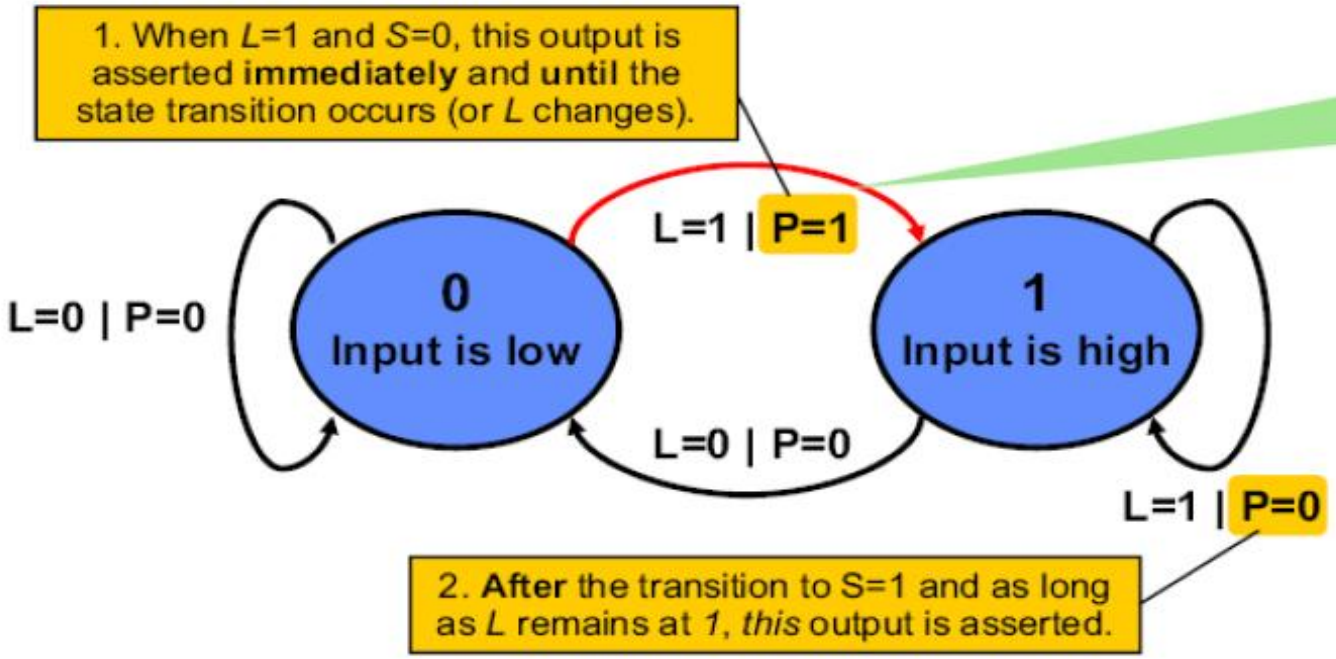
* State transition diagram

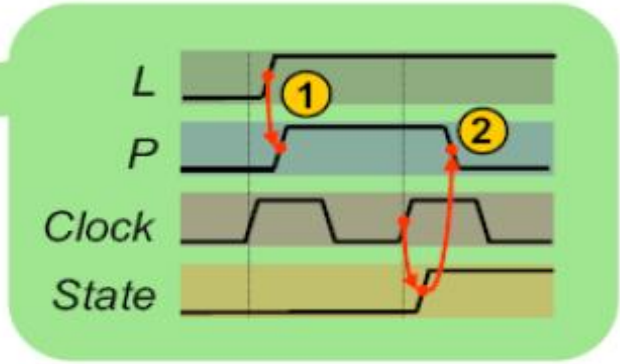


5. Design of a Mealy Level-to-Pulse

* Characteristic

- Since outputs are determined by state and inputs, Mealy FSMs may need fewer states than Moore FSM



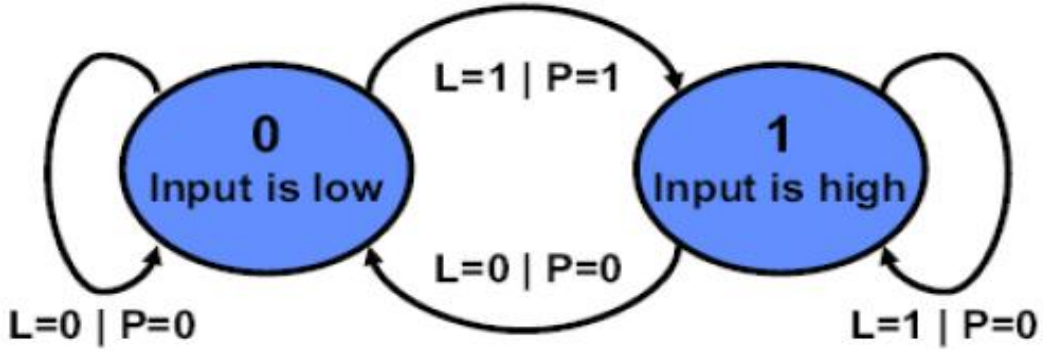


Output transitions immediately.

State transitions at the clock edge.

5. Design of a Mealy Level-to-Pulse

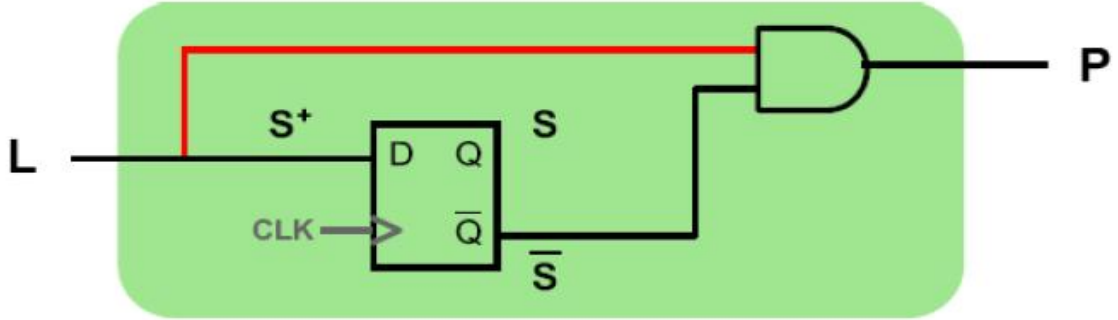
* Implementation



Pres. State	In	Next State	Out
S	*L*	*S*⁺	*P*
0	0	0	0
0	1	1	1
1	0	0	0
1	1	1	0

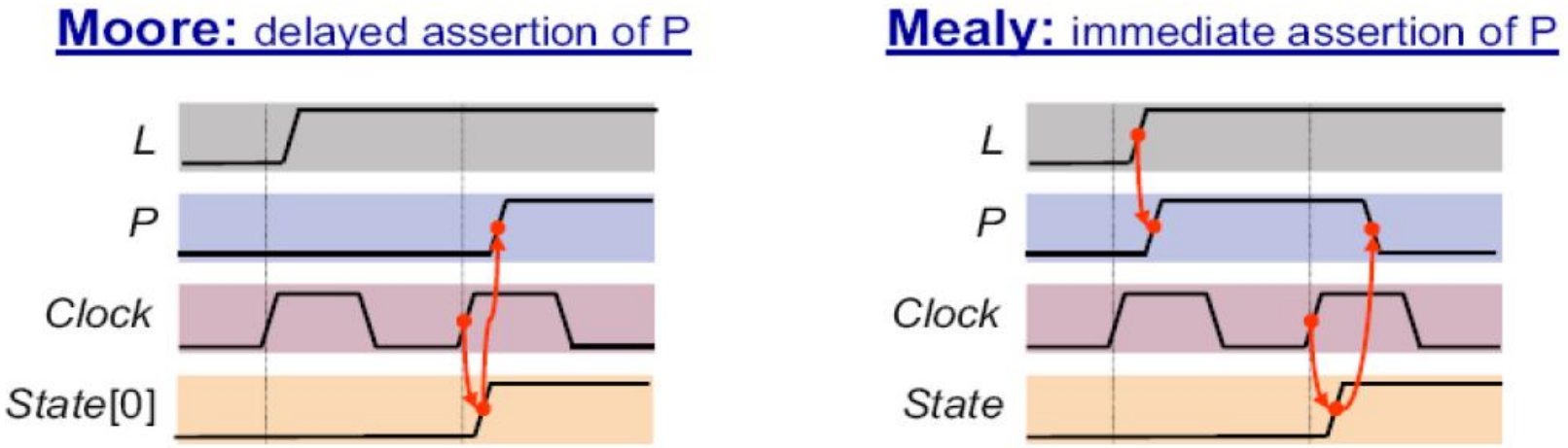
- FSM's state simply remembers the previous value of L

- Circuit benefits from the Mealy FSM's implicit single-cycle assertion of outputs during state transitions



5. Mealy vs. Moore

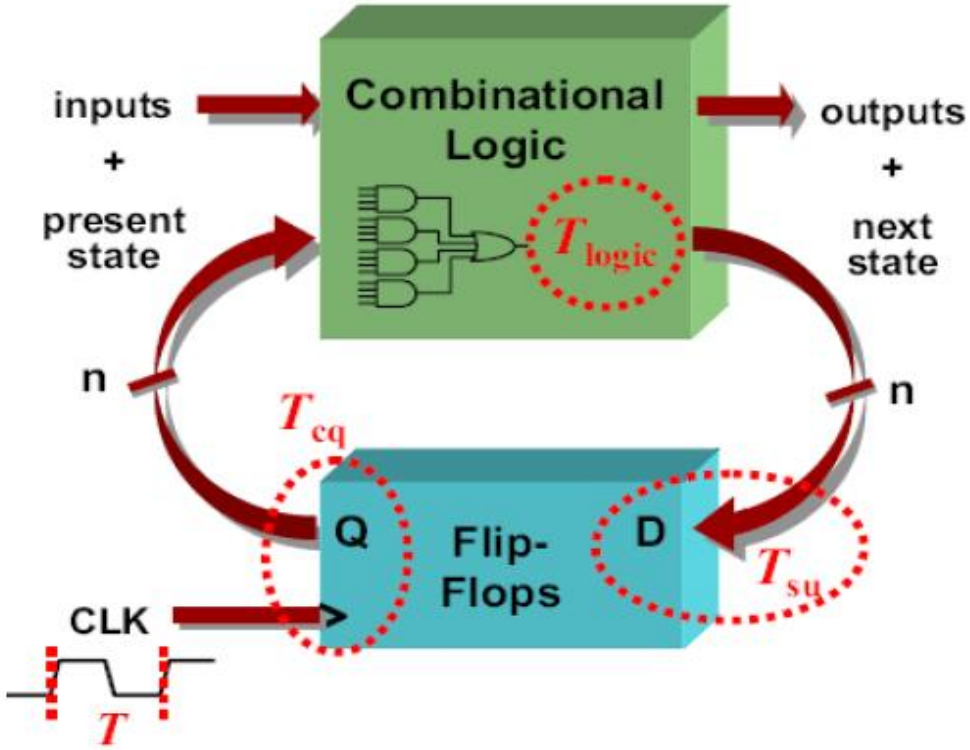
* Difference is in the output generation
  - Moore outputs are based on state only
  - Mealy outputs are based on state and input
  - Therefore, Mealy outputs generally occur one cycle earlier than a Moore



* vs. Moore
  - Be more difficult to conceptualize and design
  - Have fewer states

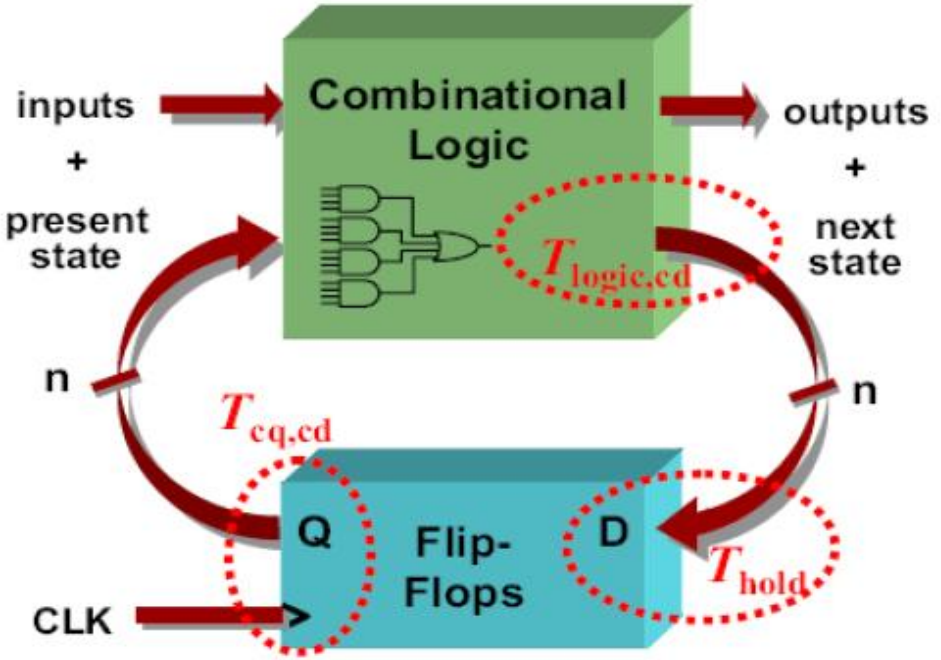
6. Timing requirements

Minimum Clock Period



$$T > T_{cq} + T_{logic} + T_{su}$$

Minimum Delay



$$T_{cq,cd} + T_{logic,cd} > T_{hold}$$

7. How to convert Mealy FSM to Moore FSM

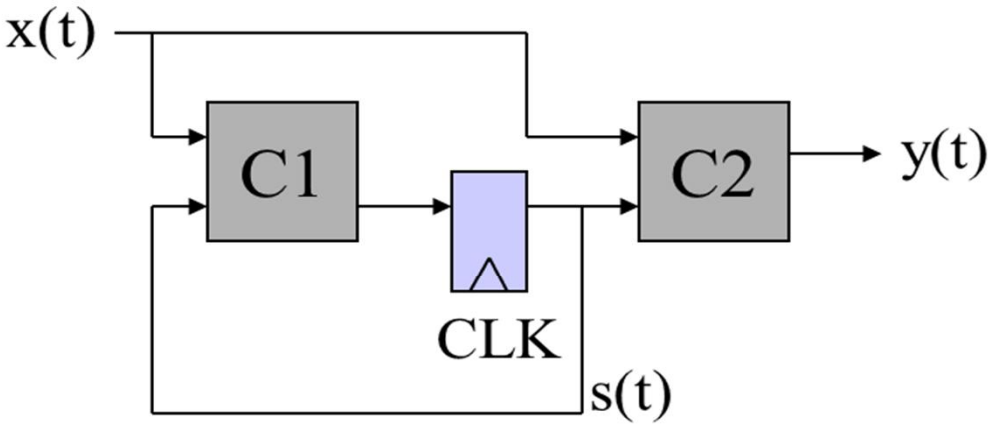
* Transform from Mealy to Moore Machine

- Mealy Machine : $y(t) = f(x(t), s(t))$

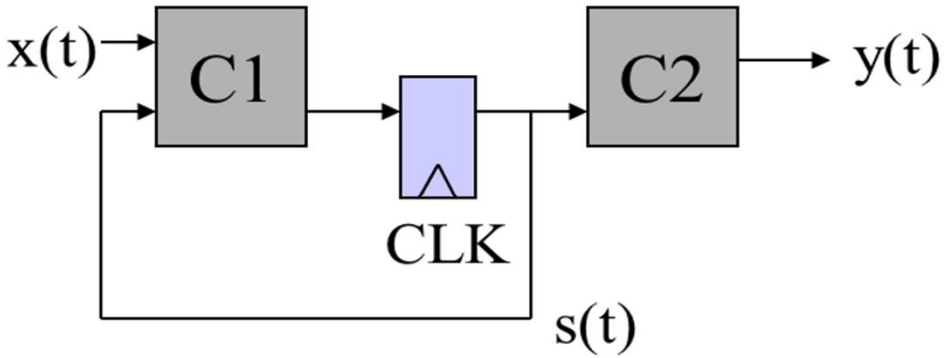
  $s(t+1) = g(x(t), s(t))$

- Moore Machine : $y(t) = f(s(t))$

  $s(t+1) = g(x(t), s(t))$



Mealy Machine



Moore Machine

7. How to convert Mealy FSM to Moore FSM

* Algorithm

1) For each NS, $z = S_i, y_j$ create a state $S_i(j)$

2) For each new state $S_i(j)$, repeat the row $PS = S_i$

3) Replace NS, $z = S_i, y_j$ with state $S_i(j)$

Set output $z = y_j$ for row $PS = S_i(j)$

Mealy Machine:

PS	00	01	10	(x,y)
A	A,0	A,1	B,0	
B	A,1	B,0	B,0	(NS, z)

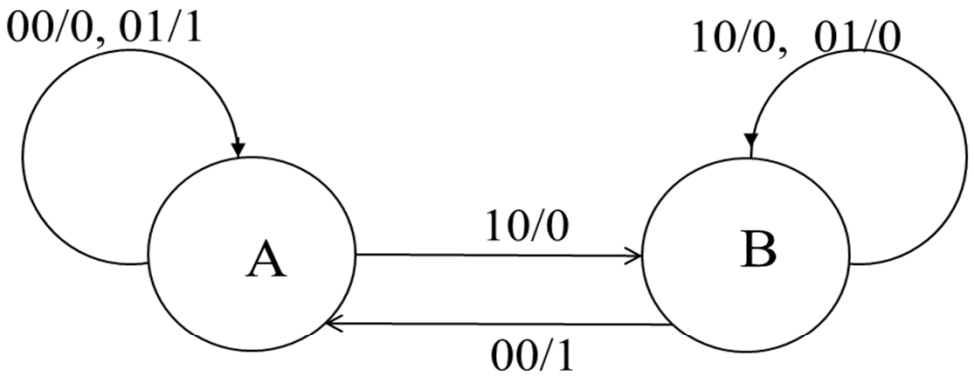
Moore Machine:

PS	00	01	10	(x,y)	z
A0	A0	A1	B		0
A1	A0	A1	B		1
B	A1	B	B		0

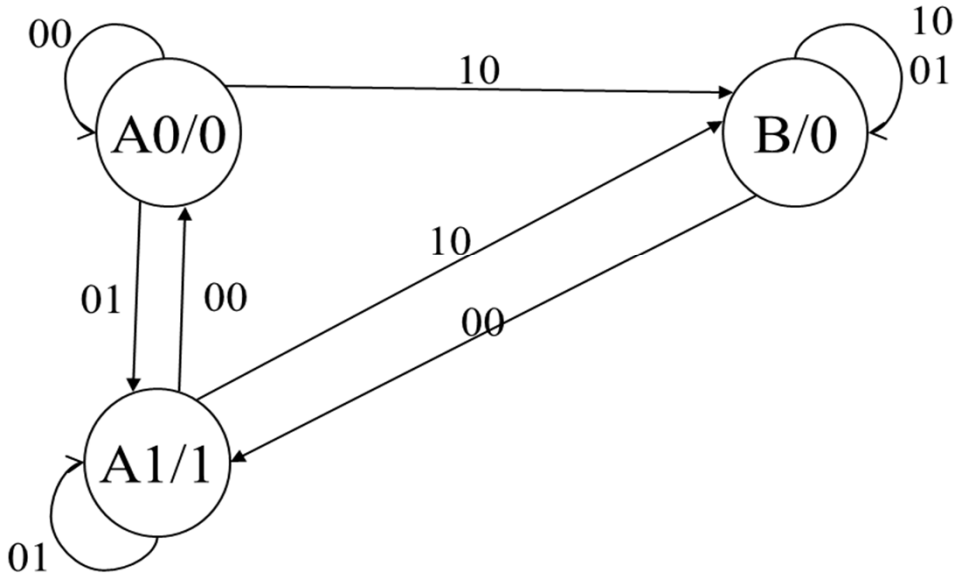
7. How to convert Mealy FSM to Moore FSM

* State diagram

Mealy Machine



Moore Machine



Wrap-up

1. Behavioral modeling

* Finite State Machine
* Mealy FSM vs. Moore FSM
  - Mealy machines tend to have less states
  - Mealy machines react faster to inputs
  - Moore machines are safer to use
* What is better?

  It depends.

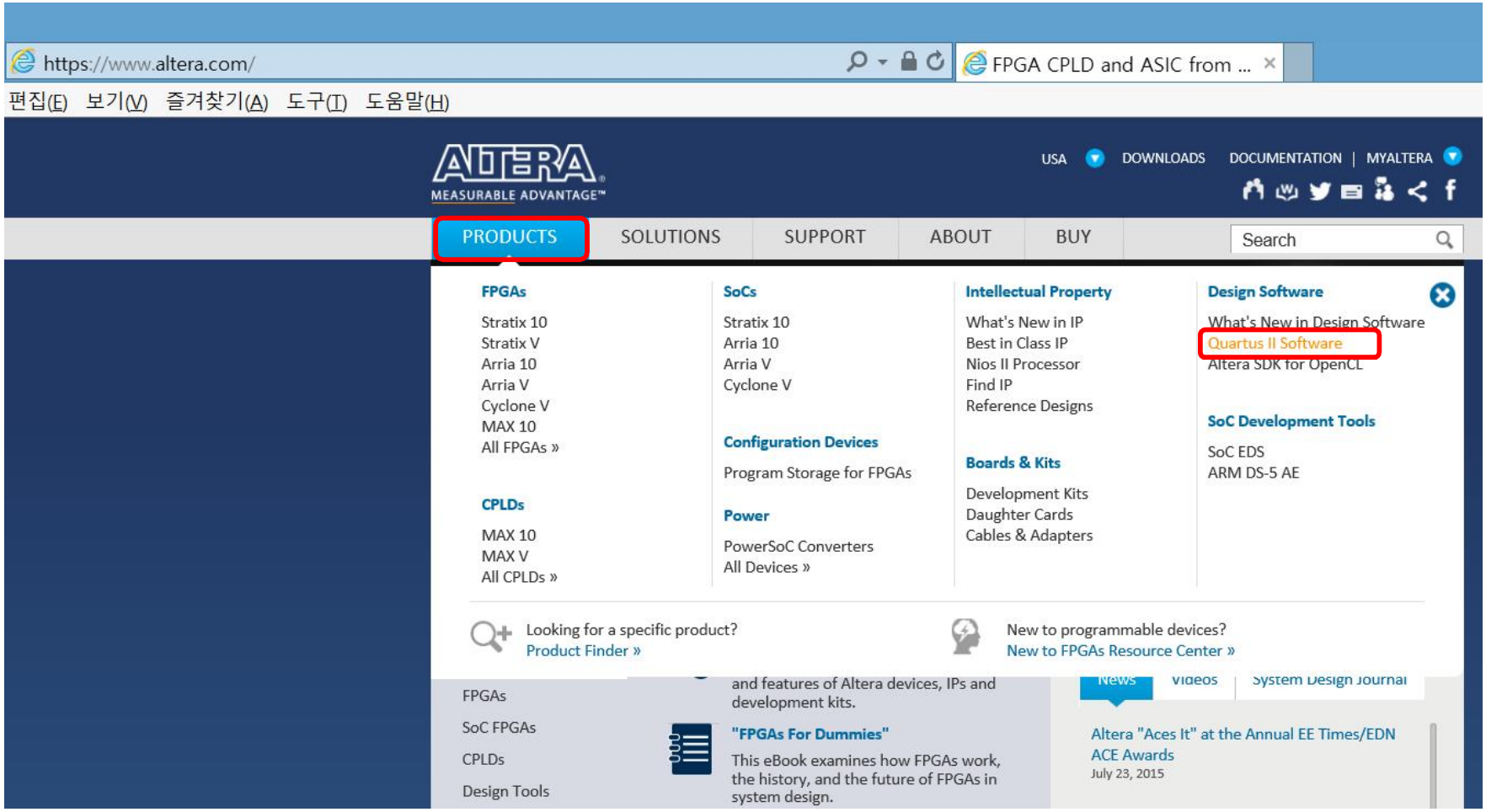
기초 설계 실습 – 환경 설정 및 설계 기초

ModelSim & Quartus II 설치하기

1. 공개버전 Download

* Link

- Altera website : http://dl.altera.com/?edition=web



ModelSim & Quartus II 설치하기

1. 공개버전 Download

* Link

- Altera website : http://dl.altera.com/?edition=web

ModelSim & Quartus II 설치하기

2. Introduction

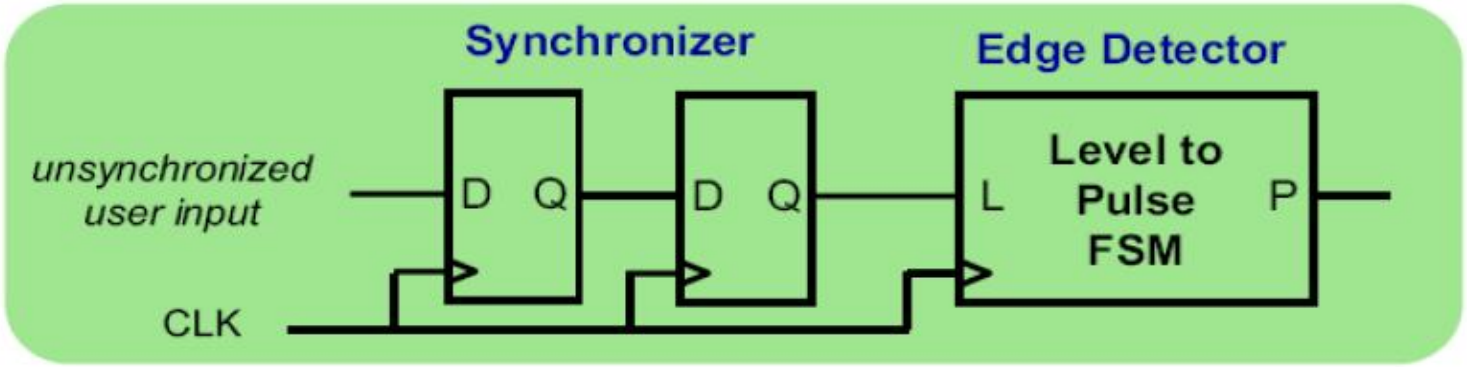
* ModelSim
  - Logical simulator from Mentor Graphics
  - Verilog & VHDL compile 지원 : 문법/구조적 체크 수행, 합성과는 별개
  - 4-value simulator : 0, 1, X, Z => X and !X = X

* Quartus II
  - FPGA 설계 software from Altera
  - 합성, physical mapping, static timing analysis, simulator 내장
  - Modelsim, Synplify 등 FPGA용 3rd party tool 사용 가능

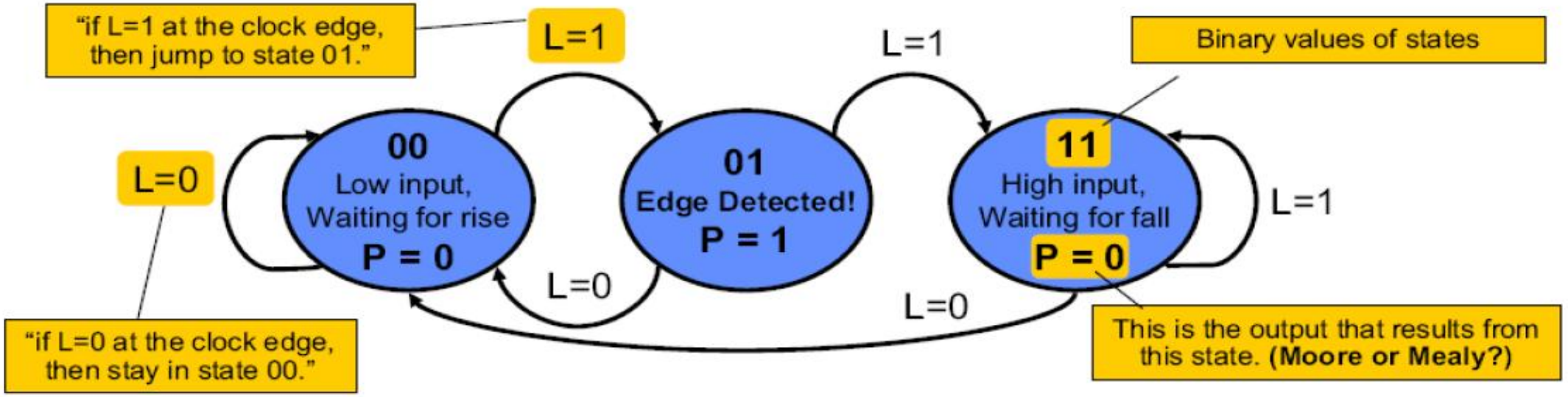
Design Description

1. Level-to-Pulse converter 설계하기

* Block diagram



* State transition diagram

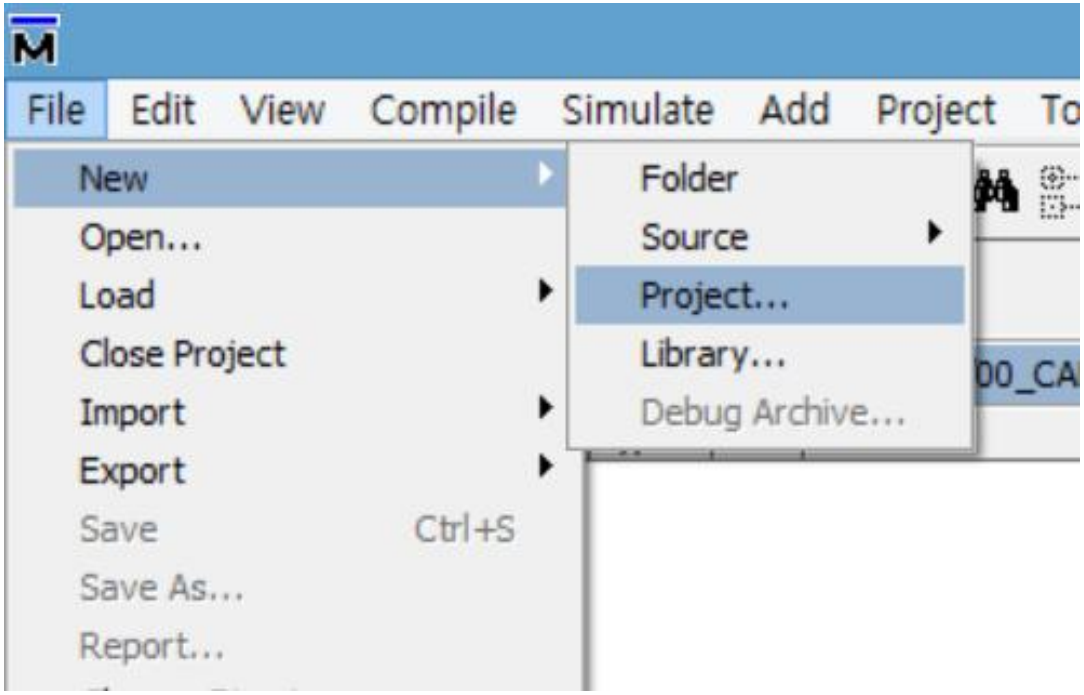


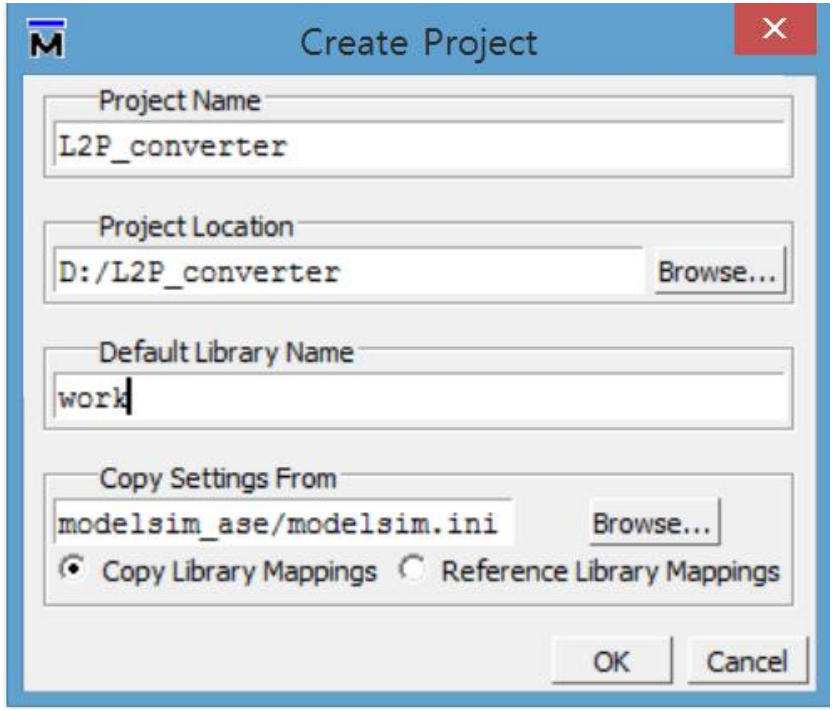
Level-to-Pulse Converter 설계하기

1. ModelSim에서 설계하기

* Project 만들기

- File => New => Project
- Project Name : L2P_converter
- Project Location, Default Library Name 설정



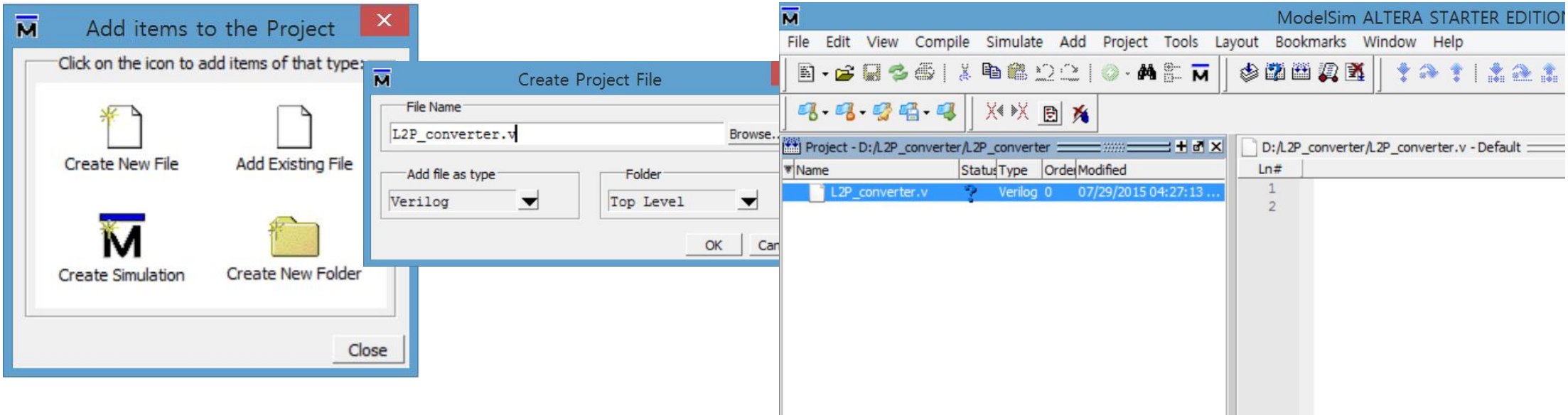


Level-to-Pulse Converter 설계하기

1. ModelSim에서 설계하기

* Design File 추가하기

- Create New File => File Name, Type(Verilog) 지정
- 기존 설계 파일은 Add Existing File로 추가
- ModelSim project 창에 파일 추가 확인, 더블클릭 혹은 우클릭 => Edit 으로 파일 편집 가능

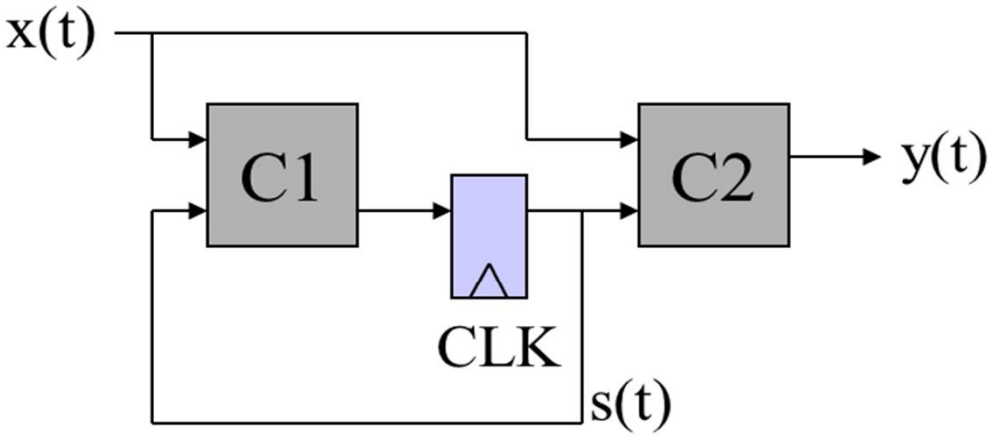


Level-to-Pulse Converter 설계하기

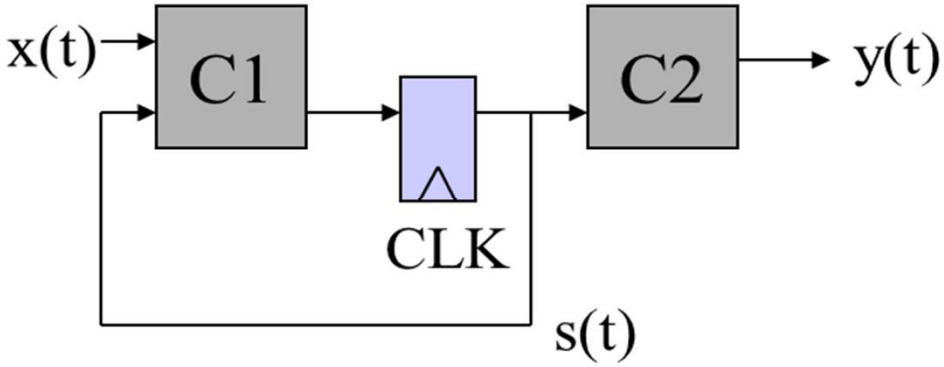
2. Verilog 설계하기

* Finite State Machine 설계하기

- State : FSM 동작 클럭의 매 사이클마다 결정됨. 즉, flip-flop으로 구현됨
- State transition : 현재 state와 입력 값에 따라 다음 state를 미리 연산함. 즉, combinational 회로 파트임
- Result output : Moore machine의 경우 현재 state에 의해 결정되고, Mealy machine의 경우 현재 state 및 입력 값에 따라 결정됨. 즉, combinational 회로로 구현됨



Mealy Machine



Moore Machine

Level-to-Pulse Converter 설계하기

2. Verilog 설계하기

* Finite State Machine 설계하기

- FSM 설계는 state transition 부, state flip-flops, result 결정부로 구성됨

- State transition : sensitivity list는 현재의 state인 "state"와 입력값인 "L"이며, 일반적으로 다음 state인 "next_state"의 값을 결정하는 case로 구현

- State : clock의 positive edge에 미리 계산된 "next_state"값을 sampling함

- Result : 간단한 연산 출력의 경우 assign 구문과 조건 연산(?)을 통해 구현 가능. 복잡해질 경우 always 구문 사용 가능

```
module L2P_converter (clk, rst_n, L, P);
input clk, rst_n;
input L;
output P;

reg [1:0] state, next_state;

// state transition part
always @ ( state or L )
begin
    case ( state )
    2'b00 : begin
        if ( L ) next_state = 2'b01;
        else next_state = 2'b00;
    end
    2'b01 :
        if ( L ) next_state = 2'b11;
        else next_state = 2'b00;
    2'b11 :
        if ( L ) next_state = 2'b11;
        else next_state = 2'b00;
    default : next_state = 2'b00;
    endcase
end

// state
always @ (posedge clk or negedge rst_n)
    if(!rst_n) state <= 2'b00;
    else state <= next_state;

// result
assign P = (state == 01)? 1'b1: 1'b0;

endmodule
```

Level-to-Pulse Converter 설계하기

3. Testbench 설계하기

* Testbench file 추가하기

  - Project 창에서 우클릭 => Add to Project => New File

  - TB_L2P_converter.v (Verilog)

* Scenario

  - clk : 주기 100MHz 인가 (period 10ns)

  - rst_n : 초반 20ns 동안 0으로 주어 리셋

  - 1 cycle(10ns) 이후 L => 1 (예상 state : 01), 1 cycle 이후 L => 0 (예상 state : 00)

  - 1 cycle 이후 L => 1 (예상 state : 01), 1 cycle 이후 L => 1 (예상 state : 11),

  - 1 cycle 이후 L => 1 (예상 state : 11), 1 cycle 이후 L => 0 (예상 state : 00)

```
`timescale 1ns/100ps

module TB_L2P_converter;
reg clk;
reg rst_n;
reg L;
wire P;

// Instantiation DUT
L2P_converter dut(.clk(clk), .rst_n(rst_n), .L(L), .P(P));

// Clock generation
initial clk = 0;
always #5 clk = ~clk;

// Scenario
initial
begin
    rst_n = 0; L = 0;
    #20 rst_n = 1; L = 0;
    #10 L = 1;
    #10 L = 0;
    #10 L = 1;
    #10 L = 1;
    #10 L = 1;
    #10 L = 0;
end
endmodule
```

Level-to-Pulse Converter 설계하기

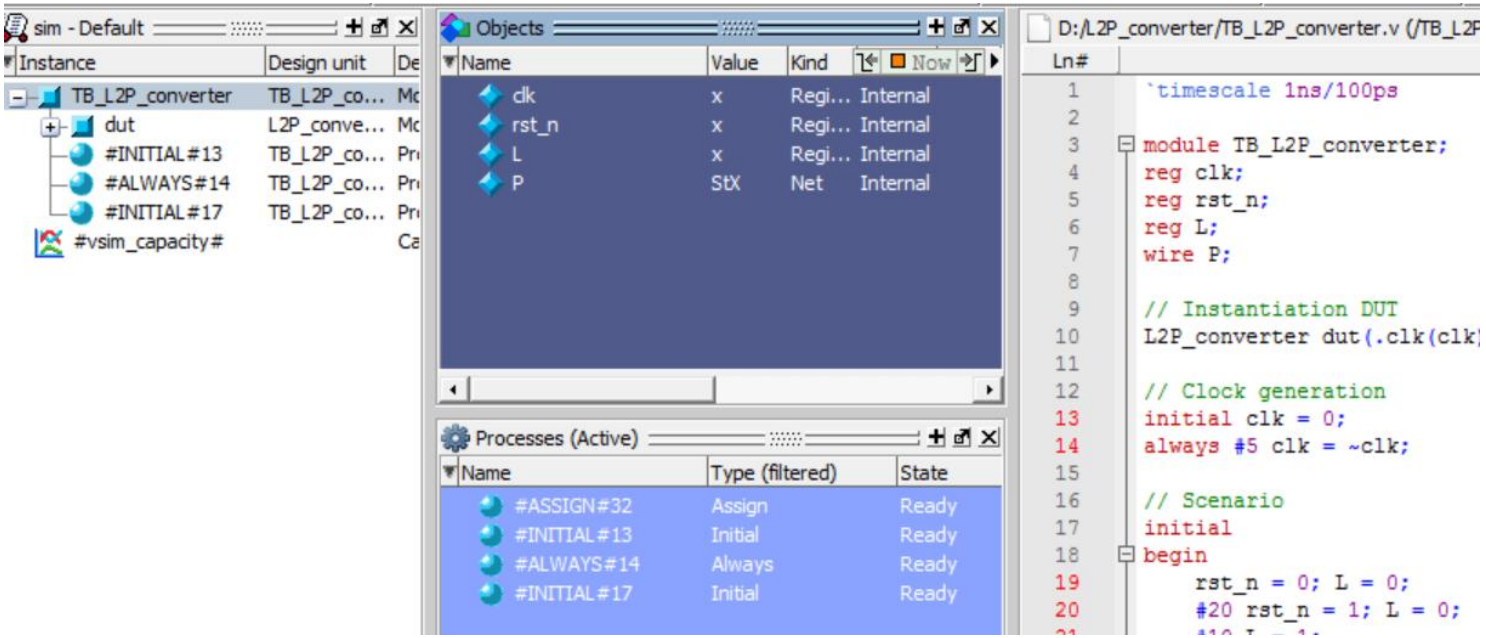
4. Simulation 수행하기

* Compile

  - Compile => Compile All 혹은 파일 선택 => Compile => Compile Selected : Check 표시 나올 경우 OK. Transcript 창에 Error 있을 경우 더블클릭하면 Error 내용 출력

* Simulate

  - Library 창 => work(default working space) => Testbench 우클릭 => Simulate



Level-to-Pulse Converter 설계하기

4. Simulation 수행하기

* Waveform 실행하기

  - Sim 창에서 dut(L2P_converter) 선택, Object 창에 dut의 input/output/wire/reg list 보임

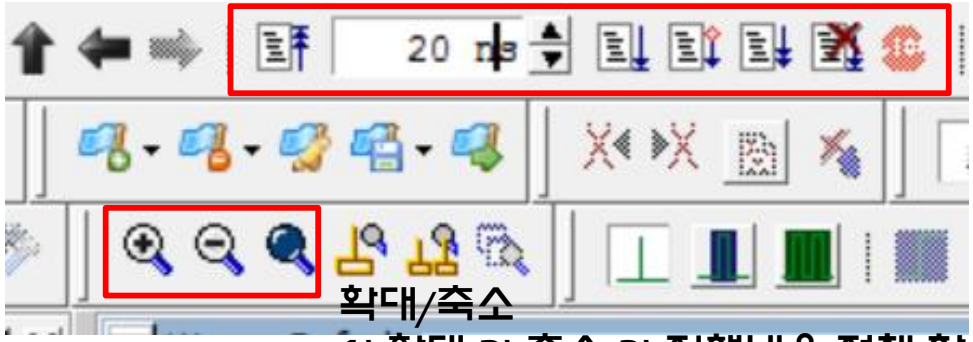
  - Waveform에서 확인할 signal 선택(ctrl, shift 사용 다중 선택 가능) => 우클릭 => Add Wave

Level-to-Pulse Converter 설계하기

4. Simulation 수행하기

* Simulation 진행하기

- Simulation unit time : 100 ps => 20 ns(임의 가능)



Simulation 진행 버튼
1) 재시작 2) 1 unit 진행 3) breakpoint까지 진행
4) 계속 진행 5) endpoint까지 진행 6) 멈춤

확대/축소
1) 확대 2) 축소 3) 진행내용 전체 한 화면에 표시

1. Quartus Tutorial

* File => New Project Wizard

  - Introduction : Next

  - Environment setting (1/5) : Project directory, name, top-entity name (L2P_converter)

  - Add files (2/5) : Modelsim에서 디자인을 완성한 경우 추가

  - FPGA setting (3/5) : Family – Cyclone, Device – EP1C6Q240C8, Target Device – Specific device..

  - EDA setting (4/5) : Next

  - Summary (5/5) : Finish

1. Quartus Tutorial

* Assignment => Pin Planner
- Node Name : 현재 디자인 top-level entity에 존재하는 pin
- Direction : Input/Output/Inout
- Location : FPGA pin mapping
  : Location double click, Drop-down menu에서 핀 선택

* Compile
- Analysis & Synthesis
- Fitter (Place & Route)
- Assembler (Generate Programming Files)
- TimeQuest Timing Analysis
- EDA Netlist Writer

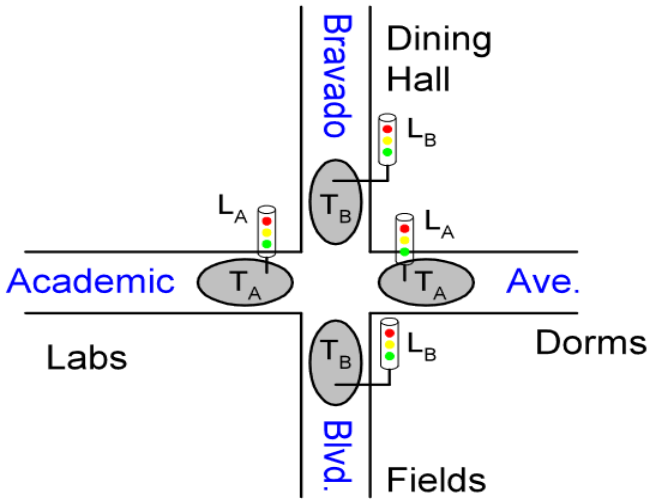
심화 설계 실습 – 교통 신호 시스템 설계

반도체설계교육센터
IC DESIGN EDUCATION CENTER

심화 설계 실습 – 교통 신호 시스템 설계

1. 교통 신호 시스템 SPEC.

* 입출력 신호 정의
  - 교통 감지(입력) : T_A, T_B
  - 신호기(출력) : L_A, L_B
  - Assignment => Pin Planner



* Scenario
  - 두 개 도로 모두 직진만 가능 : 즉, red, yellow, green 만 출력
  - Reset 후 Academic Ave.에 green light, Bravado Blvd.는 red light
  - 교통 감지 T_A 참(교통량 존재) : 계속 Academic Ave. green light
  - 교통량 T_A 거짓(교통량 없음) : Bravado Blvd. green light 전환
  - 교통량 T_B 거짓일 때까지 Bravado Blvd. green light
  - Green => red 신호 전환 사이에는 yellow light 출력 필요(중간 state)

심화 설계 실습 – 교통 신호 시스템 설계

2. Finite State Machine 정의하기

* FSM 그리기

Thank you