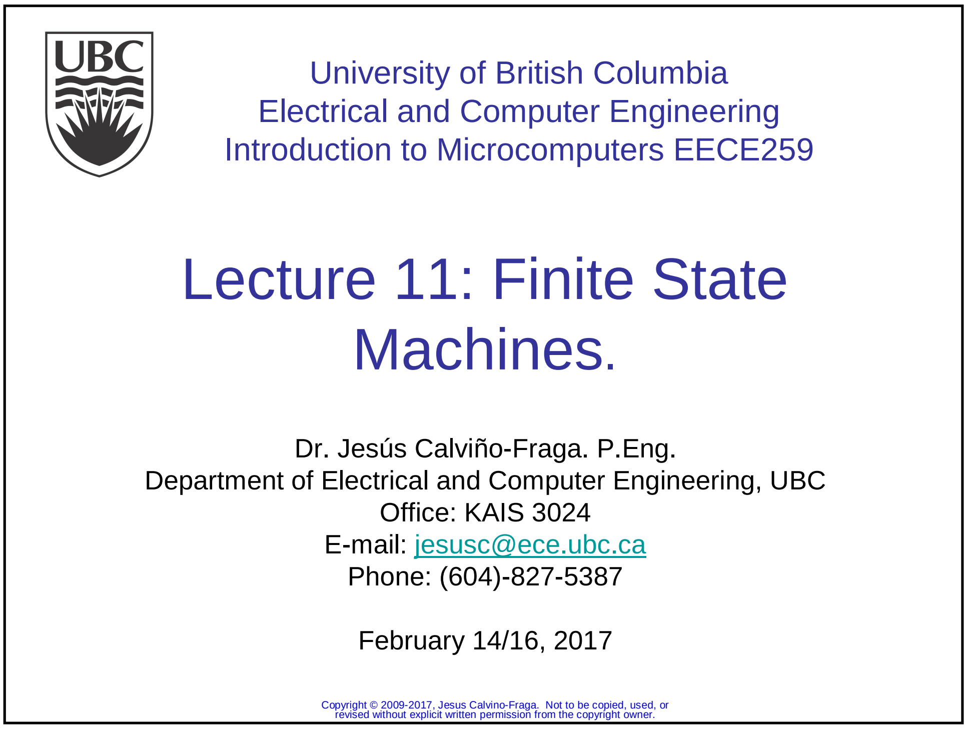
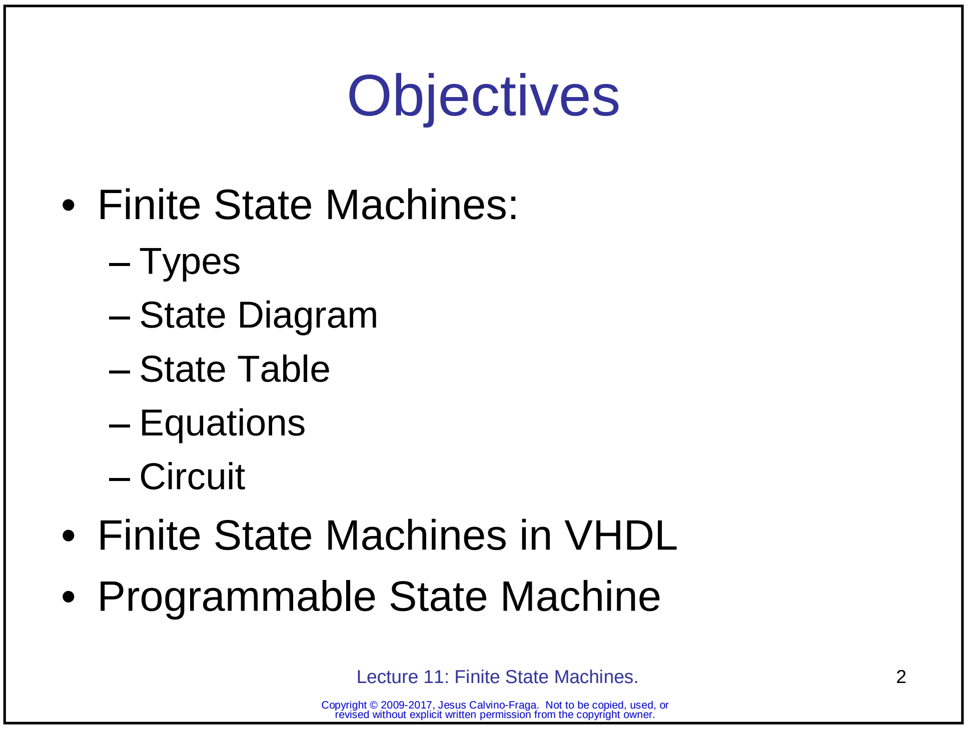


University of British Columbia
Electrical and Computer Engineering
Introduction to Microcomputers EECE259

Lecture 11: Finite State Machines.

Dr. Jesús Calviño-Fraga. P.Eng.
Department of Electrical and Computer Engineering, UBC
Office: KAIS 3024
E-mail: jesusc@ece.ubc.ca
Phone: (604)-827-5387

February 14/16, 2017



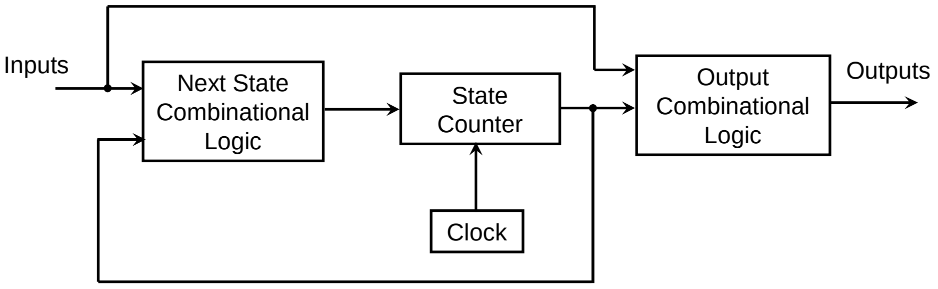
Objectives

- Finite State Machines:
  - – Types
  - – State Diagram
  - – State Table
  - – Equations
  - – Circuit
- Finite State Machines in VHDL
- Programmable State Machine

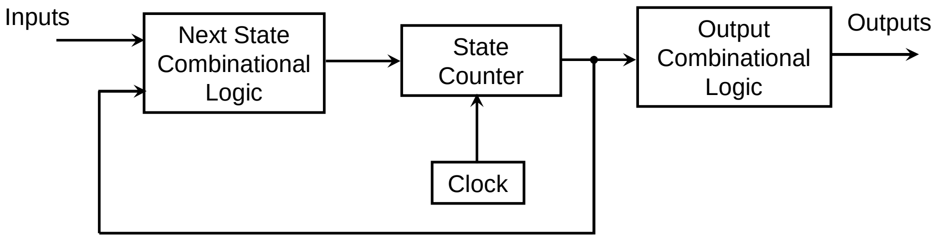
Finite State Machines

- Finite State Machines (FSM) are a mathematical abstraction used to design sequential logic circuits as well as to write computer programs.
- In the previous lecture we studied synchronous counters. They are FSMs! The out of each state in a counter is the number of the state.
- There are two kinds of FSMs:
  - FSMs whose outputs are only a function of the current state. They are known as Moore FSMs. They are synchronous FSMs. The synchronous counters we studied in the previous lecture are all Moore FSMs.
  - FSMs whose outputs are a both a function of the current state and the inputs. They are know as Mealy FSMs they are asynchronous FSMs.
- The design of FSMs is very similar to the design of arbitrary count synchronous counters.

Mealy Type FSM



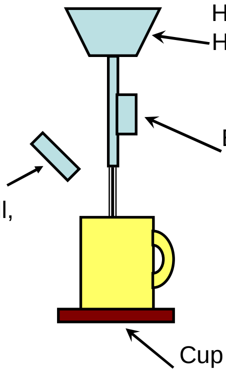
Moore Type FSM



Design Example 1

- Design a FSM that controls the automatic ~~coffee~~ dispenser shown below

Hot Chocolate



Heater and temperature sensor. H=1:Heat on, H=0 Heat off, T=1 hot, T=0 Not hot enough

Electromechanical valve. V=1:open. V=0:Close,

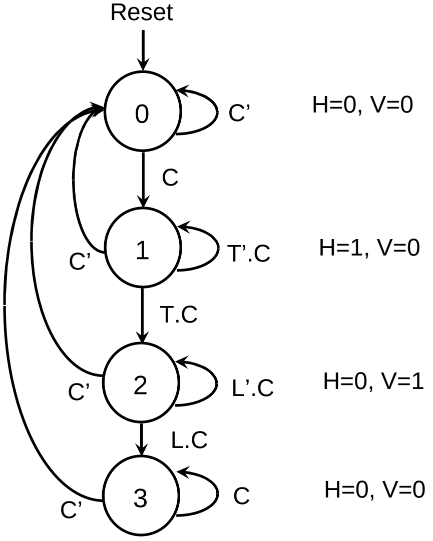
Ultrasonic Level Detector. L=1:Cup Full. L=0:Cup not Full,

(Beverage tank not shown)

Cup Detector. C=1:Cup Present, C=0: No Cup

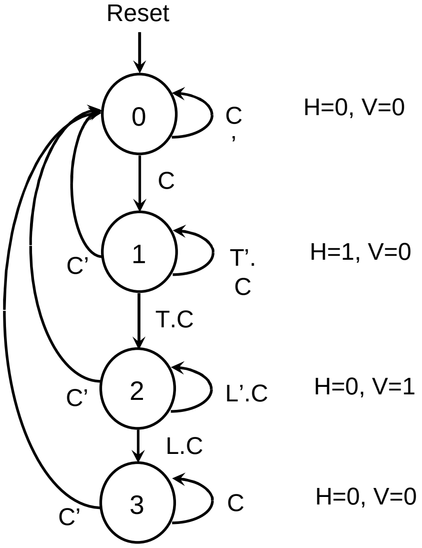
State Diagram

C: Cup sensor
L: Cup Full sensor
V: Valve on/off
H: Heater on/off
T: Temperature sensor



State Table

Inputs			State		Next		Outputs	
C	L	T	Q_1	Q_0	D_1	D_0	H	V
0	X	X	0	0	0	0	0	0
1	X	X	0	0	0	1	0	0
0	X	X	0	1	0	0	1	0
1	X	0	0	1	0	1	1	0
1	X	1	0	1	1	0	1	0
0	X	X	1	0	0	0	0	1
1	0	X	1	0	1	0	0	1
1	1	X	1	0	1	1	0	1
0	X	X	1	1	0	0	0	0
1	X	X	1	1	1	1	0	0



Get Equations

$$D_1=C.T.Q'_1.Q_0+C.L'.Q_1.Q'_0+C.L.Q_1.Q'_0+C.Q_1.Q_0 =$$

$$D_1=C.T.Q'_1.Q_0+C.Q_1.Q'_0+C.Q_1.Q_0 =$$

$$D_1=C.T.Q'_1.Q_0+C.Q_1 =$$

$$D_1=C.(T.Q'_1Q_0+Q_1)$$

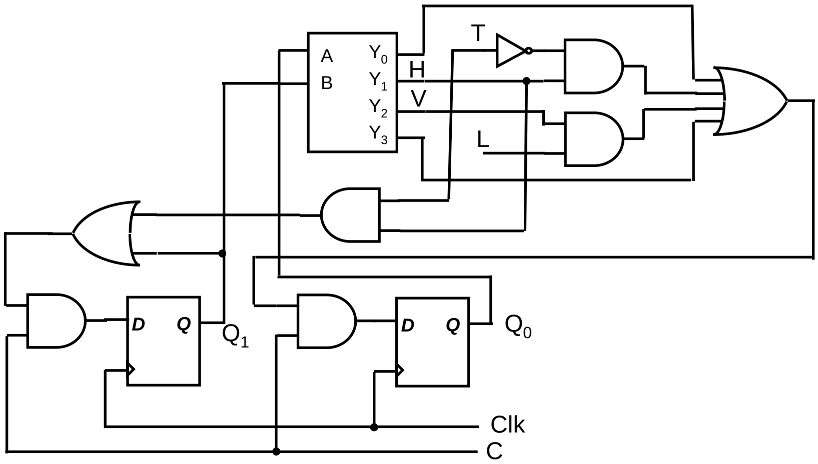
$$D_0=C.Q'_1.Q'_0+C.T'.Q'_1.Q_0+C.L.Q_1.Q'_0+C.Q_1.Q_0 =$$

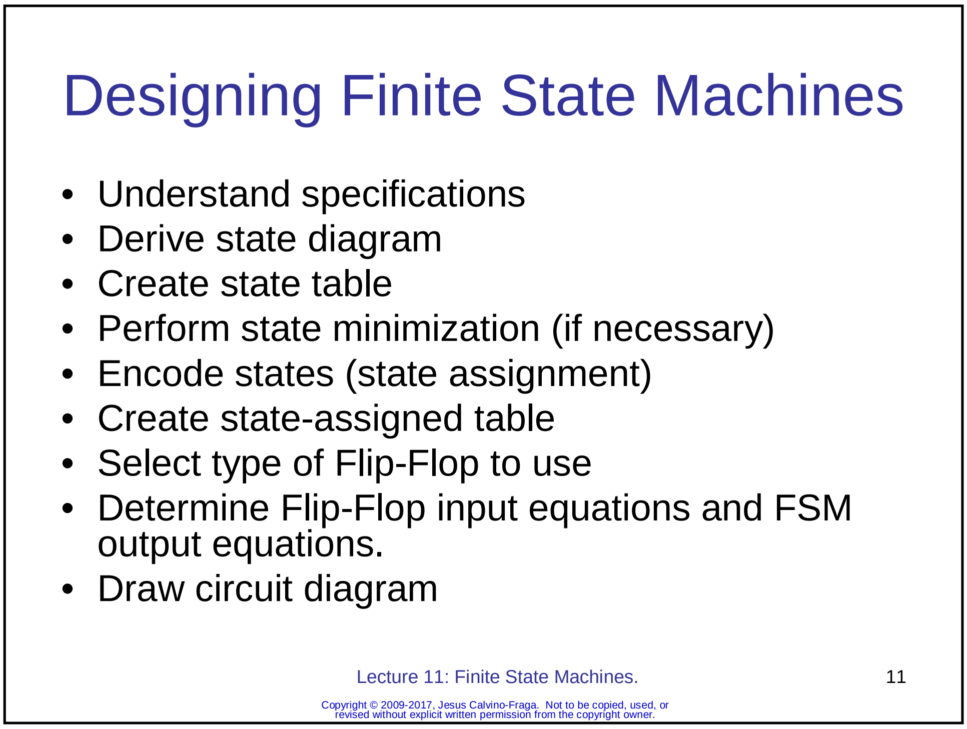
$$D_0=C.(Q'_1.Q'_0+T'.Q'_1.Q_0+L.Q_1.Q'_0+Q_1.Q_0)$$

$$H=Q'_1.Q_0$$

$$V=Q_1.Q'_0$$

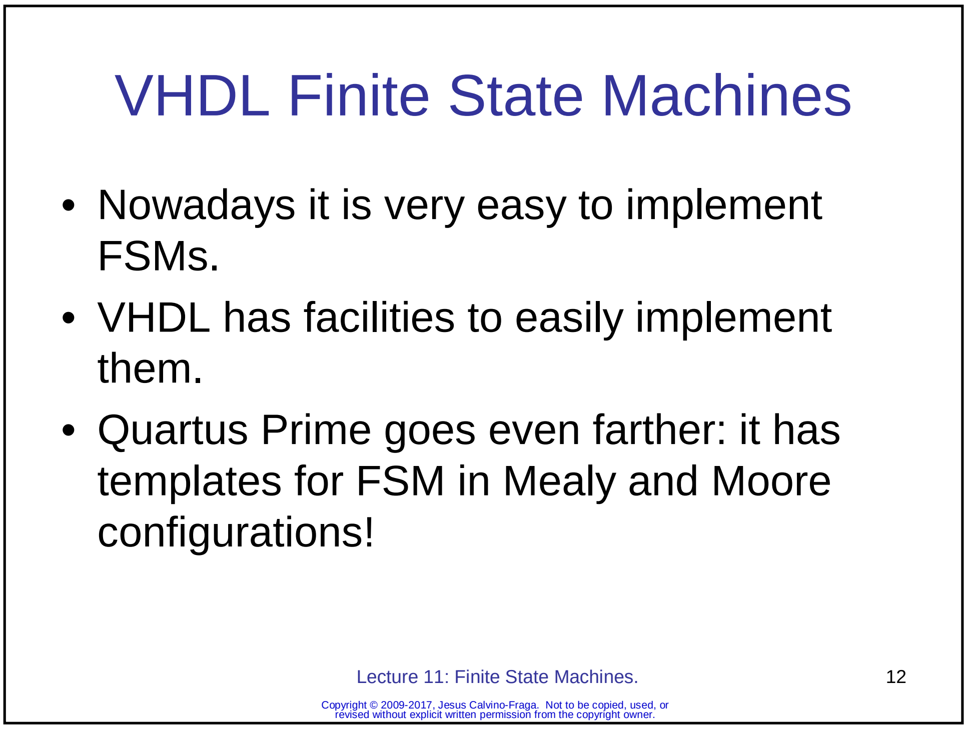
Draw the Circuit





Designing Finite State Machines

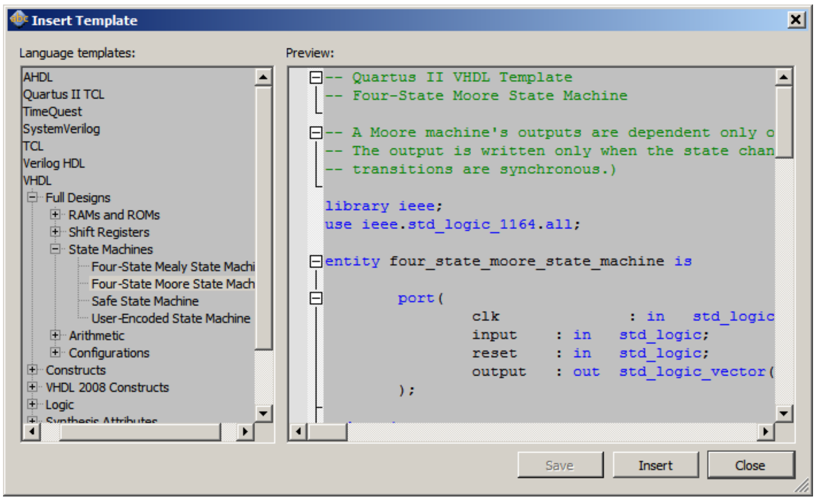
- Understand specifications
- Derive state diagram
- Create state table
- Perform state minimization (if necessary)
- Encode states (state assignment)
- Create state-assigned table
- Select type of Flip-Flop to use
- Determine Flip-Flop input equations and FSM output equations.
- Draw circuit diagram



VHDL Finite State Machines

- Nowadays it is very easy to implement FSMs.
- VHDL has facilities to easily implement them.
- Quartus Prime goes even farther: it has templates for FSM in Mealy and Moore configurations!

VHDL Finite State Machines



```
-- Quartus II VHDL Template
-- Four-State Moore State Machine

-- A Moore machine's outputs are dependent only o
-- The output is written only when the state chan
-- transitions are synchronous.)

library ieee;
use ieee.std_logic_1164.all;

entity four_state_moore_state_machine is

        port(
                clk             : in    std_logic
                input   : in    std_logic;
                reset   : in    std_logic;
                output  : out   std_logic_vector(
        );
```

VHDL FSM

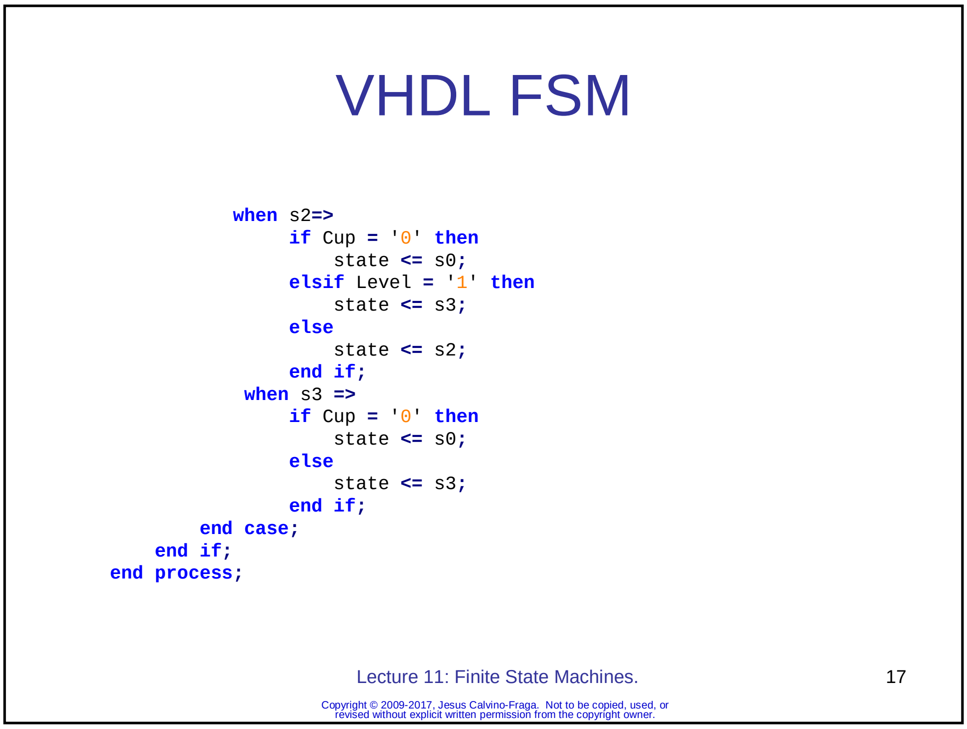
```
-- Quartus II VHDL Template
-- Four-State Moore State Machine

-- A Moore machine's outputs are dependent only on the current state.
-- The output is written only when the state changes.  (State
-- transitions are synchronous.)

library ieee;
use ieee.std_logic_1164.all;

entity fsm1 is

    port(
        clk, reset              : in  std_logic;
        Cup, Level, Temperature : in  std_logic;
        Ready                   : out std_logic;
        Heater, Valve           : out std_logic
    );

end entity;
```

VHDL FSM

```
architecture rtl of fsm1 is

    -- Build an enumerated type for the state machine
    type state_type is (s0, s1, s2, s3);

    -- Register to hold the current state
    signal state   : state_type;

begin
```

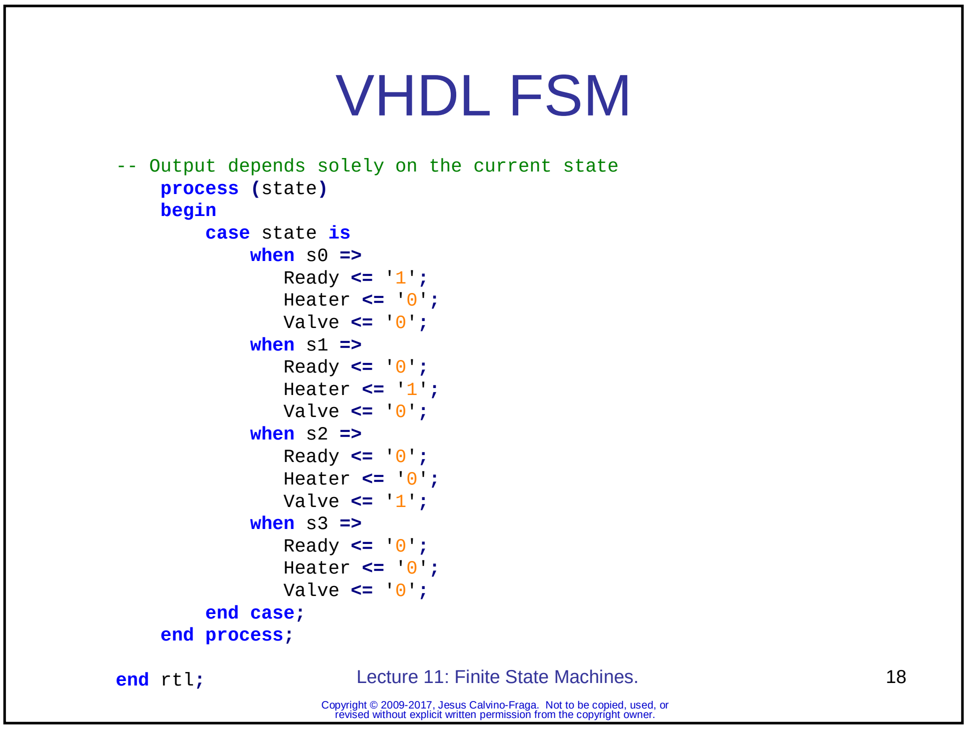
VHDL FSM

```
-- Logic to advance to the next state
    process (clk, reset)
    begin
        if reset = '0' then
            state <= s0;
        elsif (rising_edge(clk)) then
            case state is
                when s0=>
                    if Cup = '1' then
                        state <= s1;
                    else
                        state <= s0;
                    end if;
                when s1=>
                    if Cup = '0' then
                        state <= s0;
                    elsif Temperature = '1' then
                        state <= s2;
                    else
                        state <= s1;
                    end if;
```



VHDL FSM

```
           when s2=>
                if Cup = '0' then
                    state <= s0;
                elsif Level = '1' then
                    state <= s3;
                else
                    state <= s2;
                end if;
            when s3 =>
                if Cup = '0' then
                    state <= s0;
                else
                    state <= s3;
                end if;
        end case;
    end if;
end process;
```



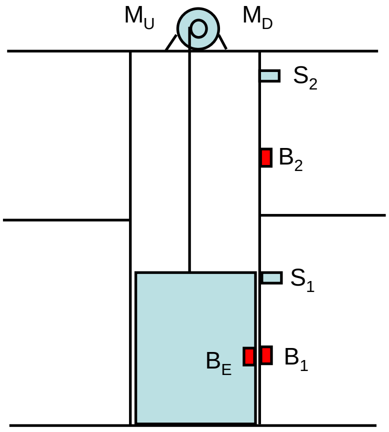
VHDL FSM

```
-- Output depends solely on the current state
    process (state)
    begin
        case state is
            when s0 =>
               Ready <= '1';
               Heater <= '0';
               Valve <= '0';
            when s1 =>
               Ready <= '0';
               Heater <= '1';
               Valve <= '0';
            when s2 =>
               Ready <= '0';
               Heater <= '0';
               Valve <= '1';
            when s3 =>
               Ready <= '0';
               Heater <= '0';
               Valve <= '0';
        end case;
    end process;

end rtl;
```

Pin Assignments in DE0-CV Board

Signal Name	DE0-CV Board Name	FPGA Pin #
clk	50 MHz Clock	PIN_M9
reset	KEY0 (active low!)	PIN_U7
Cup	SW0	PIN_U13
Level	SW1	PIN_V13
Temperature	SW2	PIN_T13
Ready	LEDR0	PIN_AA2
Heater	LEDR1	PIN_AA1
Valve	LEDR2	PIN_W2

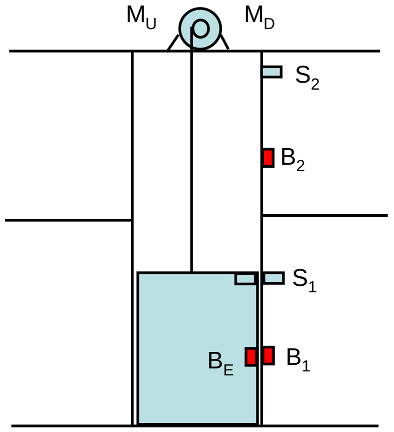
Example 2: Simple Elevator

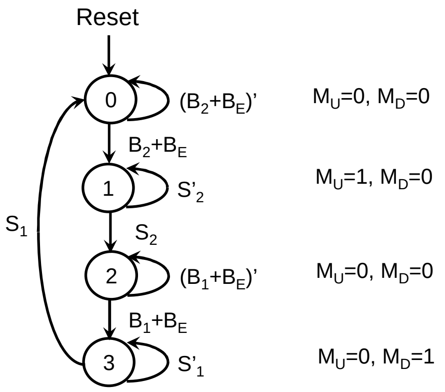


- M_U and M_D make the elevator go up or down.
- S_1 and S_2 detect if the elevator is in floor one or two
- B_1 and B_2 are used to call the elevator to each floor
- B_E is used to make the elevator move to the other floor.

After Reset the elevator is in the bottom floor

Example 2: Simple Elevator





Example 2: State Table

B_1+B_E	B_2+B_E	S_1	S_2	Q_1	Q_0	D_1	D_0	M_U	M_D
X	0	X	X	0	0	0	0	0	0
X	1	X	X	0	0	0	1	0	0
X	X	X	0	0	1	0	1	1	0
X	X	X	1	0	1	1	0	1	0
0	X	X	X	1	0	1	0	0	0
1	X	X	X	1	0	1	1	0	0
X	X	0	X	1	1	1	1	0	1
X	X	1	X	1	1	0	0	0	1

$$D_1=S_2.Q'_1.Q_0+(B_1+B_E)'Q_1.Q'_0+(B_1+B_E).Q_1.Q'_0+S'_1.Q_1.Q_0$$

$$D_0=(B_2+B_E).Q'_1.Q'_0+S'_2.Q'_1.Q_0+(B_1+B_E).Q_1.Q'_0+S'_1.Q_1.Q_0$$

$$M_U=Q'_1.Q_0$$

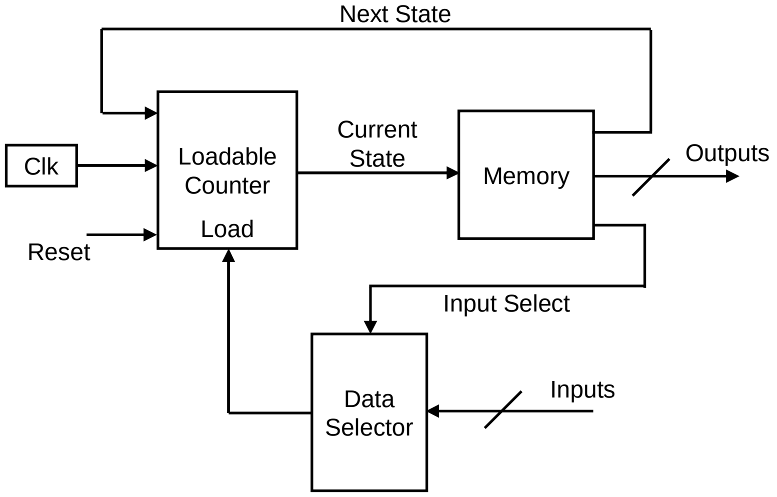
$$M_D=Q_1.Q_0$$

Circit is exercise
at the end!

Programmable State Machine

- From the previous examples notice that the hardware implementations of FSMs may get very complex.
- For large FSM it takes a long time to design and assemble.
- An alternative is to use VHDL, but we are just moving the problem from one technology to another.
- A better option is to have a FSM that is easy to design with and easy to setup.
- A programmable state machine is both easy to design with and setup. It is the direct ancestor of modern processors!

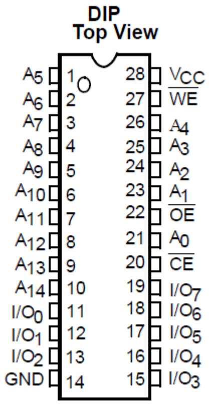
Programmable State Machine



Memory

- Typically, memory has the following signals:
  - An address bus. This allows us to select each particular memory location.
  - A data bus. Here is where we put the information to be stored to the memory, or where we get the information previously stored.
  - A control bus. These are the signals that tell the memory to write, read, or disable/enable the whole memory.

Random Access Memory (RAM)



CY62256N

256K (32K x 8) Static RAM

Address Bus: Signals A_0 to A_{14}. Since we have 15 address signals, the memory has 2^{15}=32768 locations.

Data Bus: Signals I/O_0 to I/O_7. It is an 8-bit data bus.

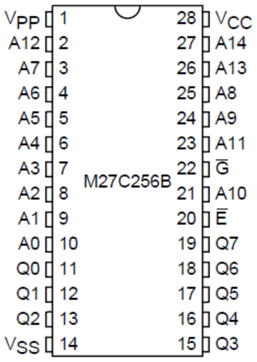
Control Bus: Signals CE*, OE*, and WE*.

CE*: Enables this IC for read or write.

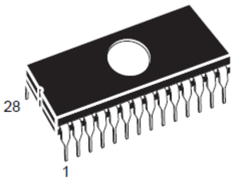
OE*: Makes the data bus an output. Read from memory.

WE*: Makes the data bus an input. Write to memory

Read Only Memory (ROM)



256 Kbit (32Kb × 8) UV EPROM (Ultra Violet Electrically Programmable Read Only Memory)



Address Bus: Same as previous slide.

Data Bus: Signals Q_0 to Q_7.

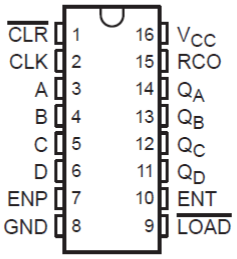
Control Bus: Signals E*, G*, and VPP.

E*: Enables this IC for read.

G*: Makes the data bus an output.

VPP: Voltage (12.5V) used to write to the memory using an external programmer.

Loadable Counter



74HC161 4-bit synchronous binary counter

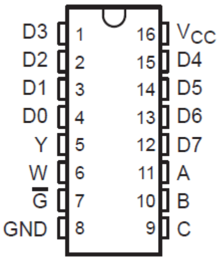
The ‘161 is one of many counters available. Not happy with this counter? Design your own using flip-flops or VHDL!

According to the datasheet, to count: ENT, ENP, LOAD*, and CLR* must be ‘1’.

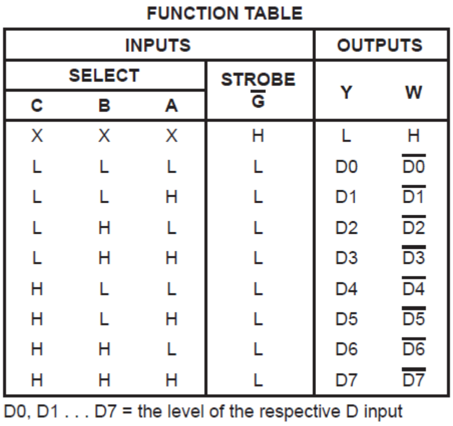
When LOAD*=0, the current count is set to whatever is in inputs A to D.

Signals RCO, ENT, and ENP are used to connect several chips together to make a bigger counter (8-bits, for example)

Data Selector



74HC151 8 to 1 data selector



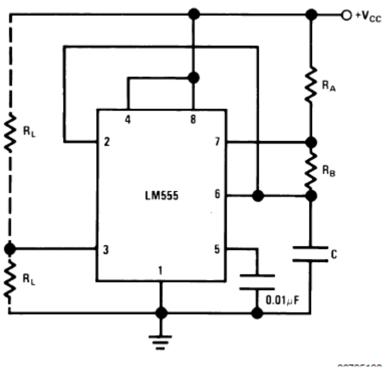
FUNCTION TABLE

INPUTS				OUTPUTS	
SELECT			STROBE $\overline{G}$	Y	W
C	B	A			
X	X	X	H	L	H
L	L	L	L	D0	$\overline{D0}$
L	L	H	L	D1	$\overline{D1}$
L	H	L	L	D2	$\overline{D2}$
L	H	H	L	D3	$\overline{D3}$
H	L	L	L	D4	$\overline{D4}$
H	L	H	L	D5	$\overline{D5}$
H	H	L	L	D6	$\overline{D6}$
H	H	H	L	D7	$\overline{D7}$

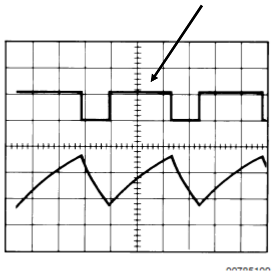
D0, D1 . . . D7 = the level of the respective D input

Clock Generation

Old good LM555 IC in A-stable configuration!

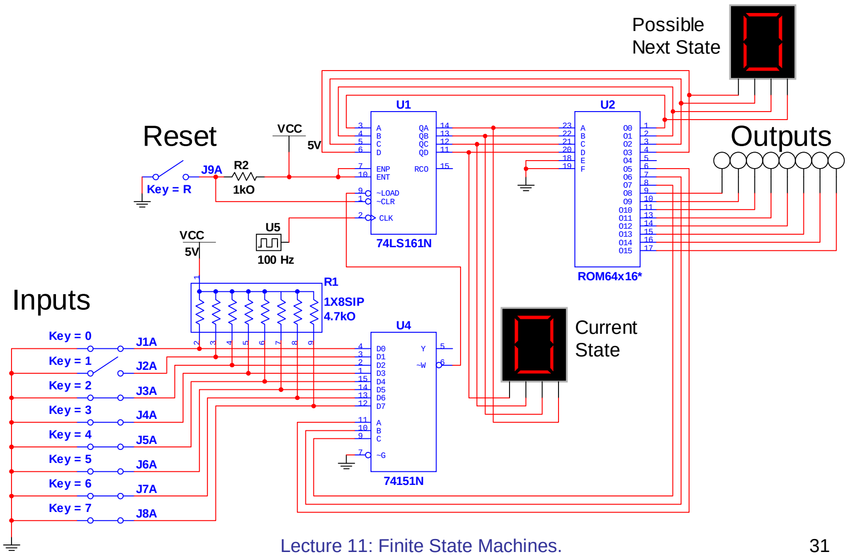


Output at pin 3



$$f = \frac{1.44}{(R_A + 2R_B)C}$$

PSM: Simulation using Multisim



PSM: Simulation using Multisim

- The program is stored in the 64x16 ROM (64 words, 16-bit each).
- To change the program in Multisim, double click the memory, then click “Edit Model”.
- Modify the output of the memory for each location according to your program. Then click “Change part model”, followed by Ok.
- To simulate the Programmable State Machine press F5.

PSM: Operation Code

- Each instruction in our programmable state machine consist of 16-bits with the following meanings:

Q15	Q14	Q13	Q12	Q11	Q10	Q9	Q8	Q7	Q6	Q5	Q4	Q3	Q2	Q1	Q0
U7	U6	U5	U4	U3	U2	U1	U0	IC	IB	IA	X	ND	NC	NB	NA

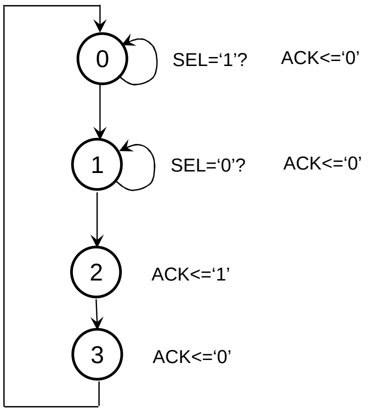
General purpose outputs

Input select

Possible next state, depending of input

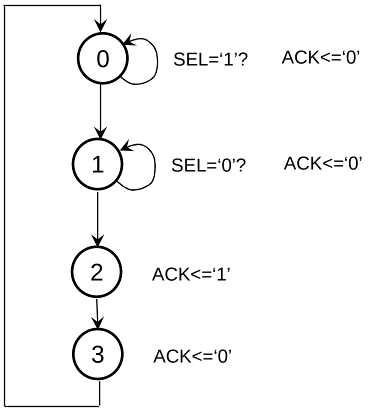
PSM Example

- Program the following state diagram:



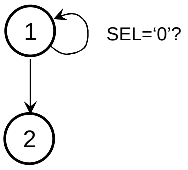
PSM Example

- Some observations:
  - State 2 increments unconditionally to state 3. Therefore we need to force an input to always zero. Say it is input zero.
  - State 3 jumps unconditionally to state 0. Therefore we need to force an input to always one. Say it is input one.
  - State 1 jumps if the condition is zero. Our PSM can jump only with one. We can solve this in different ways as outlined in the next slide.

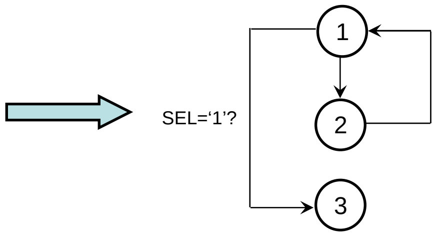


PSM: How to jump when an input is zero.

- Use an inverter (NOT gate) connected to an extra input.
- Add an extra state:

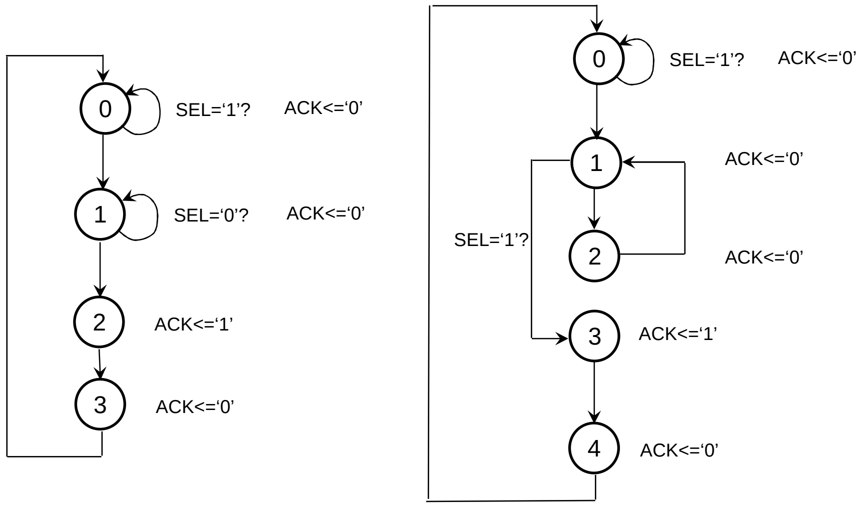


Not implementable with current design



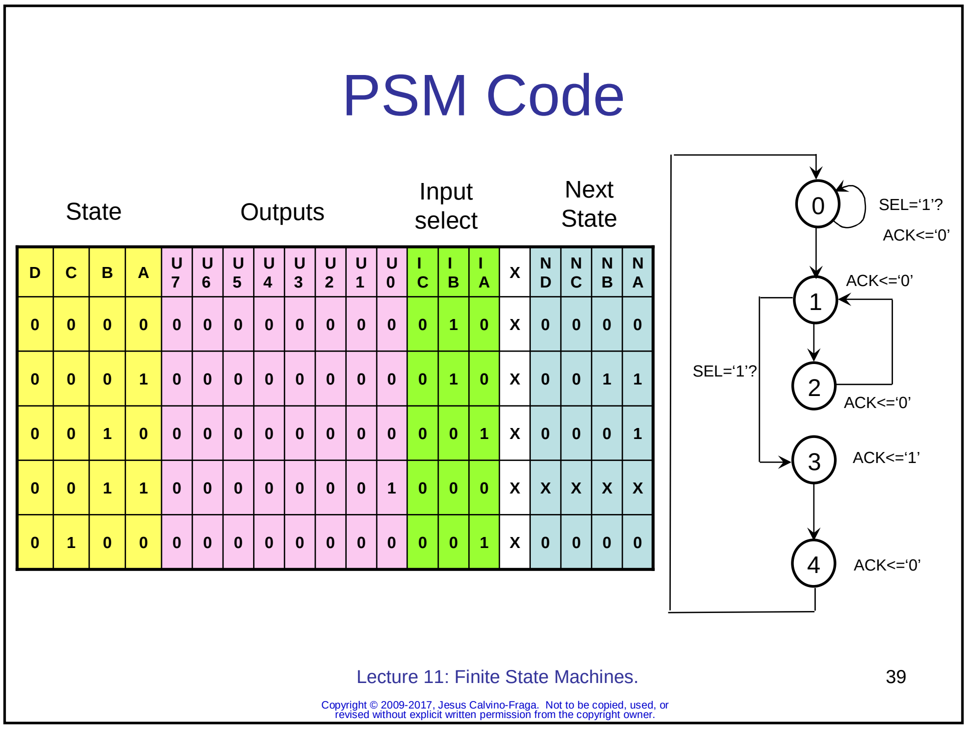
Implementable!

PSM with converted state



PSM: Signal Assignments

Signal	PSM
'0'	Input 0 (I_A=0, I_B=0, I_C=0)
'1'	Input 1 (I_A=1, I_B=0, I_C=0)
'SEL'	Input 2 (I_A=0, I_B=1, I_C=0)
'ACK'	Output 0 (U0)

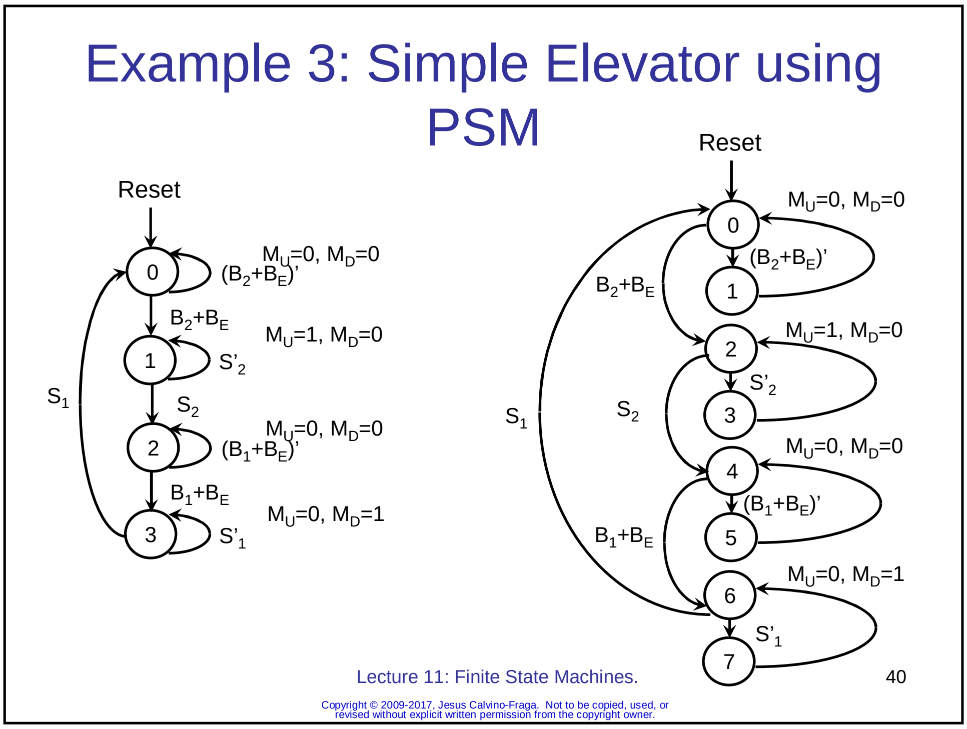


PSM Code

State				Outputs								Input select				Next State			
D	C	B	A	U 7	U 6	U 5	U 4	U 3	U 2	U 1	U 0	I C	I B	I A	X	N D	N C	N B	N A
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	X	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	X	0	0	1	1
0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	X	0	0	0	1
0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	X	X	X	X	X
0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	X	0	0	0	0

0
SEL='1'?
ACK<='0'
1
ACK<='0'
SEL='1'?
2
ACK<='0'
3
ACK<='1'
4
ACK<='0'

Lecture 11: Finite State Machines.



Example 3: Simple Elevator using PSM

Lecture 11: Finite State Machines.

Example 3: Simple Elevator using PSM

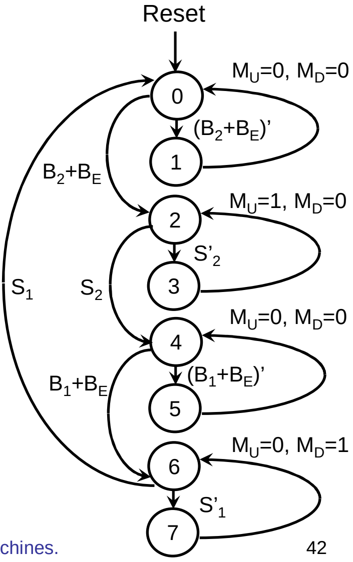
Signal	PSM
0	Input 0
1	Input 1
B_1+B_E	Input 2
B_2+B_E	Input 3
S_1	Input 4
S_2	Input 5
M_U	Output 0 (U0)
M_D	Output 1 (U1)

Pin Assignments

Example 3: Simple Elevator using PSM

B1+BE	Input 2
B2+BE	Input 3
S1	Input 4
S2	Input 5

D	C	B	A	U 7	U 6	U 5	U 4	U 3	U 2	M D	M U	I C	I B	I A	X	N D	N C	N B	N A
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	X	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	X	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	1	1	0	1	X	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	1	0	0	1	X	0	0	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	X	0	1	1	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	X	0	1	0	0
0	1	1	0	0	0	0	0	0	0	1	0	1	0	0	X	0	0	0	0
0	1	1	1	0	0	0	0	0	0	1	0	0	0	1	X	0	1	1	0



Exercises

1. Draw the circuit for example 2 given in class.
2. Write a VHDL program that implements the state diagram of example 2.
3. Solve example 1 using the PSM given in class.
4. Using a two input XOR gate and memory output ‘U7’, modify the PSM presented in this lecture so it can be programmed to jump either with zero or one.