**Finite State Machines**

Combinational circuits just depend on inputs. By just looking at input, you can see the outputs. At each time, when you see the same inputs, you will get the same outputs. There is no any changing because outputs definitely depend on inputs.

In sequential circuits it is different because output also depends on the current state. Output depends on input and output. You cannot represent the output of the sequential circuit by using boolean algebra.

In sequential circuits:

If you press on a button, output 1 for 3 cycles.

If you press on a button in 1, 0, 1, 1 order (press, wait 1 cycle, press 2 times); output 1 for 2 cycles.

Controller

Sequential circuit that controls the output depending on current inputs and time-ordered behaviour.

Example:

Consider a circuit that outputs 1 for 30 ns when a button is pressed (assume that a cycle is 10 ns).

That means I will press button for 10 ns, and then output is going to be 1 for 3 cycles (30 ns).

Diagram

Description automatically generated



At first clock signal Q1 is going to be 1 and Q is going to be 1.  
At next clock signal Q2 is going to be 1, Q1 is going to be 0 (I pressed the button for 1 clock cycle), and Q is going to be 1 again.  
At next clock signal Q3 is going to be 1, Q1 & Q2 are going to be 0, and Q is going to be 1 again.

There is a problem: What if within the 3 cycle, user press the button again?

For example, let Q2 be 1 and Q1 & Q3 be 0. User press the button at this situation. I don’t see the effect of this press because output is already 1. Output will be 1 for 20 ns because of the previous press. I cannot see the real effect of last press.

Also if you always press 1, you always get 1 at the output.

Thought that this is a laser scanner. If you press on the button, it should be active only for 3 cycles. More than 3 cycles laser output could be dangerous for a person. If you press 2 times that means you can give 5 cycles at the output.

We should do like this: If you are within 3 cycles, don’t care about the inputs. We should define behaviour of the circuit.

We prevent this by finite state machines. We are not gonna draw, we just focus on the behaviour. From behaviour, we can design the circuit.

Finite State Machine

In pandemic conditions what you do are:

* sleeping (state 0)
* eating (state 1)
* studying (state 2)
* watching (stste 3)

x : actions / outputs  
arrow : initial state - where I have to start  
clk^ : you will move at the rising edge

I have to move from one state to other one. I can move by using the clock signal.

Diagram

Description automatically generated

For each finite state machines, we will have 4 things:

1. states
2. inputs
3. outputs / actions
4. transitions / arrows

Diagram

Description automatically generated

Finite state machines are not machines, they are just diagrams. They are not a device nor circuit.

By using that state diagrams, we will design a circuit.

You can have more than 1 outputs.

A picture containing whiteboard

Description automatically generated

Example:

When you press on a button, we will get 1 at the outputs for 3 cycles. If you press again during that cycles we will ignore that one.

Diagram

Description automatically generated

ON1 ON2 ON3

Diagram

Description automatically generated

x = 1

x = 1

x = 1

x = 0

A picture containing text, shoji

Description automatically generatedSo b (input) doesn’t have any effect here. It can be 0 or 1.

*clk^ will be always there because you are activating the circuit by clock signal. We don’t need to show it.*

Example:

Consider a color coded keypad in a hospital door. When a person press red, blue, green, and red button in order, the door opens. As shown in figure:

Diagram

Description automatically generated

(output)

a = 1 when a button is pressed.

Diagram

Description automatically generated

**u = 0**

What happens when you at start state, somebody press red, green, and blue at the same time?

You move to red1 step. If somebody press again 3 buttons at the same time, you will move to green state, …. The user can open the door with doing this at every state.

You can solve this by that:

Diagram

Description automatically generated

But if you are at the start, if you press red-green-blue at the same time, ar and a(r’+g+b) conditions are both satisfied. So you have to change this like that:

Diagram

Description automatically generated

red1’deyken green ve blue’ya basıldı diyelim. Bu sefer waite gitmeli. Yani a(g’+r+b) gerekli.

Only one of the arrows must be true.

Implementing Finite State Machine as a Sequential Circuit

We will use registers to store the states of a circuit and we will use a combinational circuit to implement.

Example:

For a FSM with 4 states and a button input, how could it be?

At different states, x should give different outputs. I want to manipulate combinational circuit. I want it to behave depending on state.

I can represent 4 states by 2 input. So I have 2 inputs (s1, s0) for states.

I will change states and depending on the states different combinations are made. At each output, states can be changed. Depending on the new states, I will feed that states to the combinational circuit again. So I have to store the states as well.

s1, s0 🡪 current states  
n1, n0 🡪 next states

Diagram

Description automatically generated

**Steps to Design Circuit from FSM**

1. Capture / Create FSM
2. Create the architecture (List what you have)
3. Encode the states
4. Create the state table
5. Implement the combinational logic

**Example:**

When you press on a button, we will get 1 at the outputs for 3 cycles. If you press again during that cycles we will ignore that one.

STEP 1

Diagram

Description automatically generated

STEP 2

* 4 states
* b is input
* x is output
* 2 bit state register (because we have 4 states)

STEP 3

|  |  |
| --- | --- |
|  | s1s0 |
| OFF | 00 |
| ON1 | 01 |
| ON2 | 10 |
| ON3 | 11 |

STEP 4

Inputs for combinational circuit : s1, s0, b

Outputs for combinational circuit : n1, n0, x

I have formulized combinational circuit and it has 3 input, 3 outputs.

*(Recall n1 and n0 are next steps)*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| s1 | s0 | b | n1 | n0 | x |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 |

Diagram

Description automatically generated

We split inputs by green-red-green-red for each state 🡪 00 (OFF), 01 (ON1), 10 (ON2), 11 (ON3)

STEP 5

n1 = s1’s0b’ + s1’s0b + s1s0’b’ + s1s0’b = s1’s0(b’+b) + s1s0’(b’+b) = s1’s0 + s1s0’ = s0 s1

n0 = s1’s0’b + s1s0’

x = s1 + s1’s0 = s1 + s0 🡪 son 4 satır sadece s1’e bağlı olduğundan direkt s1 yazdık

We can also simplify x in the K-map:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| s0b  s1 | 00 | 01 | 11 | 10 |
| 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

So 🡪 x = s1 + s0

A picture containing text, whiteboard

Description automatically generated

Bold 2 gates are AND; other ones are OR, XOR.

EXAMPLE: Design a button which produces 1 for a duration of 1 cycle and returns 0

Diagram

Description automatically generated

Zile bir kez bastığında basılı tutsa bile 1 kez çalsın. Zile basıp bıraktıktan sonra tekrar basarsa yine zil 1 kez çalsın.

Step 1: Draw FSM

A picture containing text, map, wire

Description automatically generated

Diagram

Description automatically generated

Step 2: Architecture

* 3 states
* 2 bit register
* bi : input
* bo : output

Step 3: Encoding

WAIT: 00  
ON : 01  
OFF : 10

Step 4: State table

WAIT

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| s1 | s0 | bi | n1 | n0 | bo |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0  ON  WAIT | 1  WAIT | 0  WAIT | 0  WAIT | 0 | 1  WAIT |
| 0  OFF | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1  X | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | x | x | x |
| 1 | 1 | 1 | x | x | x |

Step 5: Implement the combinational logic

bo ----> s1’s0

n0 ----> s1’s0’bi

n1 ----> s1’s0bi + s1s0’bi = bi(s1 ⊕ s0)

Diagram

Description automatically generated

EXAMPLE: Design a sequence generator that will produce sequence 0001, 0011, 1100, 1000 in order and after 1000 it returns 0001 again.

* Diagram

  Description automatically generated4 states
* 2 input register
* 6 outputs (4 data + 2

register outputs)

A picture containing diagram

Description automatically generatedA : 00  
B : 01  
C : 10  
D : 11

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| s1 | s0 | n1 | n0 | x | y | z | w |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |

n1 = s1 ⊕ s0

n0 = s1’s0’ + s1s0’ = s0’(s1’+s1) = s0’

x = s1

y = s1s0’

z = s1’s0

w = s1’

EXAMPLE: The car keys have a chip inside (we are designing this chip), includes a code and when the drive turns the key, the car’s computer ask key to send the embedded code via radio signal and checks it. If it is correct it will turn on the engine. Assume the embedded code is 1101 and the behaviour in time diagram is as follows:

Diagram, engineering drawing

Description automatically generated

Diagram, schematic

Description automatically generated

We have:

* 5 states
* 1 input (a)
* 1 output (b)
* 3 bit register

Diagram, engineering drawing

Description automatically generated

🡪 This is controller

wait : 000  
c1 : 001  
c2 : 010  
c3 : 011  
c4 : 100

WAIT

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| s2 | s1 | s0 | a | n2 | n1 | n0 | b |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0  C1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0  C2 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0  C3 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1  C4 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | x | x | x | x |
| 1  X | 0 | 1 | 1 | x | x | x | x |
| 1 | 1 | 0 | 0 | x | x | x | x |
| 1  X | 1 | 0 | 1 | x | x | x | x |
| 1 | 1 | 1 | 0 | x | x | x | x |
| 1  X | 1 | 1 | 1 | x | x | x | x |

X: unused

n2 : s2’s1s0

n1 : s2’s1s0’ + s2’s1’s0

n0 : s2’s1s0’ + s2’s1’s0’a

b : s2’s1s0’ + s2’s1’s0 + s2s1’s0’

EXAMPLE: Convert a sequential circuit to a FSM.

Diagram

Description automatically generated

STEP 1: Get equation for each output

y = s1’

z = s1s0’

n1 = (s1 ⊕ s0)x = (s1’s0 + s1s0’)x = s1’s0x + s1s0’x

n0 = s1’s0’x

STEP 2: Create state table

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| s1 | s0 | x | n1 | n0 | y | z |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 |

STEP 3: Architecture

We have:

* 4 states (we may not use all the states, we will see)
* inputs : x
* outputs : y, z

STEP 4: Encoding

A : 00  
B : 01  
C : 10  
D : 11

STEP 5: Create FSM

Diagram

Description automatically generatedThere isn’t any way to go to D state. That’s why D is unused. We can ignore it. We have only 3 states.

At each clock signal, we have to take a movement.

Two properties needs to be fulfilled on FSM design:

Condition 1

At transition only one condition should be true.

A graph with numbers and symbols

Description automatically generated with low confidenceWhat if a and b are both 1 at the same time?

Condition is not fulfilled.

We can fix this condition by:

A picture containing shoji

Description automatically generated

Condition 2

At each clock, one condition should be true.

A picture containing shoji

Description automatically generatedWhat if a and b are both 0 at the same time?

Condition is not fulfilled.

We can fix this condition by:

A picture containing shoji, text

Description automatically generated

How can we check these conditions are fulfilled?

To validate condition 1:

* AND of all the transitions should produce 0

To validate condition 2:

* OR of all the transitions should produce 1

Sadece clock ile geçtiğimiz yerde aslında condition var: a 1’ken de a 0’ken de geç

