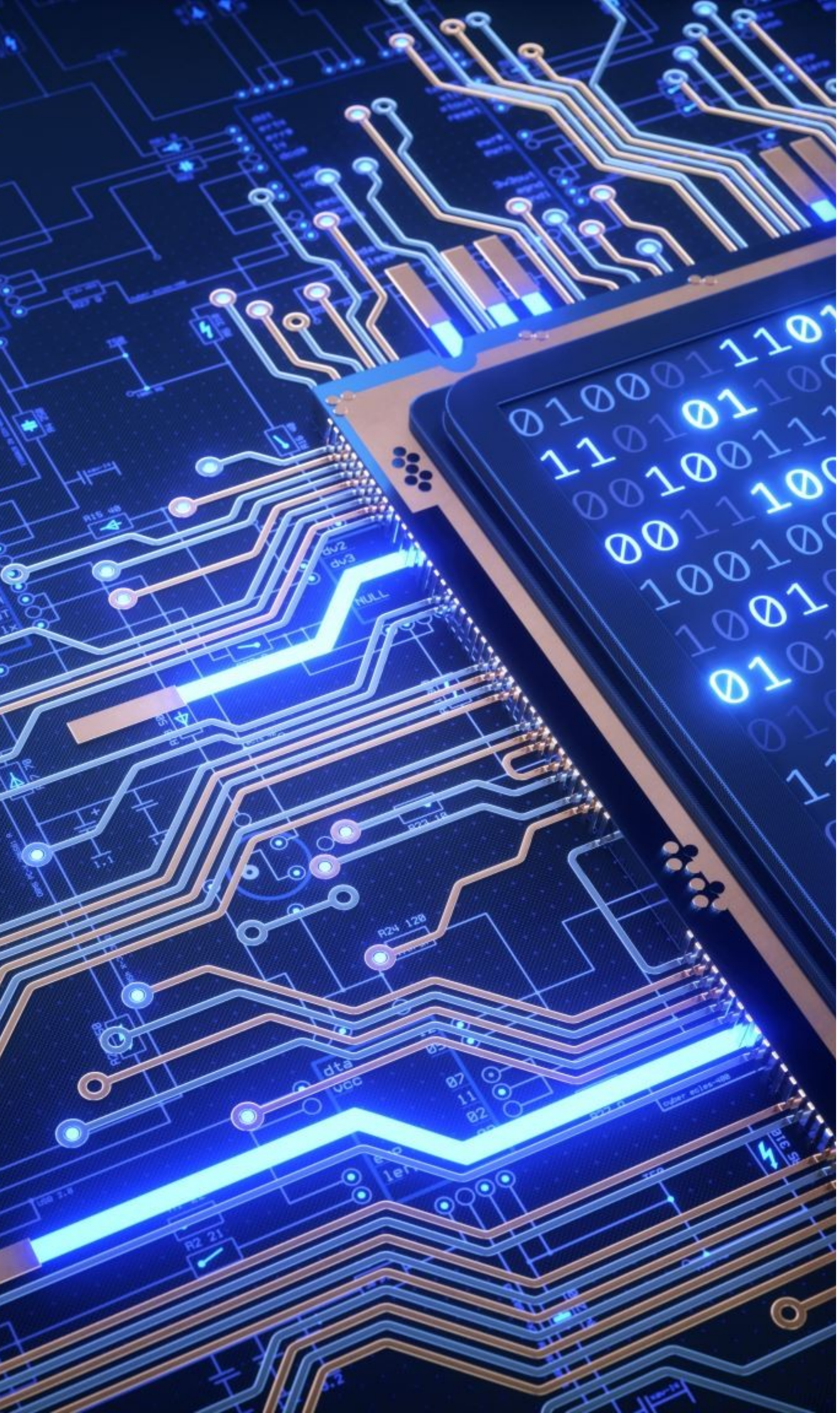
Finite State Machine

Dr. Min He



Outline

- Finite State Machines
- Moore FSM
- Model Traffic Light Controller as Moore FSM
- Implement Traffic Light Controller
- Reading Materials and Assignments

Finite State Machine

A Finite State Machine (FSM) is used to model embedded systems with a set of inputs, a set of outputs, and finite number of states and transitions.

- ❖ Inputs: Sensors
- ❖ Outputs: Actuators (a device that causes a machine or other device to operator).
- ❖ State: Description of current conditions
- ❖ Controller/Engine: Software that takes inputs, generates outputs, and changes state.
- ❖ Tools to defines input/output relationship:
  - ➢ State table
  - ➢ State graph

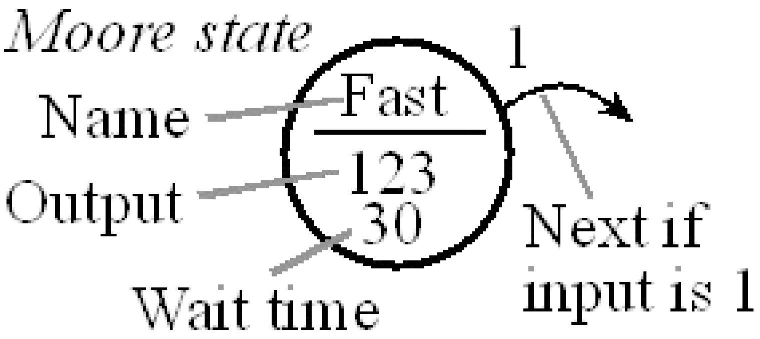
Two Types of FSM

Moore FSM:
output depends only on the current state.

Mealy FSM:
output depend on its current state and current input.

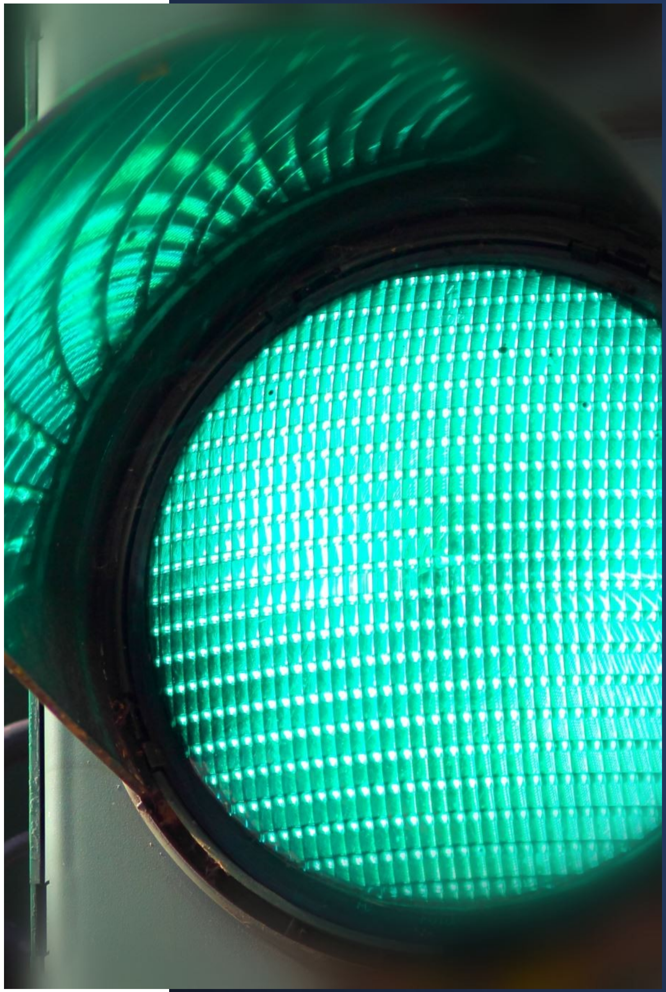
Moore FSM

- Output value depends **only** on the current state.
- State change is based on inputs & current state.
- Timed vs not-timed: determines when output and state can be changed.
- Significance is being in a state.



A Simple Traffic Light Controller

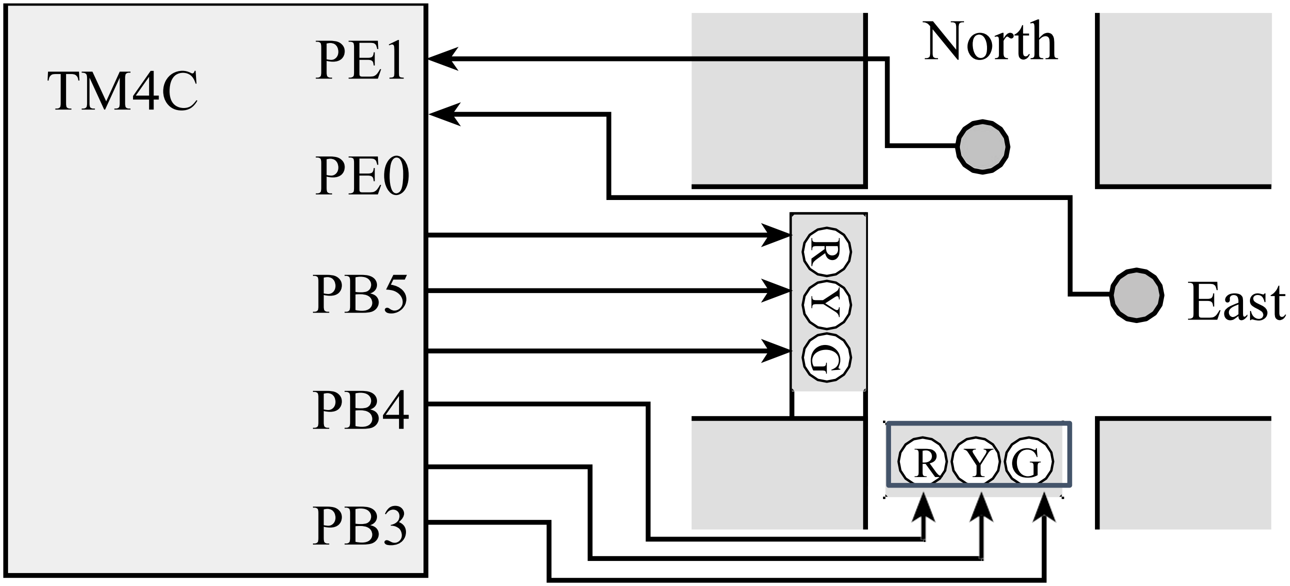
- If no cars are coming, stay in a green state
- When changing from green to red, show yellow for 1 seconds
- Green lights last at least 2 seconds
- If cars are detected in only one direction, move to and stay green in that direction
- If cars are detected in both directions, cycle traffic lights to allow cars to pass in both directions (eg North-South, East-West, North-South, …)



Traffic Light Control

PE1=0, PE0=0 means no cars exist on either road
PE1=0, PE0=1 means there are cars on the East road
PE1=1, PE0=0 means there are cars on the North road
PE1=1, PE0=1 means there are cars on both roads

North **East** **Time**



goN, PB5-0 = 100001 makes it green on North and red on East
waitN, PB5-0 = 100010 makes it yellow on North and red on East
goE, PB5-0 = 001100 makes it red on North and green on East
waitE, PB5-0 = 010100 makes it red on North and yellow on East

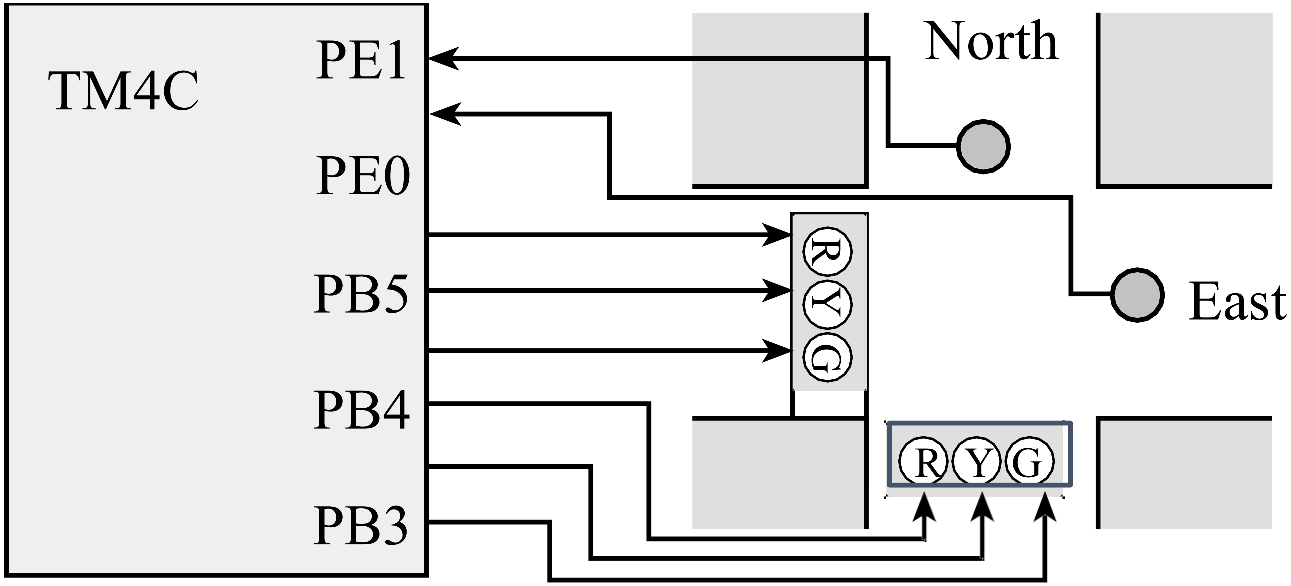
4 possible states

Outputs for each possible state

Traffic Light Control

PE1=0, PE0=0 means no cars exist on either road
PE1=0, PE0=1 means there are cars on the East road
PE1=1, PE0=0 means there are cars on the North road
PE1=1, PE0=1 means there are cars on both roads

North	East	Time
GREEN	RED	2
YELLOW	RED	1
RED	GREEN	2
RED	YELLOW	1



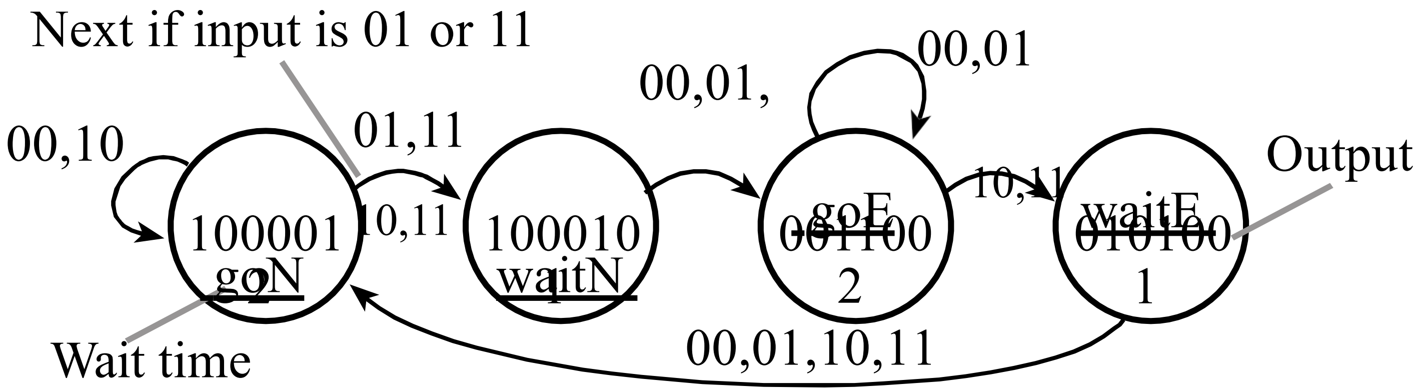
goN, PB5-0 = 100001 makes it green on North and red on East
waitN PB5-0 = 100010 makes it yellow on North and red on East
, goE, PB5-0 = 001100 makes it red on North and green on East
PB5-0 = 010100 makes it red on North and yellow on East
waitE
,

4 possible states

Outputs for each possible state

Moore FSM:

Model Traffic Light with State Graph and State Table



State(output, wait time)	Inputs			
	00	01	10	11
goN(100001, 2)	goN	waitN	goN	waitN
waitN(100010,1)	goE	goE	goE	goE
goE(001100,2)	goE	goE	waitE	waitE
waitE(010100,1)	goN	goN	goN	goN

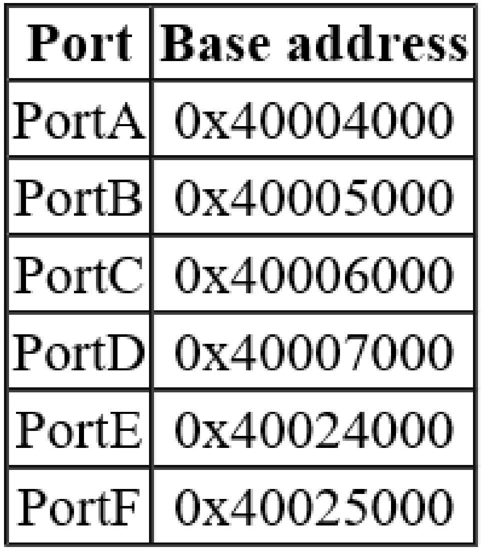
Moore FSM Engine: Execution Sequence

Each time through the main loop we do the following steps in order:

- Set output based on the current state
- Wait the prescribed amount of time for the current state
- Read inputs
- Update state based on inputs and current state

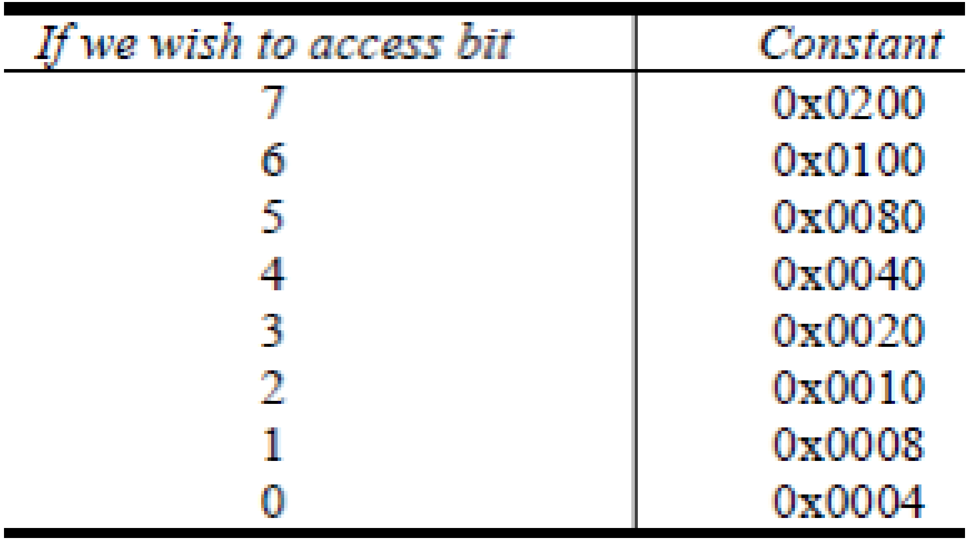
Define I/O Bit-specific Addresses

```
// PE1, PE0 connect to the two sensor (switches)
#define SENSOR      (*((volatile unsigned long *)0x4002400C))
// PB0 to PB5 are used for traffic lights (LEDs)
#define LIGHT        (*((volatile unsigned long *)0x400050FC))
```



Port	Base address
PortA	0x40004000
PortB	0x40005000
PortC	0x40006000
PortD	0x40007000
PortE	0x40024000
PortF	0x40025000

$4*2^b$



If we wish to access bit	*Constant*
7	0x0200
6	0x0100
5	0x0080
4	0x0040
3	0x0020
2	0x0010
1	0x0008
0	0x0004

Accessing a Single I/O Pin

Instead of just defining bit specific values for all inputs/all outputs, it is sometimes useful to define them for individual input and output.

```
// PE0 connect to the East sensor (switch)
#define SENSOR_E    (*((volatile unsigned long *)0x40024004)) // PE1 connect to the North sensor (switch)
#define SENSOR_N    (*((volatile unsigned long *)0x40024008))
#define N_SENSOR_MASK 0x02
```

So, we can write

```
        if (SENSOR_N == N_SENSOR_MASK )
```

instead of

```
        if ((SENSOR & 0x02) == N_SENSOR_MASK )
```

FSM Data Structure in C

```
struct State {
  uint8_t Out;        // outputs
 uint8_t Time;          // in
  second units
 uint8_t Next[4]; // list of
  next states
};
```

In ROM

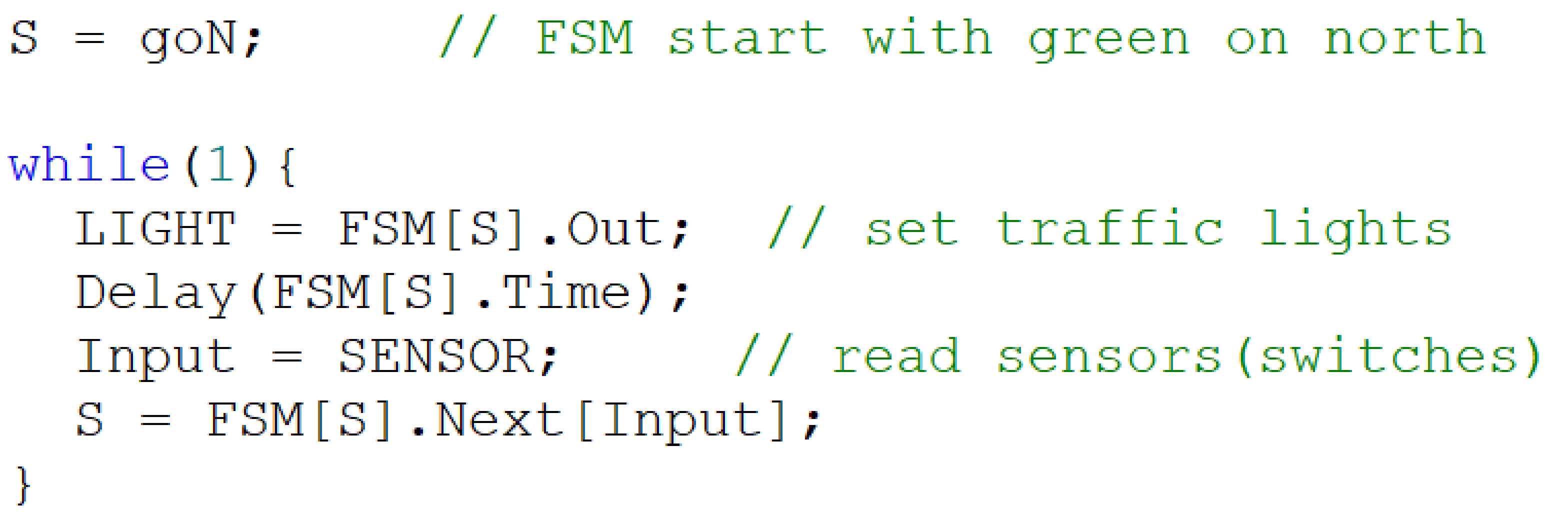
```
typedef const struct
STyp;

#define goN   0
#define waitN 1
#define goE   2
#define waitE 3
```

or

```
enum states {goN, waitN, goE, waitE};
STyp FSM[4] = {
{0x21,2,{goN,waitN,goN,waitN}},
{0x22, 1,{goE,goE,goE,goE}},
{0x0C,2,{goE,goE,waitE,waitE}},
```

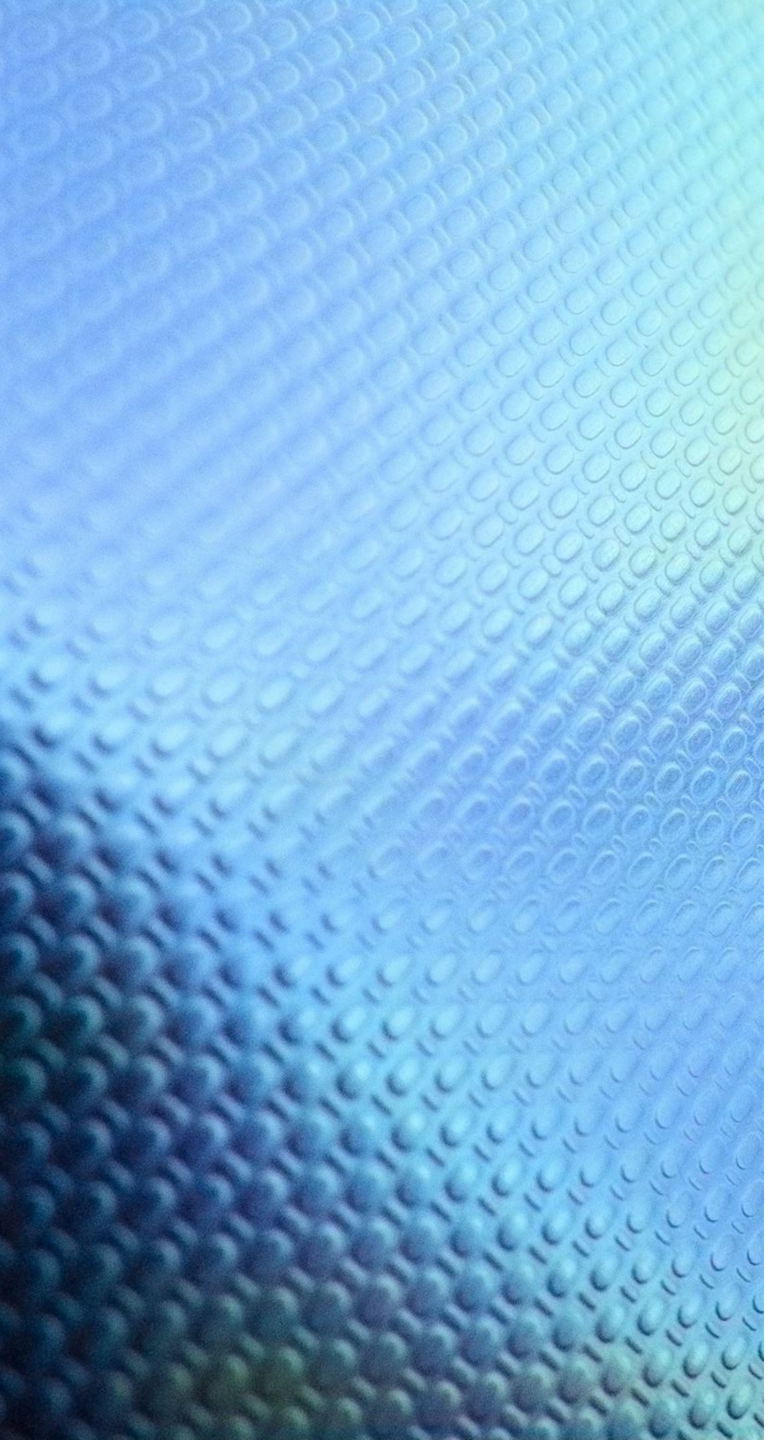
State(output, wait time)	Inputs			
	00	01	10	11
goN(100001, 2)	goN	waitN	goN	waitN
waitN(100010,1)	goE	goE	goE	goE
goE(001100,2)	goE	goE	waitE	waitE
waitE(010100,1)	goN	goN	goN	goN

FSM Engine in C



```
S = goN;       // FSM start with green on north

while(1){
   LIGHT = FSM[S].Out;  // set traffic lights
   Delay(FSM[S].Time);
   Input = SENSOR;       // read sensors(switches)
   S = FSM[S].Next[Input];
}
```

- **If SENSOR does not have inputs in the least significant bits, right shift (>>) the bits to move them to the least significant bits.**
- **Make sure the bits in Input count 0, 1, … in decimal for the Next[Input] line to work as expected**



Reading Materials and Assignments

- Textbook Chapter 6: 6.1 – 6.6
- Tiva™ TM4C123GH6PM Microcontroller Data Sheet.
- TM4C123 Launchpad User’s Guide
- Example Project: SimpleTrafficLight
- Lab Assignment: Lab 3