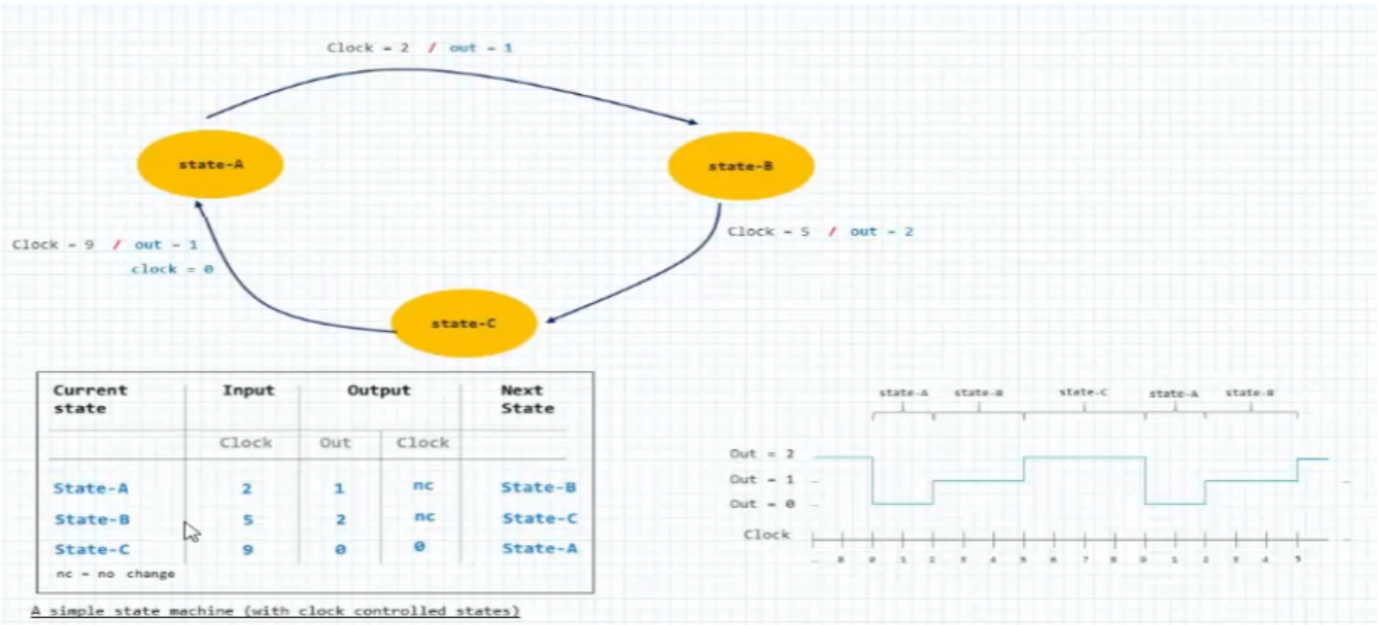
Finite State Machine

A Simple Example



Current state	Input	Output		Next State
	Clock	Out	Clock	
State-A	2	1	nc	State-B
State-B	5	2	nc	State-C
State-C	9	0	0	State-A

nc = no change

A simple state machine (with clock controlled states)

```c
#include <stdio.h>

enum states{
    STATE_A = 0,
    STATE_B,
    STATE_C
};

typedef enum states State_Type;

void state_a_function(void);
void state_b_function(void);
void state_c_function(void);
void state_machie_init(void);

static void (*state_table[])(void) = {
                                        state_a_function,
                                        state_b_function,
                                        state_c_function
                                    };

static State_Type current_state;
static int Clock;

int main(void) {
    state_machie_init();

    while(1) {
        state_table[current_state]();
        Clock++;
    }

    return 0;
}

void state_machie_init(void) {
    current_state = STATE_A;
    Clock = 0;
}

void state_a_function(void) {
    if(Clock == 2){
        current_state = STATE_B;
        printf("This is the exection of STATE A \n\r");
    }
}

void state_b_function(void) {
    if(Clock == 5){
        current_state = STATE_C;
        printf("This is the exection of STATE B \n\r");
    }
}

```

```
55 void state_c_function(void) {
56      if(Clock == 9){
57         Clock = 0;
58         current_state = STATE_A;
59         printf("This is the exection of STATE C \n\r");
60     }
61 }
```

```c
#include <stdio.h>
#include <unistd.h>
#include <stdint.h>
#include <time.h>

#define MAX_STATE 6

// enum states{
//     STATE_A = 0,
//     STATE_B,
//     STATE_C
// };

// typedef enum states State_Type;

void state_a_function(void);
void state_b_function(void);
void state_c_function(void);
void state_machie_init(void);
void SysTick_Handler(void);

static void (*state_table[])(void) = {
                                        state_a_function,
                                        state_a_function,
                                        state_b_function,
                                        state_a_function,
                                        state_c_function,
                                        state_b_function
                                    };

uint8_t current_state = 0;
static int Clock;

int main(void) {
    state_machie_init();

    while(1) {
        state_table[current_state]();
        sleep(1);
        Clock++;

        if (current_state == MAX_STATE) {
            current_state = 0;
        }
    }

    return 0;
}

void state_machie_init(void) {
    Clock = 0;
}

uint32_t sa_prev_time = 0;
```

```
55  uint32_t sa_now;
56  float sa_delta;
57
58  void state_a_function(void) {
59      if(Clock % 2){
60          current_state++;
61
62          sa_now = clock();
63          sa_delta = sa_now - sa_prev_time;
64          /* To seconds */
65          sa_delta /= 1000;
66          sa_prev_time = sa_now;
67          printf("This is the output of STATE A : %f seconds ago \n\r", sa_delta);
68          // printf("This is the exection of STATE A \n\r");
69      }
70  }
71
72  uint32_t sb_prev_time = 0;
73  uint32_t sb_now;
74  float sb_delta;
75
76  void state_b_function(void) {
77      if(Clock % 5){
78          current_state++;
79
80          sb_now = clock();
81          sb_delta = sb_now - sb_prev_time;
82          /* To seconds */
83          sb_delta /= 1000;
84          sb_prev_time = sb_now;
85          printf("This is the output of STATE B : %f seconds ago \n\r", sb_delta);
86          // printf("This is the exection of STATE B \n\r");
87      }
88  }
89
90  uint32_t sc_prev_time = 0;
91  uint32_t sc_now;
92  float sc_delta;
93
94  void state_c_function(void) {
95       if(Clock % 9){
96          current_state++;
97
98          sc_now = clock();
99          sc_delta = sc_now - sc_prev_time;
100         /* To seconds */
101         sc_delta /= 1000;
102         sc_prev_time = sc_now;
103         printf("This is the output of STATE C : %f seconds ago \n\r", sc_delta);
104         // printf("This is the exection of STATE C \n\r");
105     }
106 }
107
108 void SysTick_Handler(void) {
109
```

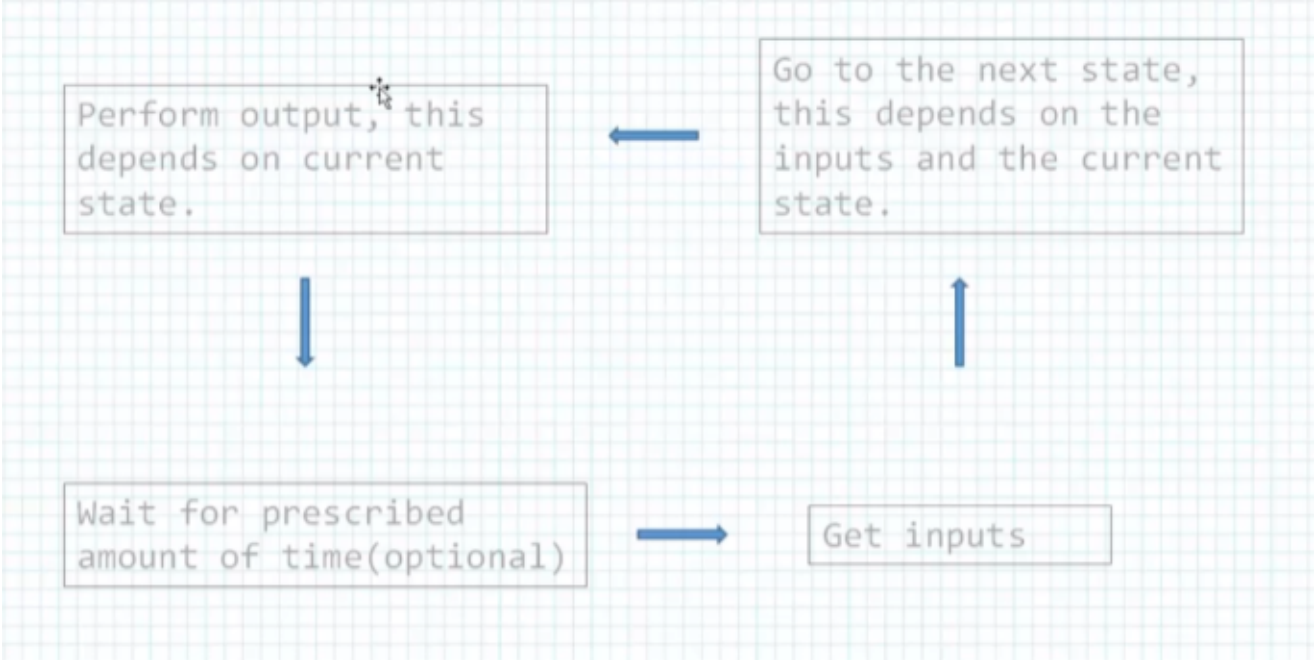
```
110 }
```

Moore and Mealy FSM

Introduction

1. Moore FSM

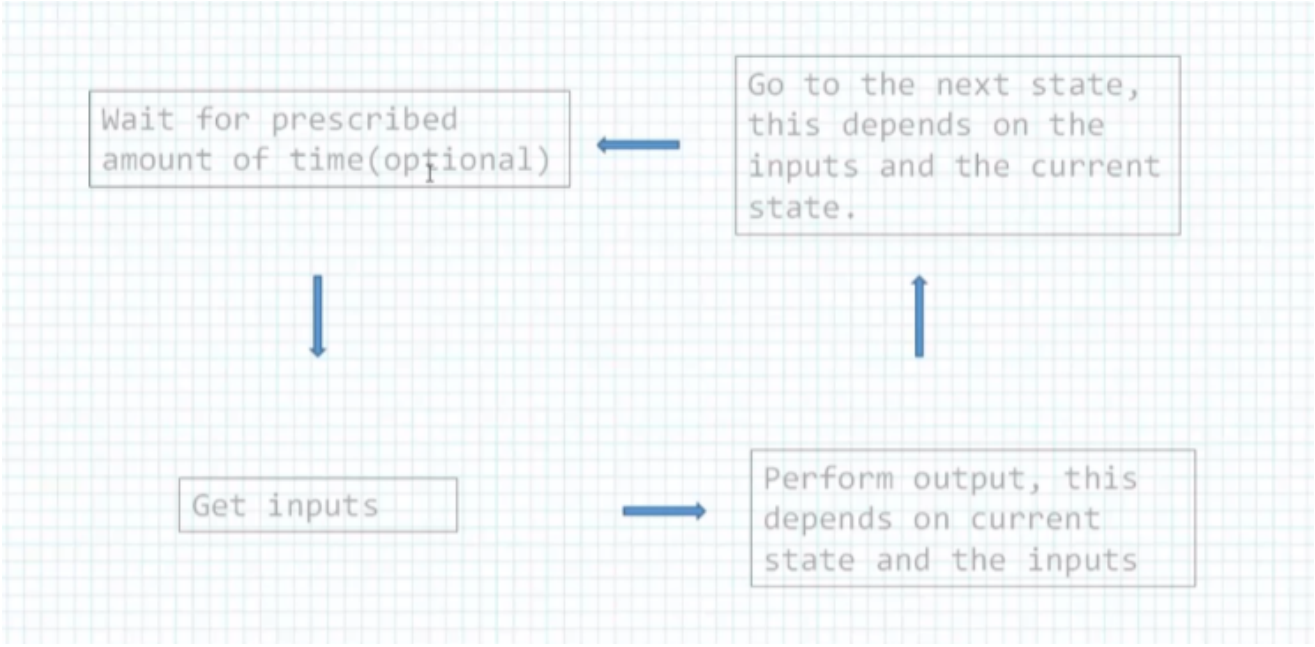
- Output value depends on only current state.
- Output is necessary to be a state.

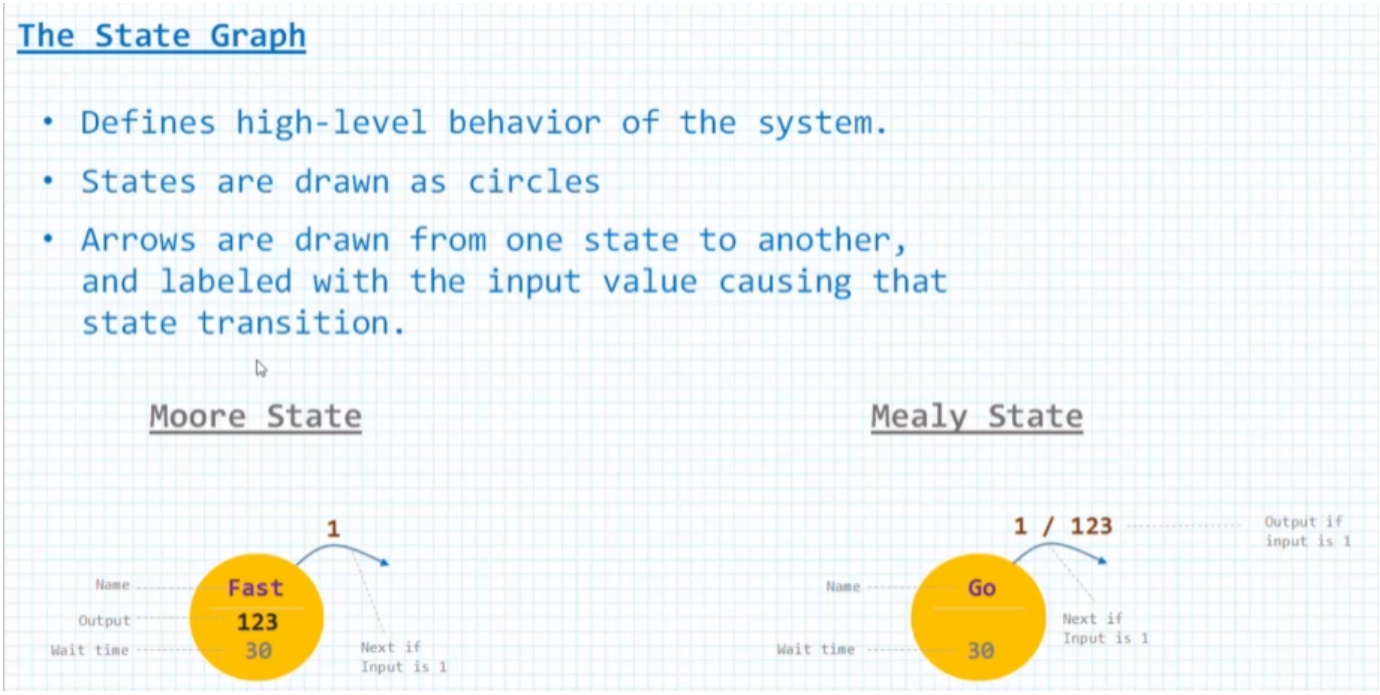


2. Mealy FSM

- Output value depends on both current state and inputs.
- No specific output value is necessary to be a state.

- Output is required to transit the machine from one state to the next.

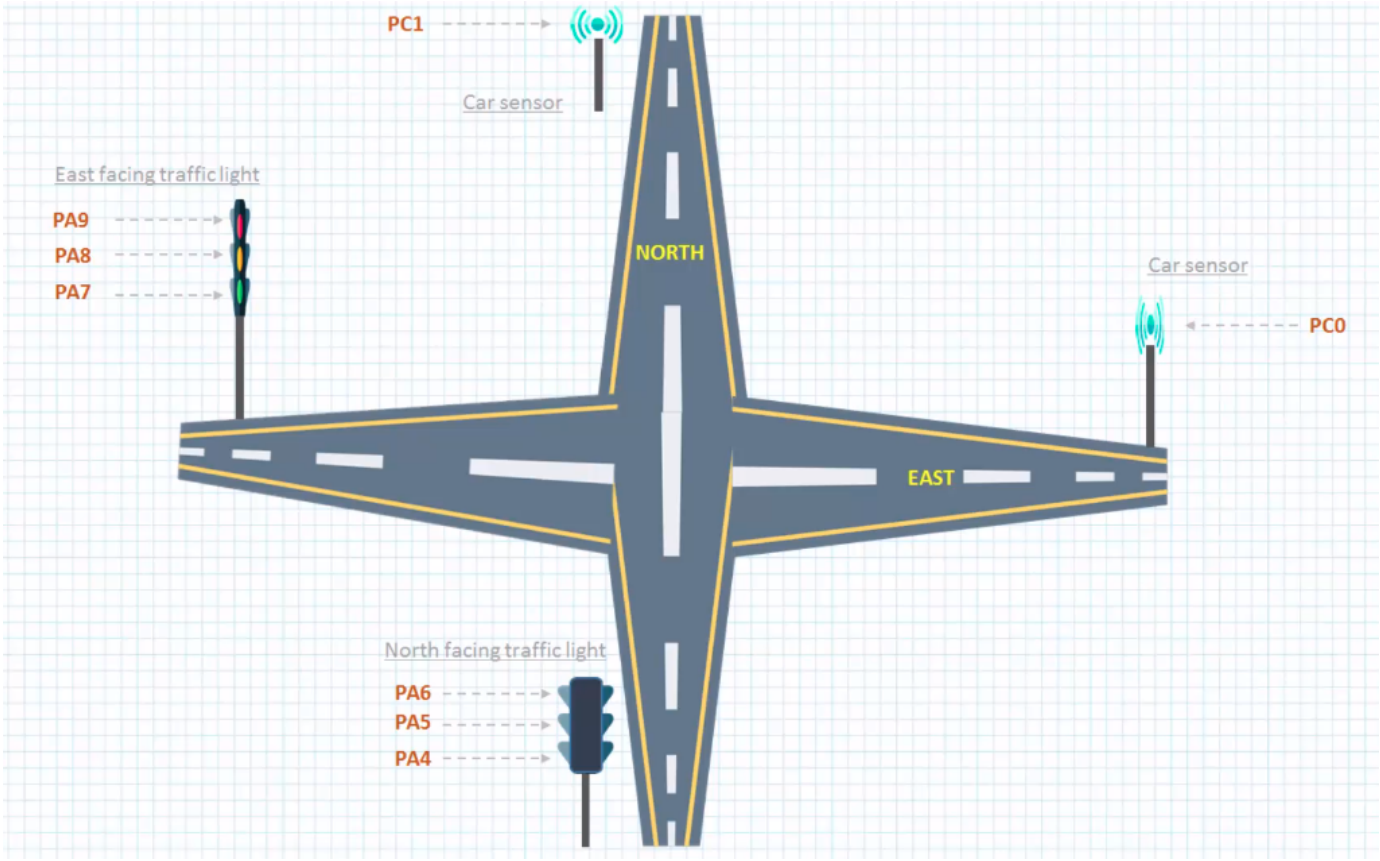




3. FSM Implementation

- We use a linked structure with fixed size.
- There should be one-to-one mapping between the FSM state graph and the data structure.

Case Study : Traffic Light System Implementattion using Moore FSM



3 Steps

Define what a state is.
E.g. light patterns

↓

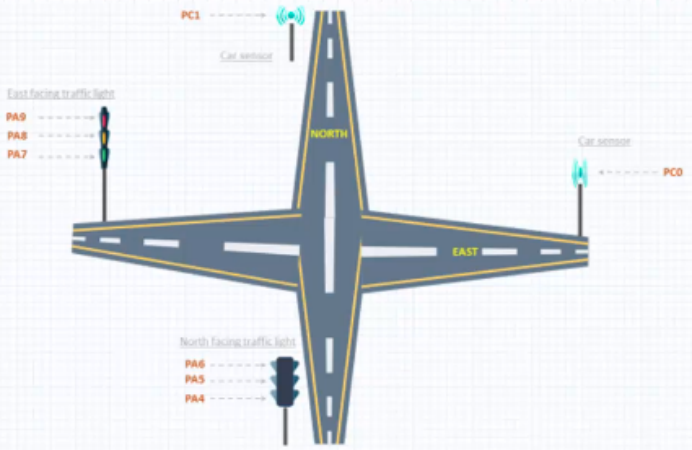
Make a list of various states in which system might exist.

↓

Add outputs and inputs to enable the system to affect external environment and collect Information about environment.

Key points about the system

- State describes which road has authority to cross the intersection.
- Light pattern defines which road has right of way over the other.



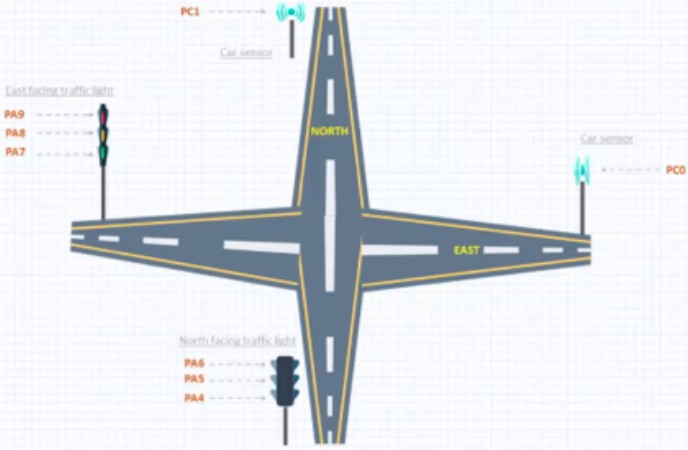
System I/O summary

Inputs

- Car sensor on north
- Car sensor on east

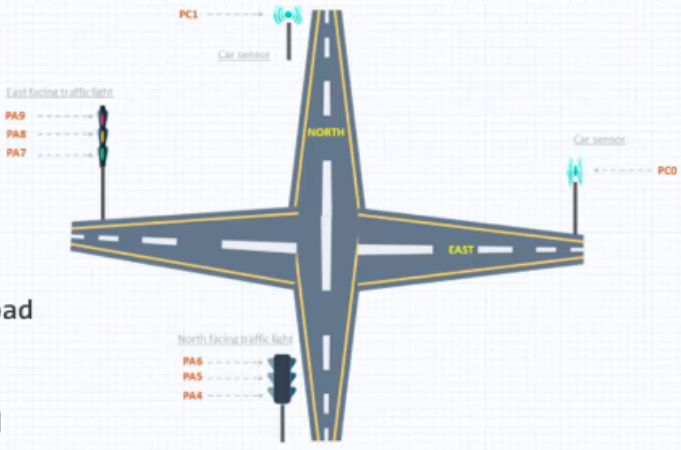
Outputs

- 6 outputs one for each light in the traffic signal



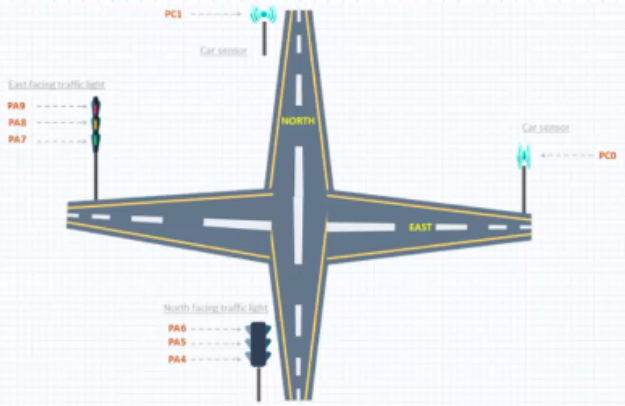
Step 1 : Inputs

PC0 : East car sensor
PC1 : North car sensor



PC0	PC1	Meaning
0	0	No cars exist on either road
0	1	Cars are on the east road
1	0	Cars are on the north road
1	1	Cars are on both roads

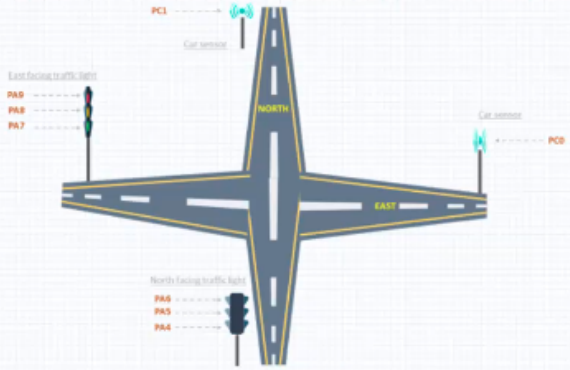
Step 2 : States(and outputs)



Create state and give each state a symbolic name.

	PA9	PA8	PA7	PA6	PA5	PA4		
go_north	1	0	0	0	0	1	= 0x210	/* Make RD on at East and GR on at north */
wait_north	1	0	0	0	1	0	= 0x220	/* Make RD on at East and YL on at north */
go_east	0	0	1	1	0	0	= 0x0C0	/* Make GR on at East and RD on at north */
wait_east	0	1	0	1	0	0	= 0x140	/* Make YL on at East and RD on at north */

Decision rules



1. If no cars are coming stay in green state

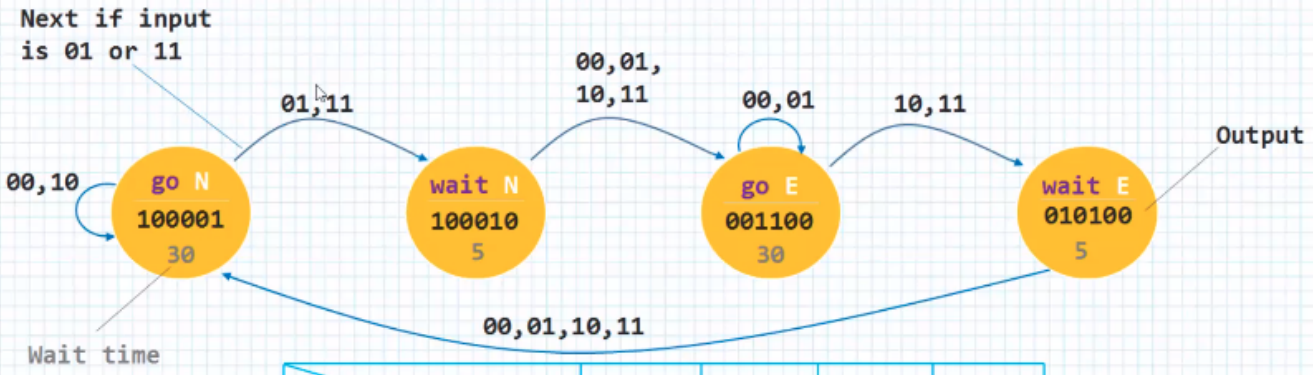
2. To change from green to red implement yellow for exactly 5 seconds.

3. Green lights should last for at least 30 seconds

4. If cars are only coming in one direction, move to that direction and stay in green.

5.If cars are coming in both directions cycle through all 4 states.

Draw state graph and state table



State \ Input	00	01	10	11
goN (100001,30)	goN	waitN	goN	waitN
waitN (100010,5)	goE	goE	goE	goE
goN (001100,30)	goE	goE	waitE	waitE
waitN (010100,5)	goN	goN	goN	goN

Map FSM Graph unto data structure

```
struct State{

    uint32_t Out; //E.g 100001
    uint32_t Time;//E.g 30
    const struct state * Next[4];//E.g. goN,waitN,goN,waitN
}
```

```
struct State{
    uint32_t output;
    uint32_t time;
    uint8_t next_state[4];
};

typedef const struct State stateType;

#define go_north         0  // 00
#define wait_north       1  // 01
#define go_east          2  // 10
#define wait_east        3  // 11

stateType STATE_MACHINE[4] = {
    // Using 3sec and 0.5 sec for demo
    {0x210, 3000, {go_north, wait_north, go_north, wait_north}},
    {0x220, 500,  {go_east, go_east, go_east, go_east}},
    {0x0C0, 3000, {go_east, go_east, wait_east, wait_east}},
    {0x140, 500, {go_north, go_north, go_north, go_north}}
};
```