State Machine

Last updated 10/29/20

State Machine

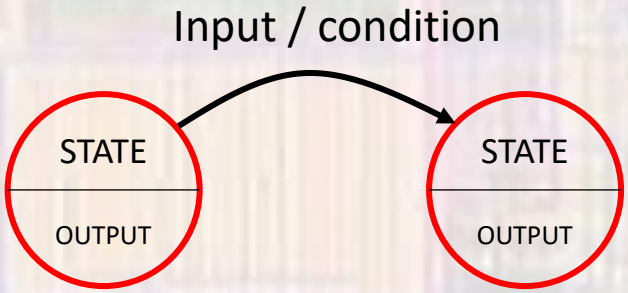
- These slides introduce state machines
- Upon completion: You should be able interpret and code using state machines

State Machine

- State Machine
  - Logical structure used to control the actions/outputs of a system

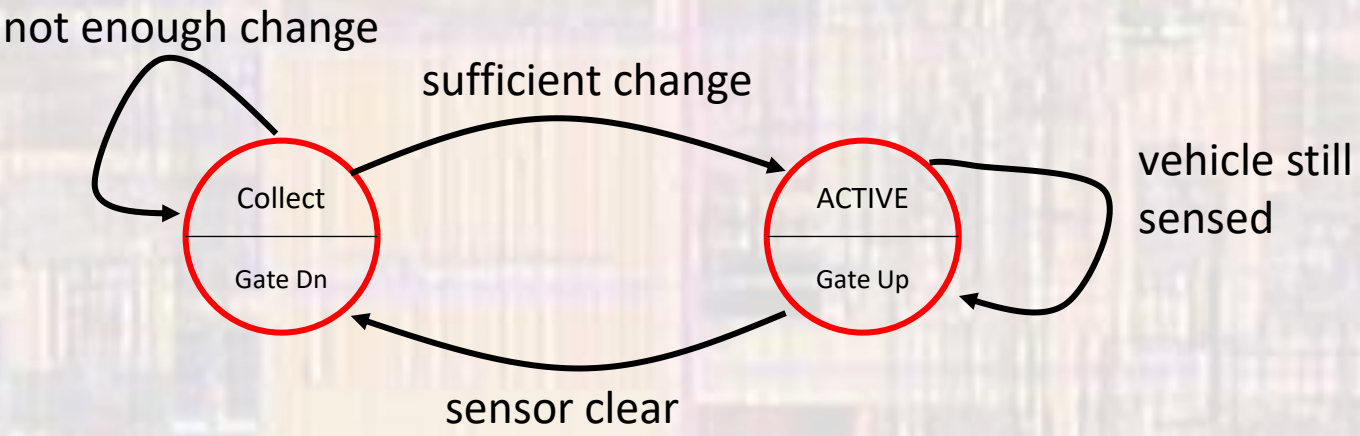
State Machine

- State Machine - Moore
  - State Diagram



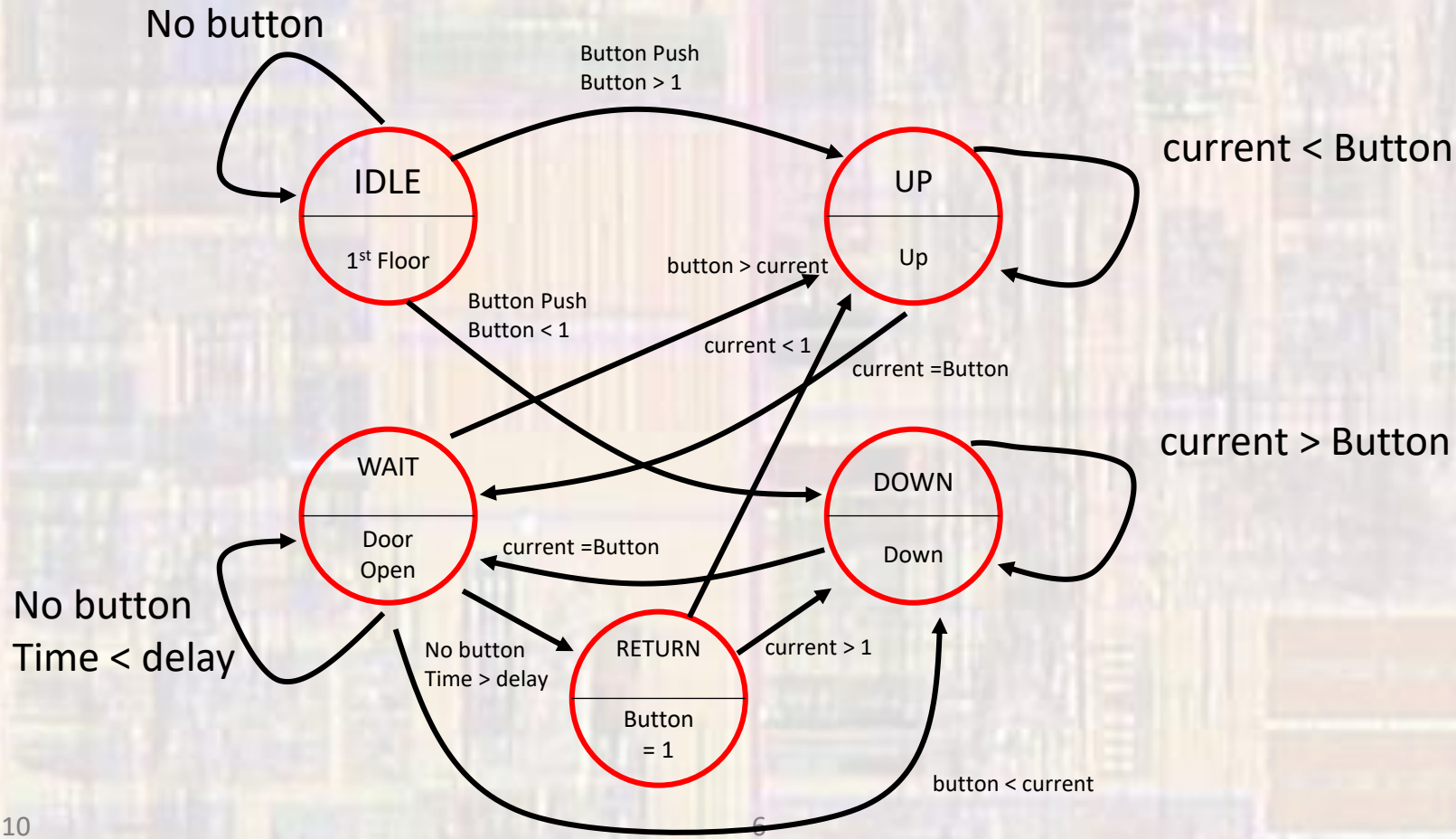
State Machine

- State Machine - example
  - Toll Gate



State Machine

- State Machine - example
  - Elevator - simplified



State Machine

- State Machine – C
  - Define states with enum

```
enum typeName {list of values};

enum EL_states {IDLE, UP, DOWN, WAIT, RETURN};
```

  - Create 2 state variables

```
enum EL_states myState;
enum EL_states myState_next;
```

State Machine

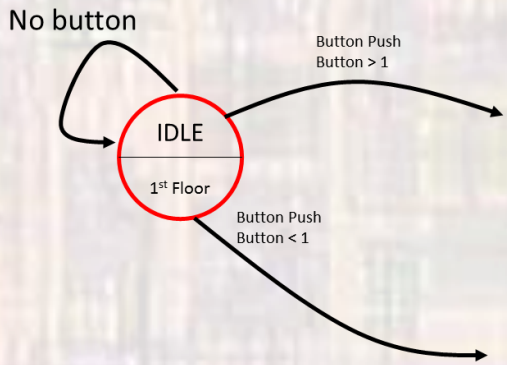
- State Machine – C
  - Use if-else construct to manage states

```
if(myState == IDLE){
…
else if(myState == UP){
…
else if(myState == DOWN){
…
```

State Machine

- State Machine – C
  - Use if-else construct to manage transitions

```
if(myState == IDLE){
    if(button == 0){
        myState_next = myState;
    }
    else if (button > 1){
        myState_next = UP;
    }
    else{
        myState_next = DOWN;
    }
}
else if(myState == UP){
...
```



State Machine

- State Machine – C
  - Enclose the whole state machine in a while loop
  - Update the state each cycle

```
while (1){
    if(myState == IDLE){
        if(button == 0){
            myState_next = myState;
    …

    myState = myState_next;
}
```

State Machine

- State Machine – C
  - Use switch statement instead

```
switch(myState){
    case IDLE:
        …
    case UP:
        …
    …
    default:
        …
}
```

State Machine

- State Machine – C
  - Enclose the whole state machine in a while loop
  - Update the state each cycle

```
while (1){
    switch(myState){
        case IDLE:
            …

    …
    myState = myState_next;
}
```

State Machine

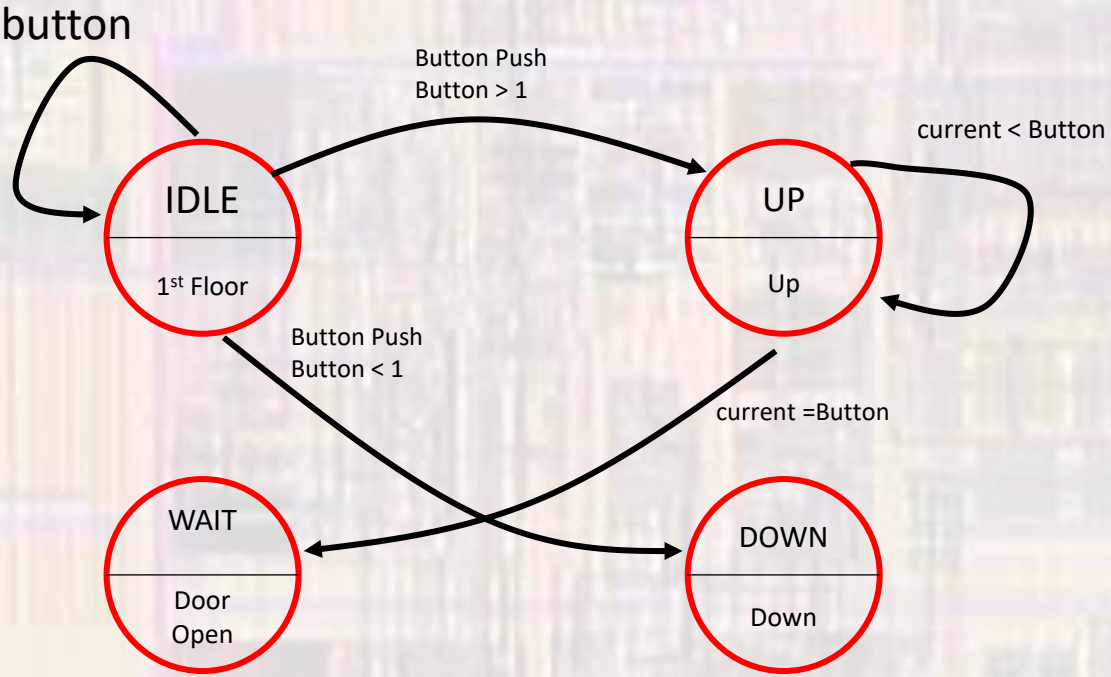
- State Machine – Elevator

```
#include <stdio.h>
#include <unistd.h>

// Prototypes
void travel(int* floor, int val);

// global variables
enum EL_STATE {IDLE, UP, DOWN, WAIT, RETURN};
enum EL_STATE myState;
enum EL_STATE myState_next;


int main(void){
   setbuf(stdout, NULL);  // disable buffering

   int button;
   int current;
   int waittime;
   int delaytime;
   delaytime = 5;
```

State Machine

- State Machine – Elevator

```
while (1){
  // State machine
  switch(myState){
    case IDLE:
      if(button == 0)
        current = 1;
      else if (button > 1)
        myState_next = UP;
      else if (button < 0)
        myState_next = DOWN;
      else
        ; // not necessary
      break;
    case UP:
      travel(&current, 1);
      if(current < button)
        myState_next = UP;
      else if(current == button)
        myState_next = WAIT;
      else
        ; // no error checking
      break;
```

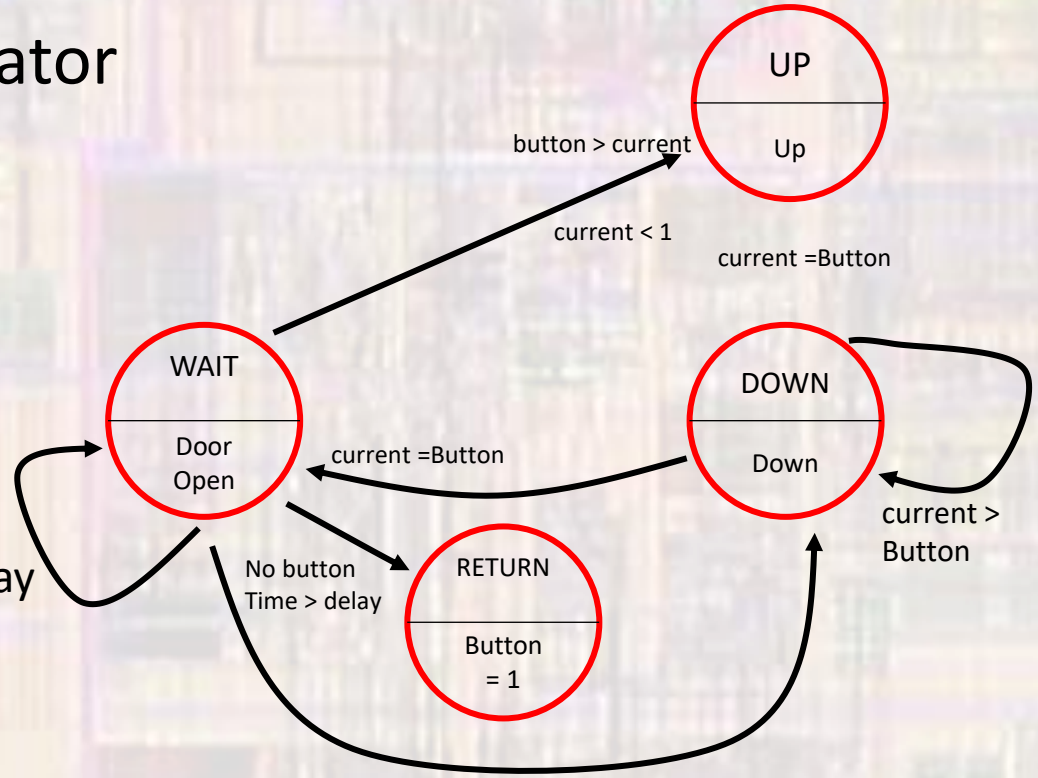


State Machine

- State Machine – Elevator

```
case DOWN:
  travel(&current, -1);
  if(current > button)
    myState_next = DOWN;
  else if(current == button)
    myState_next = WAIT;
  else
    ; // no error checking
  break;
case WAIT:
  if(waittime < delaytime){
    myState_next = WAIT;
    waittime++;
  }
  else{
    waittime = 0;
    if(!button)
      myState_next = RETURN;
    else if(current > button)
      myState_next = DOWN;
    else if(current < button)
      myState_next = UP;
    else
      myState_next = WAIT; //same floor - do nothing
  }
  break;
```

No button
Time < delay



State Machine

- State Machine – Elevator

```
        case RETURN:
           button = 1;
           if(current < 1)
             myState_next = UP;
           else
             myState_next = DOWN ;
           break;
         default:
           myState_next = IDLE;
           break;
       } // end switch

       //Update state
       myState = myState_next;
} // end while

  return 0;
} // end main
```