State Machine Review

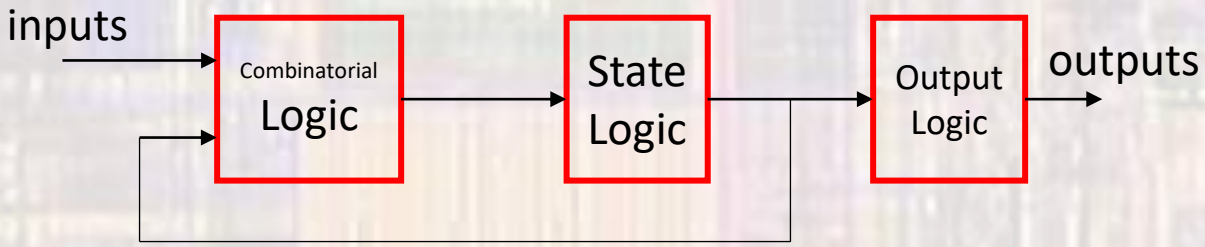
Last updated 1/9/23

State Machine

- Hardware State Machine
  - Logical structure used to control the actions/outputs of a system
  - Combines combinatorial and sequential logic

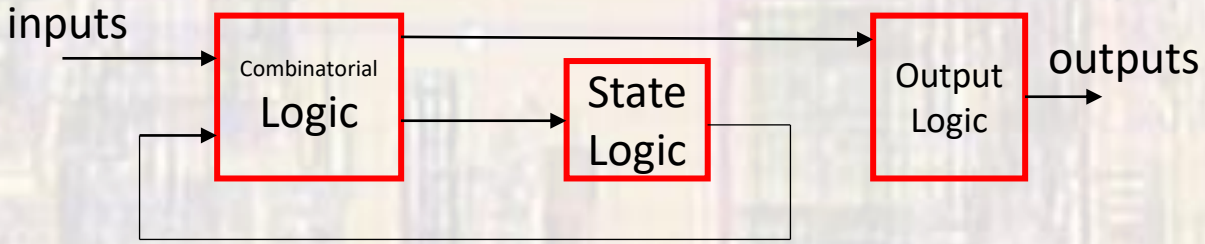
State Machine

- Hardware State Machine - Types
  - Moore
    - Outputs only dependent on current state



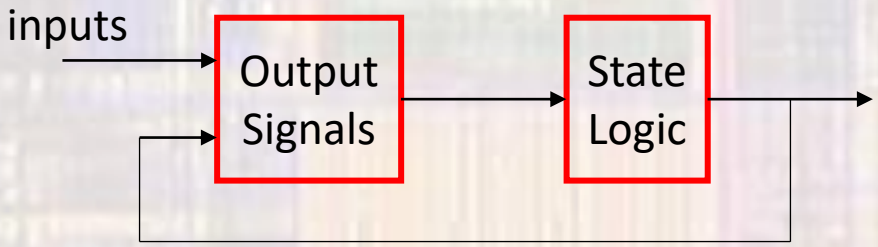
Require non-sequential (parallel) operation

  - Mealy
    - Outputs dependent on current state and inputs



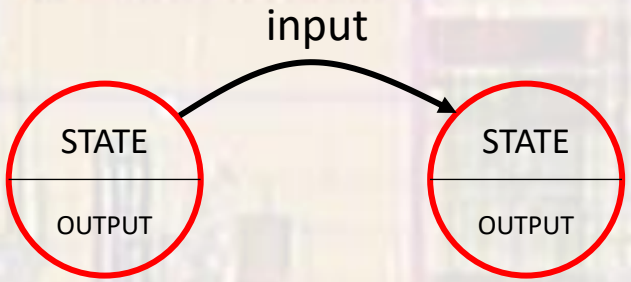
State Machine

- Software State Machine
  - Flow Diagram



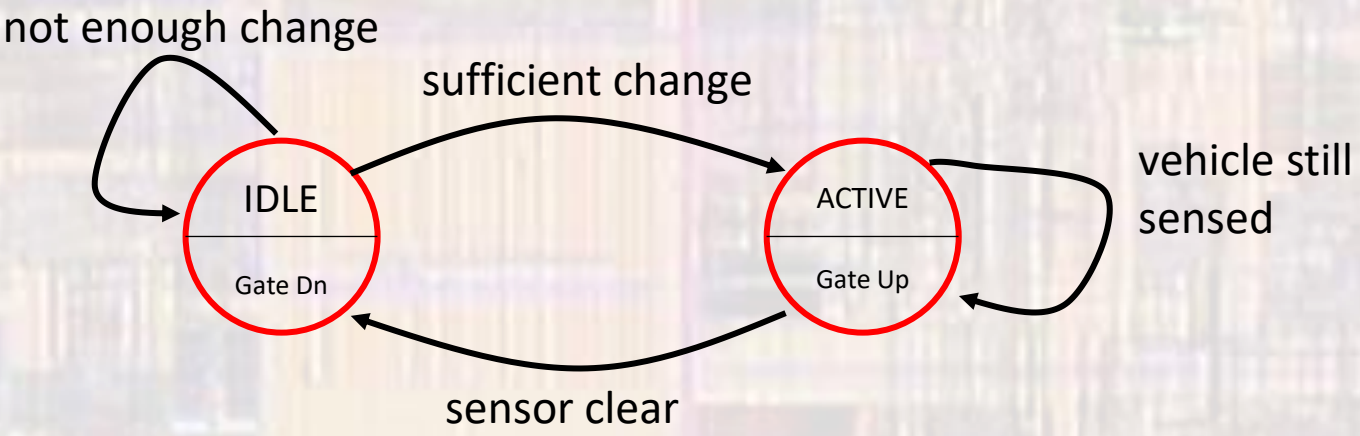
Only require sequential operation

  - State Diagram



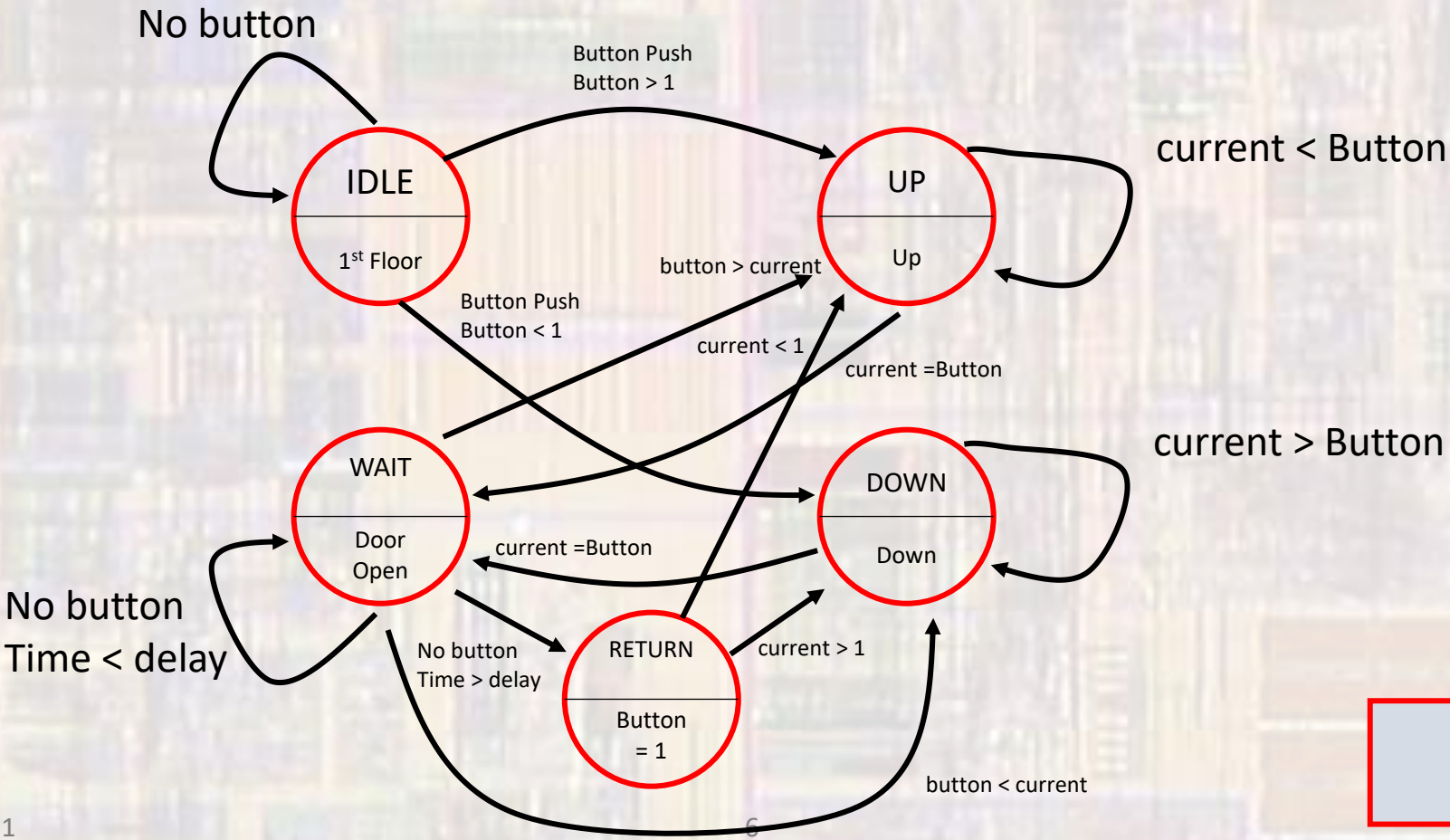
State Machine

- 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

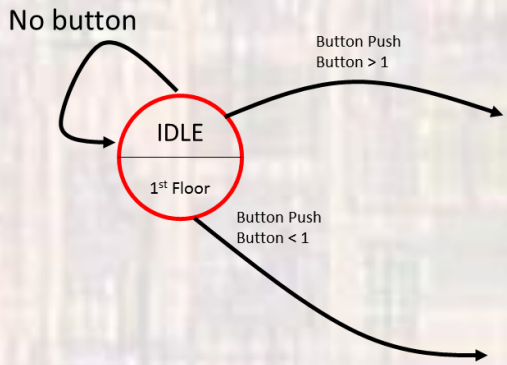
- State Machine – C
  - Use if-else construct to manage states

```
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
  - 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
  - Update the state each cycle

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

   …
   myState = myState_next;
}
```

State Machine

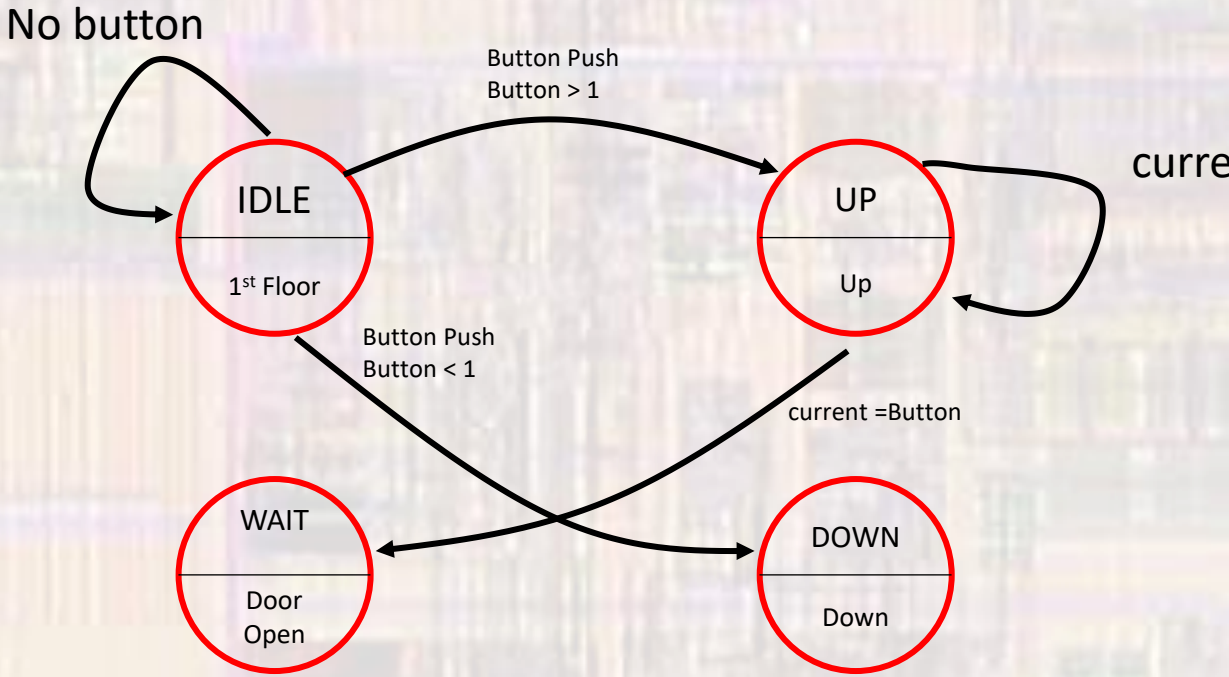
- State Machine – Elevator

```
// Includes
#include "msp432.h"

// Prototypes
void travel(int val);
int check_button(void);
int update_time();

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

int main(void){
  int button;
  int current;
  button = 1;
```

State Machine

- State Machine – Elevator

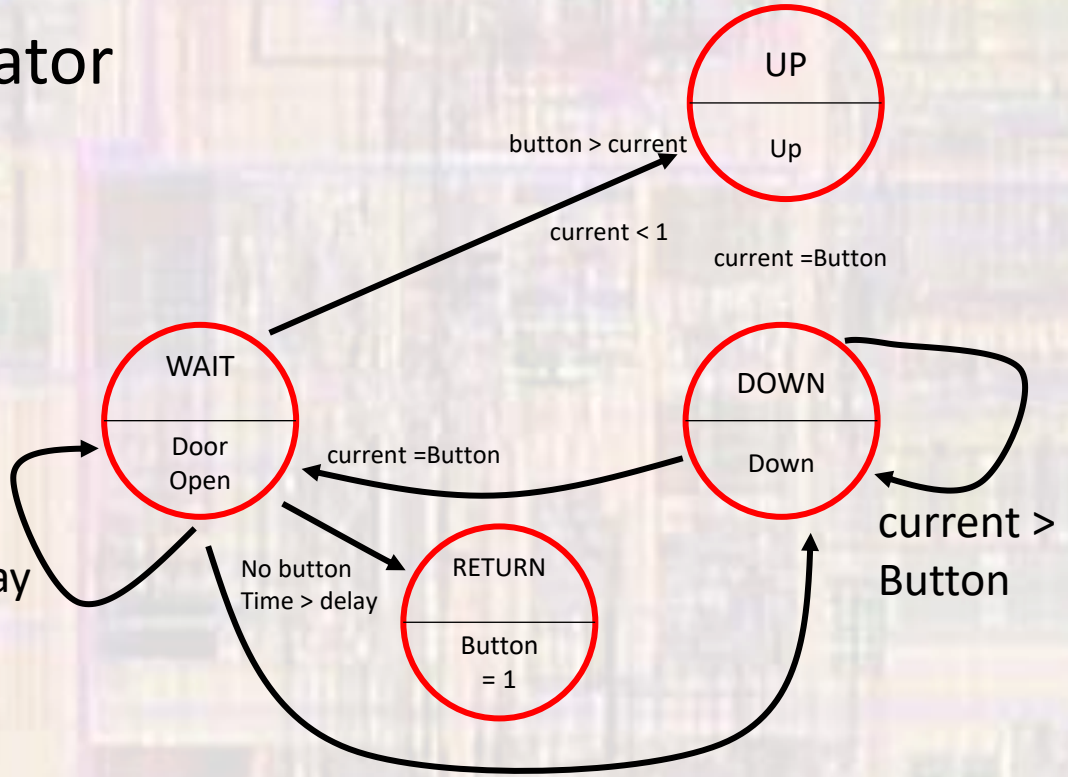
```
while (1){
  // get button val
  button = check_button();

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



State Machine

- State Machine – Elevator

```
case DOWN:
    travel(-1);
    if(current > button)
      myState_next = DOWN;
    else if(current == button)
      myState_next = WAIT;
    else
      ; // no error checking
  case WAIT:
    if(waittime < delaytime)
      myState_next = WAIT;
    else{
      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
    }
```



State Machine

- State Machine – Elevator

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

  //Update state
  myState = myState_next;

  //update time
   waittime = update_time;

} // end while

void travel(int val){
  // function to move elevator
  current = current + val;
}
```