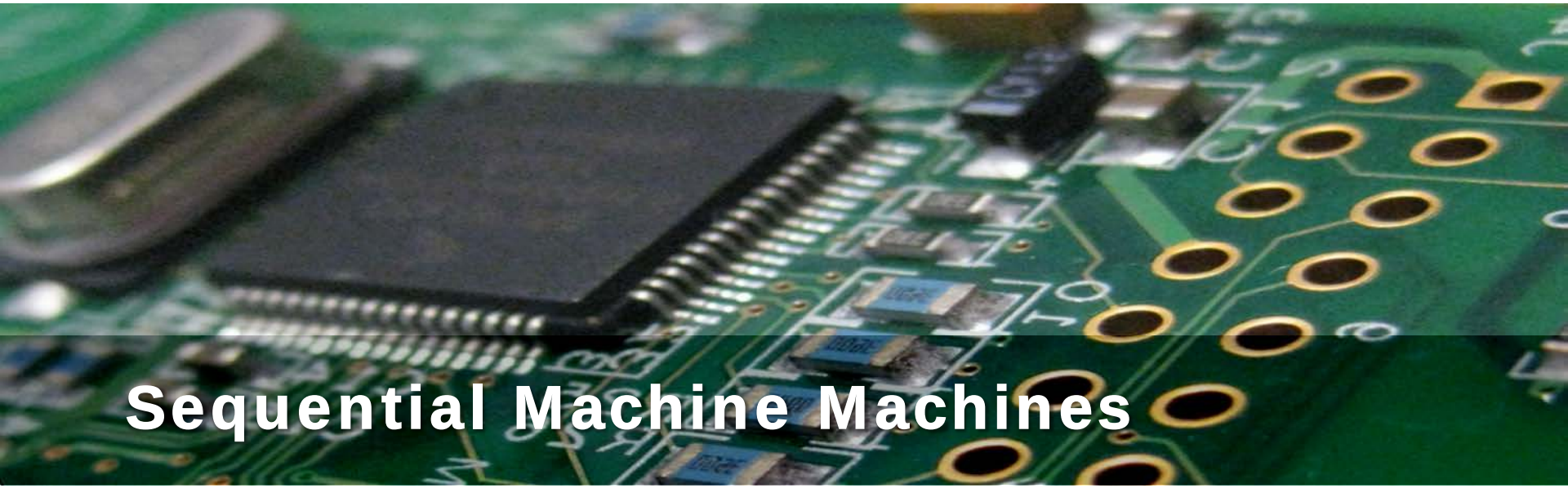
Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur



# Sequential Machine Machines

*“Having now a lot of things together, let’s use it with a simple state machine.”*

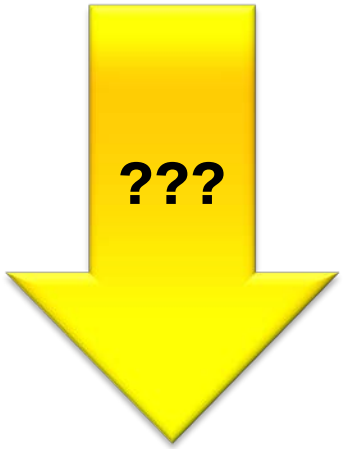
**Prof. Erich Styger**
*erich.styger@hslu.ch*
*+41 41 349 33 01*

**Scriptum:**
**State Machines**

# Learning Goals

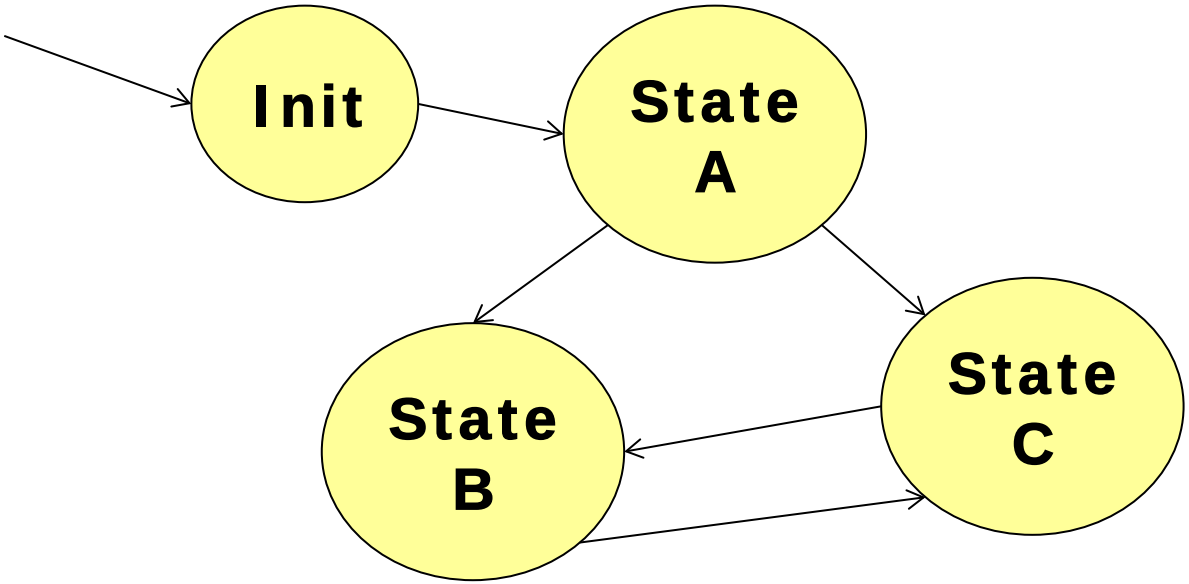
- Problem: Robot will need state machine (debouncing, modes during Sumo, ...)

- State Machines
  - Function
  - If-elseif-else
  - switch
  - Mealy for Testing



# State Machines

- Design pattern
  - States
  - Transition between states



# State Machine Implementation: Functions

- Each function implements state
- Transition between states by function calls
- No global state variable needed
- Disadvantage, possible issues:
  - Stack overflow
  - Recursion

function verwenden:

```
void State_A(void) {
  if (KeyIsPressed()) {
    State_B()
  }
  State_C();
}
```

```
void State_B(void) {
  LED1_On();
  if (MotorIsOff()) {
    State_C()
  }
  while(KeyIsPressed()) {}
}
```

# State Machine Implementation: If-Else

- Periodic 'Process' Function call
- If-elseif-else on (global) state variable
- Changing state
  - During processing
  - Outside: volatile/reentrancy!
- Size/Length/performance of function

```
typedef enum {
  STATE_INIT,
  STATE_A,
  STATE_B
} State;

State state;
```

if we need to change this state:
it's not that easy!

```
void Process(void) {
  if (state==STATE_INIT) {
    BlinkRed();
    state = STATE_A;
  } else if (state==STATE_A) {
    Beep();
    state = STATE_C;
  } else if (state==STATE_B) {
    MotorOn();
    state = STATE_B;
  }
}
```

**Idea: using function pointers**

# State Machine Implementation: Switch

- Similar to if-elseif-else, better structure
- Compiler can find best processing way
- Switch with break/return loop: re-iterate machine again
- Problem: global variable

```
void Process(void) {
  switch(state) {
     case STATE_INIT:
        BlinkRed();
        state = STATE_A;
        break;
     case STATE_A:
        Beep();
        state = STATE_C;
        break;
     …
  } /* switch */
}
```

switch wird verlassen

dann zu STATE_A

```
void Process(void) {
  for(;;) {
    switch(state) {
     case STATE_INIT:
        BlinkRed();
        state = STATE_A;
        return; /* exit */
     case STATE_A:
        Beep();
        state = STATE_C;
        break; /* reiterate */
     …
    } /* switch */
  } /* for */
}
```

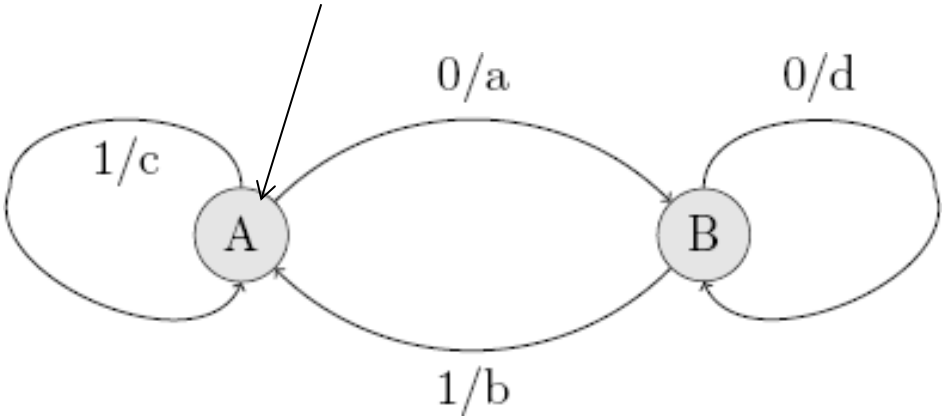
call the process every 5ms (e.g.)

get out here

wait until call again

# Input/Output:
# Mealy Sequential State Machine

Lucerne University of
Applied Sciences and Arts
HOCHSCHULE
LUZERN
Technik & Architektur

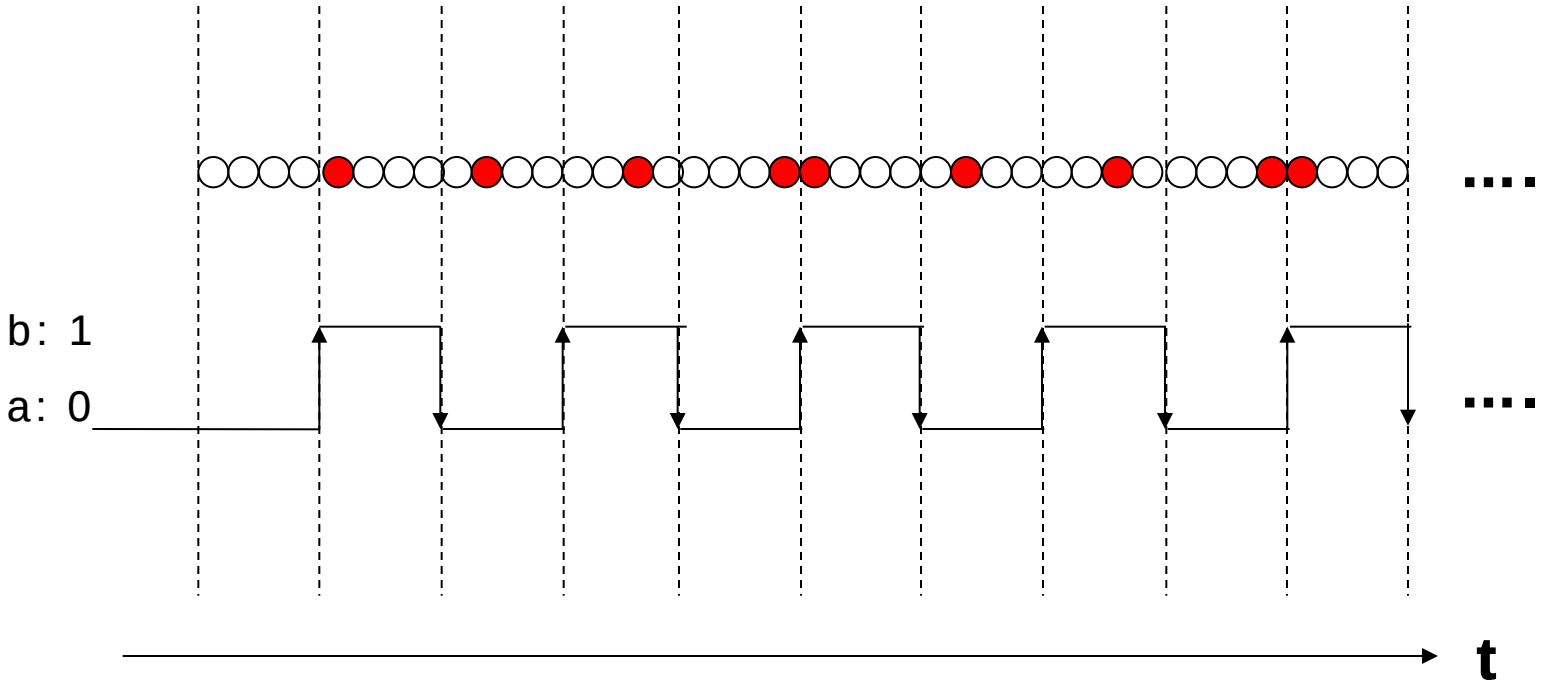



| State | 0 | 1 |
|---|---|---|
| A | B/a | A/c |
| B | B/d | A/b |

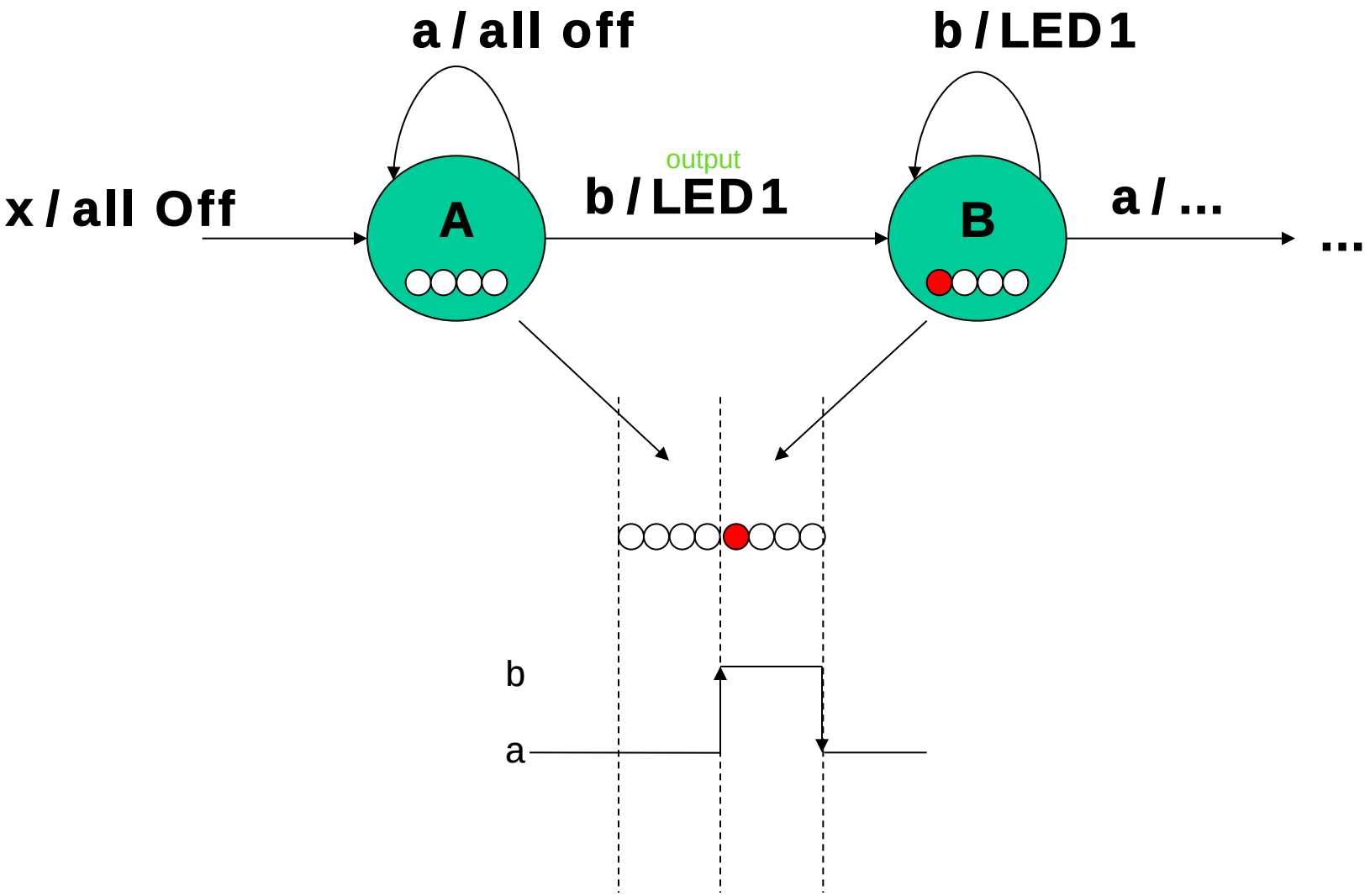
```
typedef enum {
  A, B
} StateT;

static StateT state;
static uint8_t tbl[2][2][2] =
  {{{B,a},{A,c}},
   {{B,d},{A,b}}};

void Loop(void) {
  uint8_t r;

  for(;;) {
    r = Read();
    Output(tbl[state][r][1]);
    state = tbl[state][r][0];
  }
}
```

direct from table

# Input/Output task



# FSM Representation



# Mealy Running LED

```
typedef enum MealyState {
  A, B, C, D, E} MealyState;

const uint8_t tbl[5][2][2] =
   /*     input a   input b */
 { /*A*/ {{A,0},    {B,LED1}},
   /*B*/ {{C,LED2}, {B,LED1}},
   /*C*/ {{C,LED2}, {D,LED3}},
   /*D*/ {{E,LED4}, {D,LED3}},
   /*E*/ {{E,LED4}, {B,LED1}},
 };
```

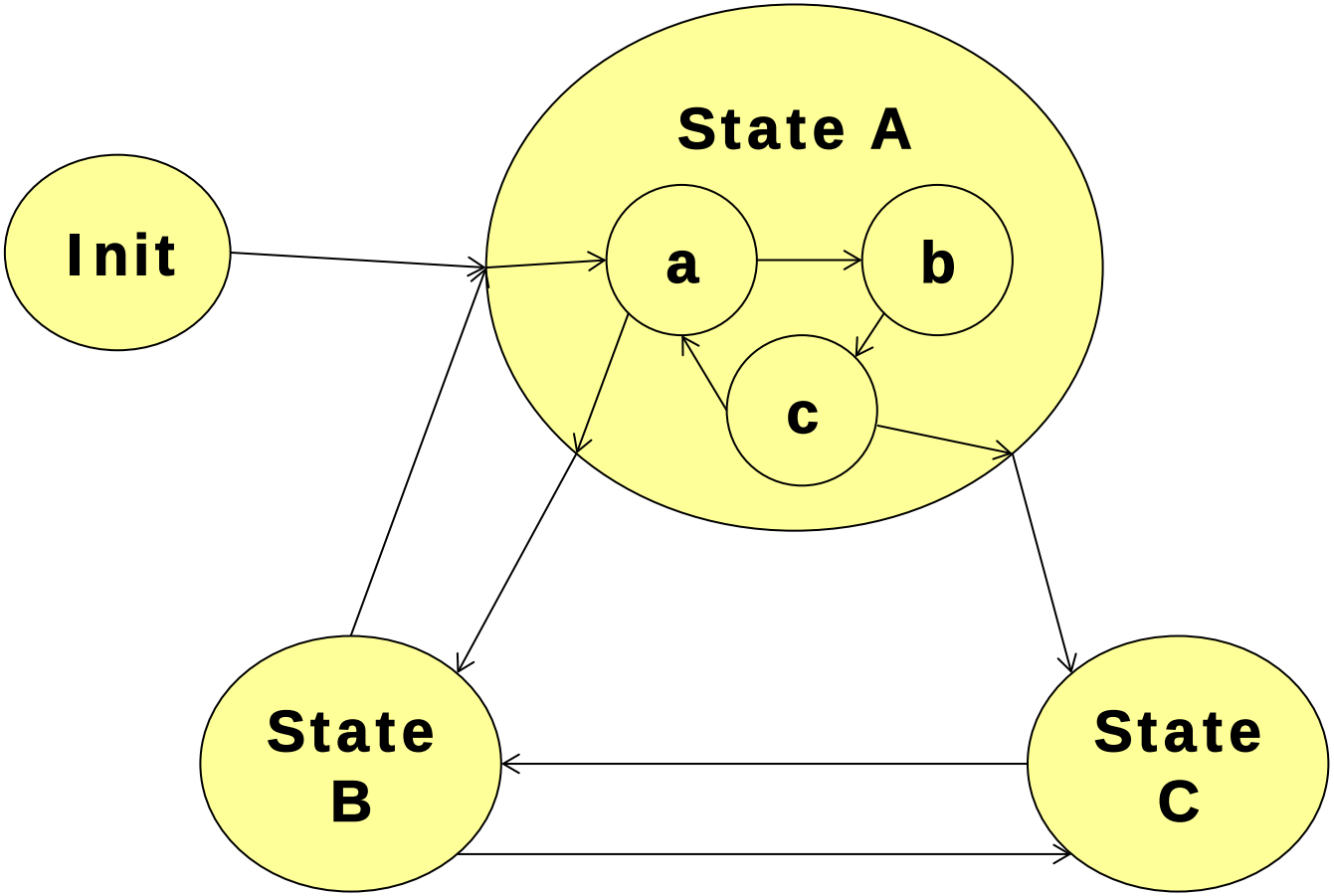
```
typedef enum InputState {
  INPUT_a, INPUT_b} InputState;

static InputState GetInput(void) {
  if (KEY1_Get()==0) {
    return INPUT_b;
  } else {
    return INPUT_a;
  }
}
```

```
void MEALY_Process(void) {
  InputState i;

  i = GetInput();
  LEDPut(tbl[state][i][1]);
  state = (tbl[state][i][0]);
}
```

```
void LEDPut(uint8_t set) {
  LED1_Put(set&LED1);
  LED2_Put(set&LED2);
  LED3_Put(set&LED3);
  LED4_Put(set&LED4);
}
```

# Hierarchical State Machines

- Combination of multiple FSM



# Summary

- Function State Machine
  - No global variable, needs stack space, recursion

- If-Else-If / Switch Processing State Machine
  - Process/task
  - Usual: Global variable

- Mealy Sequential Machine
  - Table driven SSM
  - Input → Transition with output

- Combinations, Hierarchical FSM