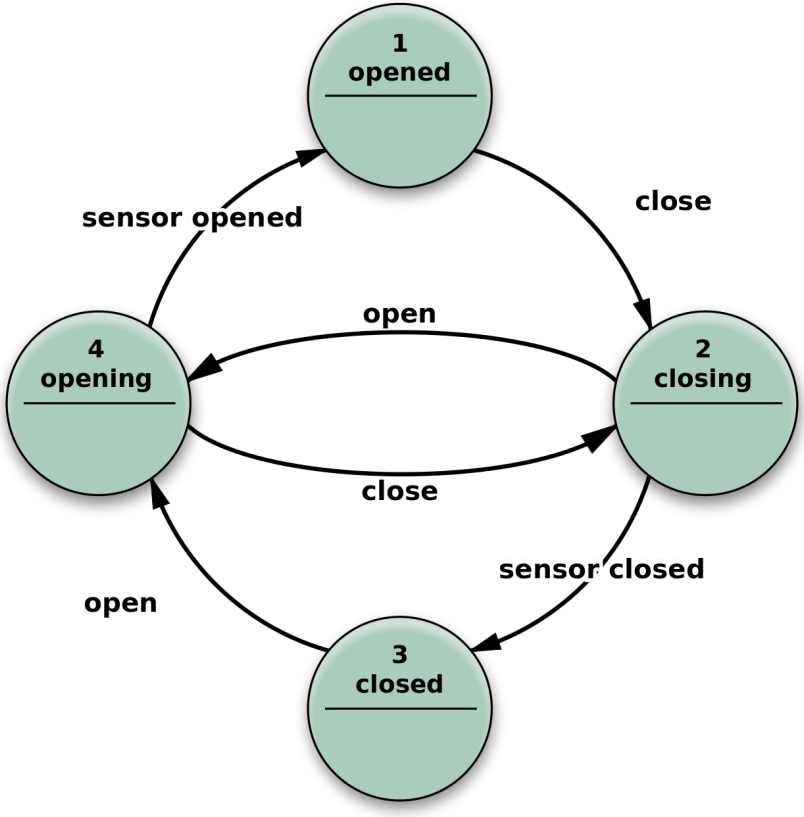
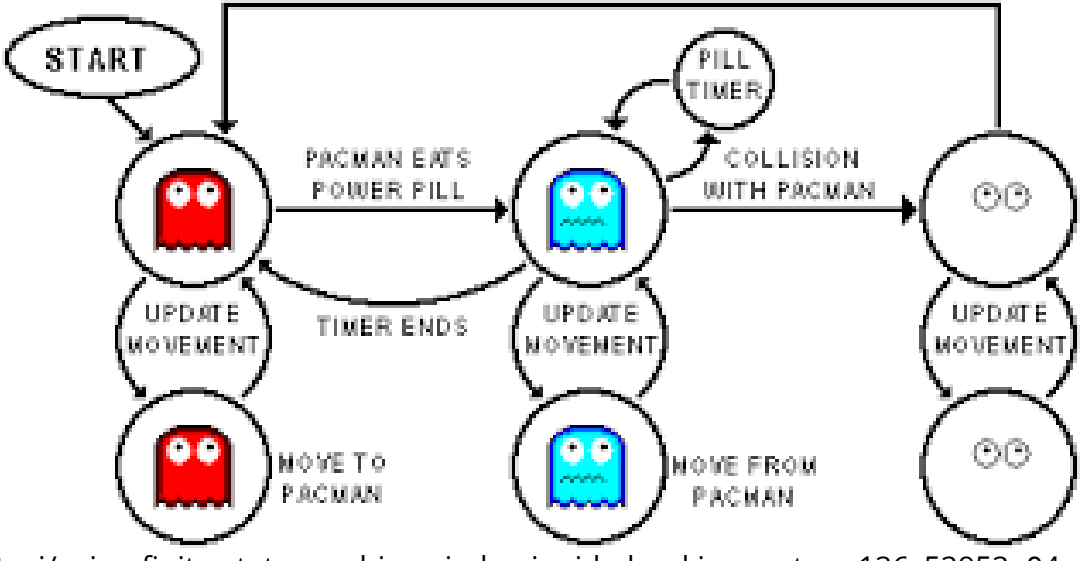
State Machines in SME

Using await/async
to implement
hardware state
machines

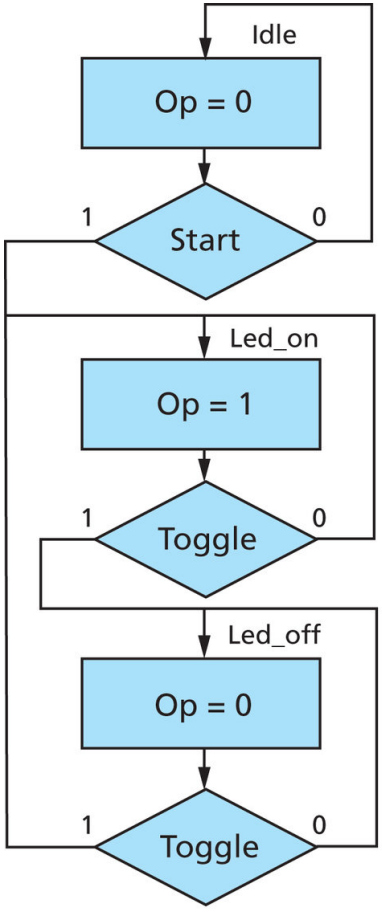
Finite state machines are very common in hardware design





https://blog.baxi.taxi/using-finite-state-machines-in-baxis-ride-booking-system-126c52952a04

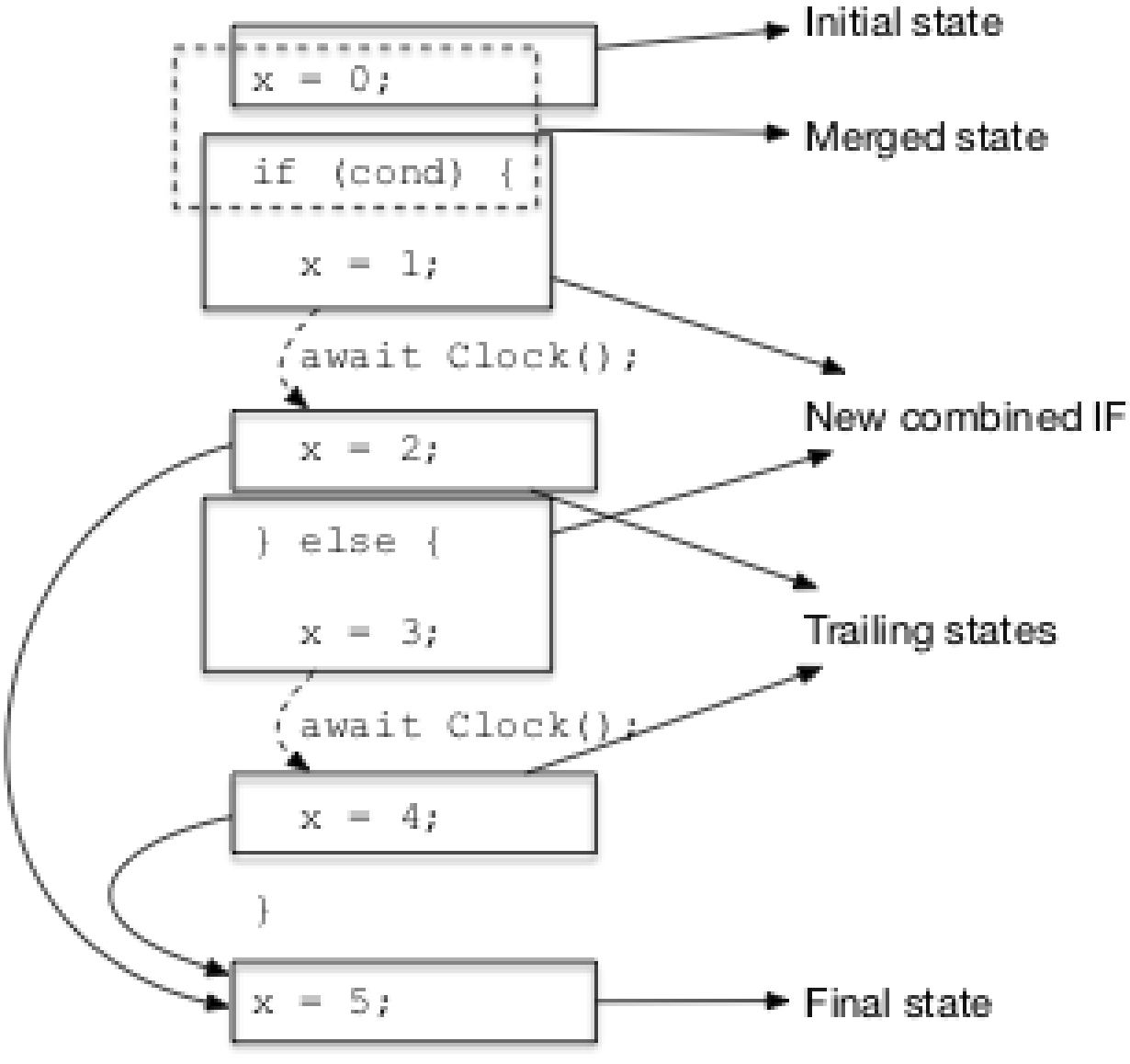
But surprisingly cumbersome to implement




```
TYPE state IS (idle, led_on, led_off);
SIGNAL current_state : state := idle;

SIGNAL timer  : unsigned(24 DOWNTO 0) := (OTHERS =>'0');
SIGNAL toggle : std_logic := '0';

BEGIN

PROCESS(clk,reset)
BEGIN
  IF reset = '1' THEN
    current_state <= idle;
    op <= '0';
  ELSIF rising_edge(clk) THEN
  CASE current_state IS
    WHEN idle =>
      op <= '0'; --output is a function of the current state only
      IF start = '1' THEN
        current_state <= led_on;
      END IF;
    WHEN led_on =>
      op <= '1';
      IF toggle = '1' THEN
        current_state <= led_off;
      END IF;
    WHEN led_off =>
      op <= '0';
      IF toggle = '1' THEN
        current_state <= led_on;
      END IF;
    WHEN OTHERS =>
      op <= '0';
      current_state <= idle;
  END CASE;
 END IF;
END PROCESS;
```

https://www.rs-online.com/designspark/how-to-implement-state-machines-in-your-fpga

Using async/await in SME



Basic example

```
async Task OnTickAsync()
{
    var tmp0 = input.a + input.b;
    var tmp1 = input.c;

    await ClockAsync();
    output.sum = tmp0 + tmp1;
}
```

```
case FSM_CurrentState is
    when FSM_State0 =>
        tmp0 := input_a + input_b;
        tmp1 := input_c;
        FSM_NextState <= FSM_State1;
    when FSM_State1 =>
        output_sum <= tmp0 + tmp1;
        FSM_NextState <= FSM_State0;
    when others =>
end case;
```

Loop example

```
async Task OnTickAsync()
{
    while (!input.valid)
        await ClockAsync();

    var count = input.count;
    for (var i = 0; i < count; i++)
    {
        output.number = i;
        output.valid = true;
        await ClockAsync();
    }
}
```

```
if FSM_RunState = FSM_State0 then
    if not (input_valid = '1') then
        FSM_NextState <= FSM_State0;
    else
        FSM_RunState := FSM_State1;
    end if;
end if;
if FSM_RunState = FSM_State1 then
    count := input_count;
    i := 0;
    FSM_RunState := FSM_State3;
end if;
if FSM_RunState = FSM_State2 then
    i := i + 1;
    FSM_RunState := FSM_State3;
end if;
if FSM_RunState = FSM_State3 then
    if TO_SIGNED(i, 32) < count then
        output_number <= TO_SIGNED(i, 32);
        output_valid <= '1';
        FSM_NextState <= FSM_State2;
    else
        FSM_NextState <= FSM_State0;
    end if;
end if;
```

Possible projects

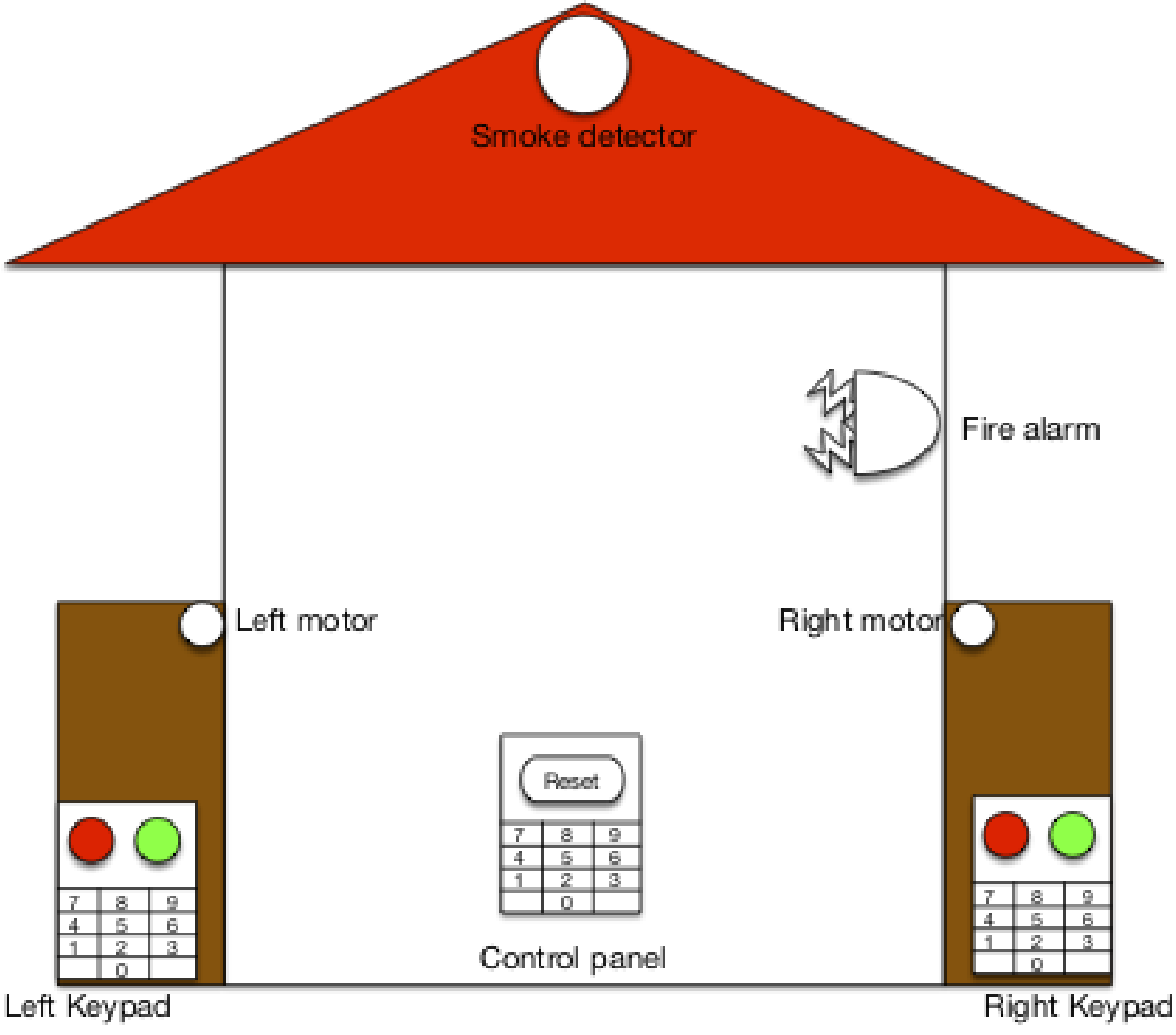
Many projects on https://projects.escience.dk

Adding async/await logic to SMEIL

Adding async function support for better composition

Introduction to the exam

Implementing a
smart house


Smoke detector
Fire alarm
Left motor
Right motor
Reset
7 8 9
4 5 6
1 2 3
0
Control panel
Left Keypad
Right Keypad

Three part exam

1. A CSP design

2. An IoT protocol

3. A hardware design

- An extension of the assignments already done, but more open-ended
- All parts have a design/analysis and an implementation phase
- Solving all sub-questions is not required for passing the exam
- Implementation is used as a method for verifying the design, not a task in itself

Need help?

- Extra practial session on the 15th 15:00
- Forum questions
- Meet us in the office (3rd floor)