ECE 327/627
Digital Hardware Systems

Lecture 6: FSM Examples

Andrew Boutros

andrew.boutros@uwaterloo.ca

Some Logistics

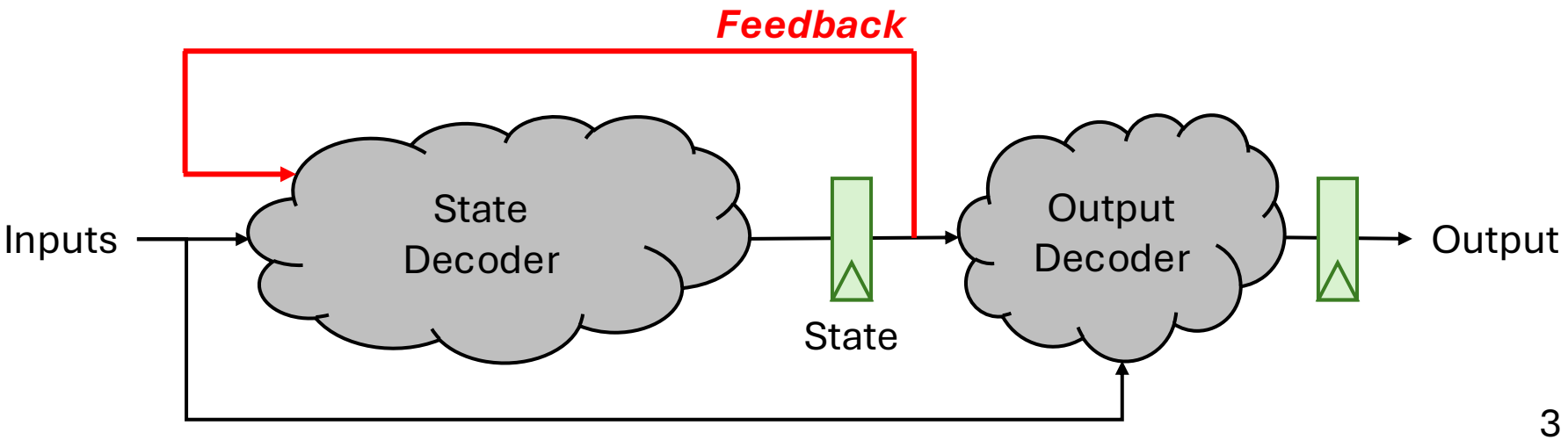
- Lab 1 is due tomorrow!
  - Worth **5%** of the total course grade for ECE 327
  - Worth **3%** of the total course grade for ECE 627
  - Deadline **January 23 @ 11:59 pm**

- Lab 1 assessment quiz in the lab sessions of next week
  - Schedule posted by Maran on Piazza
  - Please be there on time for your assessment slot

- Lab 2 will be released tomorrow … stay tuned!

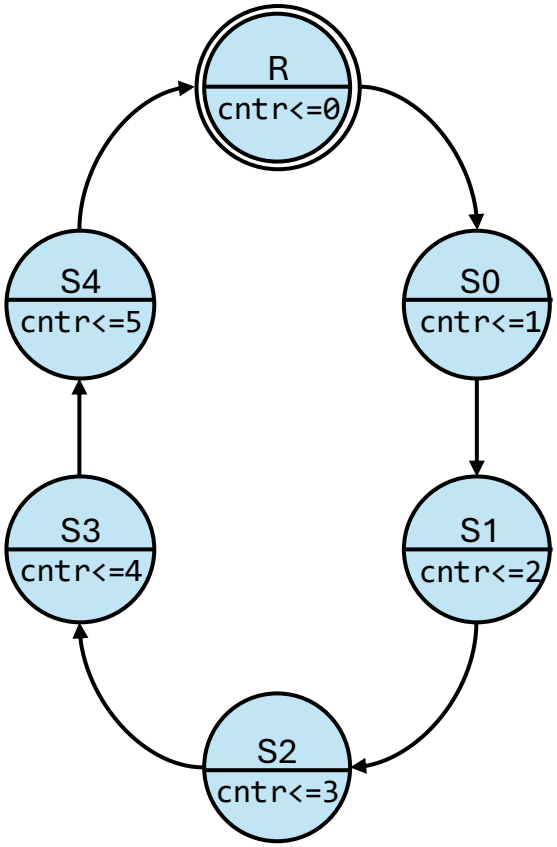
In The Previous Lecture ...

- Introduction to finite state machines (FSMs)
  - What are FSMs?
  - Visual representation of FSMs
  - FSM design recipe
- Simple counter example
  - Free-running
  - Externally enabled
  - Synchronizing outputs and state transitions



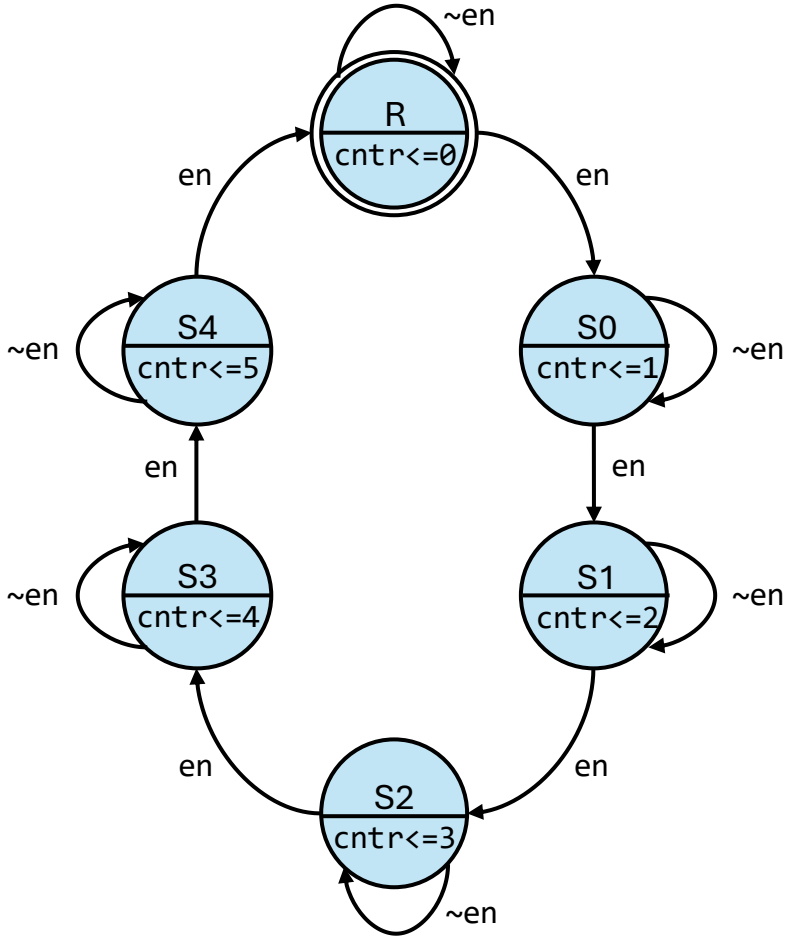
Warming Up Example

Counter FSM from last lecture …



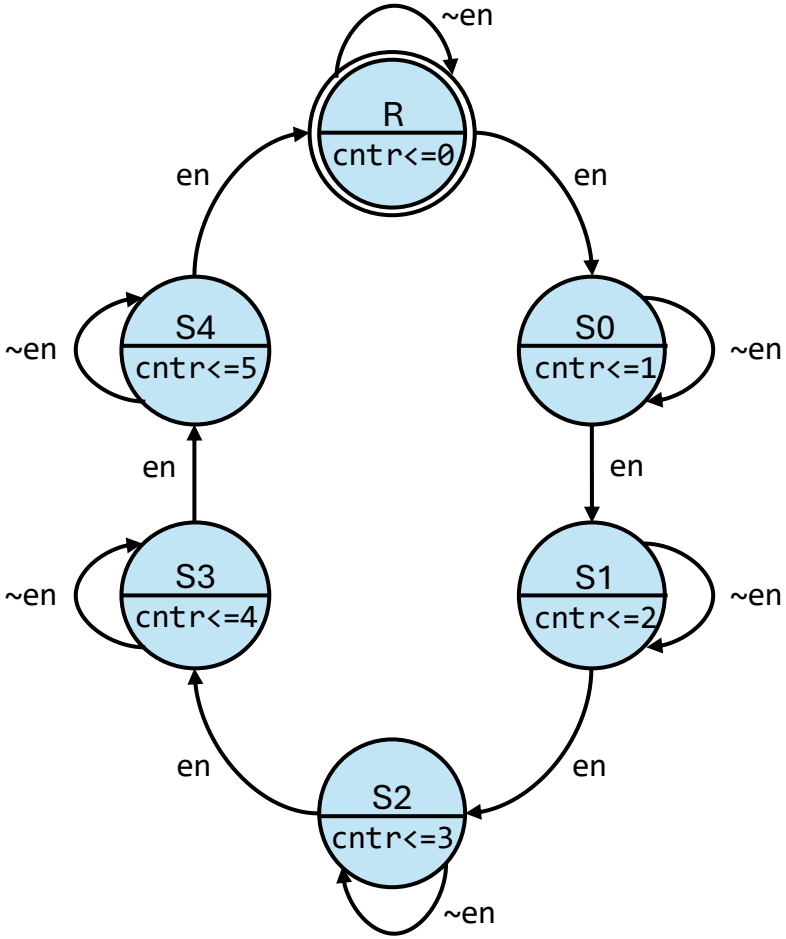
Warming Up Example

Counter FSM from last lecture with external enable signal …



Warming Up Example

What to change to add a count up/down option?



Warming Up Example

```
module cntr5_en (
      input  clk,
      input  rst,
      input  en,
      output logic [2:0] cntr
);

enum {R,S0,S1,S2,S3,S4} state, next_state;
logic [2:0] cntr_val;

always_ff @ (posedge clk) begin
    if (rst) begin
        cntr <= 'd0;
        state <= R;
    end else begin
        cntr <= cntr_val;
        state <= next_state;
    end
end
```

```
always_comb begin: state_decoder
  case (state)
    R : next_state = (en)? S0 : R;
    S0: next_state = (en)? S1 : S0;
    S1: next_state = (en)? S2 : S1;
    S2: next_state = (en)? S3 : S2;
    S3: next_state = (en)? S4 : S3;
    S4: next_state = (en)?  R : S4;
    default: next_state = R;
  endcase
end

always_comb begin: out_decoder
  case (state)
    R : cntr_val = 0;
    S0: cntr_val = 1;
    S1: cntr_val = 2;
    S2: cntr_val = 3;
    S3: cntr_val = 4;
    S4: cntr_val = 5;
    default: cntr_val = 0;
  endcase
end

endmodule
```

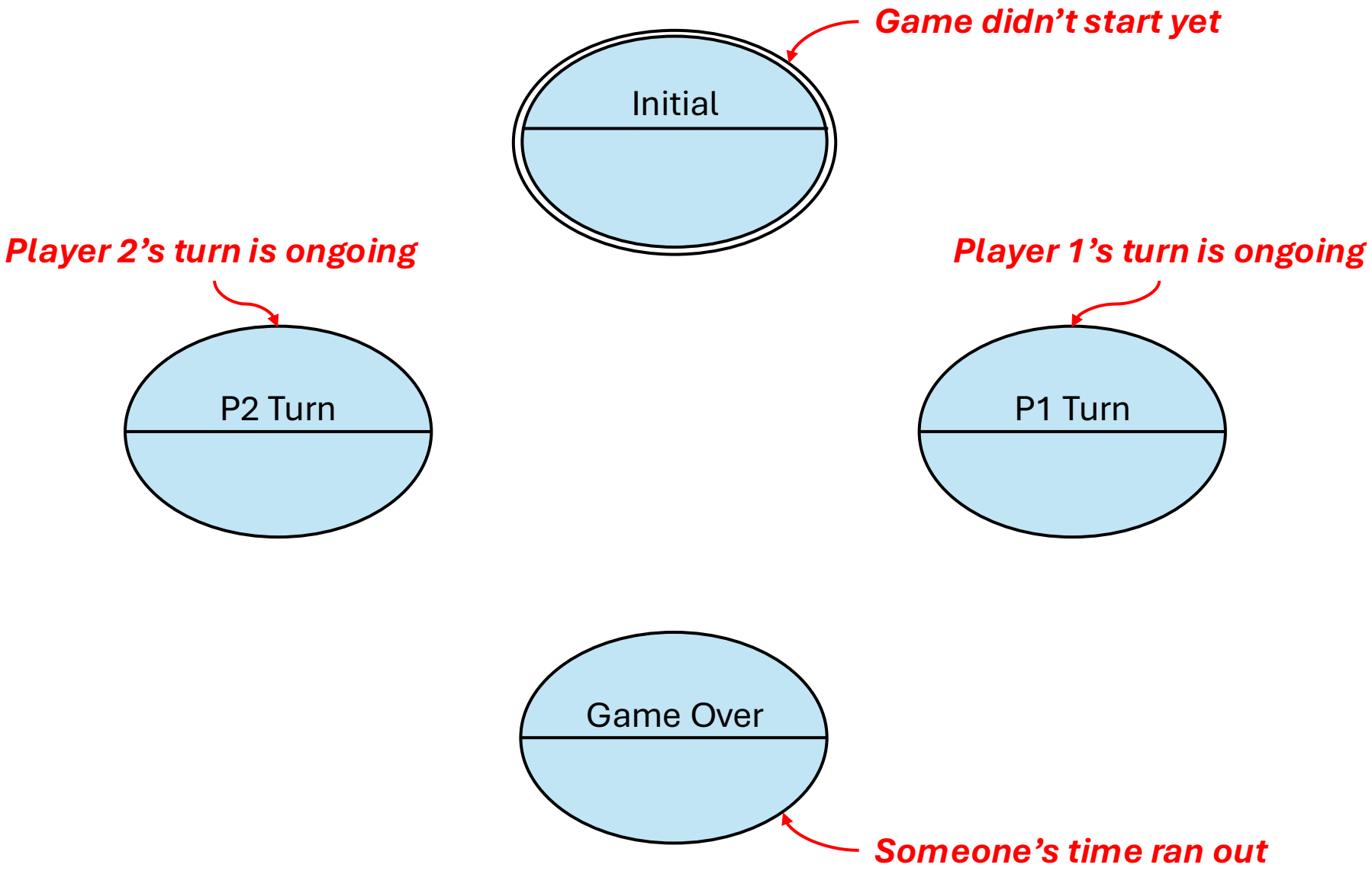
What changes in the SystemVerilog code?

Example: Chess Clock

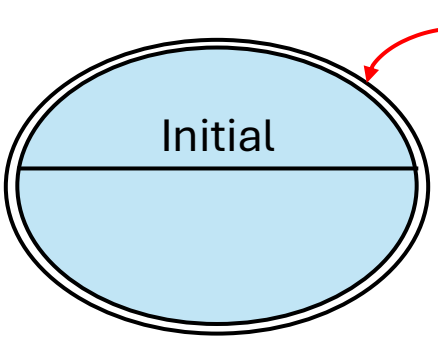
- Two count down clocks for players *P1* & *P2*
- Set initial time budget (`T_INIT`) on both clocks (e.g., 10 mins)
- A player loses when their time counts down to zero
- After *P1* finishes turn, press the button:
  - Countdown stops for *P1*
  - *A time bonus* (`T_BONUS`) *is added to P1 (e.g., 10 sec)*
  - *Countdown starts for P2*
- The two clocks are never running at the same time

Source: digitalgametechnology.com

Example: Chess Clock (FSM Design)

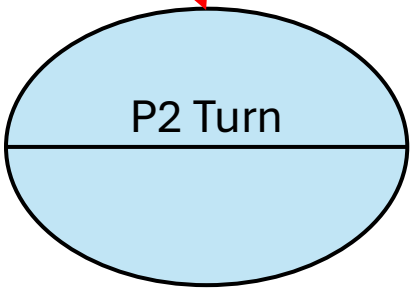


Example: Chess Clock (FSM Design)

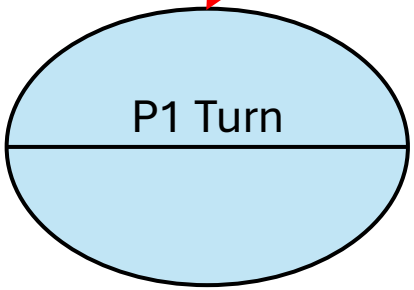


Game didn't start yet

Player 2's turn is ongoing

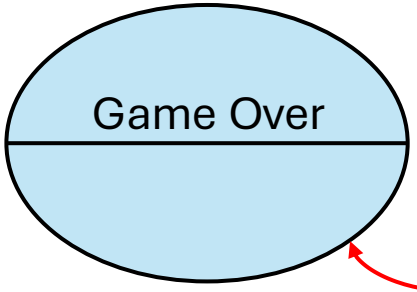


Player 1's turn is ongoing



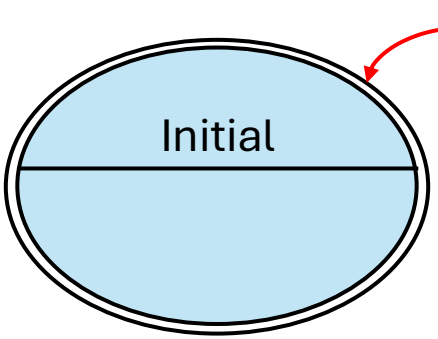
What are the inputs?

What are the outputs?



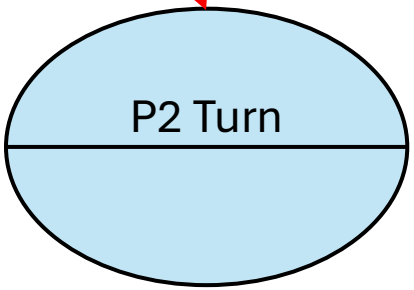
Someone's time ran out

Example: Chess Clock (FSM Design)

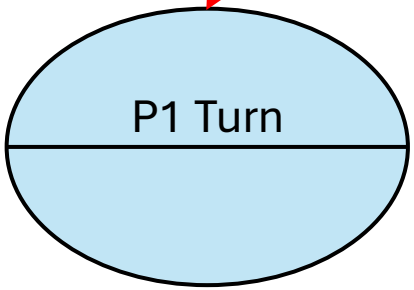


Game didn't start yet

Player 2's turn is ongoing



Player 1's turn is ongoing

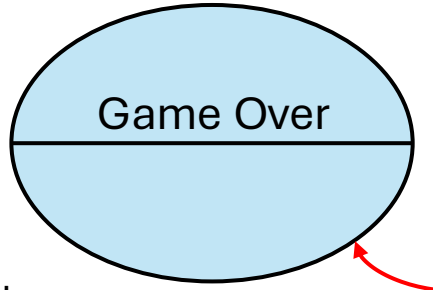


What are the inputs?

1. input button specifying which player's turn it is (`turn`)

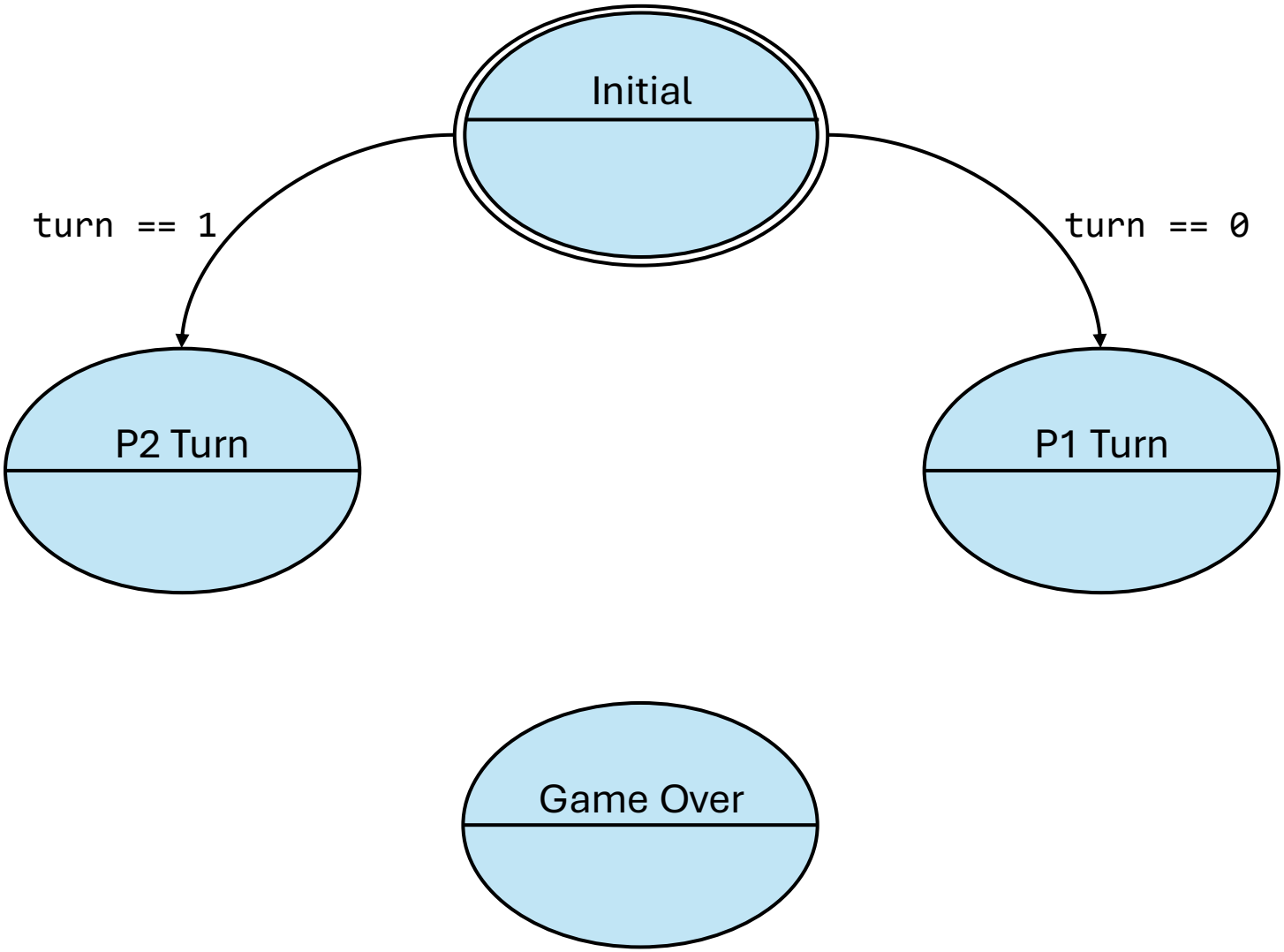
What are the outputs?

1. time remaining for both players (`p1_t, p2_t`)
2. flag specifying if a player's time is out (`gameover`)

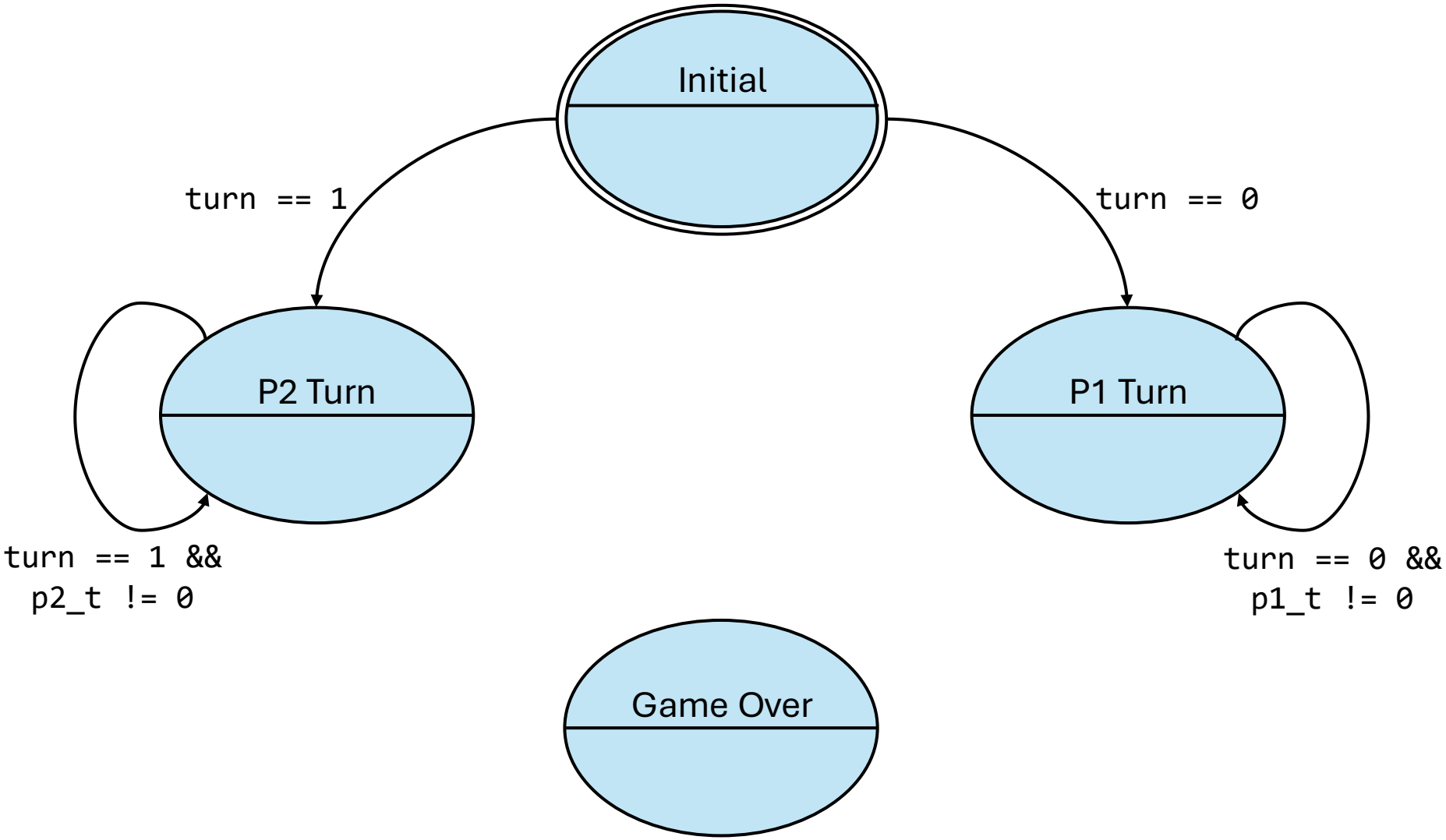


Someone's time ran out

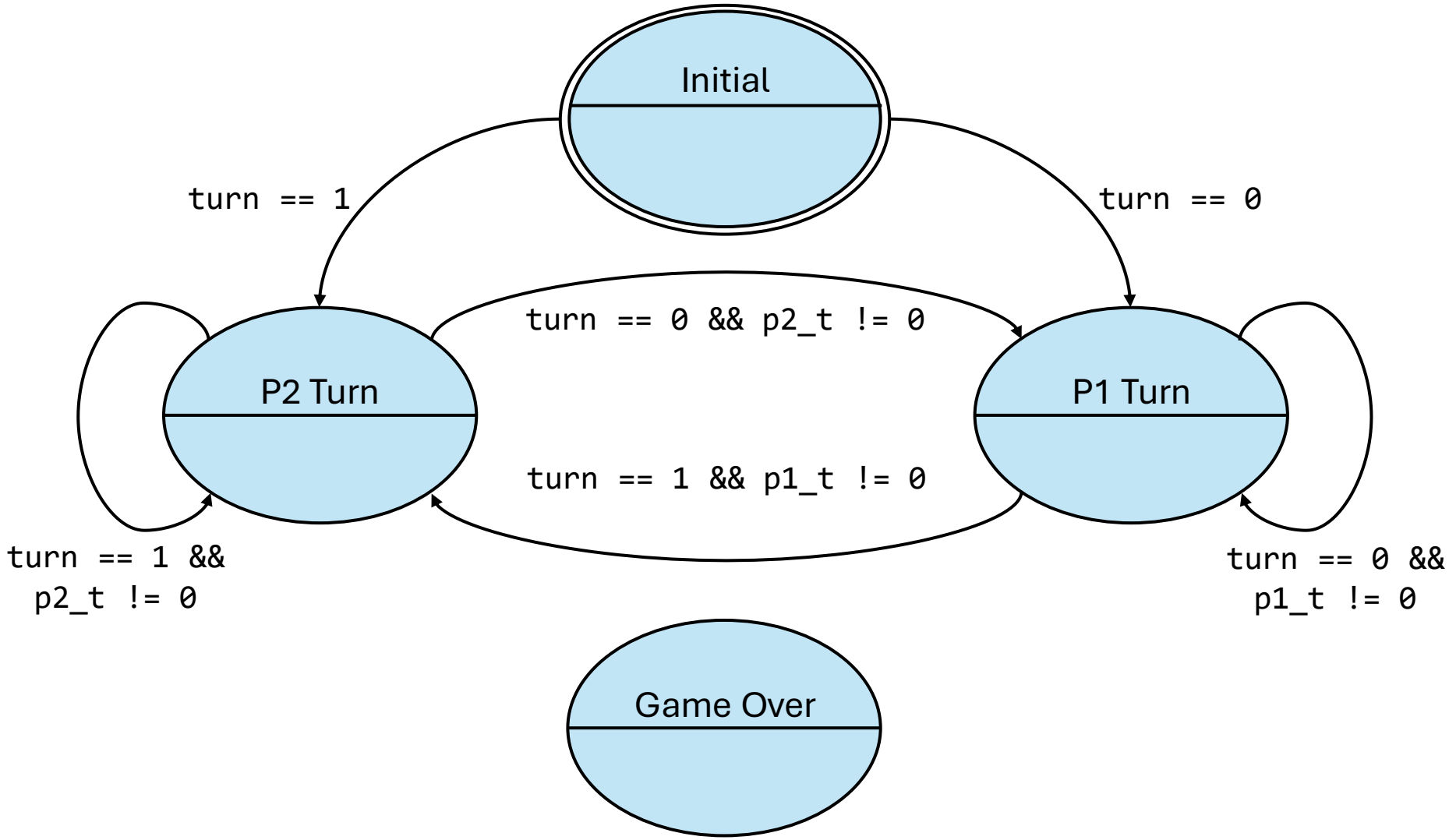
Example: Chess Clock (FSM Design)



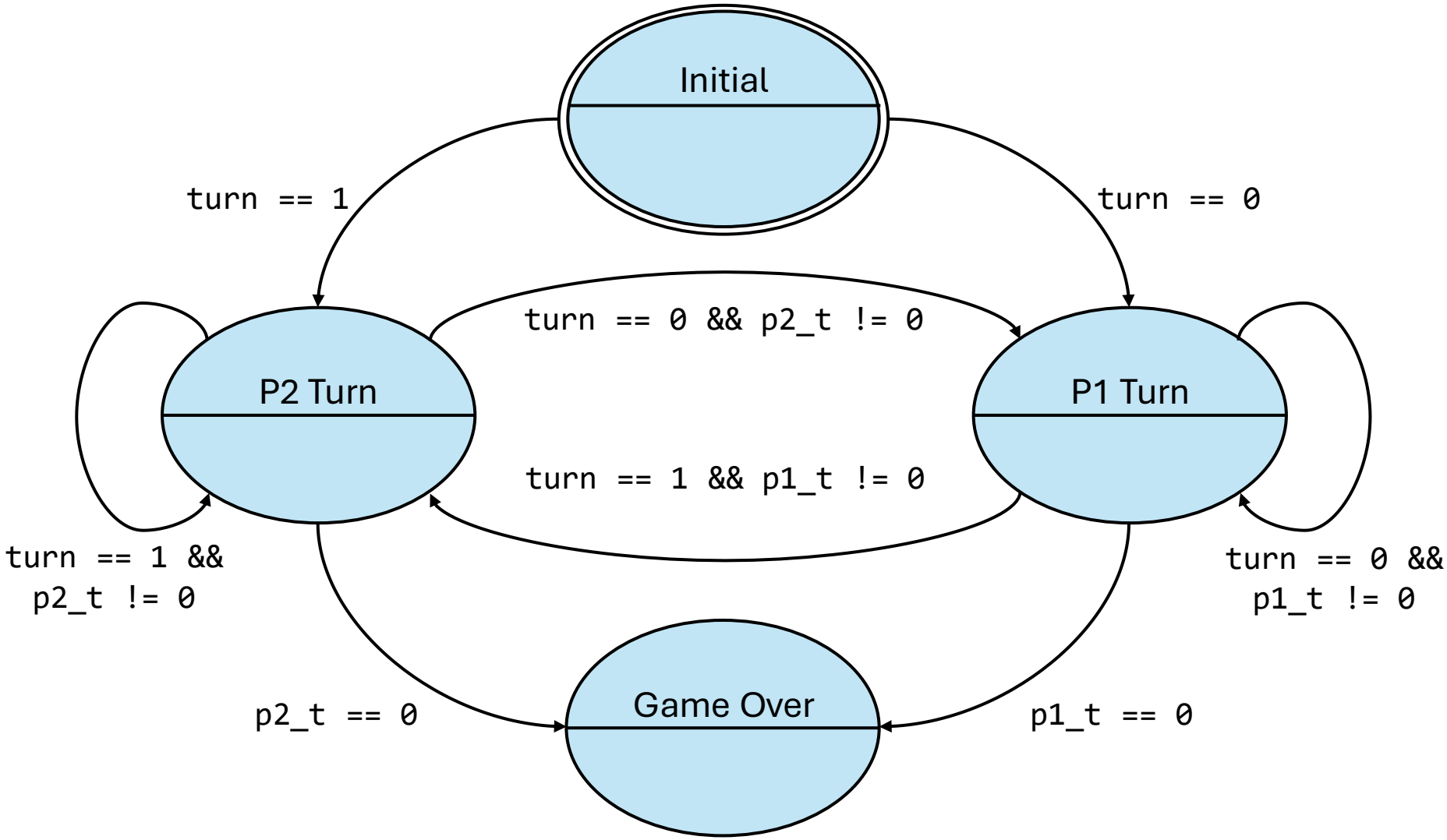
Example: Chess Clock (FSM Design)



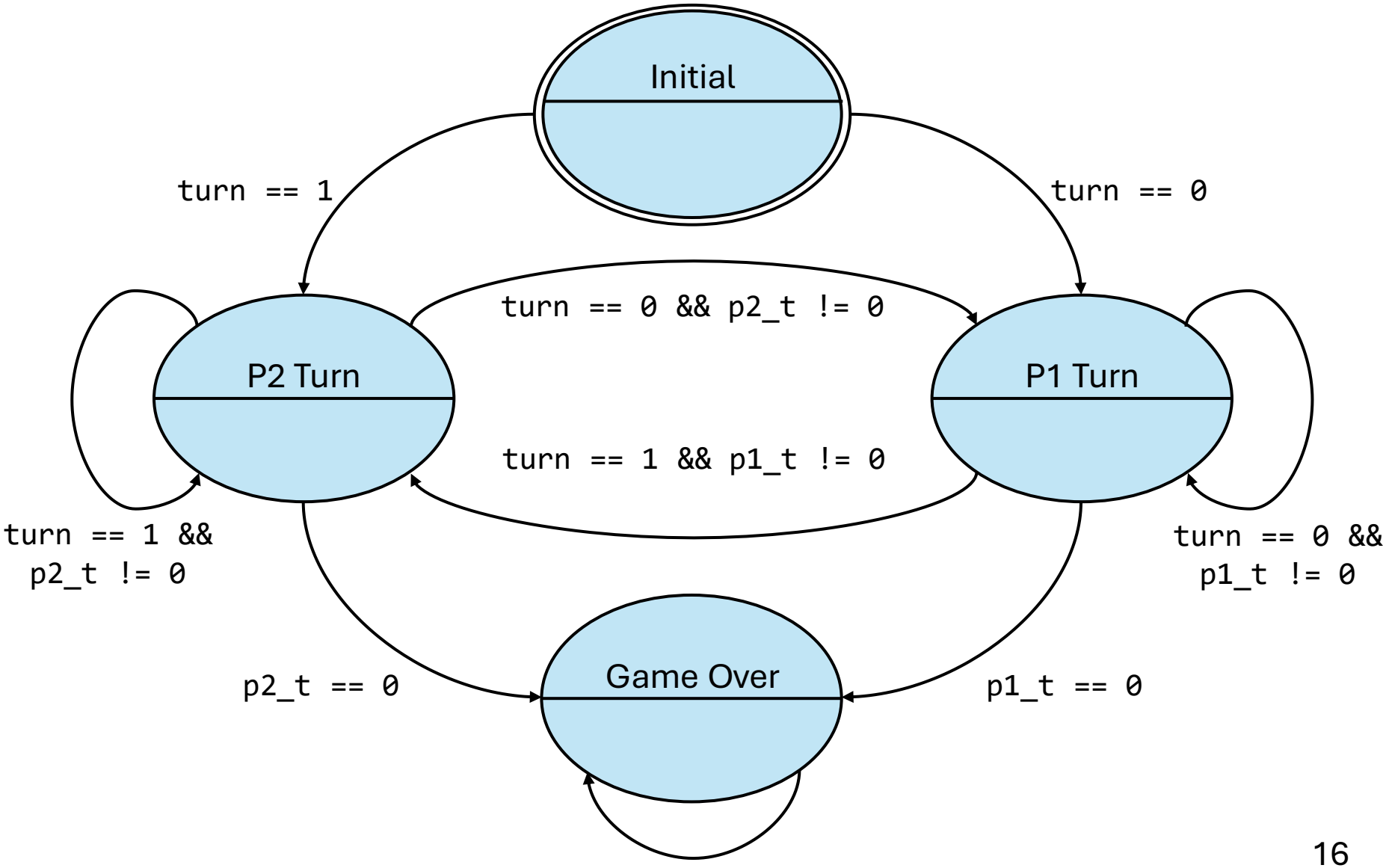
Example: Chess Clock (FSM Design)



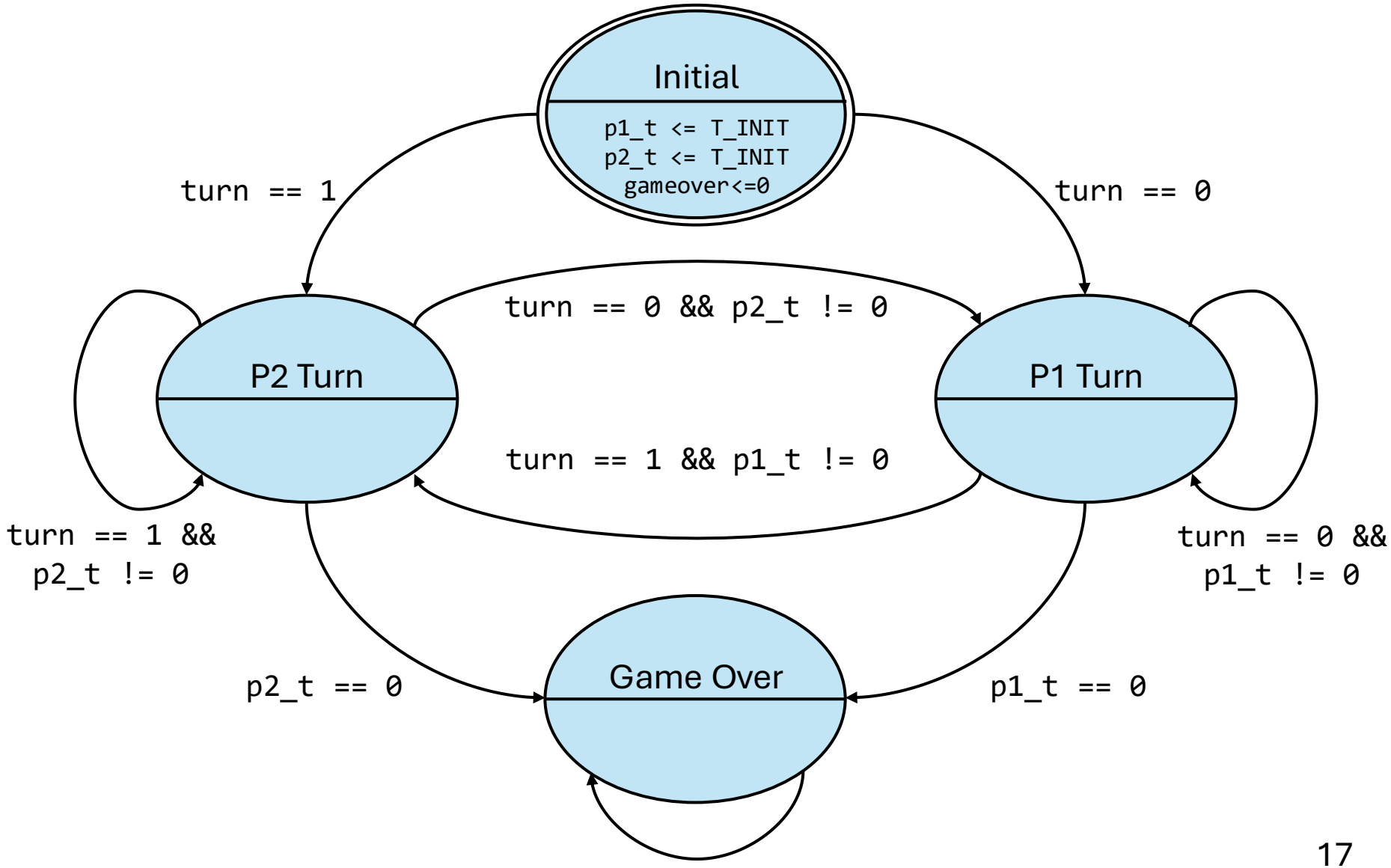
Example: Chess Clock (FSM Design)



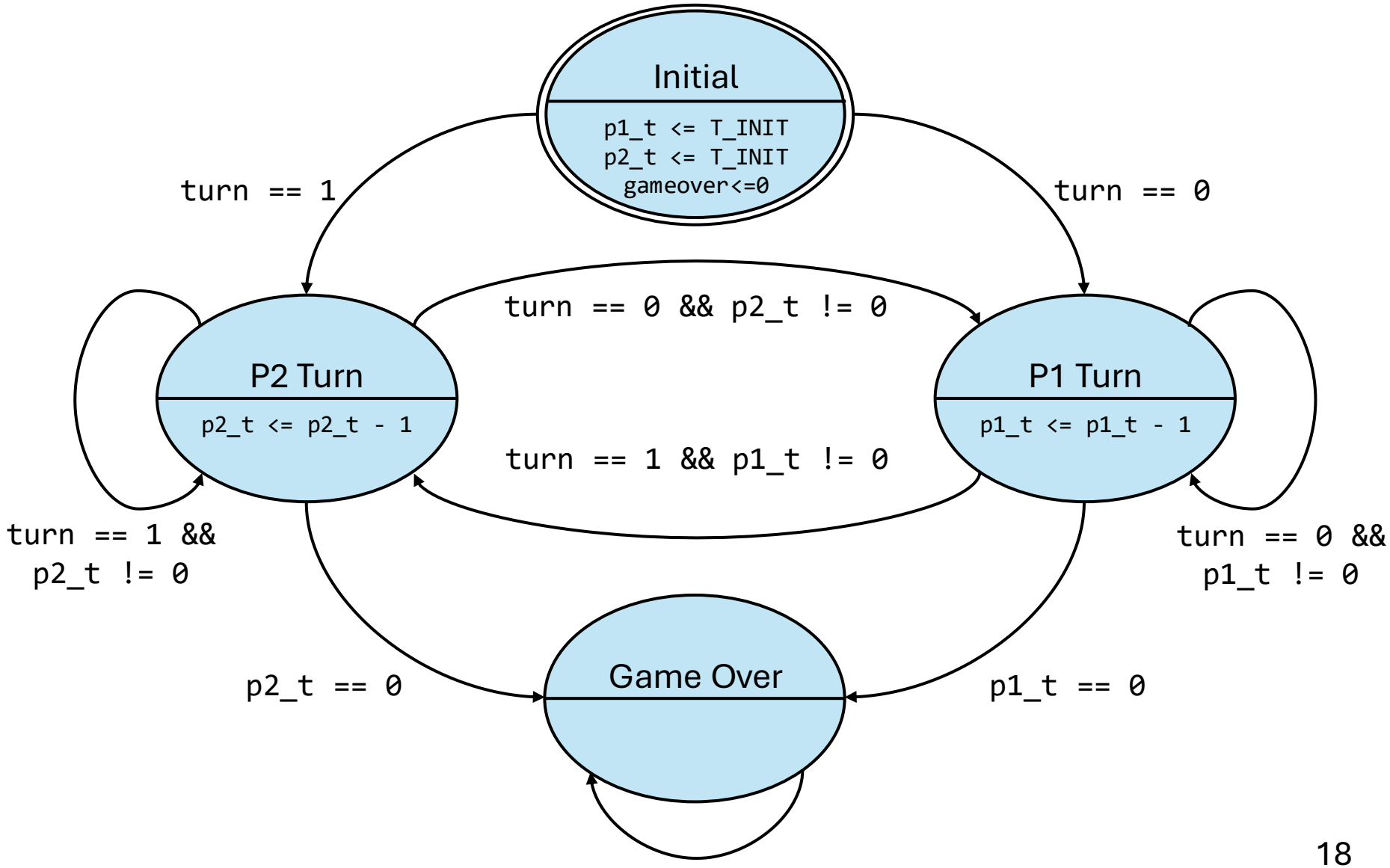
Example: Chess Clock (FSM Design)



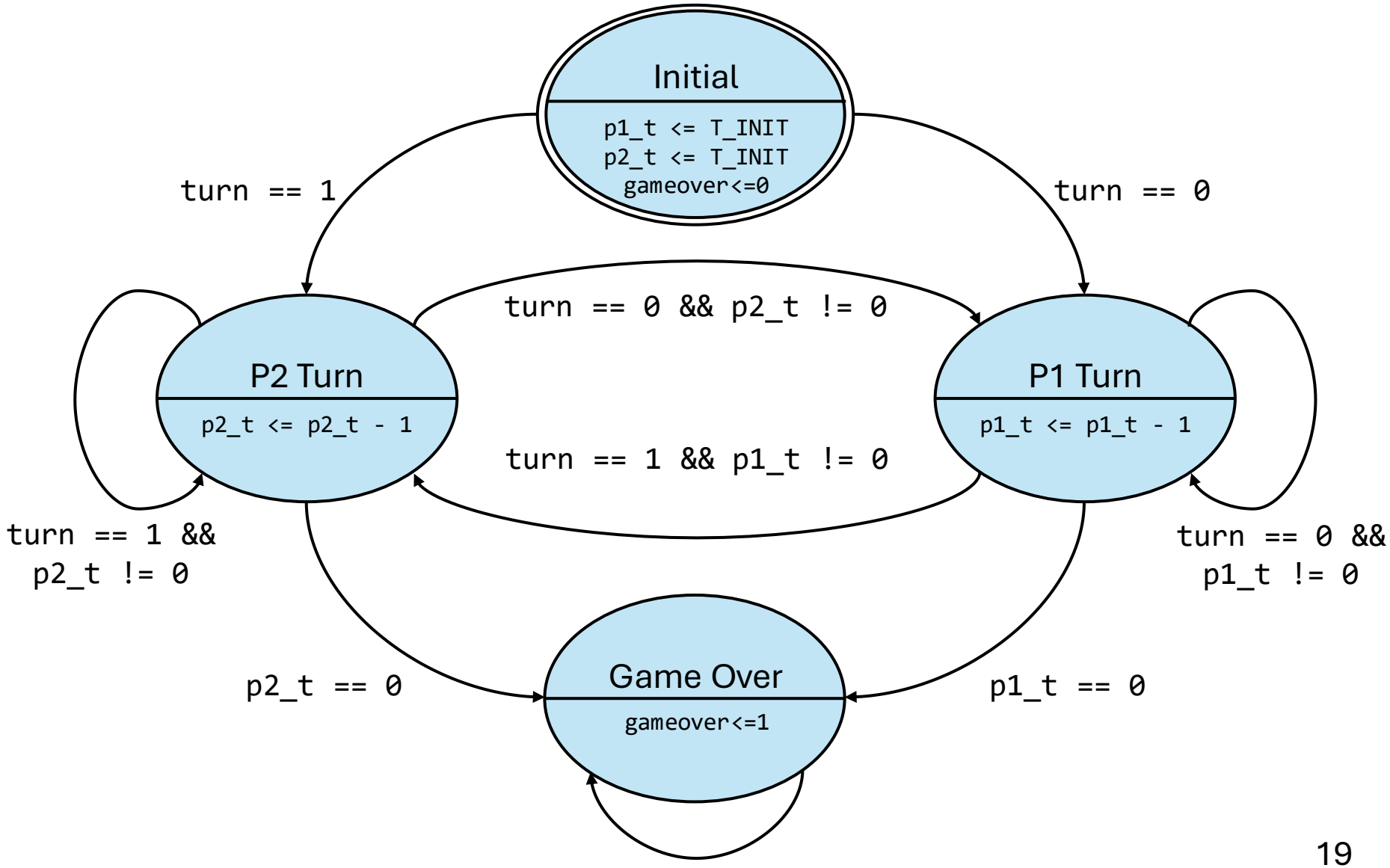
Example: Chess Clock (FSM Design)



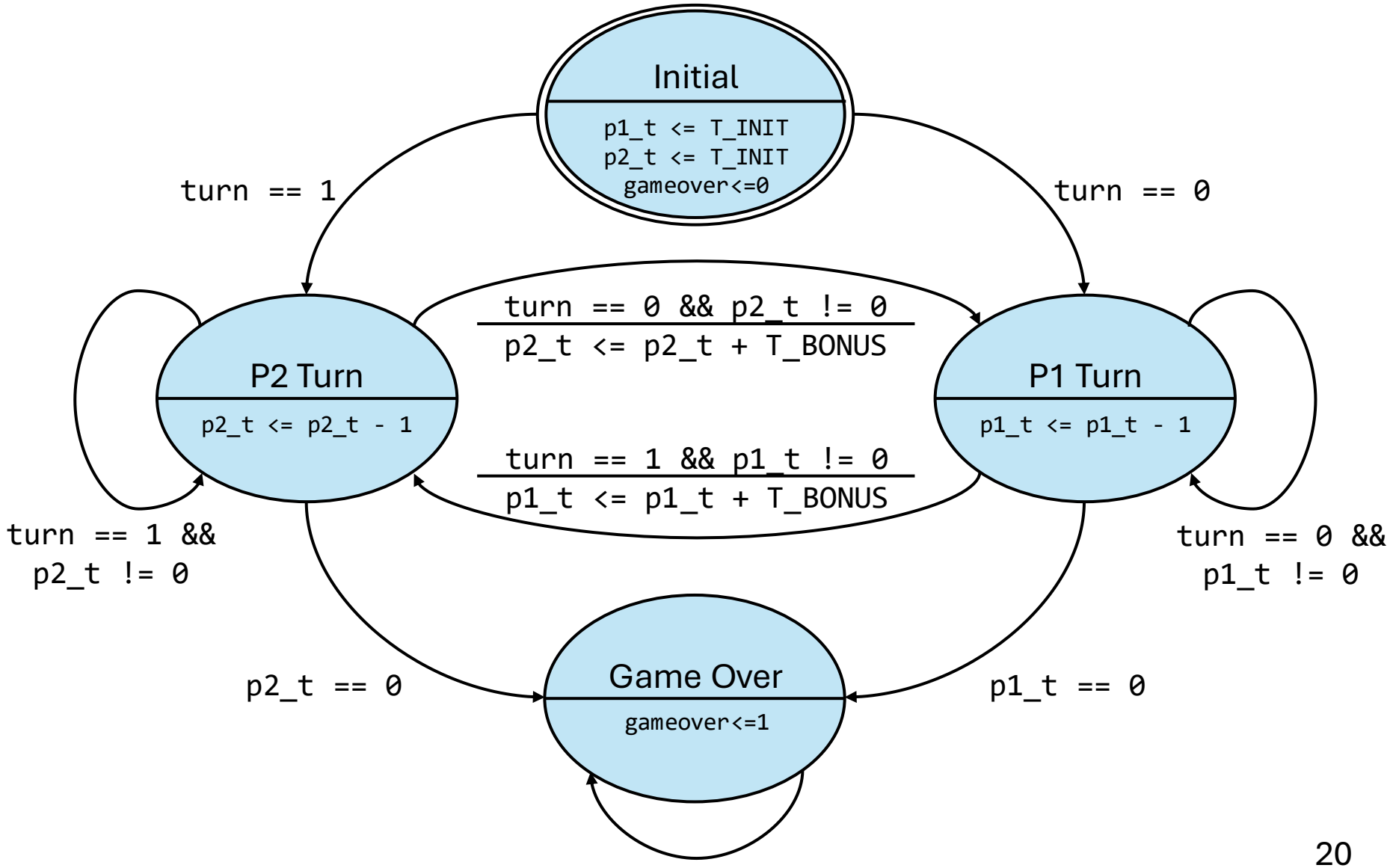
Example: Chess Clock (FSM Design)



Example: Chess Clock (FSM Design)

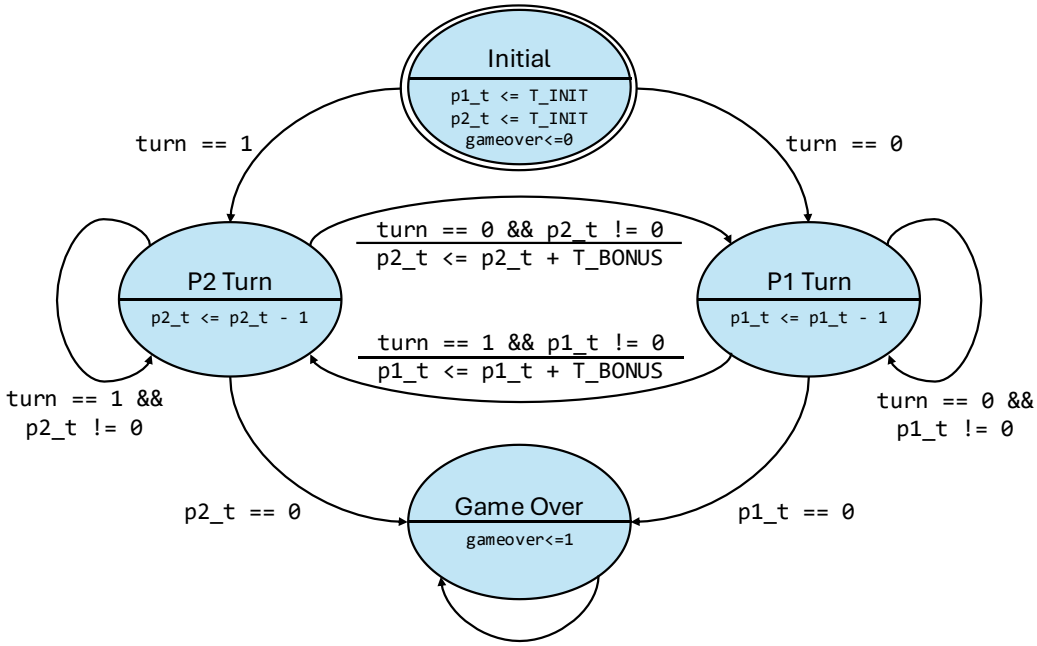


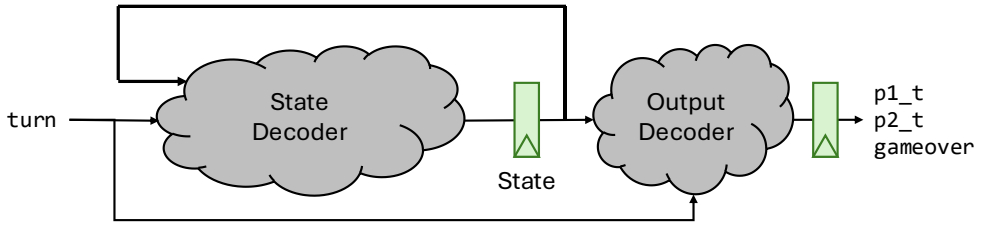
Example: Chess Clock (FSM Design)



Example: Chess Clock (Code)

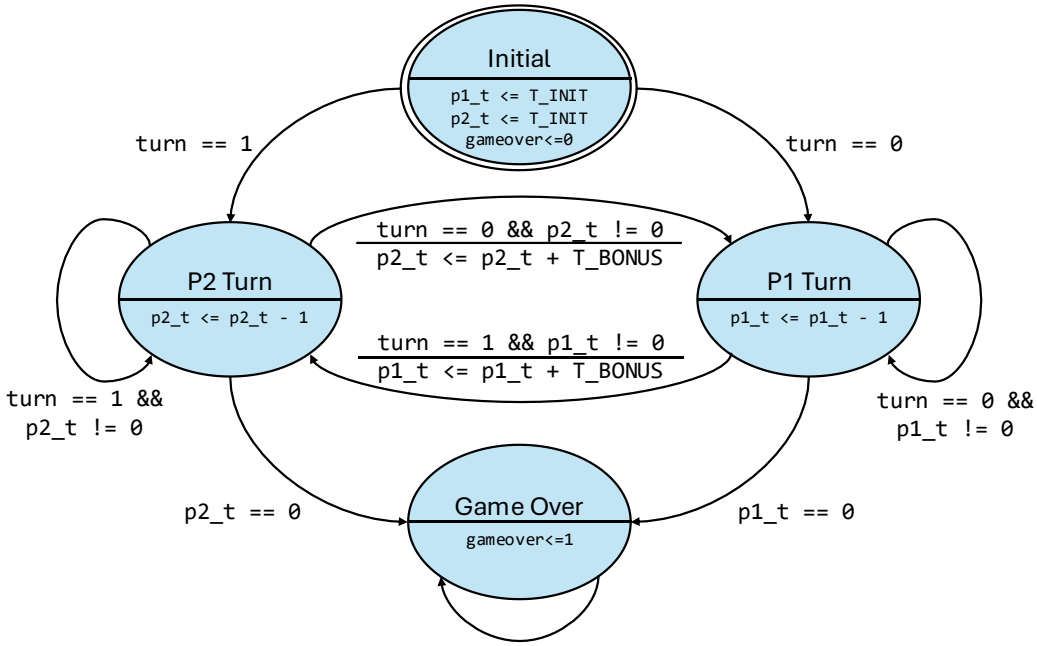
```
module chess_clock #(
      parameter T_INIT = 600,
      parameter T_BONUS = 10
)(
      input  clk,
      input  rst,
      input  turn,
      output logic [15:0] p1_t,
      output logic [15:0] p2_t,
      output logic gameover
);
```

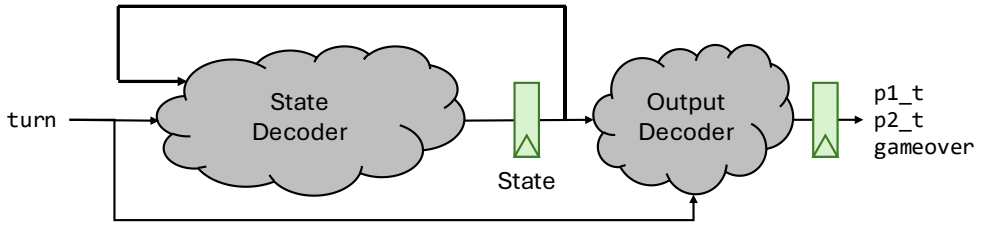




Example: Chess Clock (Code)

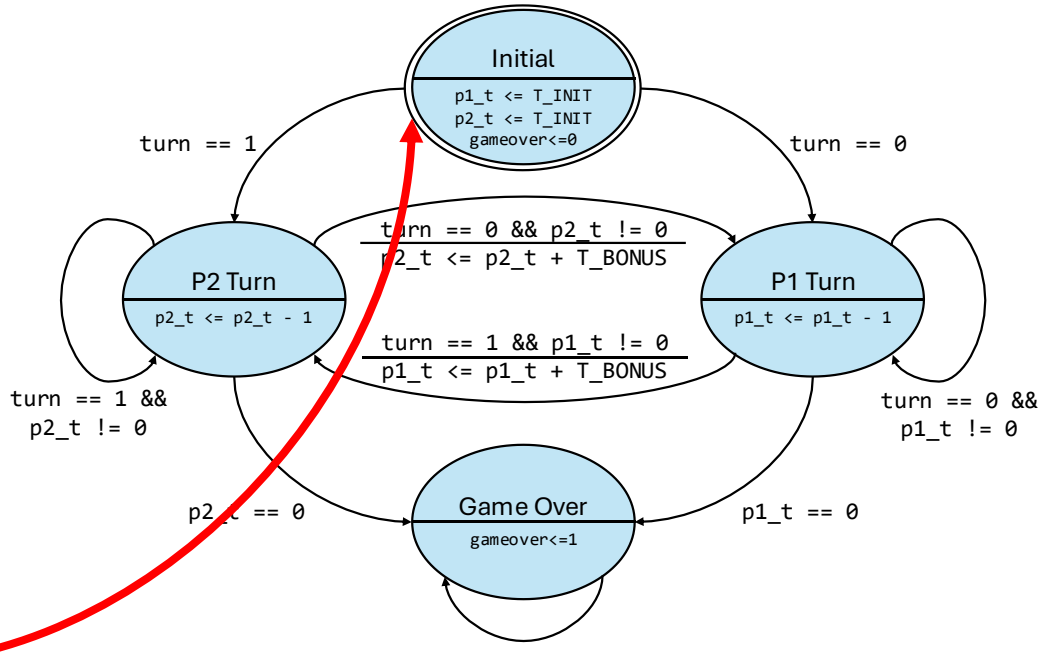
```
module chess_clock #(
      parameter T_INIT = 600,
      parameter T_BONUS = 10
)(
      input  clk,
      input  rst,
      input  turn,
      output logic [15:0] p1_t,
      output logic [15:0] p2_t,
      output logic gameover
);

enum {INIT,P1,P2,GOVER} state, next_state;
logic [15:0] p1_t_val, p2_t_val;
logic gameover_val;
```

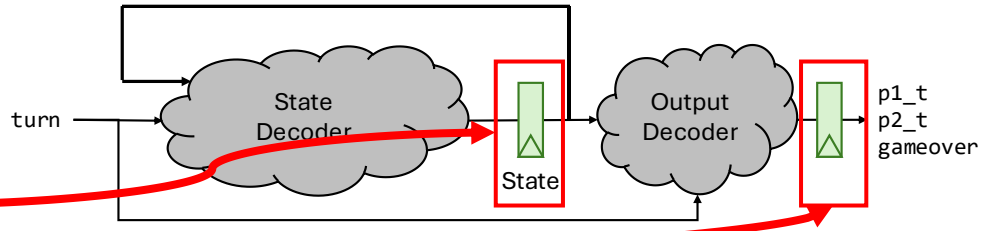




Example: Chess Clock (Code)

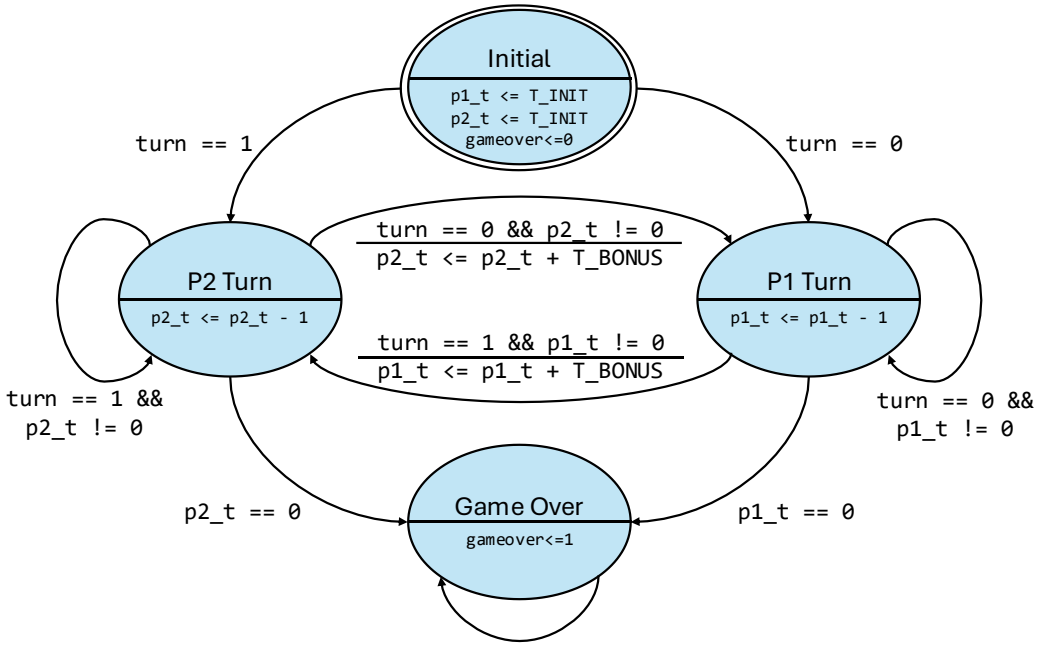
```
module chess_clock #(
      parameter T_INIT = 600,
      parameter T_BONUS = 10
)(
      input  clk,
      input  rst,
      input  turn,
      output logic [15:0] p1_t,
      output logic [15:0] p2_t,
      output logic gameover
);

enum {INIT,P1,P2,GOVER} state, next_state;
logic [15:0] p1_t_val, p2_t_val;
logic gameover_val;

always_ff @ (posedge clk) begin
    if (rst) begin
        p1_t <= T_INIT; p2_t <= T_INIT;
        gameover <= 1'b0;
        state <= INIT;
    end else begin
        p1_t <= p1_t_val; p2_t <= p2_t_val;
        gameover <= gameover_val;
        state <= next_state;
    end
end
```

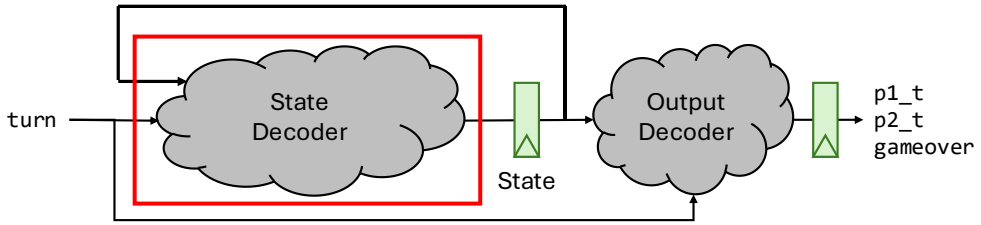




Example: Chess Clock (Code)

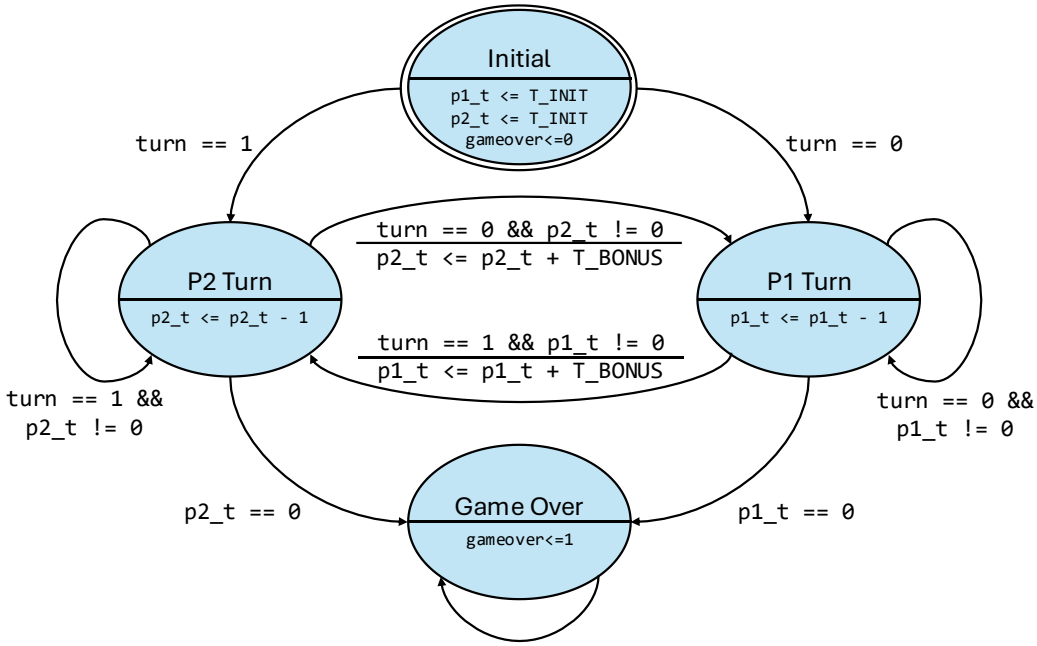
```
always_comb begin: state_decoder
  case (state)
    INIT : next_state = (turn)? P2 : P1;

    P1: begin
     if (turn && p1_t!=0) next_state = P2;
     else if (p1_t==0) next_state = GOVER;
     else next_state = P1;
    end

    P2: begin
     if (~turn && p2_t!=0) next_state = P1;
     else if (p2_t==0) next_state = GOVER;
     else next_state = P2;
    end

    GOVER: next_state = GOVER;

    default: next_state = INIT;
  endcase
end
```

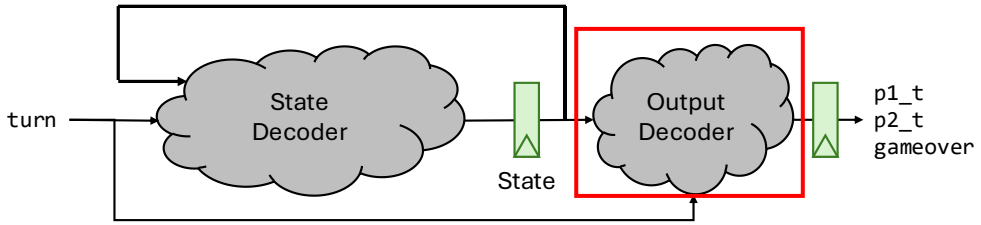




Example: Chess Clock (Code)

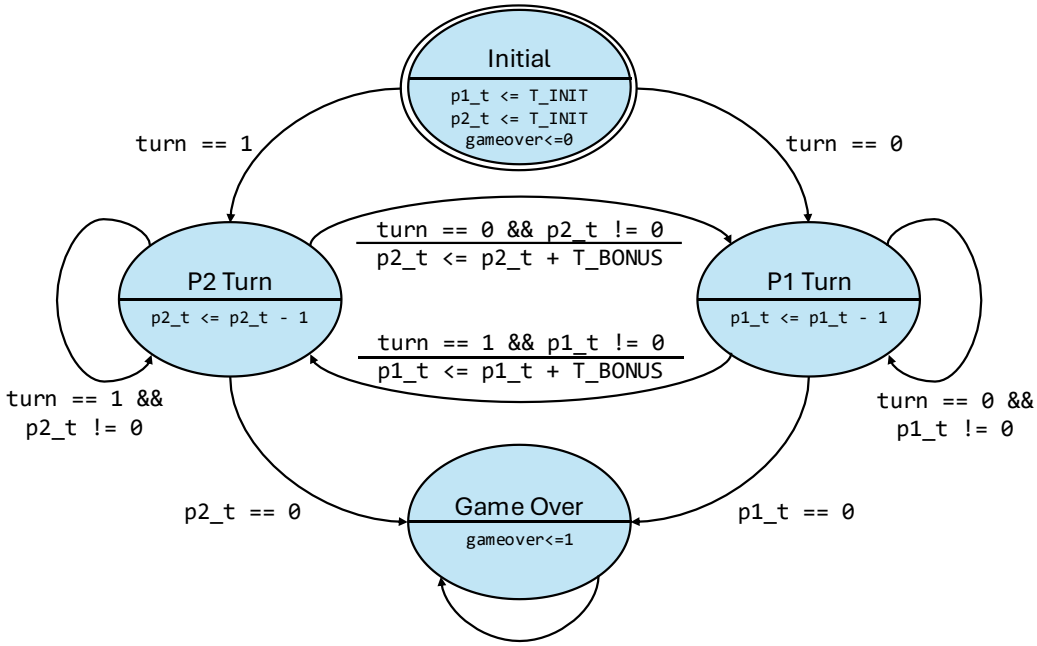
```
always_comb begin: out_decoder
  case (state)
    INIT: begin
      p1_t_val = T_INIT; p2_t_val = T_INIT;
      gameover_val = 1'b0;
    end

    P1: begin
      if (turn && p1_t!=0)
        p1_t_val = p1_t + T_BONUS;
      else p1_t_val = p1_t - 1;
    end

    P2: begin
      if (~turn && p2_t!=0)
        p2_t_val = p2_t + T_BONUS;
      else p2_t_val = p2_t - 1;
    end

    GOVER: gameover_val = 1'b1;

    default: begin
      p1_t_val = T_INIT; p2_t_val = T_INIT;
      gameover_val = 1'b0;
    end
  endcase
end

endmodule
```

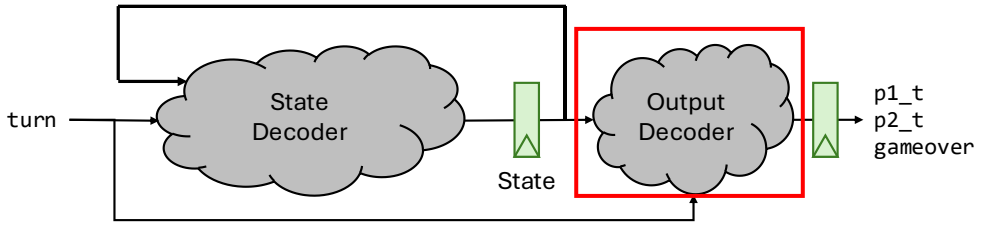




Example: Chess Clock (Code)

```
always_comb begin: out_decoder
  case (state)
    INIT: begin
      p1_t_val = T_INIT; p2_t_val = T_INIT;
      gameover_val = 1'b0;
    end

    P1: begin
      if (turn && p1_t!=0)
        p1_t_val = p1_t + T_BONUS;
      else p1_t_val = p1_t - 1;
    end

    P2: begin
      if (~turn && p2_t!=0)
        p2_t_val = p2_t + T_BONUS;
      else p2_t_val = p2_t - 1;
    end

    GOVER: gameover_val = 1'b1;

    default: begin
      p1_t_val = T_INIT; p2_t_val = T_INIT;
      gameover_val = 1'b0;
    end
  endcase
end

endmodule
```





Think how to extend to 4 players …

Example: Music Toy

- Toy gives a sequence of N music tones & waits for user to enter the same sequence correctly
- If correct tones are entered, advance to next level & give a sequence of N+1 tones
- If incorrect tones are entered, stay in the same level & give the same sequence of N tones again
- Assume the toy has a total of 3 levels (for simplicity) & the full sequence is hardcoded

My kid's music toy ☺

Example game:

Toy: Re (level 1) → User: Re

Toy: Re, Do (level 2) → User: Fa

Toy: Re, Do (level 2) → User: Re, Do

Toy: Re, Do, Mi (level 3) → User: Re, Do, Mi

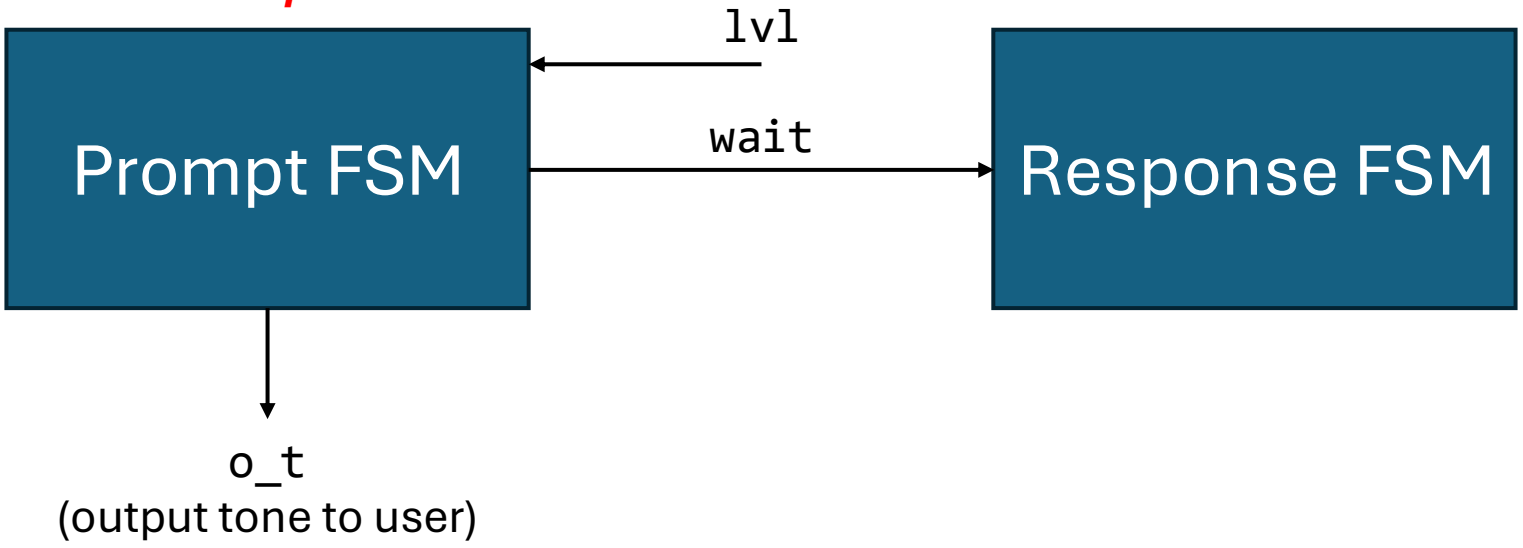
Example: Music Toy - Two Interacting FSMs

Prompt FSM

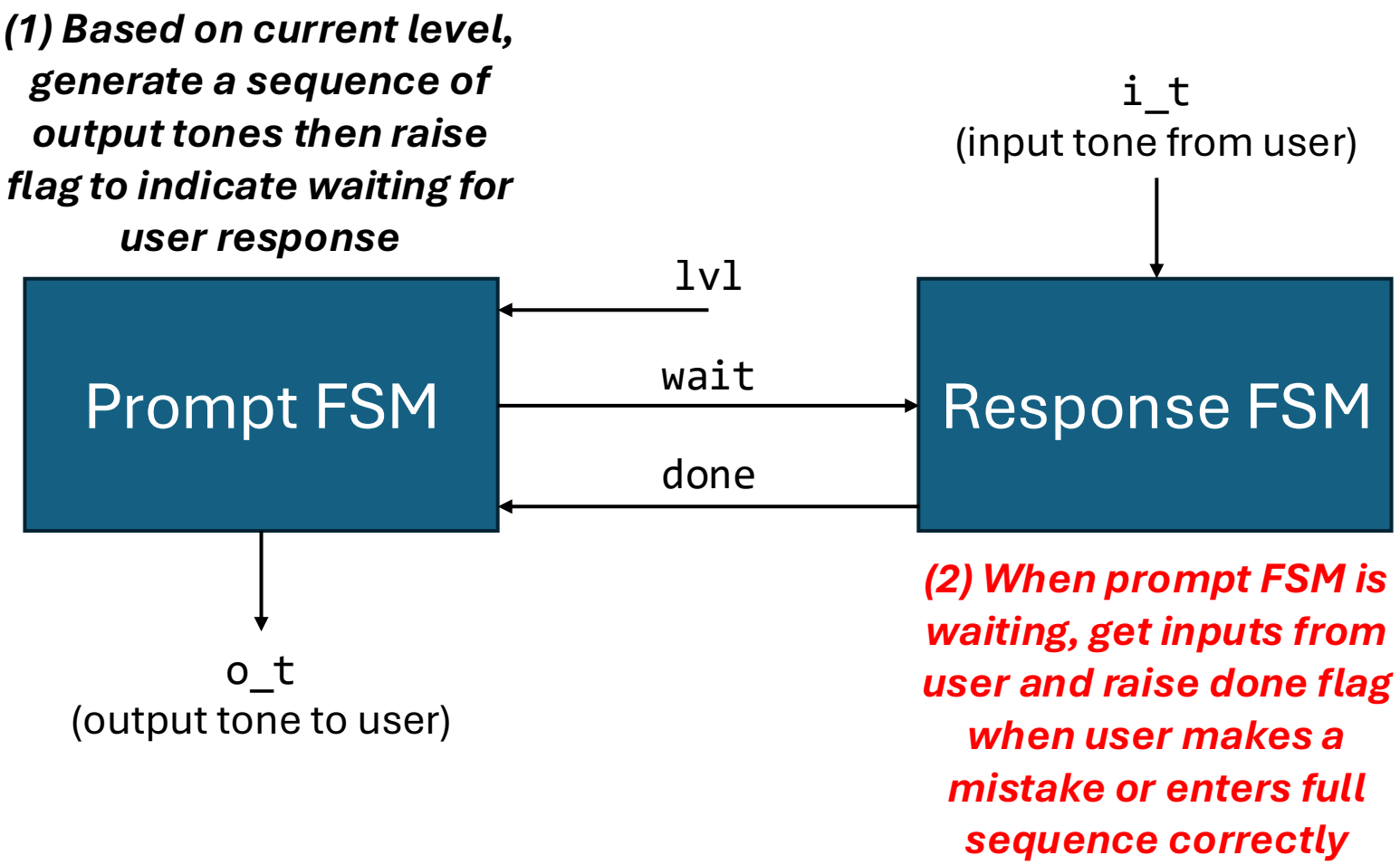
Response FSM

Example: Music Toy - Two Interacting FSMs

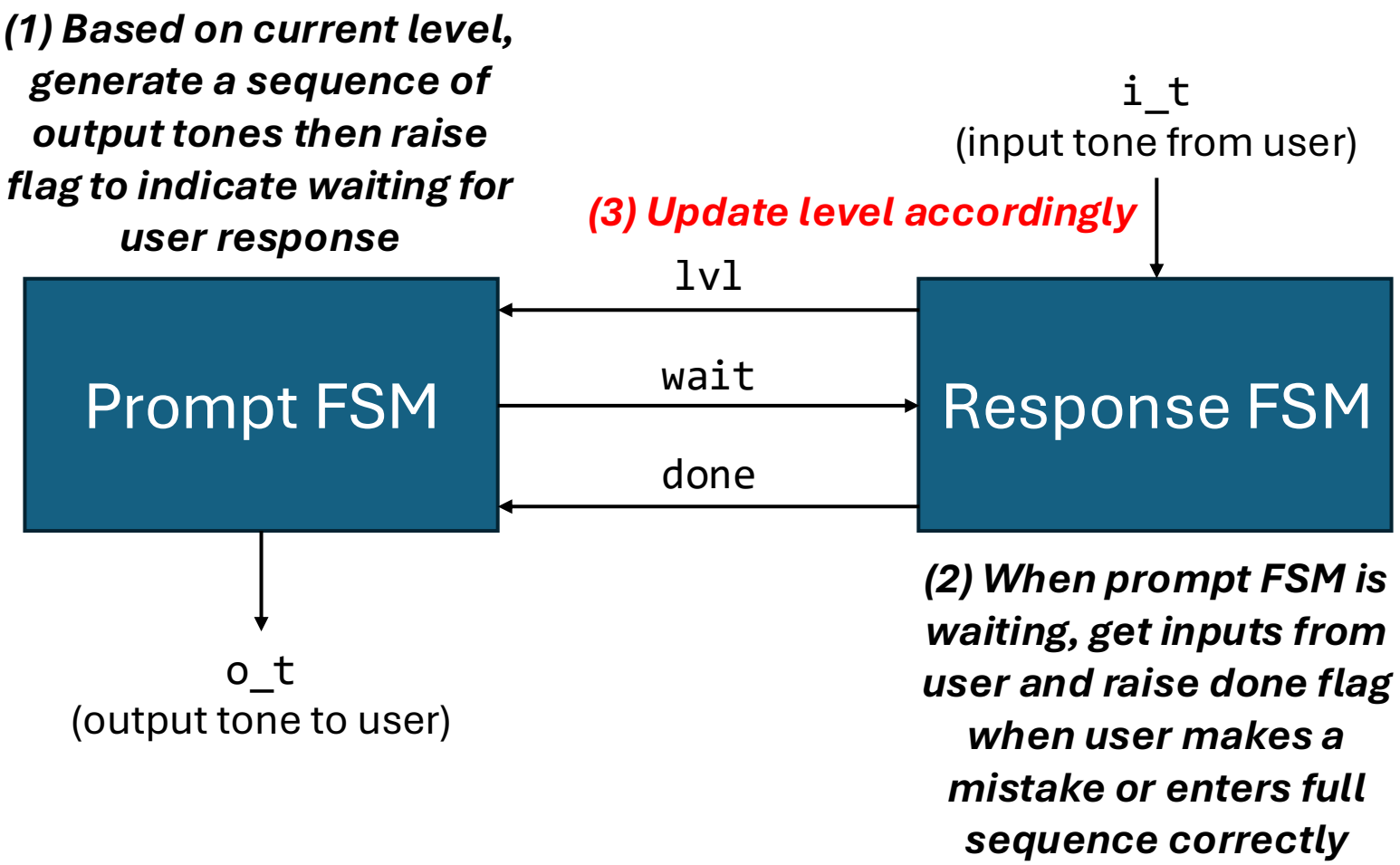
(1) Based on current level, generate a sequence of output tones then raise flag to indicate waiting for user response



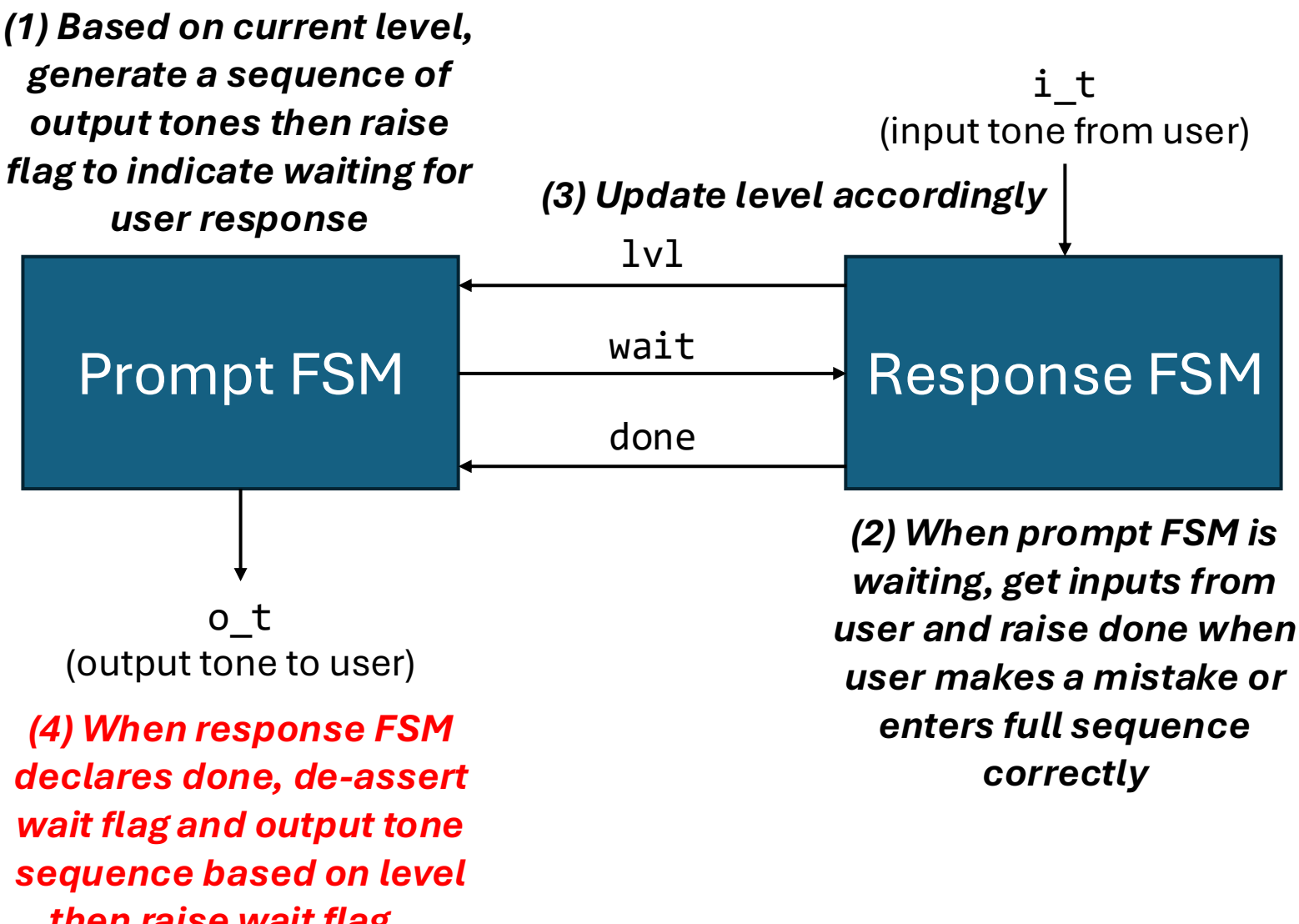
Example: Music Toy - Two Interacting FSMs



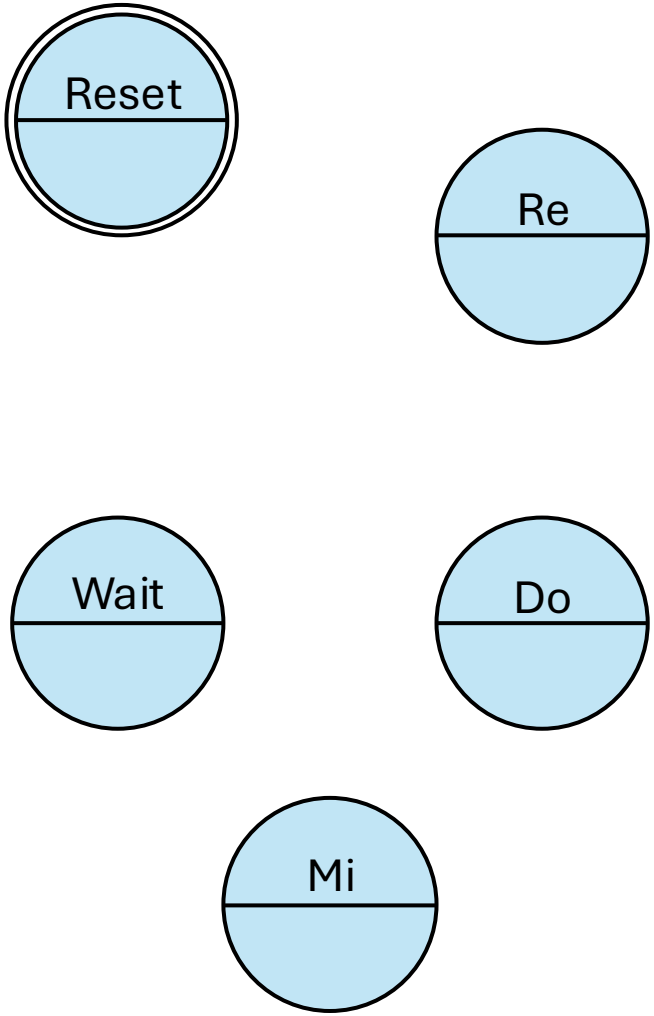
Example: Music Toy - Two Interacting FSMs



Example: Music Toy - Two Interacting FSMs

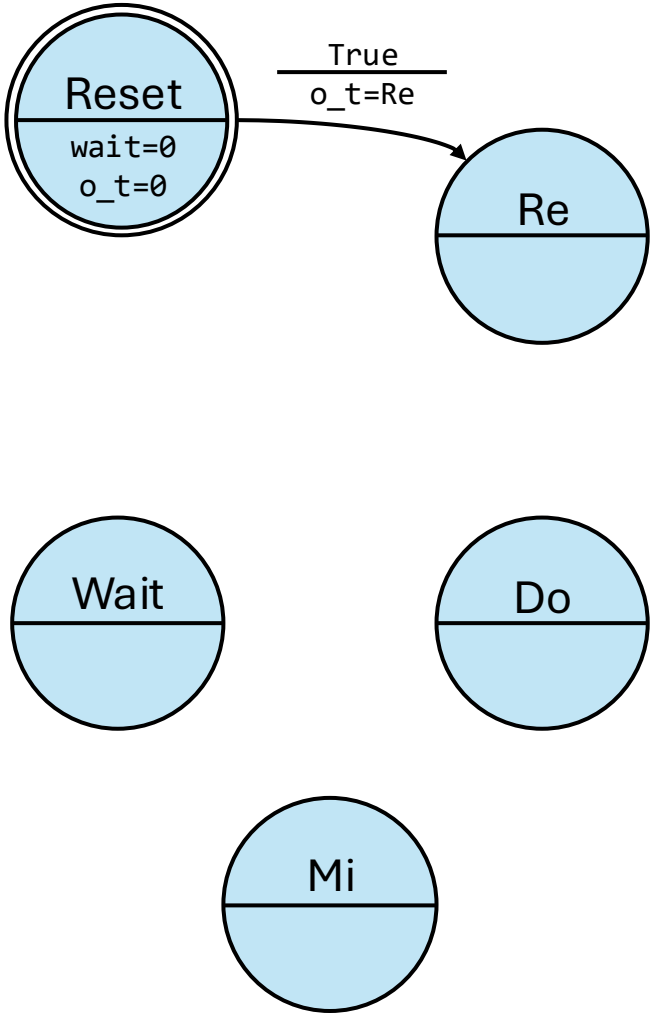


Example: Music Toy (Prompt FSM Design)

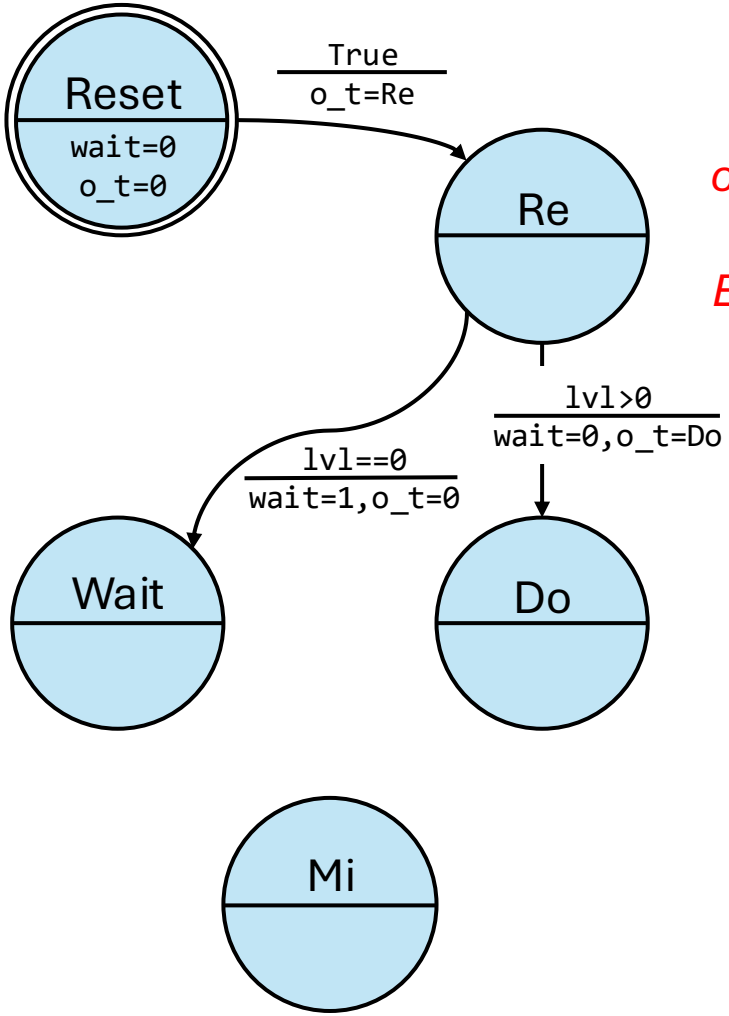


Example: Music Toy (Prompt FSM Design)

When reset is de-asserted, move to state Re and output the first tone

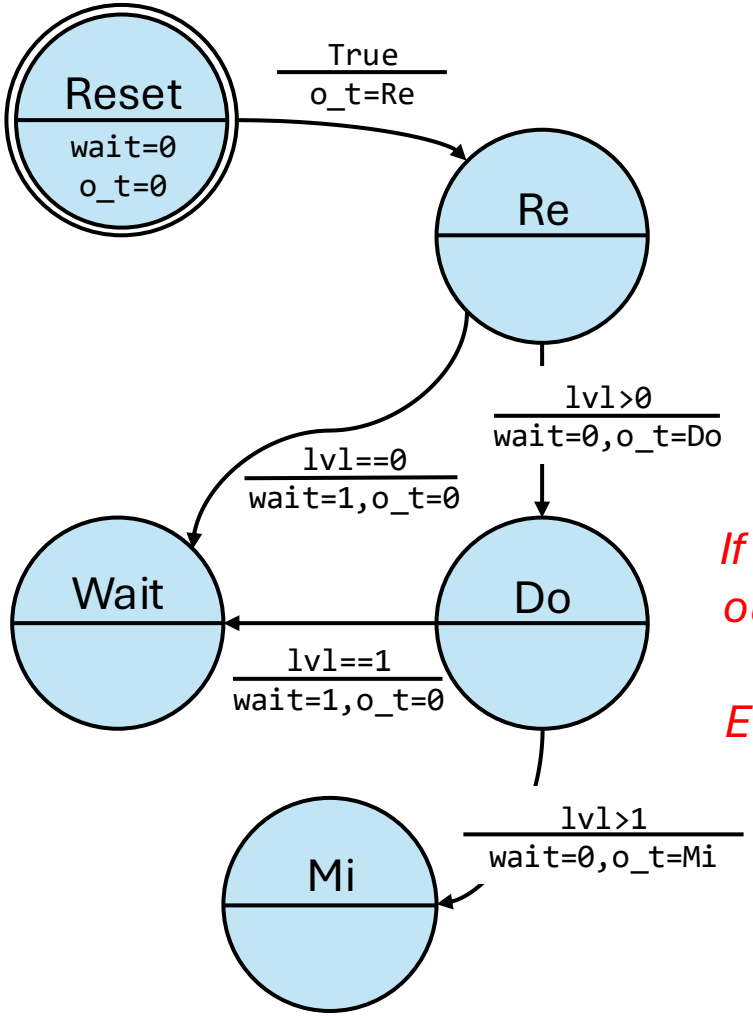


Example: Music Toy (Prompt FSM Design)



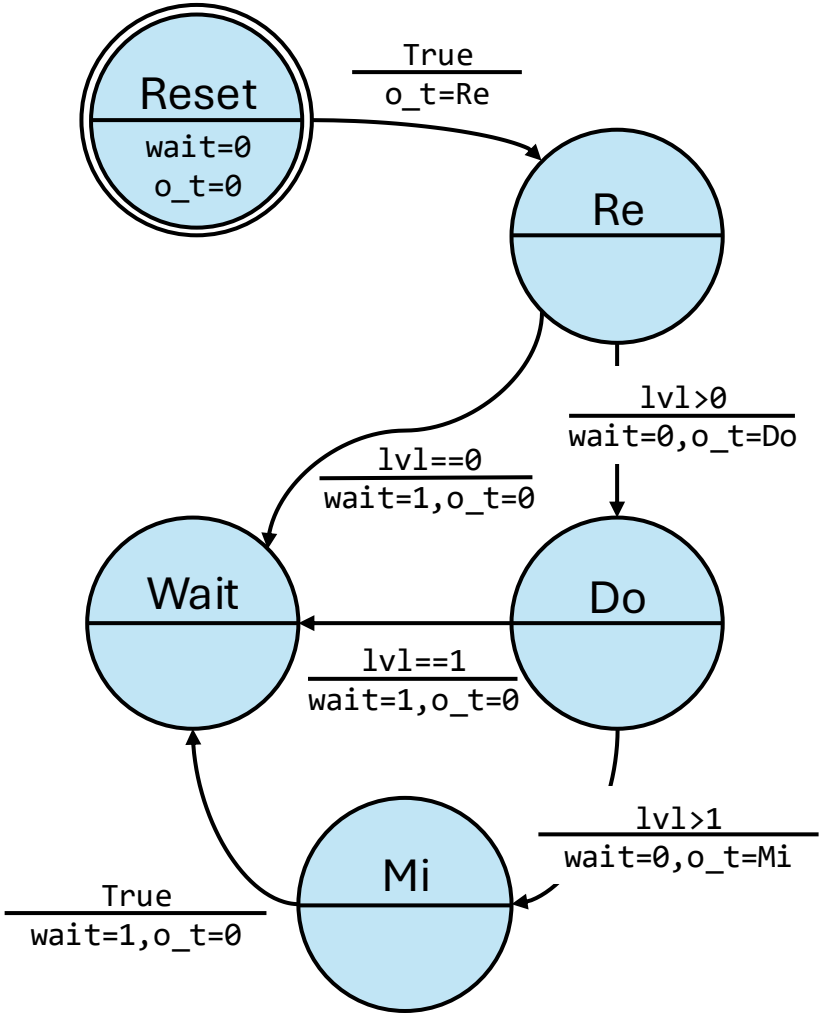
If this is the first level,
output sequence is done
→ move to wait state
Else, continue sequence

Example: Music Toy (Prompt FSM Design)



If this is the second level, output sequence is done → move to wait state Else, continue sequence

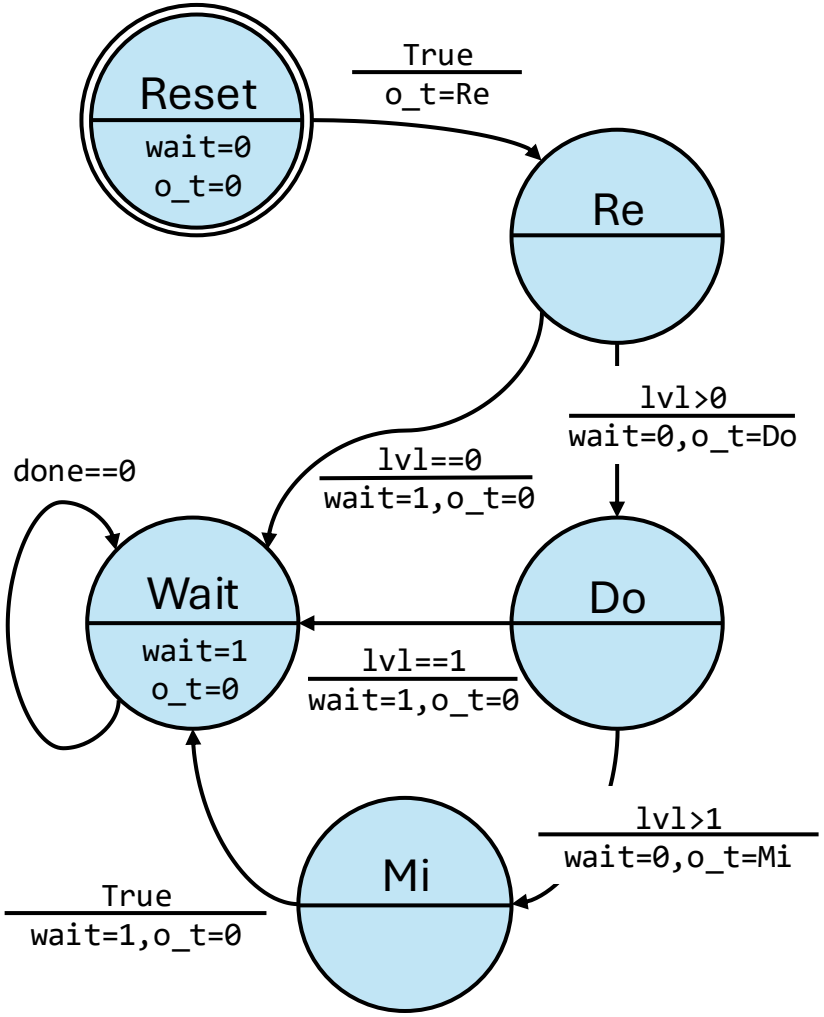
Example: Music Toy (Prompt FSM Design)



If this is the third level, output sequence is done → move to wait state

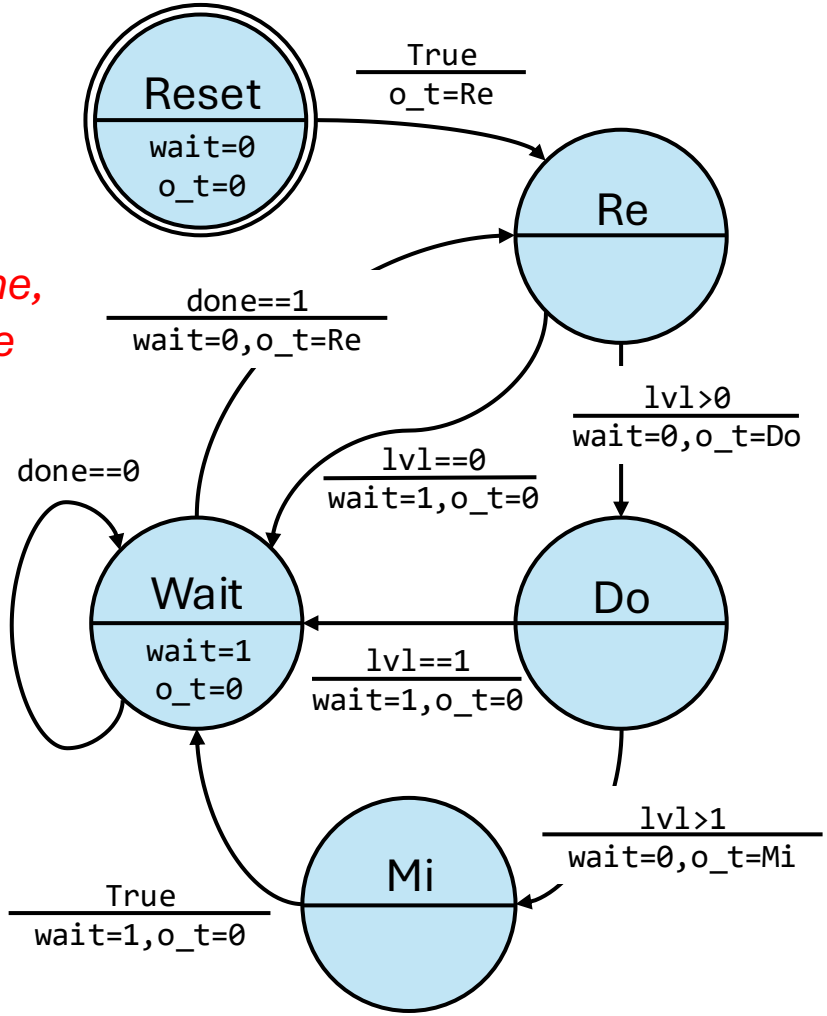
Example: Music Toy (Prompt FSM Design)

Stay in wait state as long as the response FSM did not indicate that user input is done (mistake or full correct sequence)

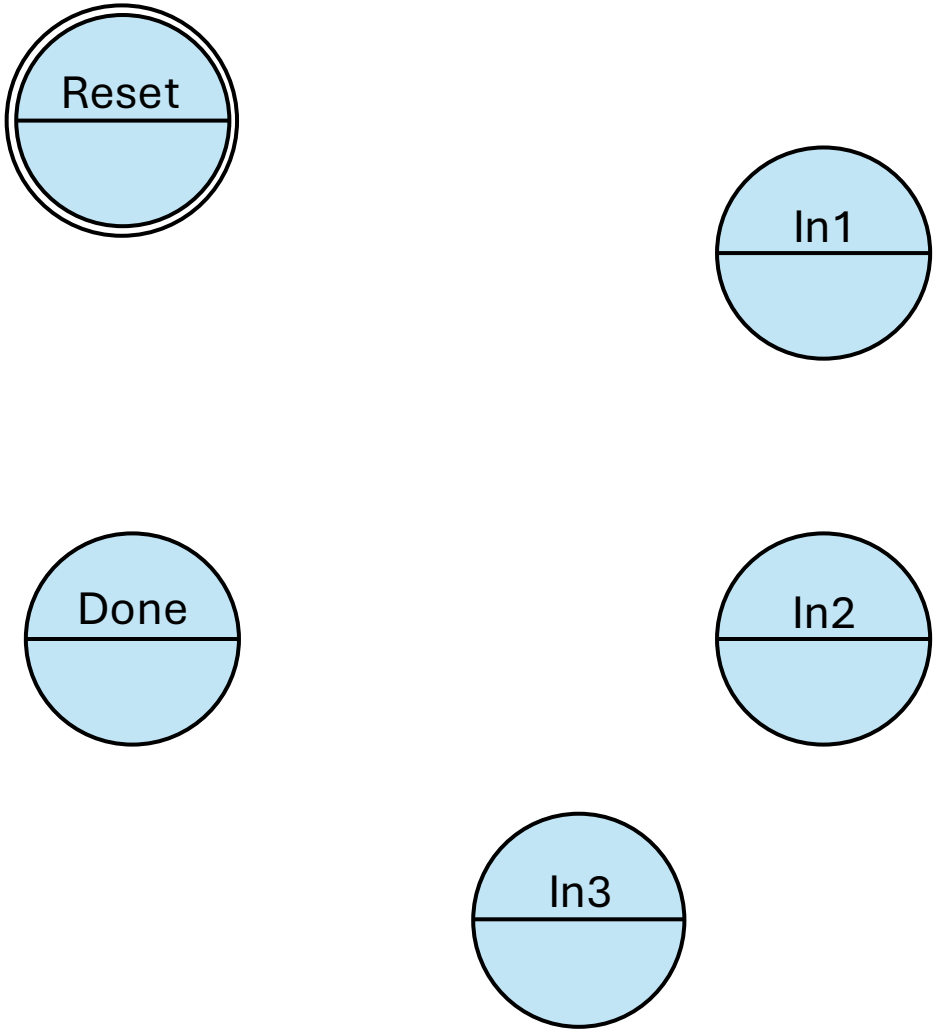


Example: Music Toy (Prompt FSM Design)

Once user input is done, start a new sequence

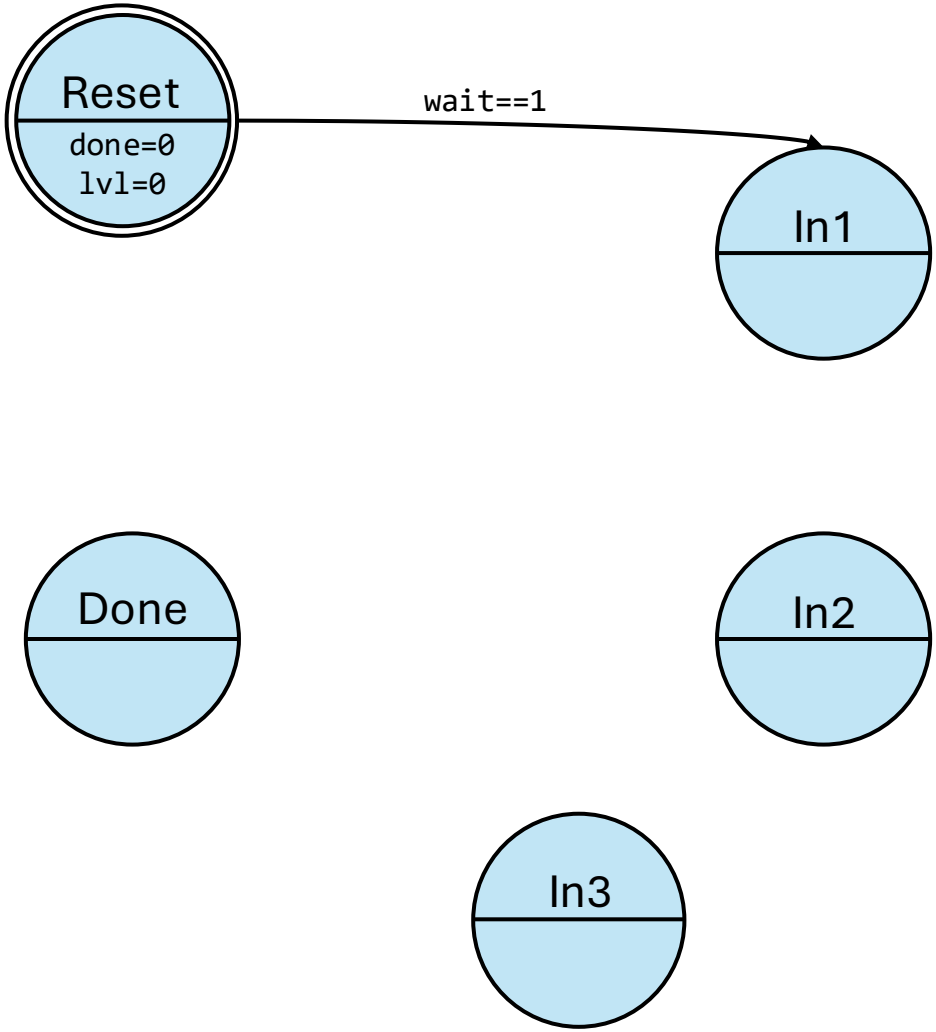


Example: Music Toy (Response FSM Design)

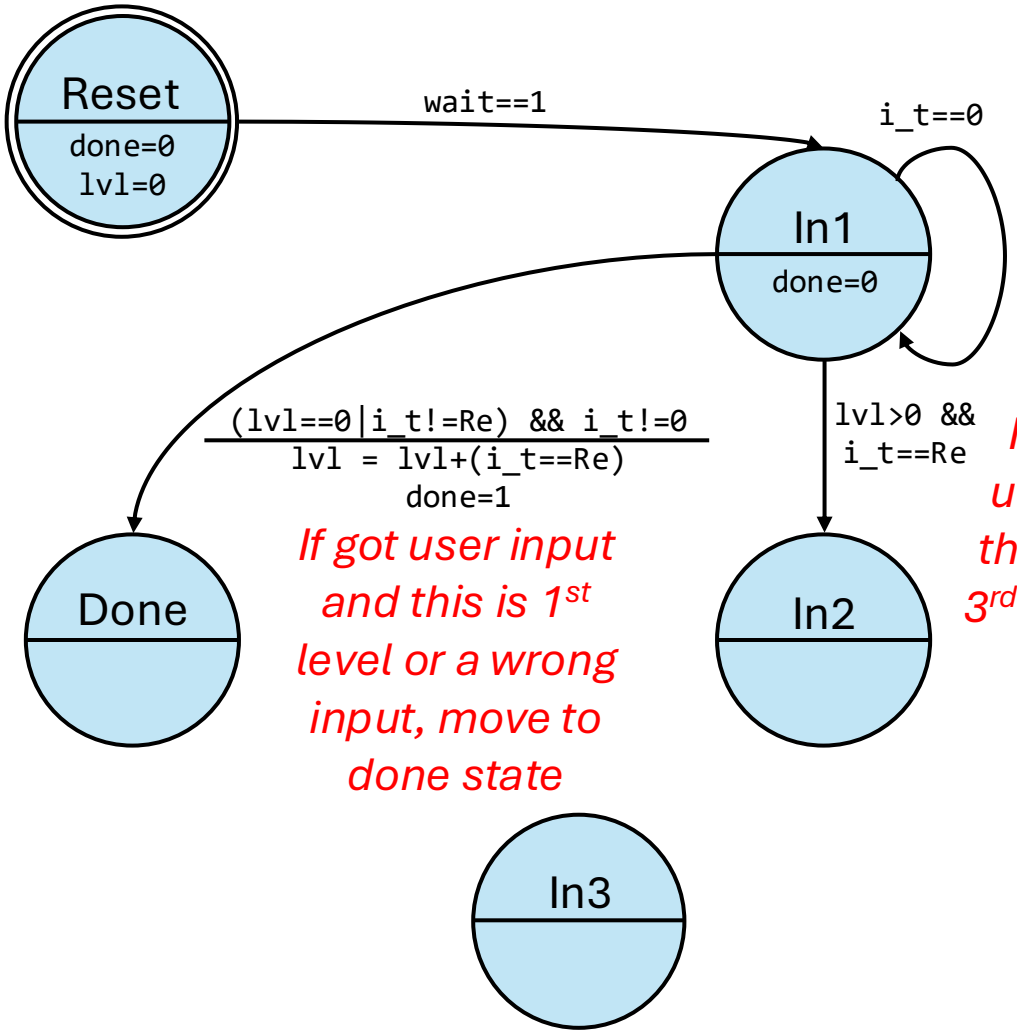


Example: Music Toy (Response FSM Design)

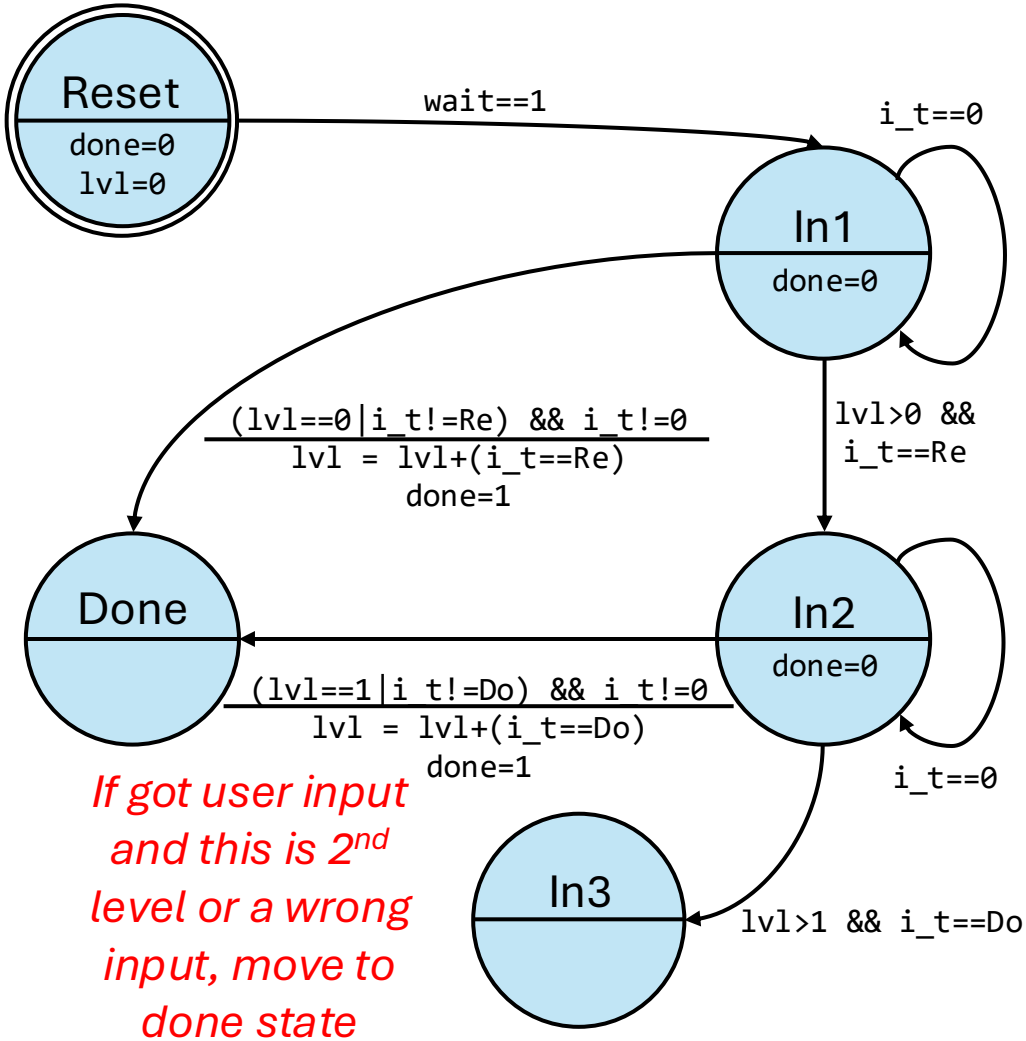
When reset is de-asserted, move to in1 state when Prompt FSM starts waiting



Example: Music Toy (Response FSM Design)



Example: Music Toy (Response FSM Design)

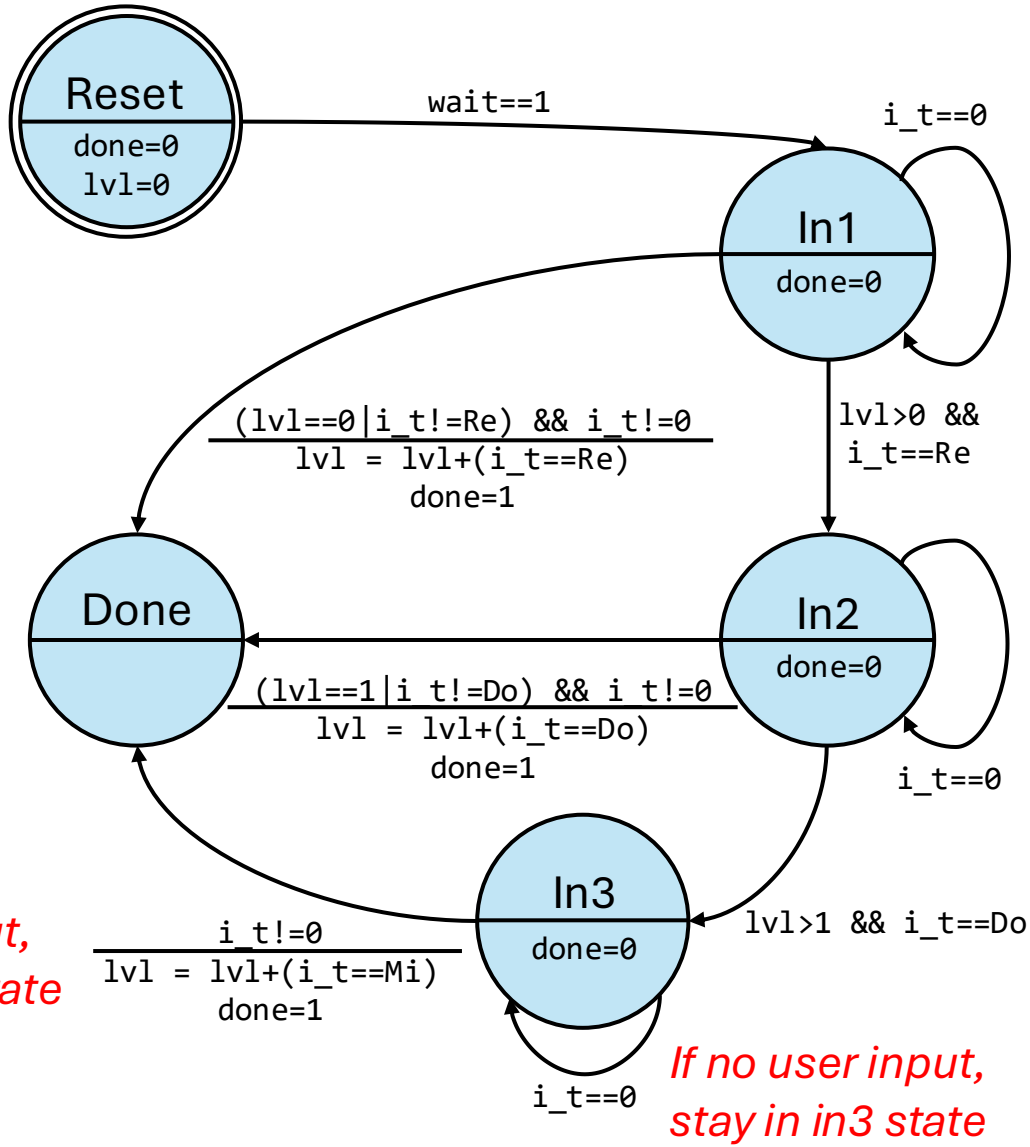


If no user input, stay in in2 state

If got user input and this is 2nd level or a wrong input, move to done state

If got correct user input and this is the 3rd level, move to in3 state

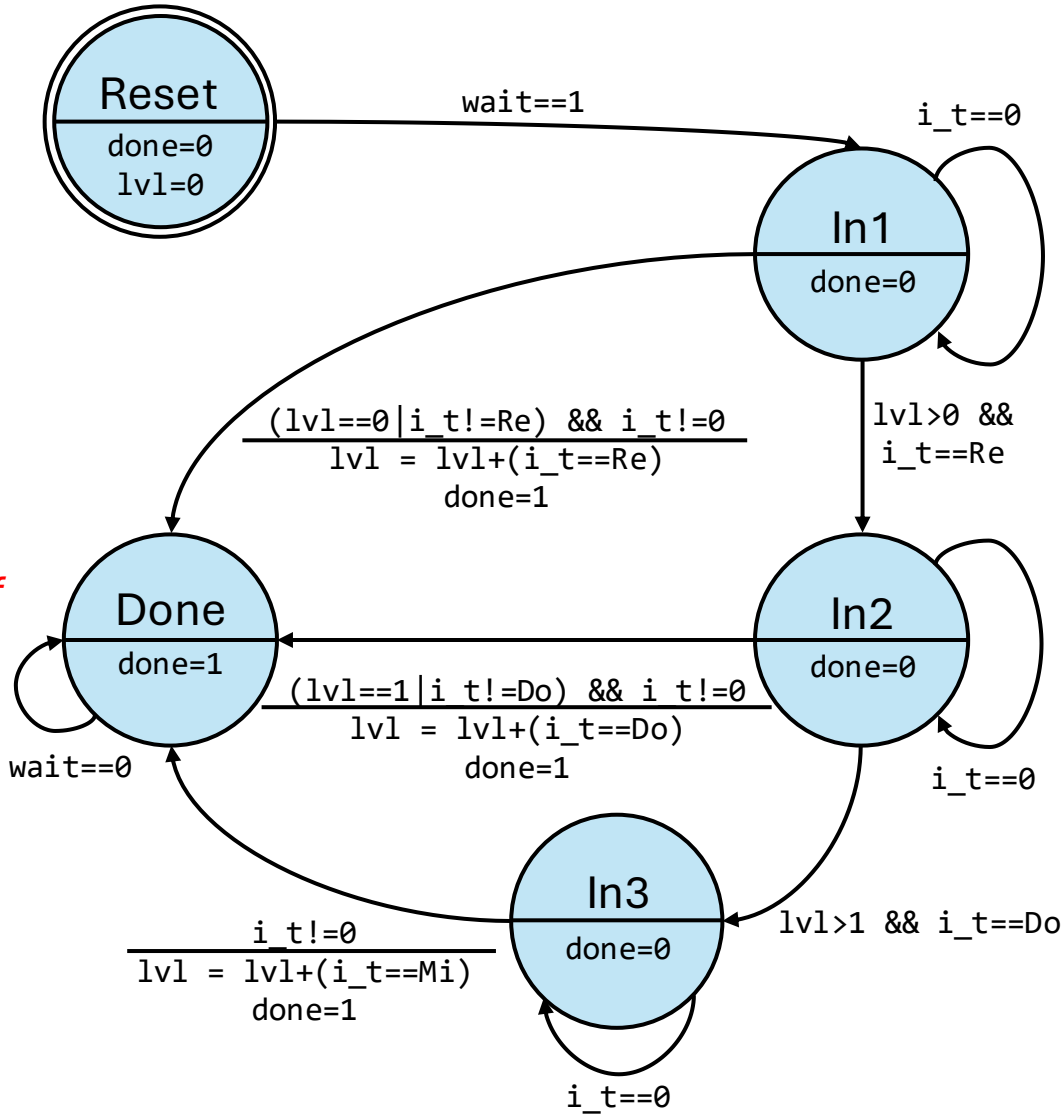
Example: Music Toy (Response FSM Design)



If got user input, move to done state

If no user input, stay in in3 state

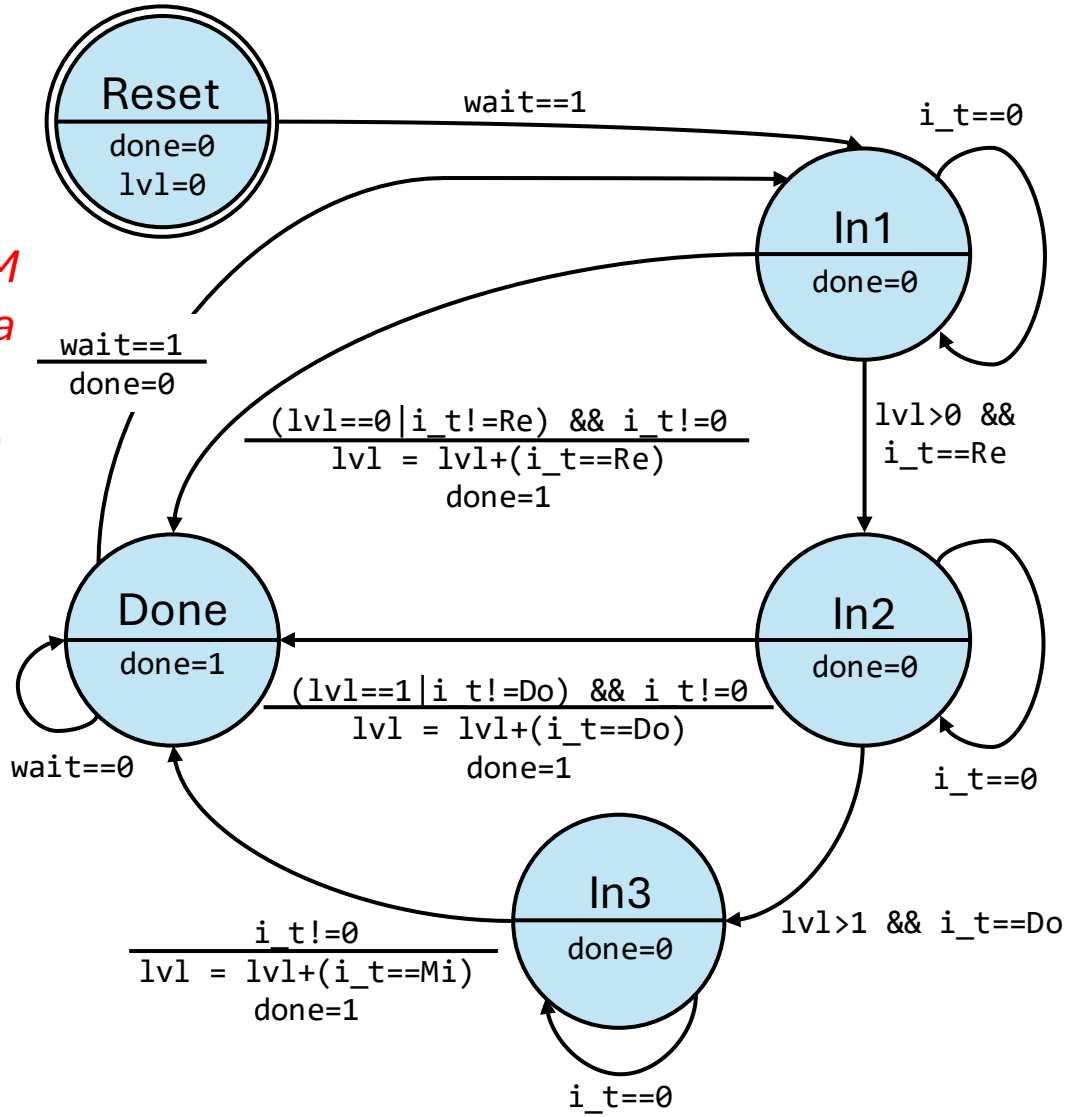
Example: Music Toy (Response FSM Design)



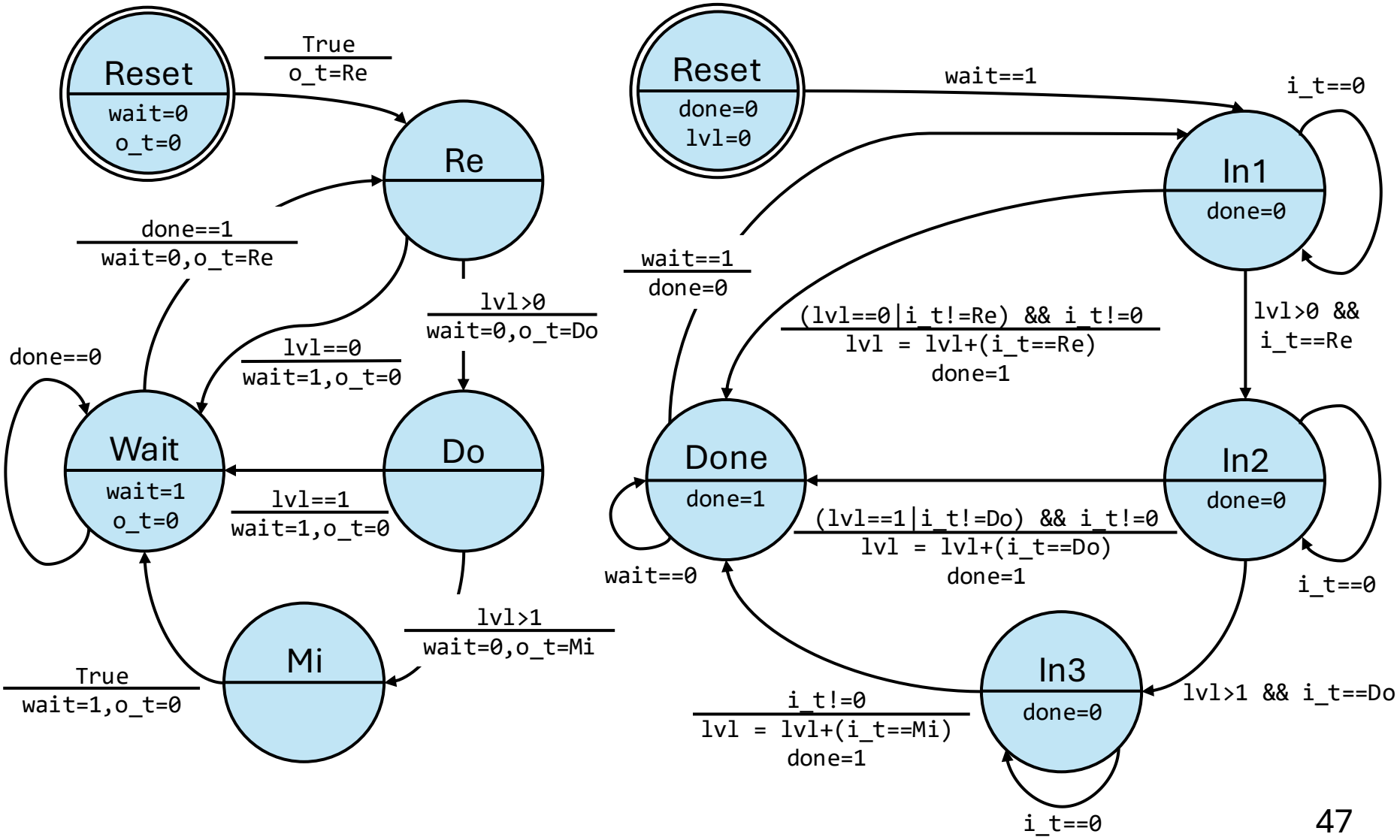
Stay in done state if Prompt FSM is not waiting for a user input sequence

Example: Music Toy (Response FSM Design)

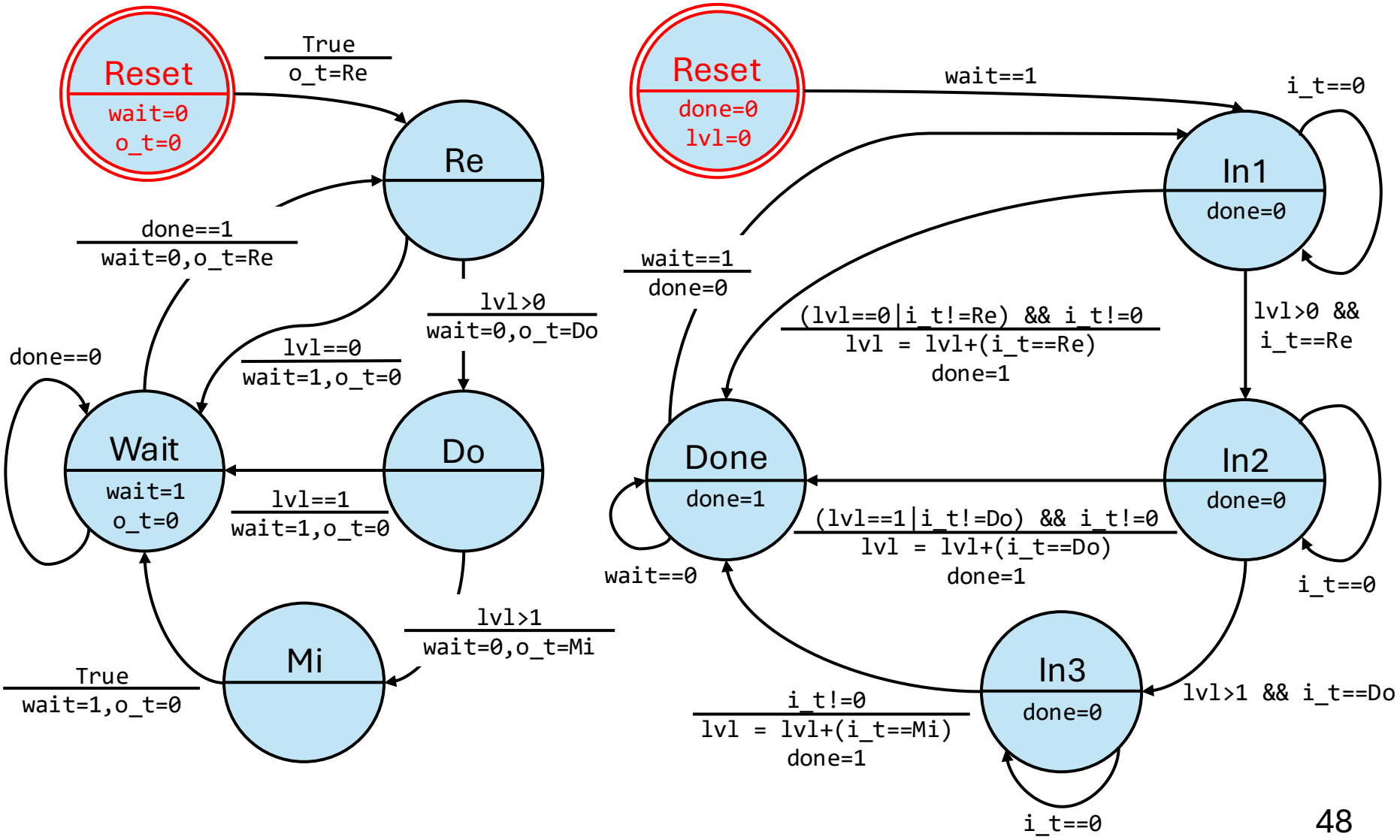
When Prompt FSM starts waiting for a user sequence, move to in1 state



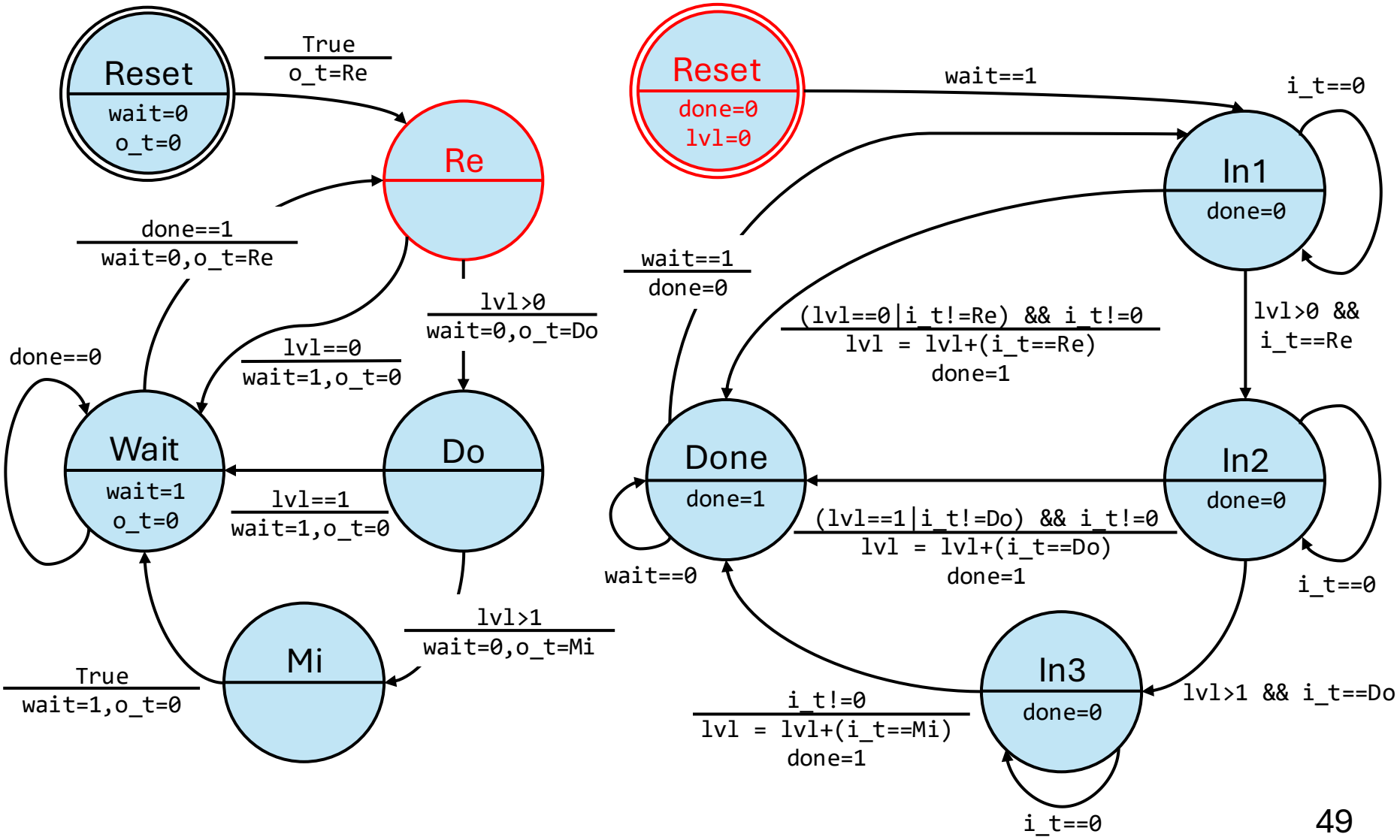
Example: Music Toy (Prompt + Response FSMs)



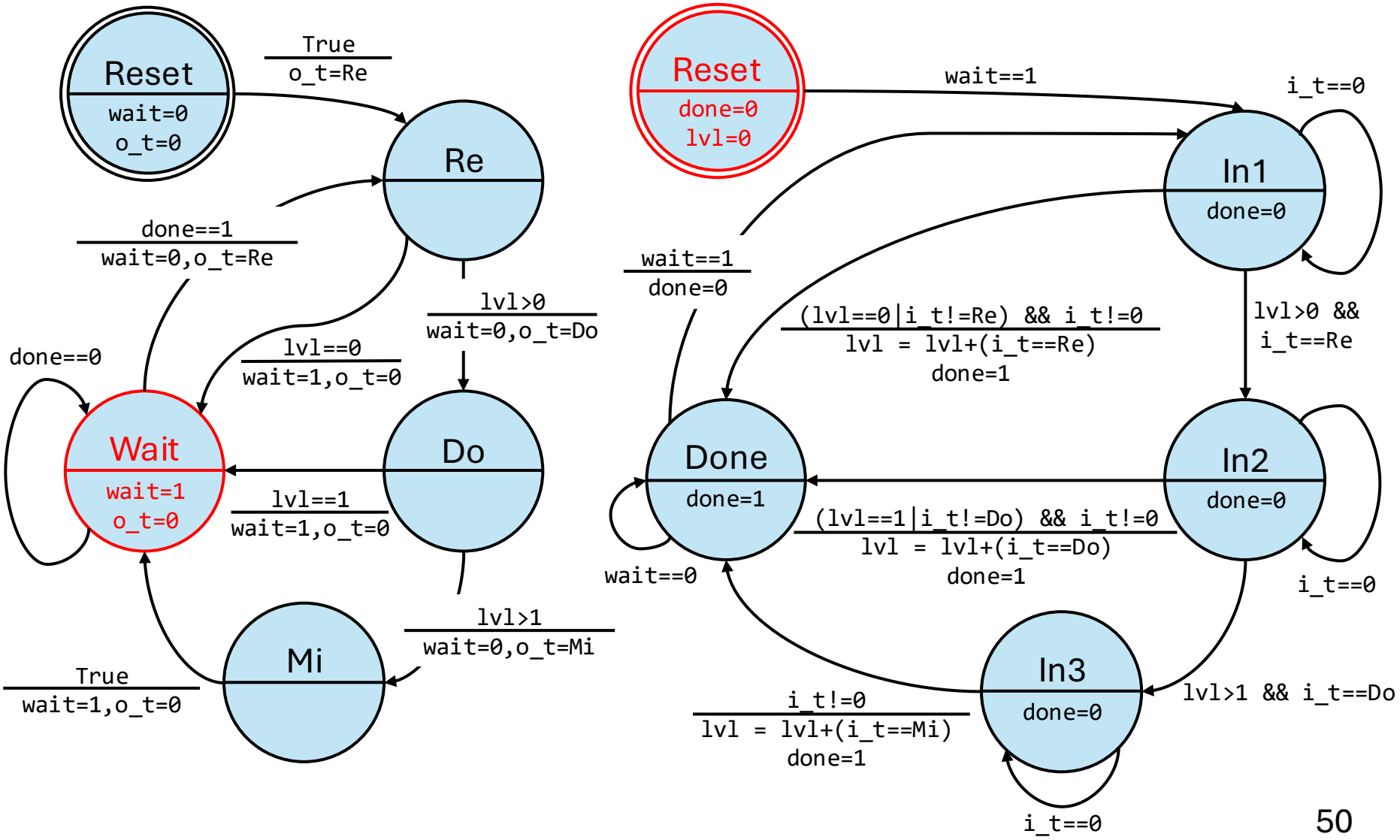
Example: Music Toy (Prompt + Response FSMs)



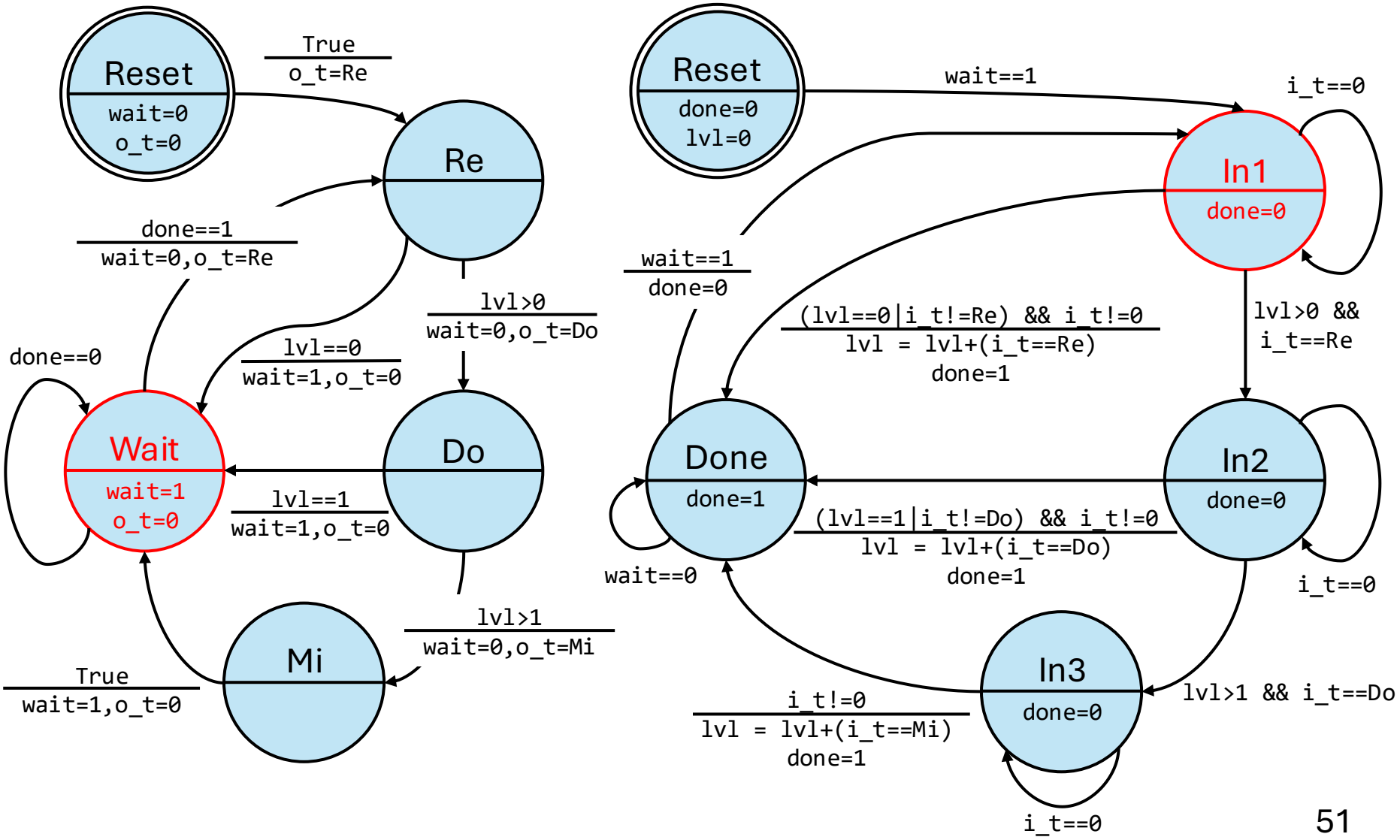
Example: Music Toy (Prompt + Response FSMs)



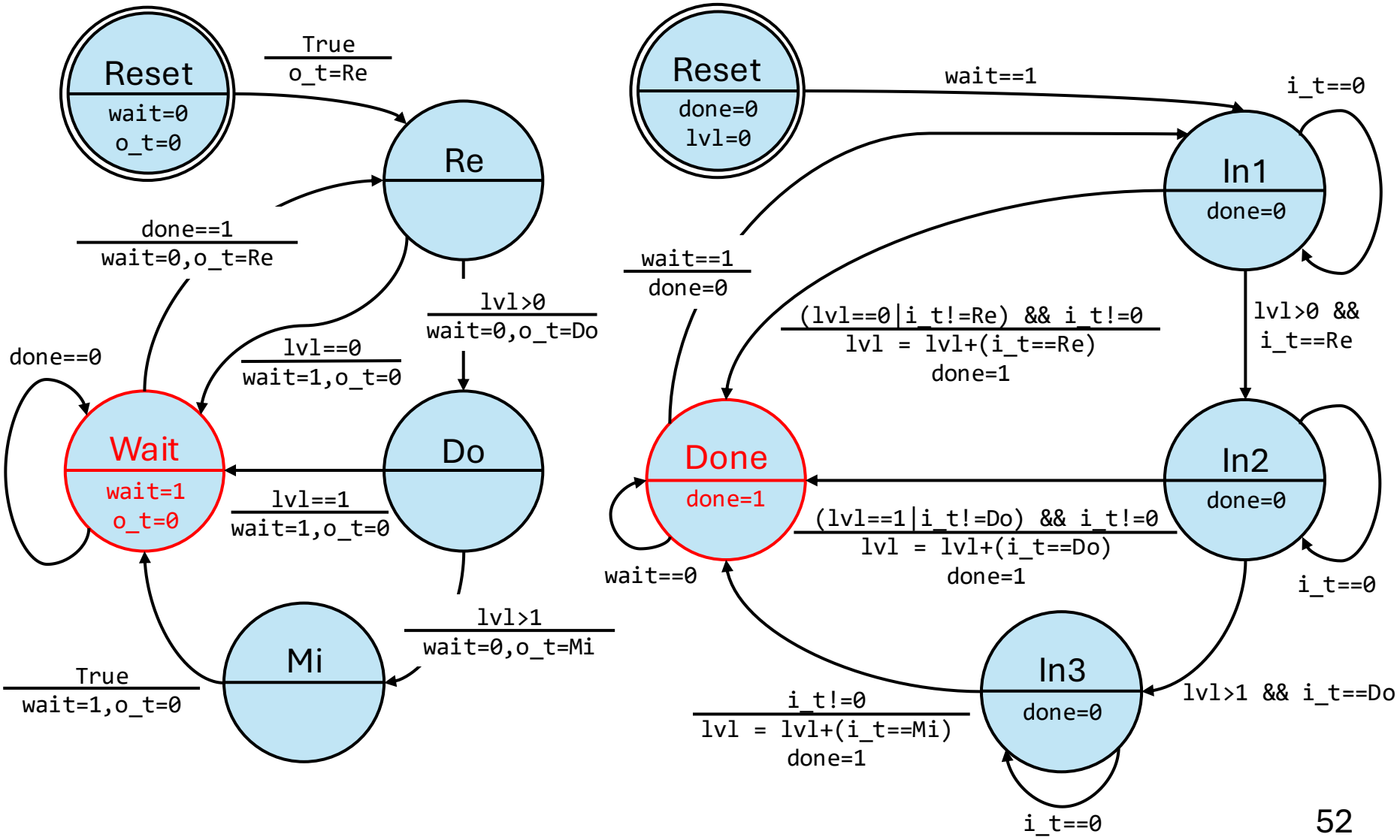
Example: Music Toy (Prompt + Response FSMs)



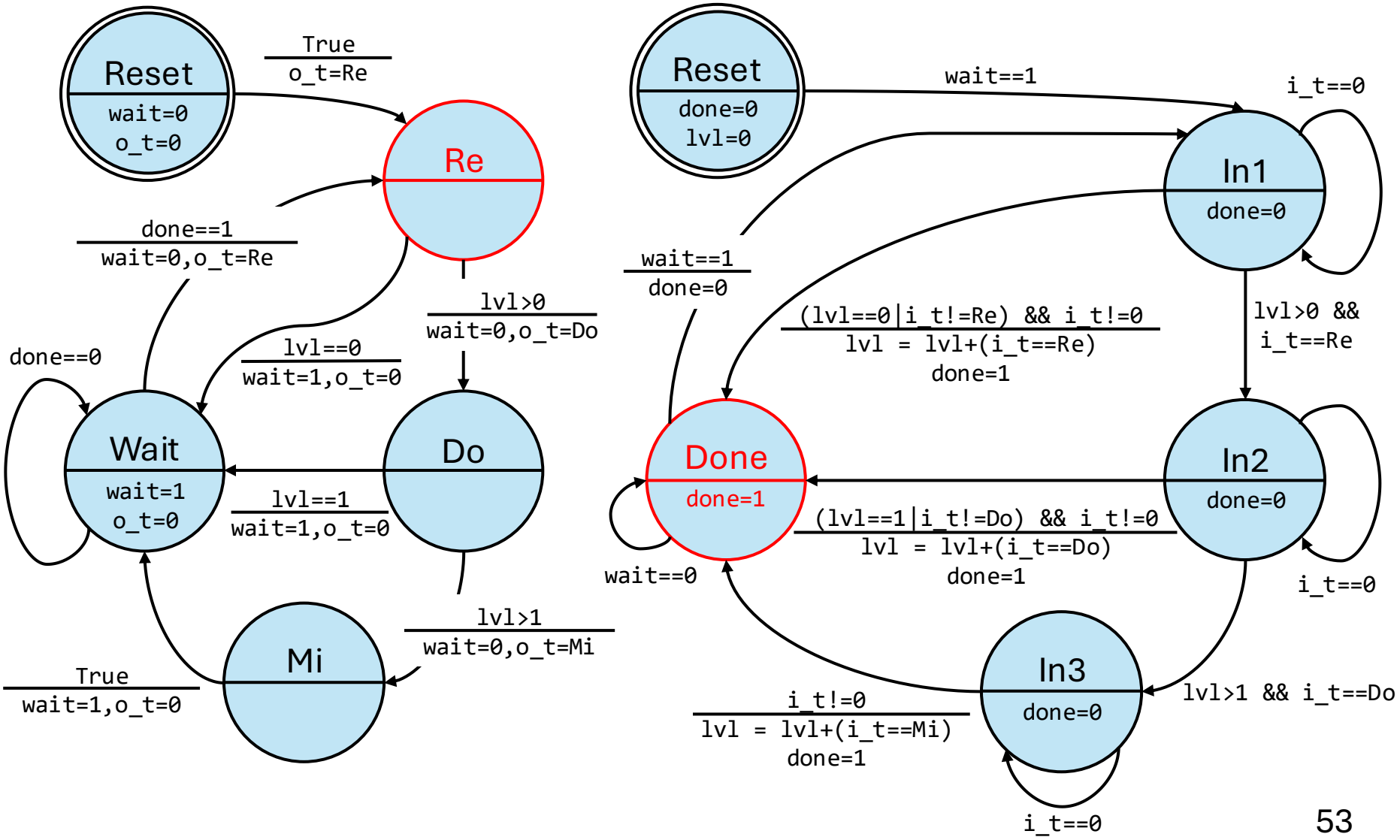
Example: Music Toy (Prompt + Response FSMs)



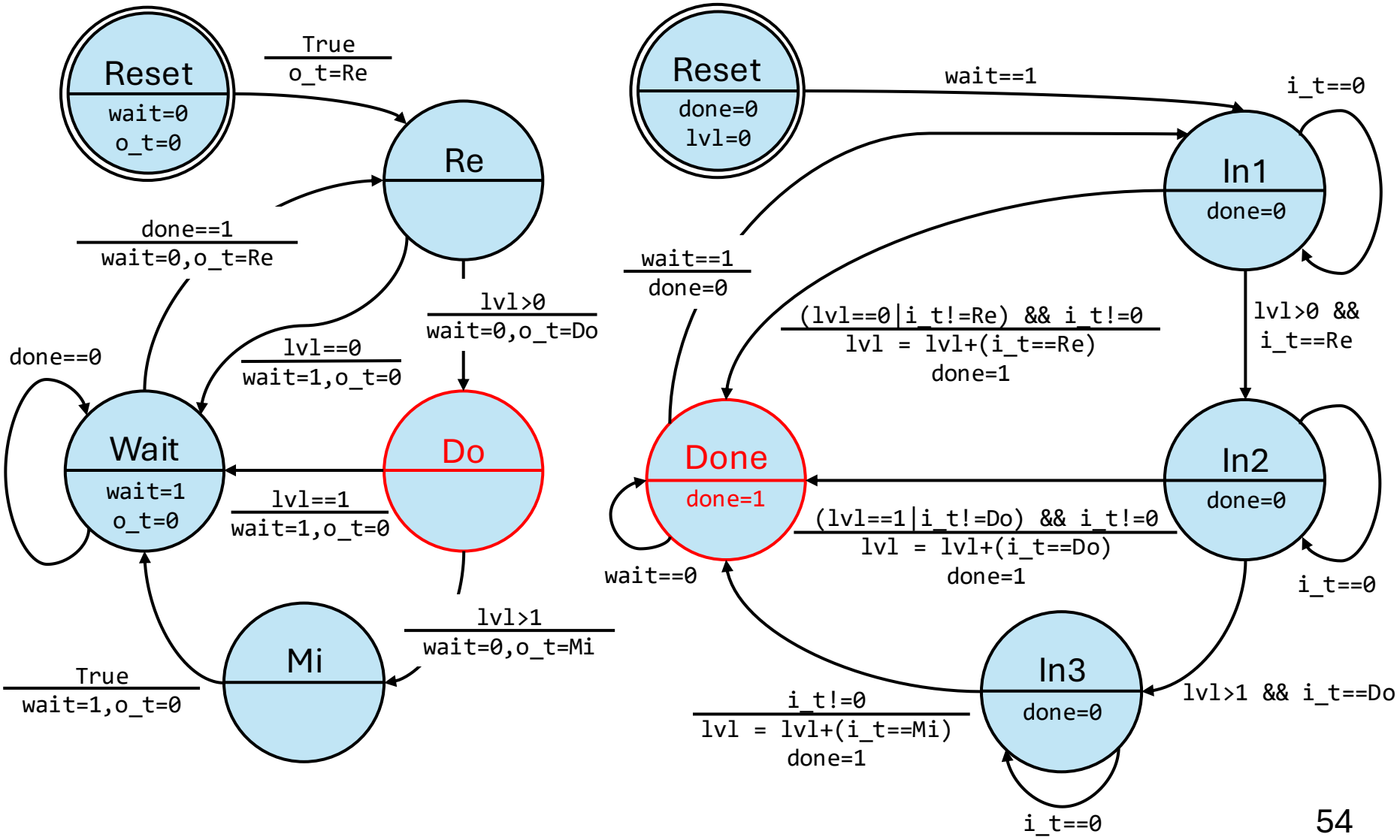
Example: Music Toy (Prompt + Response FSMs)



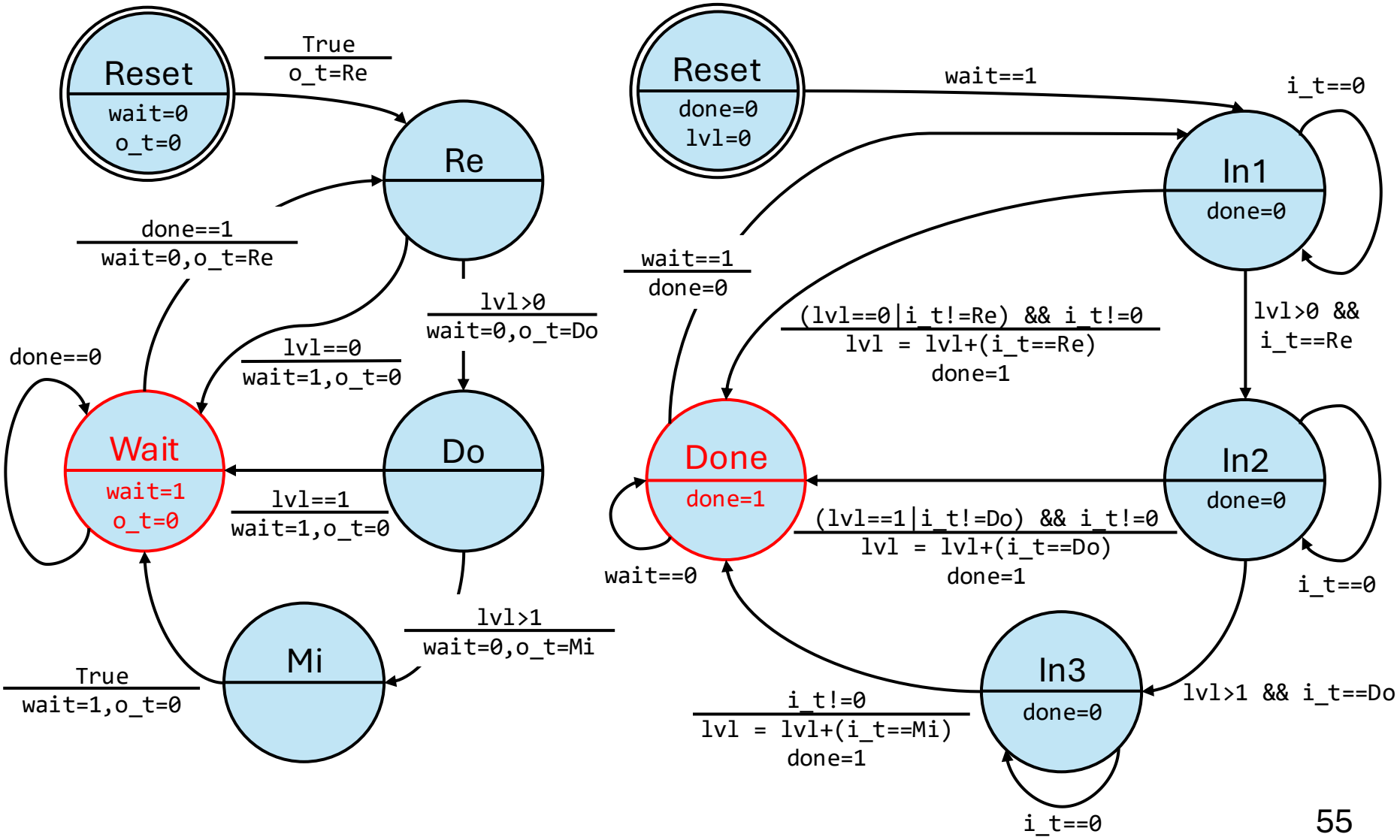
Example: Music Toy (Prompt + Response FSMs)



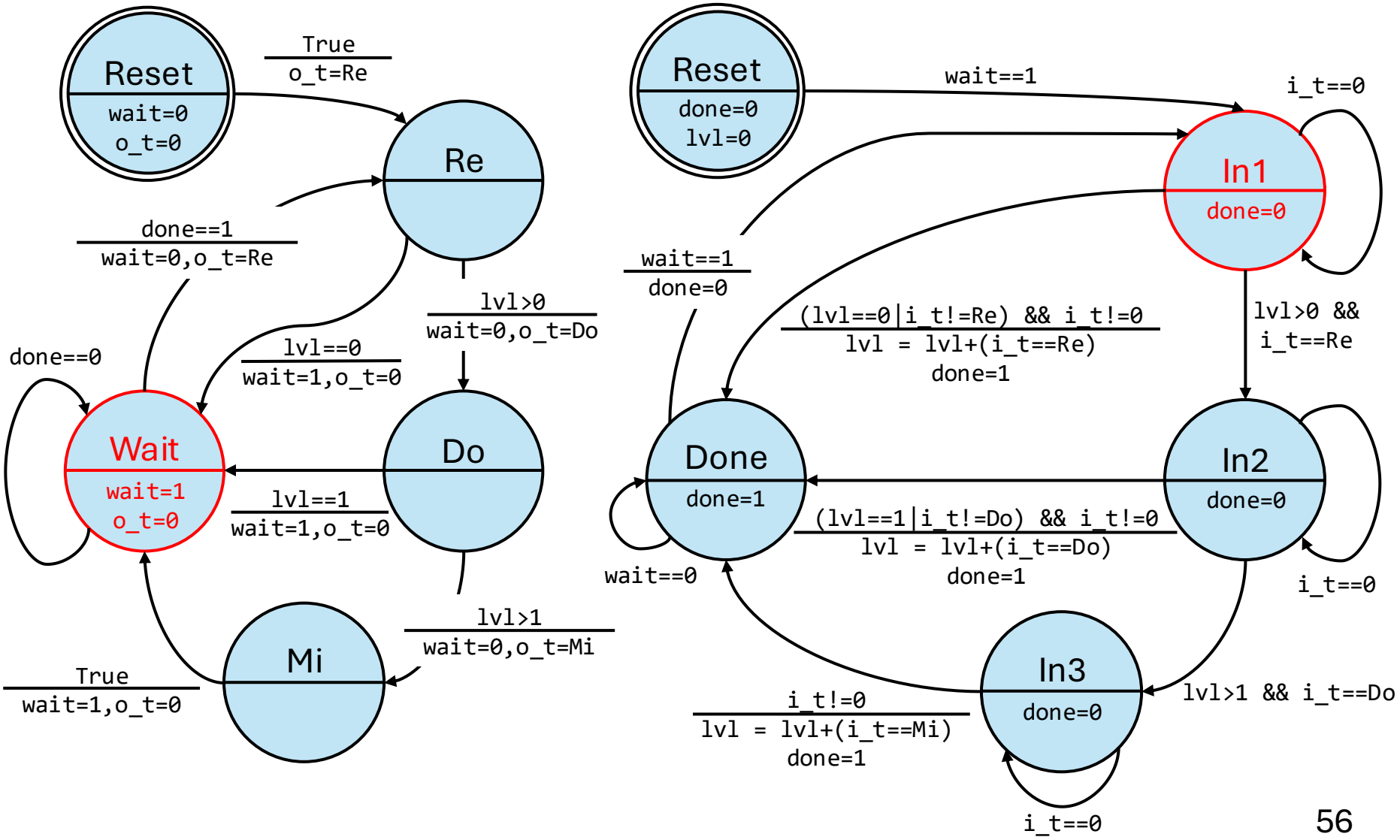
Example: Music Toy (Prompt + Response FSMs)



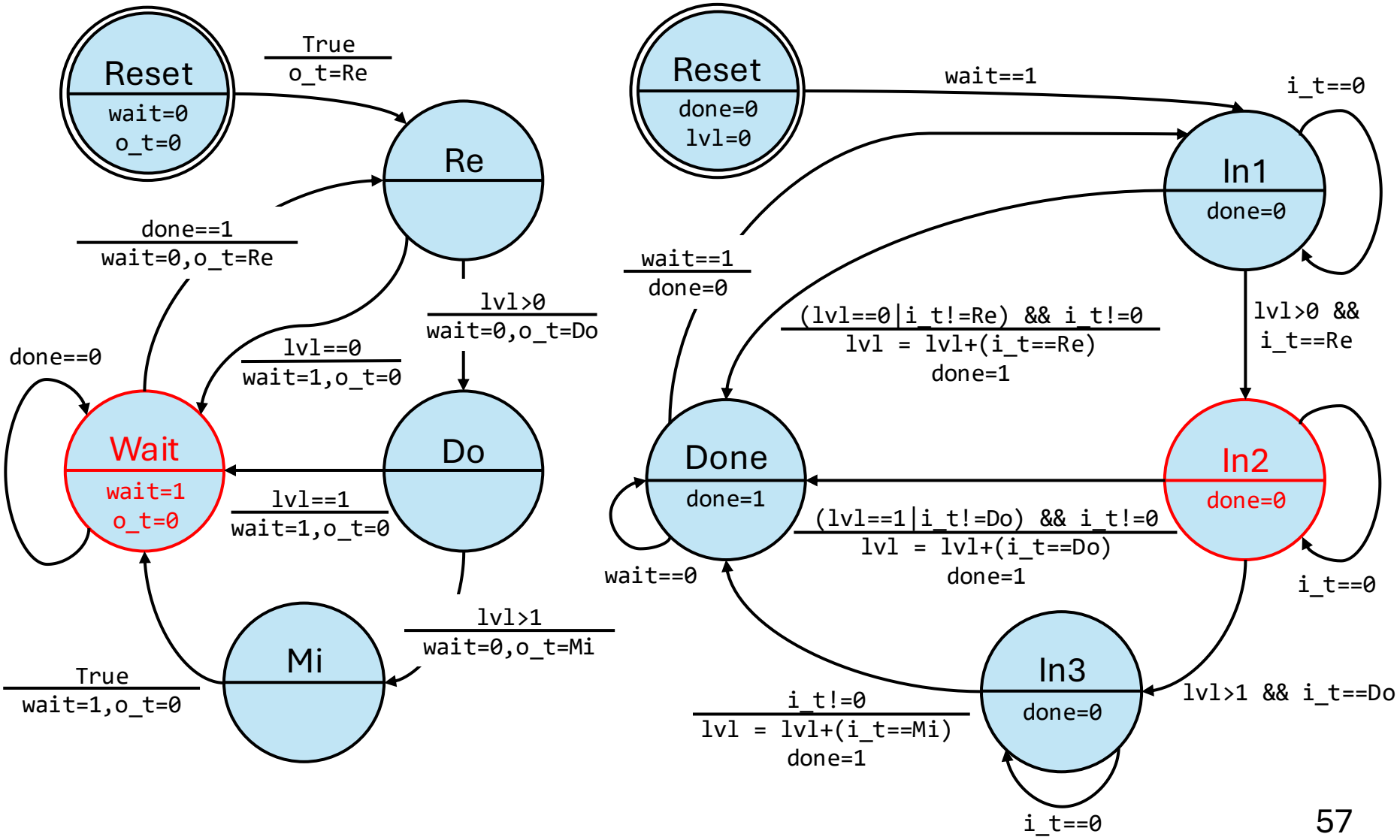
Example: Music Toy (Prompt + Response FSMs)



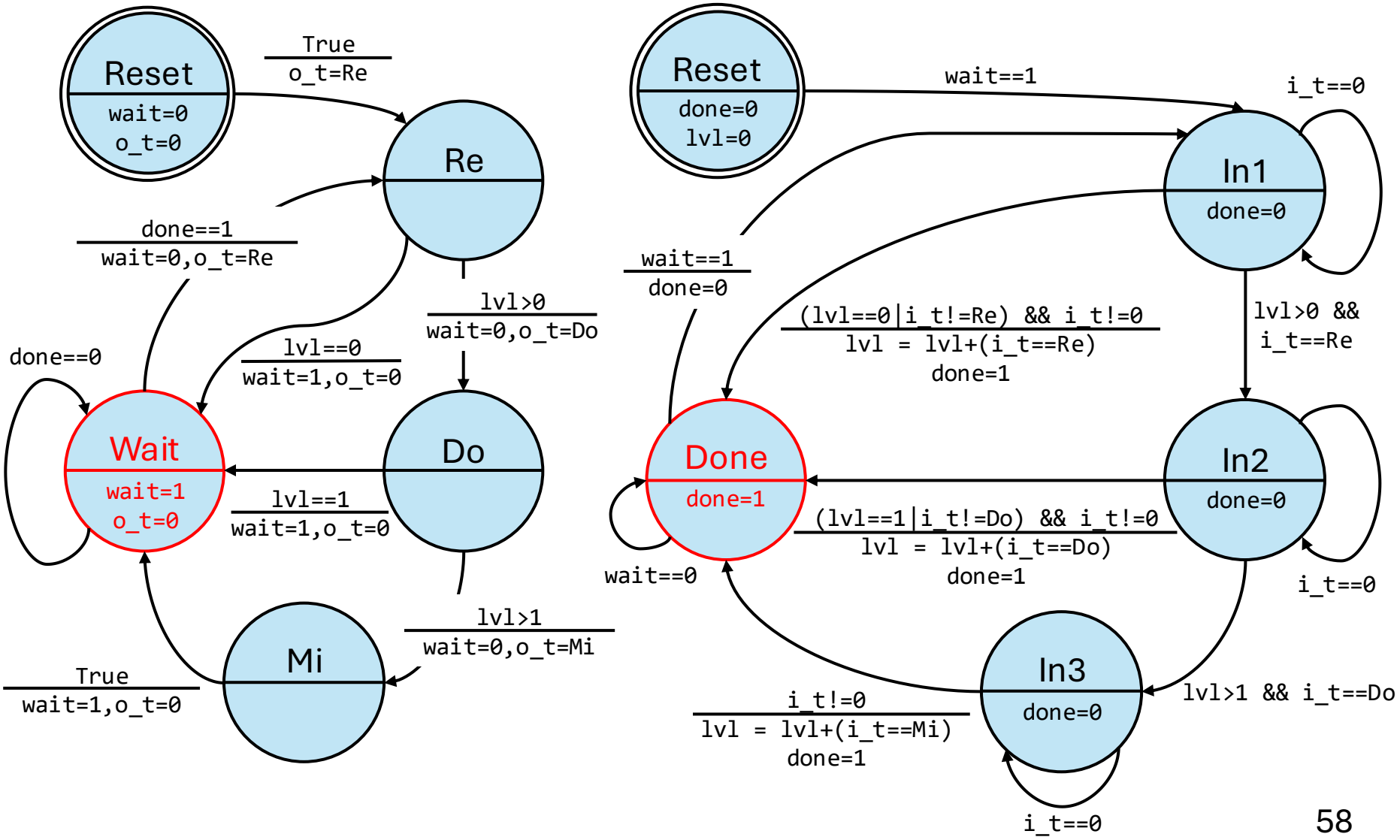
Example: Music Toy (Prompt + Response FSMs)



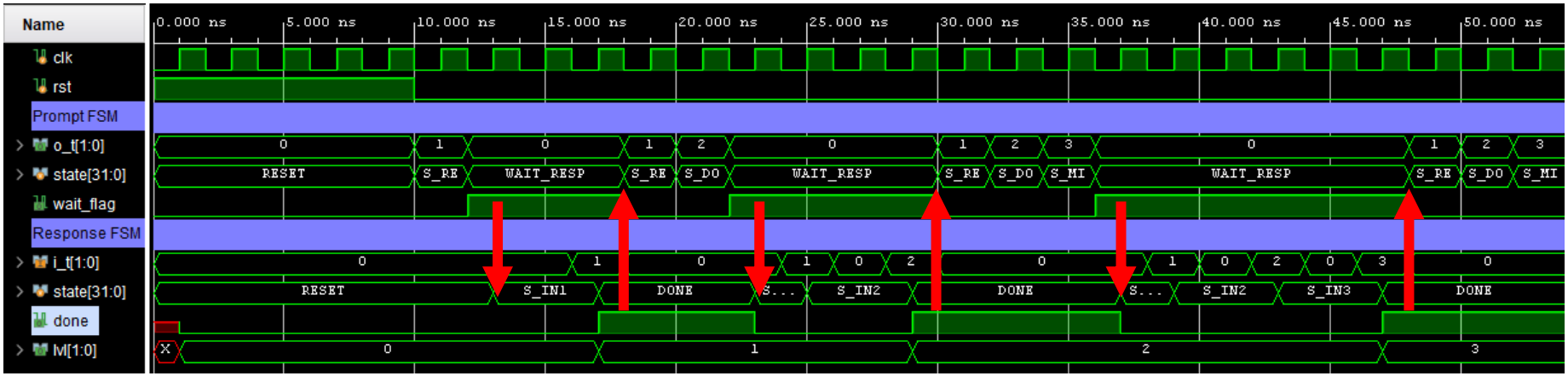
Example: Music Toy (Prompt + Response FSMs)



Example: Music Toy (Prompt + Response FSMs)



Example: Music Toy (Waveform)



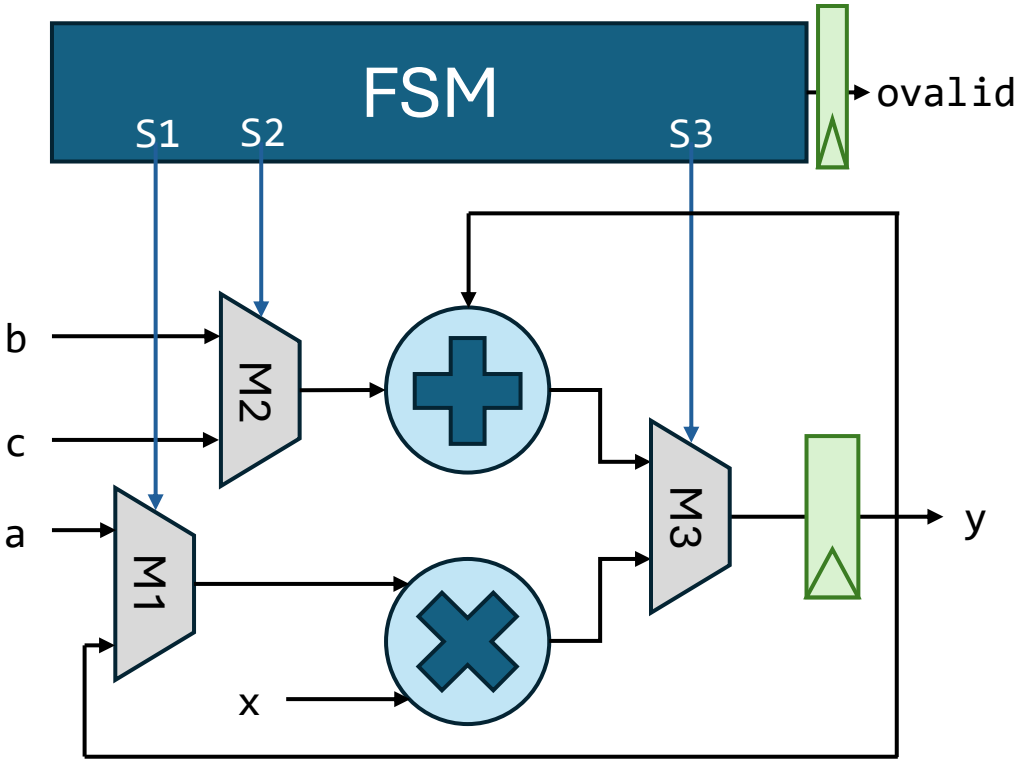
Finite State Machine + Datapath (FSMD)

- In many cases, FSMs are used to control a **time-multiplexed datapath** to perform a specific sequence of operations
  - Datapath can perform different operations (e.g., ALU from your lab 1)
  - FSM orchestrates the use of datapath in each clock cycle
- Example: In a processor …
  - FSM is the instruction decode + program counter
  - Datapath is ALU with multiplexers and bypass logic
- In FSMD, the FSM generates multiplexer select lines

Example: Polynomial Revisited

- Compute $y = ax^2 + bx + c$ using only 1 multiplier & 1 adder

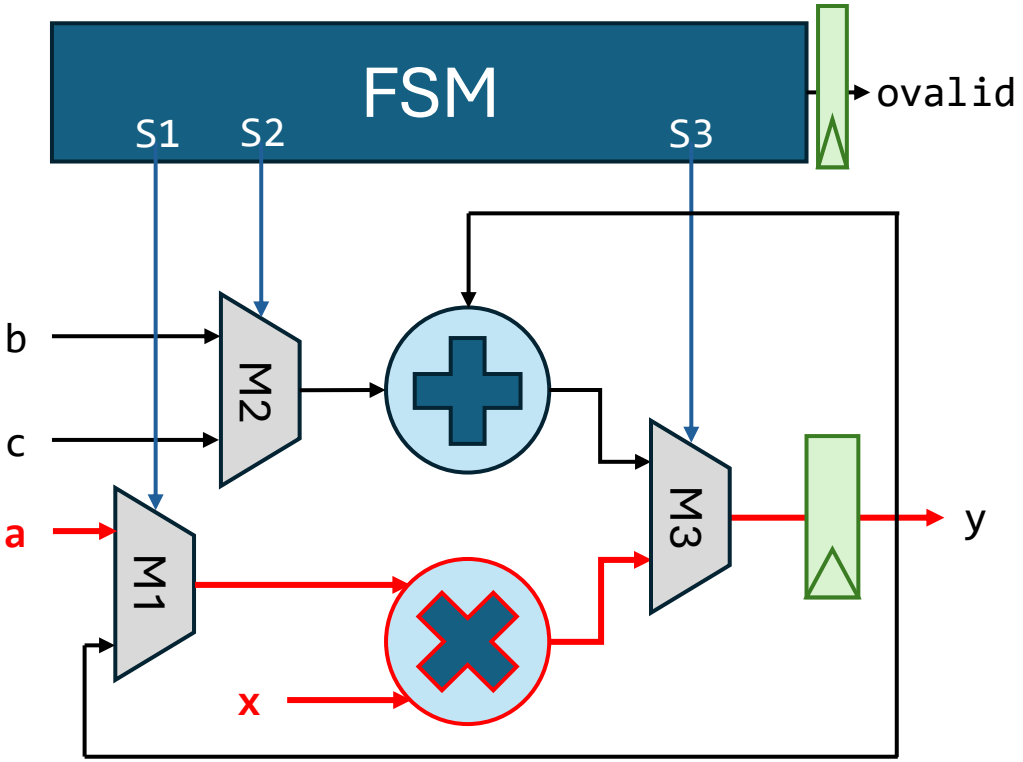
- Refactor computation as $y = [(ax+b)x]+c$ to perform over 4 steps:



Example: Polynomial Revisited

- Compute $y = ax^2 + bx + c$ using only 1 multiplier & 1 adder

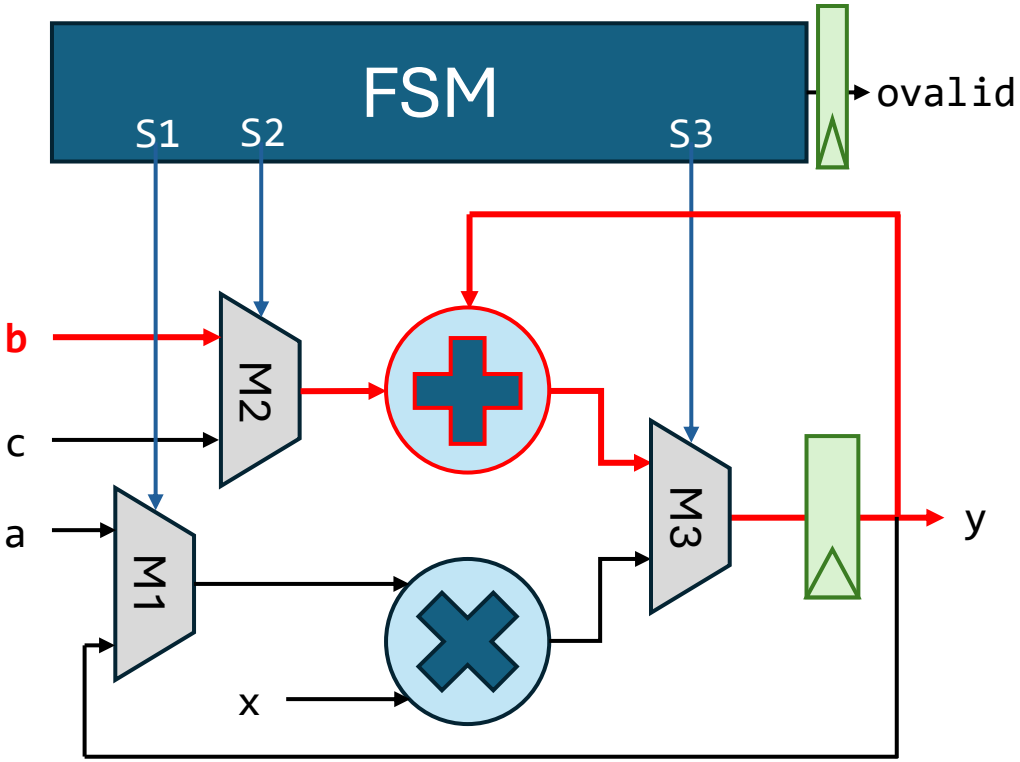
- Refactor computation as $y = [(ax+b)x]+c$ to perform over 4 steps:
  1. tmp = a * x



Example: Polynomial Revisited

- Compute $y = ax^2 + bx + c$ using only 1 multiplier & 1 adder

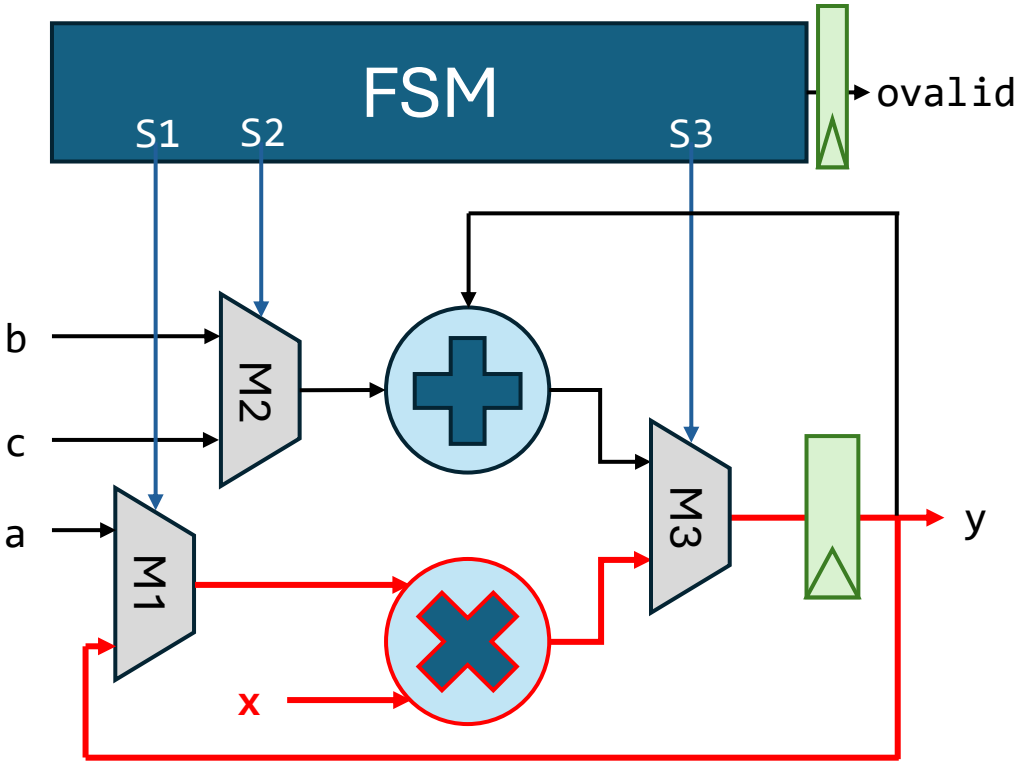
- Refactor computation as $y = [(ax+b)x]+c$ to perform over 4 steps:
  1. tmp = a * x
  2. tmp = tmp + b



Example: Polynomial Revisited

- Compute $y = ax^2 + bx + c$ using only 1 multiplier & 1 adder

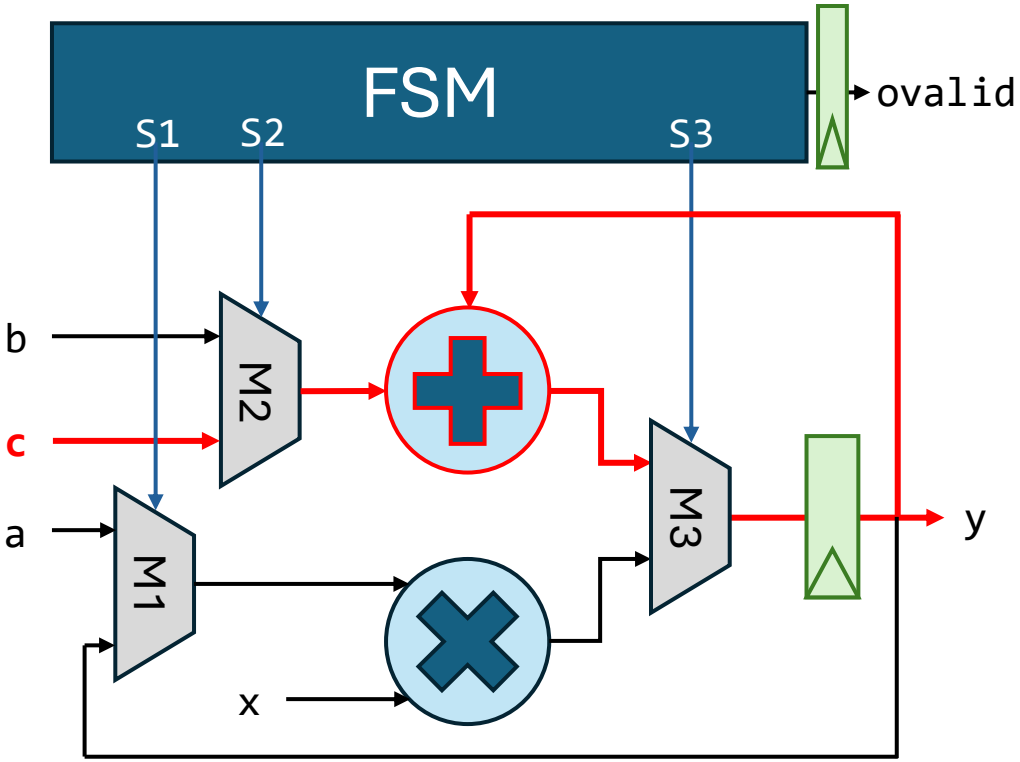
- Refactor computation as $y = [(ax+b)x]+c$ to perform over 4 steps:
  1. tmp = a * x
  2. tmp = tmp + b
  3. tmp = tmp * x



Example: Polynomial Revisited

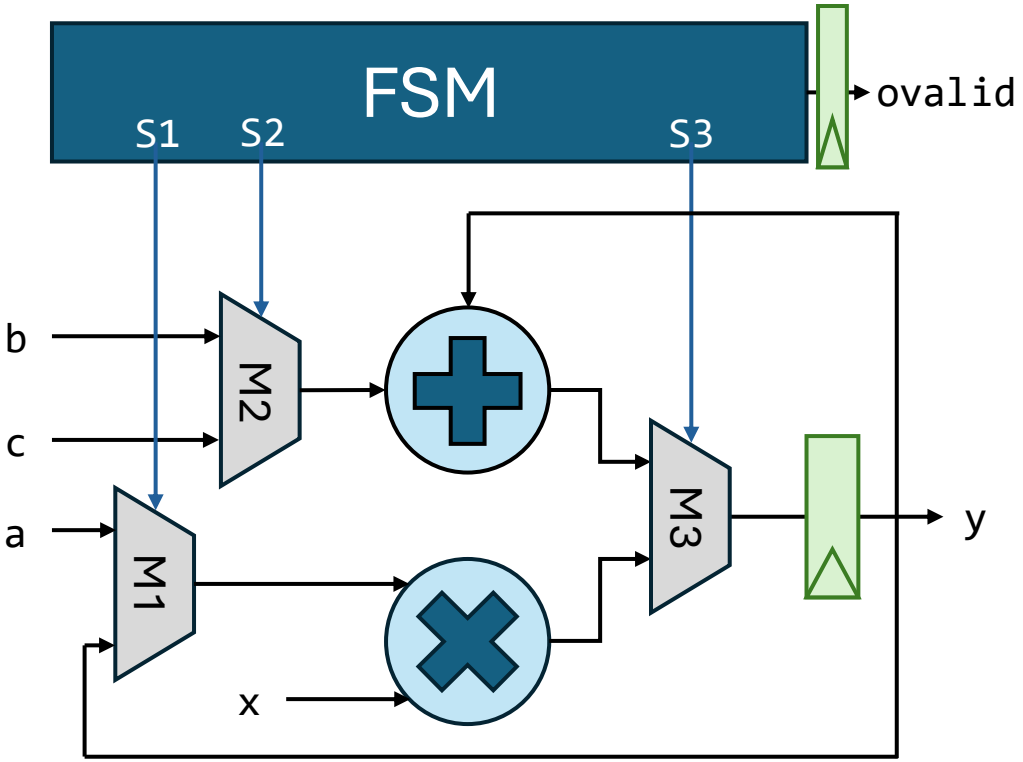
- Compute $y = ax^2 + bx + c$ using only 1 multiplier & 1 adder

- Refactor computation as $y = [(ax+b)x]+c$ to perform over 4 steps:
  1. tmp = a * x
  2. tmp = tmp + b
  3. tmp = tmp * x
  4. y = tmp + c

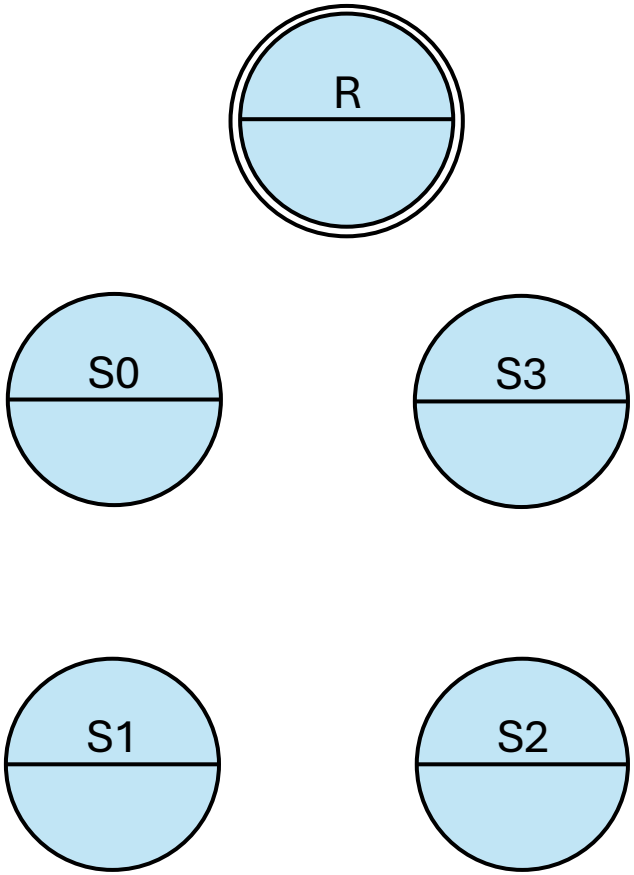


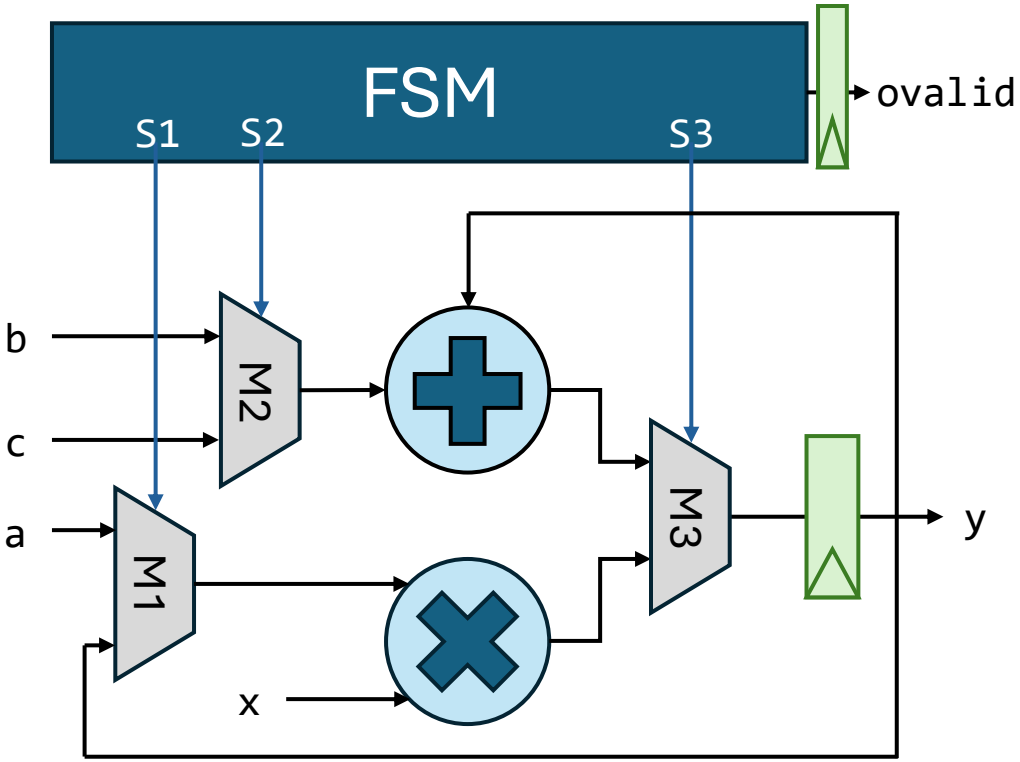
Example: Polynomial Revisited (Datapath Code)

```
module poly_datapath (
      input clk,
      input rst,
      input signed [7:0] a,
      input signed [7:0] b,
      input signed [7:0] c,
      input signed [7:0] x,
      input s1,
      input s2,
      input s3,
      output logic signed [31:0] y
);

logic signed [7:0] m1_out, m2_out, m3_out;

assign m1_out = (s1)? y : a;
assign m2_out = (s2)? c : b;
assign m3_out = (s3)? m1_out*x : m2_out+y;

always_ff @ (posedge clk) begin
    if (rst) y <= 'd0;
    else y <= m3_out;
end

endmodule
```

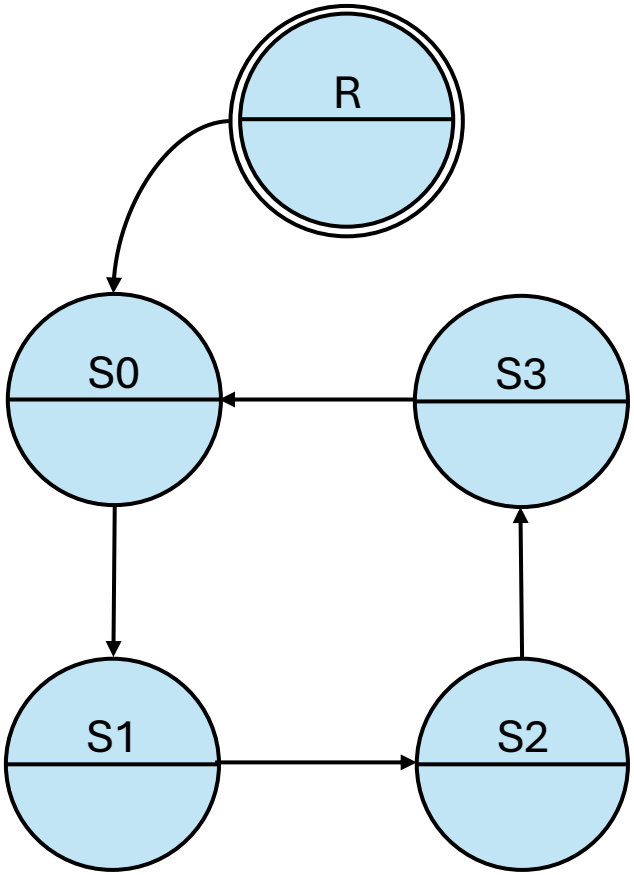


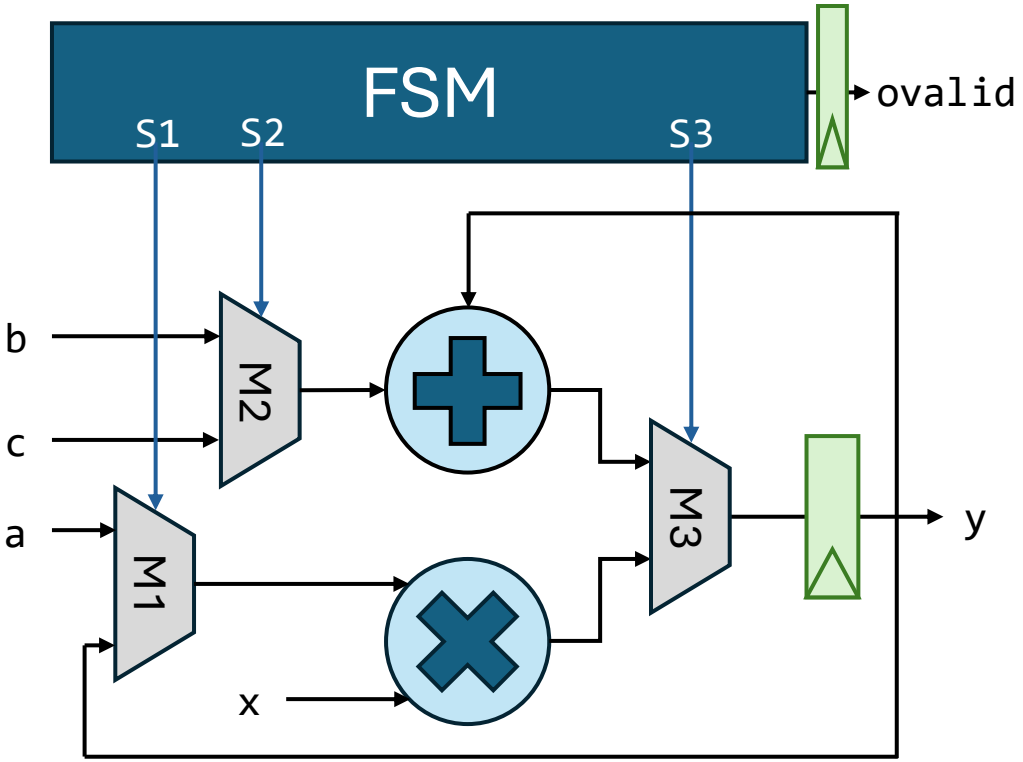
Example: Polynomial Revisited (FSM Design)



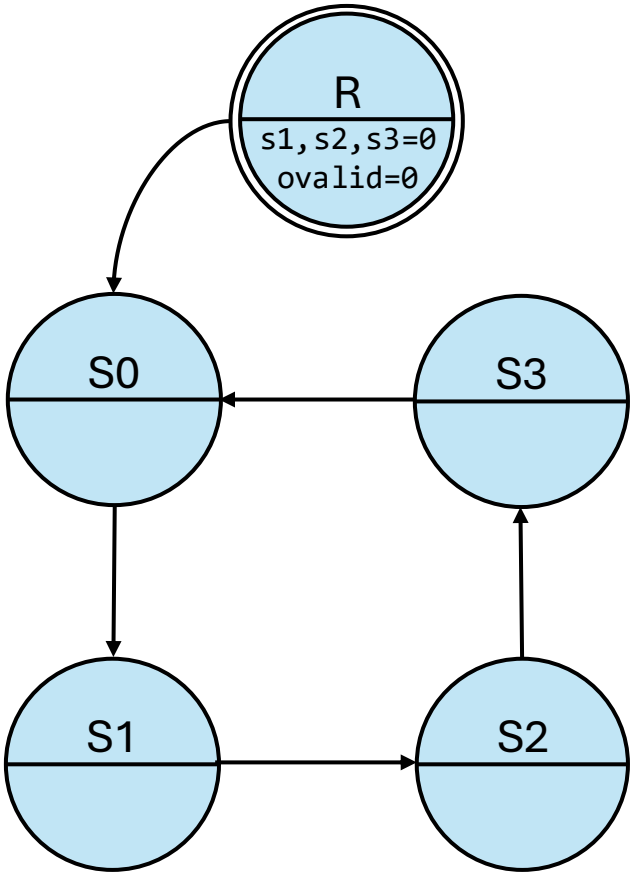


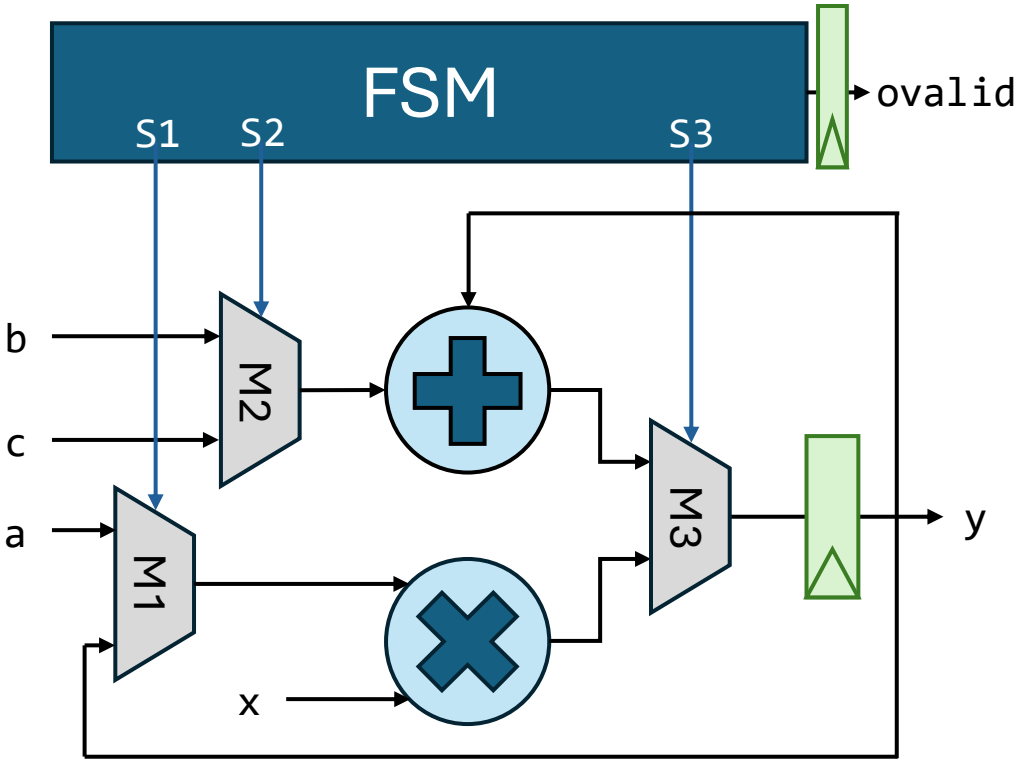
Example: Polynomial Revisited (FSM Design)



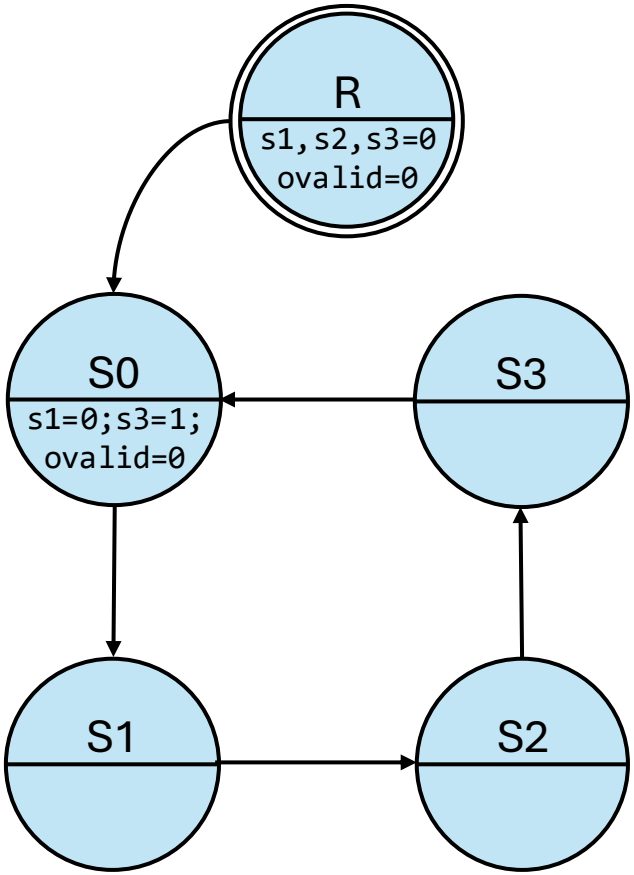


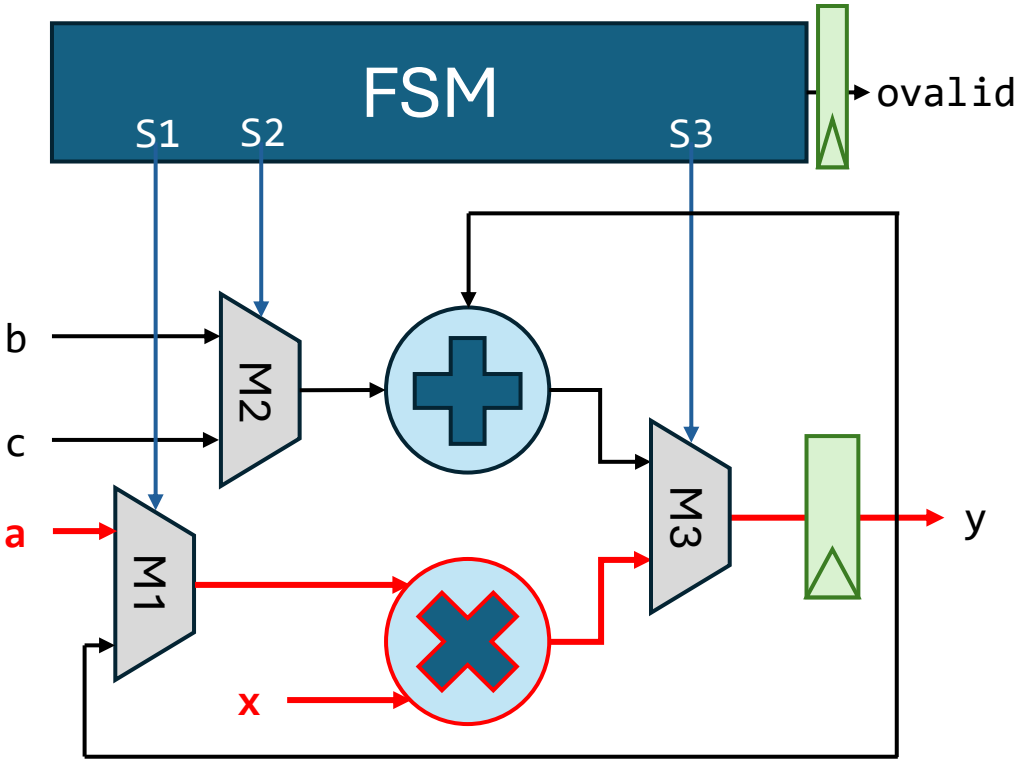
Example: Polynomial Revisited (FSM Design)



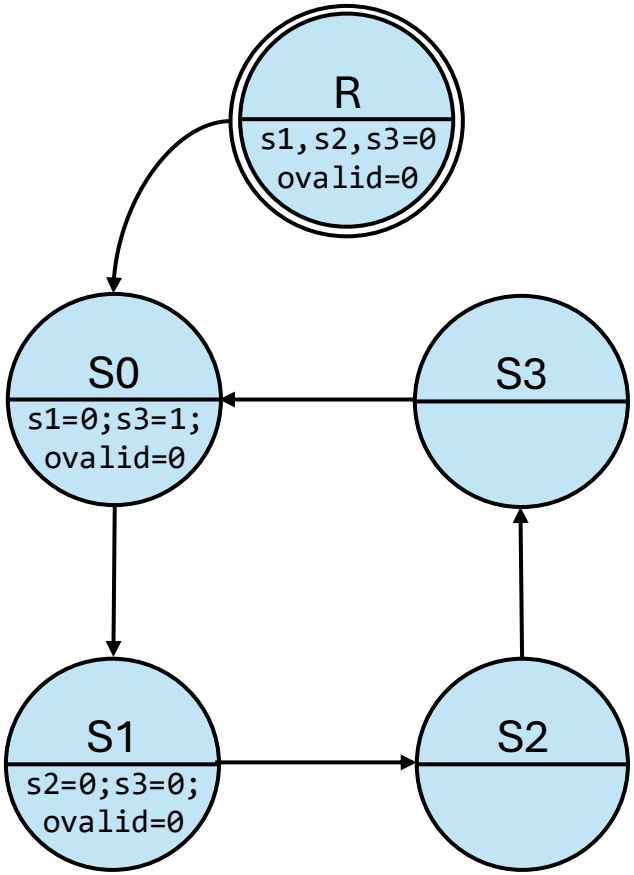


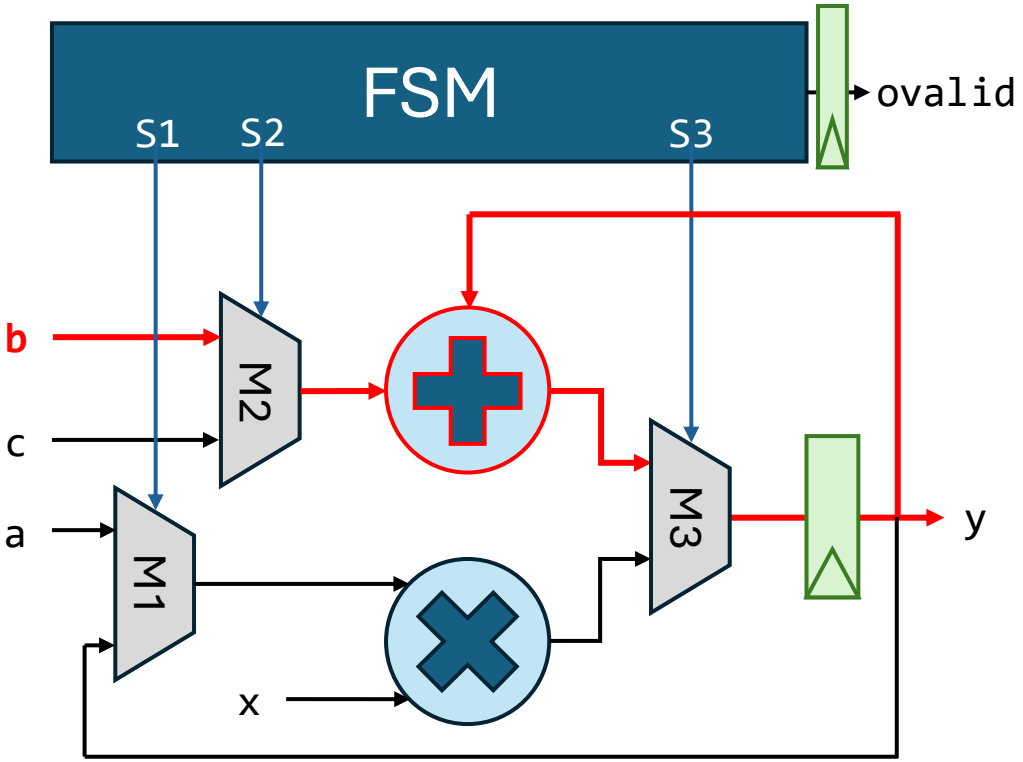
Example: Polynomial Revisited (FSM Design)



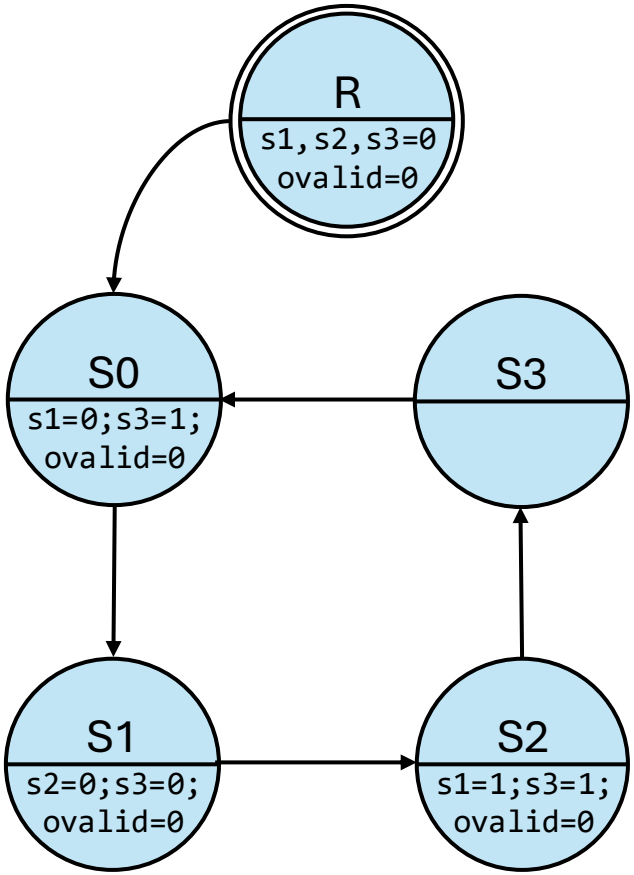


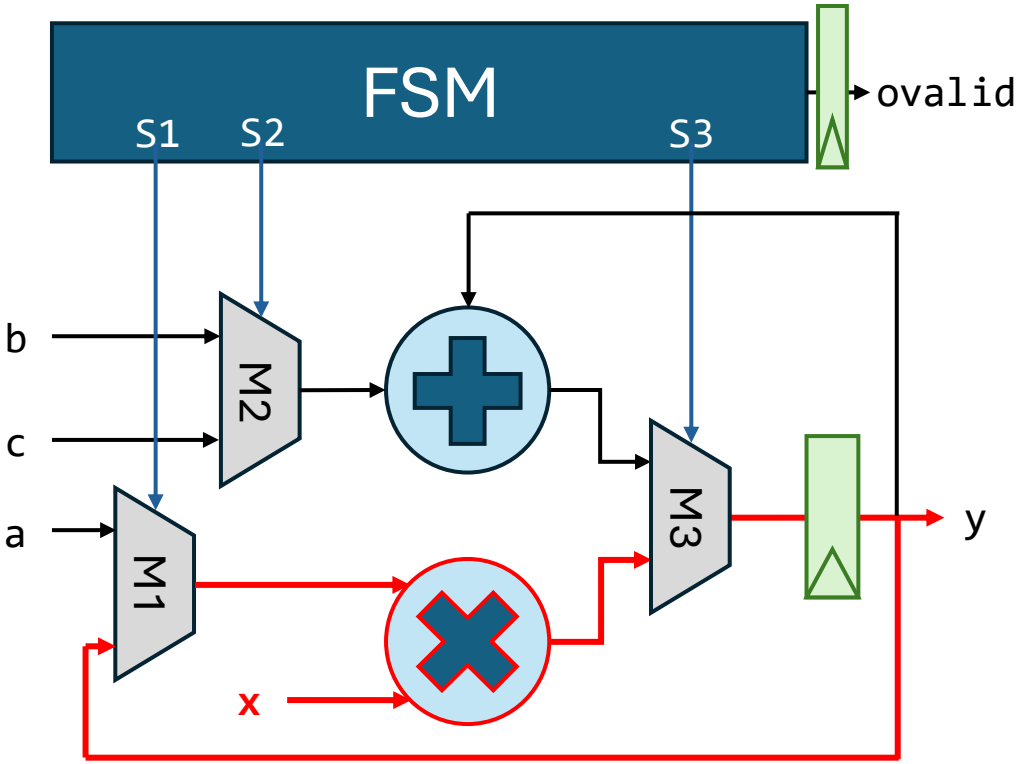
Example: Polynomial Revisited (FSM Design)



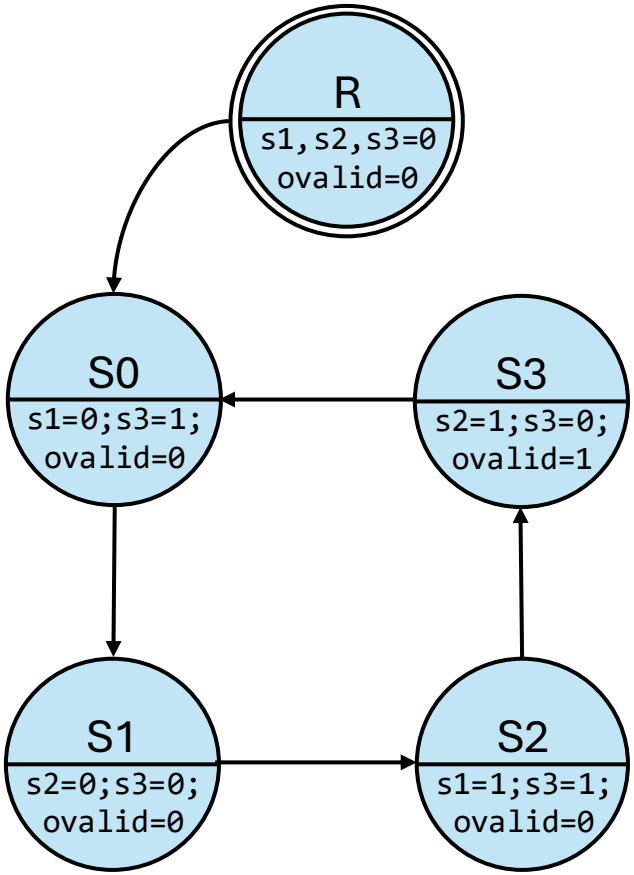


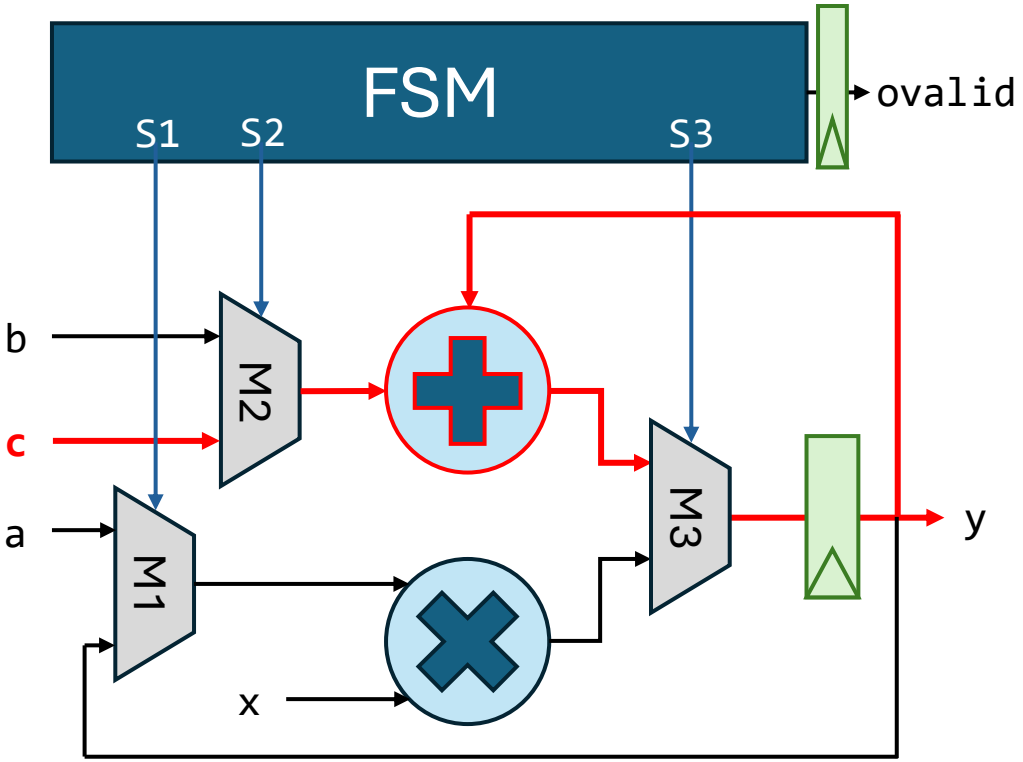
Example: Polynomial Revisited (FSM Design)





Example: Polynomial Revisited (FSM Design)





Example: Polynomial Revisited (FSM Code)

```
module poly_fsm (
      input clk,
      input rst,
      output logic s1,
      output logic s2,
      output logic s3,
      output logic ovalid
);
```

```
endmodule
```

Example: Polynomial Revisited (FSM Code)

```
module poly_fsm (
      input clk,
      input rst,
      output logic s1,
      output logic s2,
      output logic s3,
      output logic ovalid
);

enum {R,S0,S1,S2,S3} state, next_state;
logic s1_w, s2_w, s3_w, ovalid_w;
```

```
endmodule
```

Example: Polynomial Revisited (FSM Code)

```
module poly_fsm (
      input clk,
      input rst,
      output logic s1,
      output logic s2,
      output logic s3,
      output logic ovalid
);

enum {R,S0,S1,S2,S3} state, next_state;
logic s1_w, s2_w, s3_w, ovalid_w;

always_ff @ (posedge clk) begin
    if (rst) begin
        s1 <= 1'b0; s2 <= 1'b0; s3 <= 1'b0;
        state <= R; ovalid <= 1'b0;
    end else begin
        s1 <= s1_w; s2 <= s2_w; s3 <= s3_w;
        state <= next_state; ovalid <= ovalid_w;
    end
end
```

```
endmodule
```

Example: Polynomial Revisited (FSM Code)

```
module poly_fsm (
      input clk,
      input rst,
      output logic s1,
      output logic s2,
      output logic s3,
      output logic ovalid
);

enum {R,S0,S1,S2,S3} state, next_state;
logic s1_w, s2_w, s3_w, ovalid_w;

always_ff @ (posedge clk) begin
    if (rst) begin
        s1 <= 1'b0; s2 <= 1'b0; s3 <= 1'b0;
        state <= R; ovalid <= 1'b0;
    end else begin
        s1 <= s1_w; s2 <= s2_w; s3 <= s3_w;
        state <= next_state; ovalid <= ovalid_w;
    end
end

always_comb begin: state_decoder
  case (state)
    R : next_state = S0;
    S0: next_state = S1;
    S1: next_state = S2;
    S2: next_state = S3;
    S3: next_state = S0;
    default: next_state = R;
  endcase
end
```

```
endmodule
```

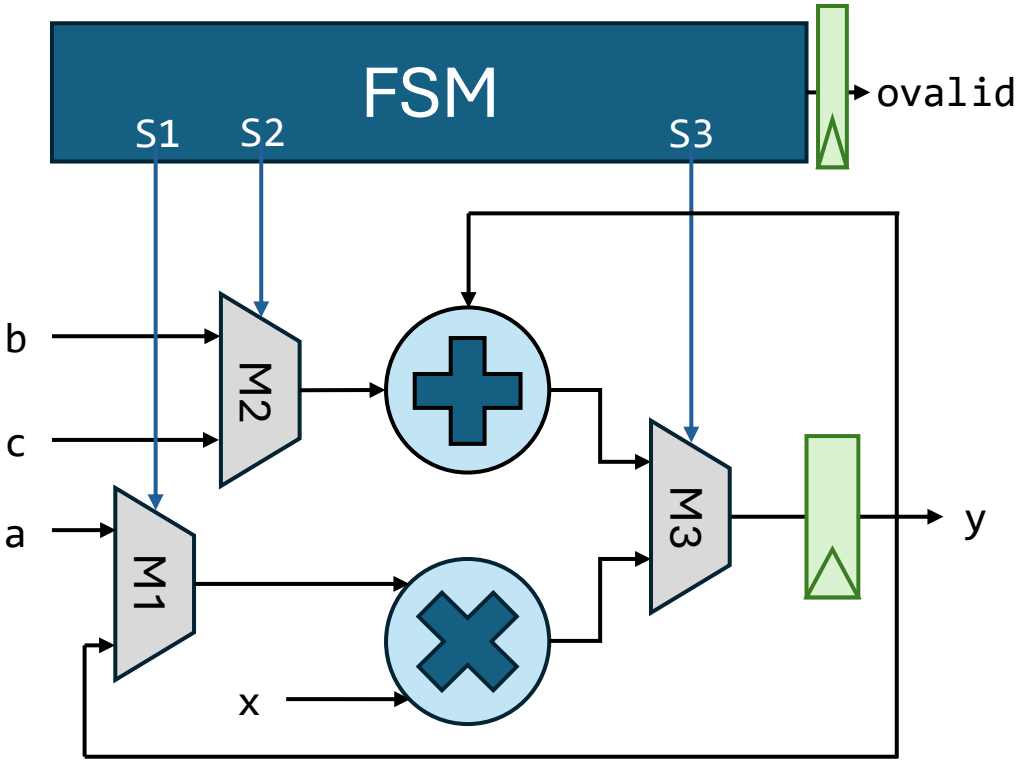
Example: Polynomial Revisited (FSM Code)

```
module poly_fsm (
      input clk,
      input rst,
      output logic s1,
      output logic s2,
      output logic s3,
      output logic ovalid
);

enum {R,S0,S1,S2,S3} state, next_state;
logic s1_w, s2_w, s3_w, ovalid_w;

always_ff @ (posedge clk) begin
    if (rst) begin
        s1 <= 1'b0; s2 <= 1'b0; s3 <= 1'b0;
        state <= R; ovalid <= 1'b0;
    end else begin
        s1 <= s1_w; s2 <= s2_w; s3 <= s3_w;
        state <= next_state; ovalid <= ovalid_w;
    end
end

always_comb begin: state_decoder
  case (state)
    R : next_state = S0;
    S0: next_state = S1;
    S1: next_state = S2;
    S2: next_state = S3;
    S3: next_state = S0;
    default: next_state = R;
  endcase
end
```

```
always_comb begin: output_decoder
  case (state)
    R : begin
      s1_w = 1'b0; s2_w = 1'b0; s3_w = 1'b0;
      ovalid_w = 1'b0;
    end
    S0: begin
      s1_w = 1'b0; s2_w = 1'b0; s3_w = 1'b1;
      ovalid_w = 1'b0;
    end
    S1: begin
      s1_w = 1'b0; s2_w = 1'b0; s3_w = 1'b0;
      ovalid_w = 1'b0;
    end
    S2: begin
      s1_w = 1'b1; s2_w = 1'b0; s3_w = 1'b1;
      ovalid_w = 1'b0;
    end
    S3: begin
      s1_w = 1'b0; s2_w = 1'b1; s3_w = 1'b0;
      ovalid_w = 1'b1;
    end
    default: begin
      s1_w = 1'b0; s2_w = 1'b0; s3_w = 1'b0;
      ovalid_w = 1'b0;
    end
  endcase
end

endmodule
```

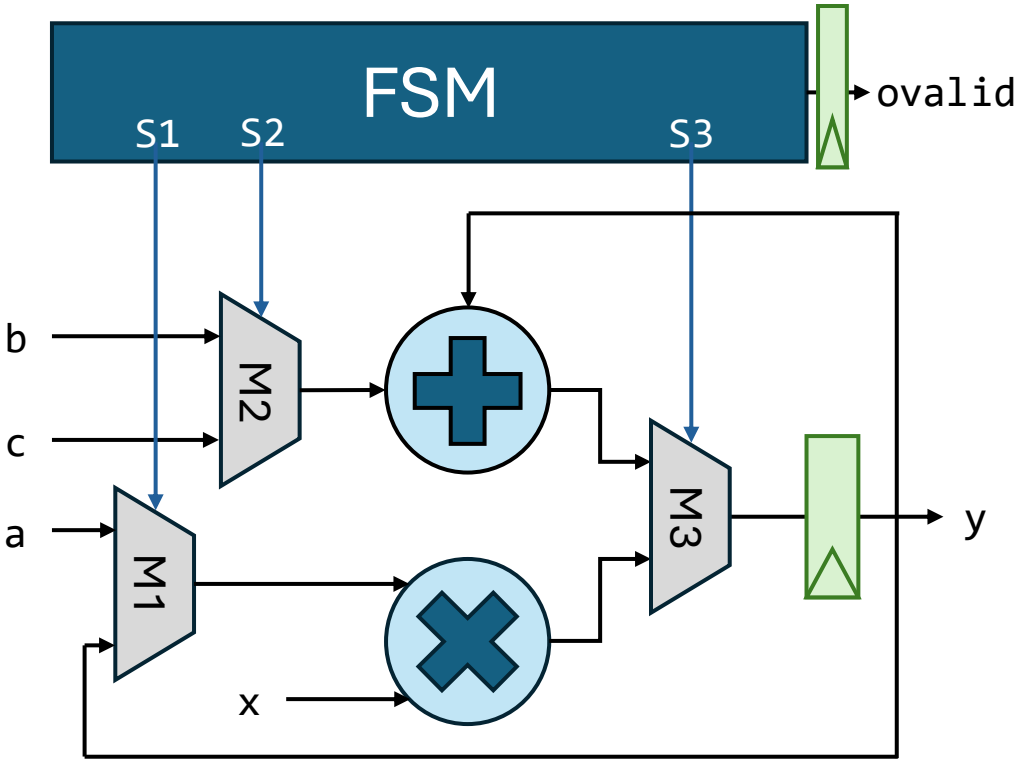
Example: Polynomial Revisited (Datapath Code)

```
module top_level (
      input clk,
      input rst,
      input signed [7:0] a,
      input signed [7:0] b,
      input signed [7:0] c,
      input signed [7:0] x,
      output signed [31:0] y,
      output logic ovalid
);

logic s1, s2, s3, fsm_ovalid;

poly_datapath dp_inst(
  clk, rst, a, b, c, x, s1, s2, s3, y
);

poly_fsm fsm_inst(
  clk, rst, s1, s2, s3, fsm_ovalid
);

always_ff @ (posedge clk) begin
  if (rst) ovalid <= 1'b0;
  else ovalid <= fsm_ovalid;
end

endmodule
```



Example: Polynomial Revisited (Datapath Code)

```
module top_level (
      input clk,
      input rst,
      input signed [7:0] a,
      input signed [7:0] b,
      input signed [7:0] c,
      input signed [7:0] x,
      output signed [31:0] y,
      output logic ovalid
);

logic s1, s2, s3, fsm_ovalid;

poly_datapath dp_inst(
  clk, rst, a, b, c, x, s1, s2, s3, y
);

poly_fsm fsm_inst(
  clk, rst, s1, s2, s3, fsm_ovalid
);

always_ff @ (posedge clk) begin
  if (rst) ovalid <= 1'b0;
  else ovalid <= fsm_ovalid;
end

endmodule
```



Why do we need this?

Summary

- FSMs can have complex control flow with multiple possible transitions from each state
  - Carefully defined state transition conditions guarantee mutual exclusivity and preserved priority
- FSMs can interact with each other
  - Must pay attention to cycle/timing of interacting signals
- FSMs can be used to control datapath (FSMD design style)
  - Decouple compute datapath from control logic

Next Lecture

Improving circuit throughput via pipelining …