Design Example: Level-to-Pulse

- A level-to-pulse converter produces a single-cycle pulse each time its input goes high.
- It’s a synchronous rising-edge detector.
- Sample uses:
  - Buttons and switches pressed by humans for arbitrary periods of time
  - Single-cycle enable signals for counters

Whenever input $L$ goes from low to high...

...output $P$ produces a single pulse, one clock period wide.
Step 1: State Transition Diagram

- Block diagram of desired system:

![Block diagram](image)

- State transition diagram is a useful FSM representation and design aid:

![State transition diagram](image)

- Binary values of states:
  - L=0
  - L=1

- This is the output that results from this state. *(Moore or Mealy?)*
Step 2: Logic Derivation

Transition diagram is readily converted to a state transition table (just a truth table)

<table>
<thead>
<tr>
<th>Current State</th>
<th>In</th>
<th>Next State</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$S_0$</td>
<td>$L$</td>
<td>$S_1^+$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- Combinational logic may be derived using Karnaugh maps
Moore Level-to-Pulse Converter

Moore FSM circuit implementation of level-to-pulse converter:

\[ y_k = f_k(S) \]
Design of a Mealy Level-to-Pulse

1. When \( L=1 \) and \( S=0 \), this output is asserted immediately and until the state transition occurs (or \( L \) changes).

2. While in state \( S=1 \) and as long as \( L \) remains at 1, this output is asserted.

Since outputs are determined by state and inputs, Mealy FSMs may need fewer states than Moore FSM implementations.
Mealy Level-to-Pulse Converter

Mealy FSM circuit implementation of level-to-pulse converter:

- FSM's state simply remembers the previous value of \( L \)
- Circuit benefits from the Mealy FSM's implicit single-cycle assertion of outputs during state transitions

<table>
<thead>
<tr>
<th>Pres. State</th>
<th>In</th>
<th>Next State</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>L</td>
<td>S'</td>
<td>P</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Moore/Mealy Trade-Offs

- How are they different?
  - Moore: outputs = f( state ) only
  - Mealy outputs = f( state and input )
  - Mealy outputs generally occur one cycle earlier than a Moore:

- Compared to a Moore FSM, a Mealy FSM might...
  - Be more difficult to conceptualize and design
  - Have fewer states
Light Switch Revisited

Level-to-Pulse FSM

Light Switch FSM
FSM Example

GOAL:
Build an electronic combination lock with a reset button, two number buttons (0 and 1), and an unlock output. The combination should be 01011.

STEPS:
1. Design lock FSM (block diagram, state transitions)
2. Write Verilog module(s) for FSM
Step 1A: Block Diagram

- Clock generator
- fsm_clock
- reset
- button
- b0_in
- button
- b1_in
- button
- fsm
- unlock
- reset
- b0
- state
- b1
- Unlock LED
- LED DISPLAY
Step 1B: State transition diagram

RESET
Unlock = 0

"0"
Unlock = 0

"01"
Unlock = 0

"01011"
Unlock = 1

"0101"
Unlock = 0

"0100"
Unlock = 0

6 states → 3 bits
Step 2: Write Verilog

```verilog
module lock(clk,reset_in,b0_in,b1_in,out);

input clk,reset,b0_in,b1_in;
output out;

// synchronize push buttons, convert to pulses

// implement state transition diagram
reg [2:0] state,next_state;
always @ (*) begin
    // combinational logic!
    next_state = ???;
end
always @ (posedge clk) state <= next_state;

// generate output
assign out = ???;

// debugging?
endmodule
```
Step 2A: Synchronize buttons

// button -- push button synchronizer and level-to-pulse converter
// OUT goes high for one cycle of CLK whenever IN makes a
// low-to-high transition.

module button(clk,in,out);
    input clk;
    input in;
    output out;

    reg r1,r2,r3;
    always @(posedge clk)
    begin
        r1 <= in; // first reg in synchronizer
        r2 <= r1; // second reg in synchronizer, output is in sync!
        r3 <= r2; // remembers previous state of button
    end

    // rising edge = old value is 0, new value is 1
    assign out = ~r3 & r2;
endmodule
Step 2B: state transition diagram

parameter S_RESET = 0; // state assignments
parameter S_0 = 1;
parameter S_01 = 2;
parameter S_010 = 3;
parameter S_0101 = 4;
parameter S_01011 = 5;

reg [2:0] state, next_state;
always @ (*) begin
  // implement state transition diagram
  if (reset) next_state = S_RESET;
  else case (state)
    S_RESET: next_state = b0 ? S_0 : b1 ? S_RESET : state;
    S_0: next_state = b0 ? S_0 : b1 ? S_01 : state;
    S_01: next_state = b0 ? S_010 : b1 ? S_RESET : state;
    S_010: next_state = b0 ? S_0 : b1 ? S_0101 : state;
    S_0101: next_state = b0 ? S_010 : b1 ? S_01011 : state;
    S_01011: next_state = b0 ? S_0 : b1 ? S_RESET : state;
    default: next_state = S_RESET; // handle unused states
  endcase
end

always @(posedge clk) state <= next_state;
Step 2C: generate output

// it’s a Moore machine! Output only depends on current state
assign out = (state == S_01011);

Step 2D: debugging?

// hmmm. What would be useful to know? Current state?
assign hex_display = {1'b0, state[2:0]};
Step 2: final Verilog implementation

```verilog
module lock(clk,reset_in,b0_in,b1_in,out, hex_display);

input clk,reset,b0_in,b1_in;
output out; output[3:0] hex_display;

wire reset, b0, b1; // synchronize push buttons, convert to pulses
button b_reset(clk,reset_in,reset);
button b_0(clk,b0_in,b0);
button b_1(clk,b1_in,b1);

parameter S_RESET = 0; parameter S_0 = 1; // state assignments
parameter S_01 = 2; parameter S_010 = 3;
parameter S_0101 = 4; parameter S_01011 = 5;

reg [2:0] state,next_state;
always @ (*) begin // implement state transition diagram
  if (reset) next_state = S_RESET;
  else case (state)
    S_RESET: next_state = b0 ? S_0 : b1 ? S_RESET : state;
    S_0: next_state = b0 ? S_0 : b1 ? S_01 : state;
    S_01: next_state = b0 ? S_010 : b1 ? S_RESET : state;
    S_010: next_state = b0 ? S_010 : b1 ? S_0101 : state;
    S_0101: next_state = b0 ? S_0101 : b1 ? S_01011 : state;
    S_01011: next_state = b0 ? S_0101 : b1 ? S_RESET : state;
    default: next_state = S_RESET; // handle unused states
  endcase
end
always @ (posedge clk) state <= next_state;
assign out = (state == S_01011); // assign output: Moore machine
assign hex_display = {1'b0,state}; // debugging
endmodule
```
Where should CLK come from?

- **Option 1: external crystal**
  - Stable, known frequency, typically 50% duty cycle

- **Option 2: internal signals**
  - **Option 2A: output of combinational logic**
    - *No!* If inputs to logic change, output may make several transitions before settling to final value → several rising edges, not just one! Hard to design away output glitches...
  
- **Option 2B: output of a register**
  - Okay, but timing of CLK2 won't line up with CLK1
Summary

• Modern digital system design:
  - Hardware description language → FPGA / ASIC

• Toolchain:
  - Design Entry → Synthesis → Implementation

• New Labkit:
  - Black-box peripherals
  - Almost all functionality is programmed in!

![Diagram showing the toolchain process: Design Entry (Verilog) → Synthesis (xst) → Implementation (map, place, route) → Programming (parallel cable)]