Finite state machines (cont’d)

- Another type of shift register
  - Linear-feedback shift register (LFSR)
    - Used to generate pseudo-random numbers
- Some FSM examples
Let's start with a shift register and scramble states

- Basic shift register

- Mobius counter

- LFSR
module LFSR (clk, reset, out);

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

    reg [0:3] out;
    wire LFSRin;

    assign LFSRin = ~(out[2] ^ out[3]);

    always @(posedge clk) begin
        if (reset) out = 0;
        else            out = {LFSRin, out [0:2]};
    end

endmodule
Adding and overflow

- Very simple: use “+”, e.g., sum[3:0] = a[3:0] + b[3:0];
  - Representing positive and negative numbers
    - shortcut: 2s complement = bit-wise complement + 1
      - 0111 -> 1000 + 0001 -> 1001 (representation of -7)
      - 1001 -> 0110 + 0001 -> 0111 (representation of 7)
  - Make sure addends and sum have same number of bits

- Overflow
  - If only positive numbers then make sum 1 bit bigger
    - If that bit is 1 then there is overflow, e.g, sum[4:0] = a[3:0] + b[3:0];
  - Add two positive numbers and end up with a negative number
  - Add two negative numbers and end up with a positive number
Some simple synthesis examples

```verilog
wire [3:0] x, y, a, b, c, d;
assign apr = ^a;
assign y = a & ~b;
assign x = (a == b) ? a + c : d + a;
```
Example FSM: a vending machine

- Release item after 15 cents are deposited
- Single coin slot for dimes, nickels
- No change
Example: vending machine (cont’d)

- Suitable abstract representation
  - tabulate typical input sequences:
    - 3 nickels
    - nickel, dime
    - dime, nickel
    - two dimes
  - draw state diagram:
    - inputs: N, D, reset
    - output: open chute
  - assumptions:
    - assume N and D asserted for one cycle
    - each state has a self loop for N = D = 0 (no coin)
Example: vending machine (cont’d)

- Suitable abstract representation
  - tabulate typical input sequences:
    - 3 nickels
    - nickel, dime
    - dime, nickel
    - two dimes
  - draw state diagram:
    - inputs: N, D, reset
    - output: open chute
  - assumptions:
    - assume N and D asserted for one cycle
    - each state has a self loop for N = D = 0 (no coin)
Reuse equivalent states

- When are states equivalent: same output AND transitions to equivalent states on every input combination
- Redraw the state diagram using as few states as possible
module vending_machine(clk, reset, N, D, open);

    input   clk, reset, N, D;
    output  open;

    reg [1:0] ps, ns;

    parameter zero = 2’b00; five = 2’b01, ten = 2’b10, fifteen = 2’b11;

    always @(*) begin
        case (ps):
            zero:    if (N) ns = five; elseif (D) ns = ten; else ns = zero;
            five:    if (N) ns = ten; elseif (D) ns = fifteen; else ns = five;
            ten:     if (N | D) ns = fifteen; else ns = ten;
            fifteen: ns = fifteen;
            default: ns = zero;
        end

        always @(posedge clk) begin
            if (reset) ps = zero;
            else        ps = ns;
        end

        assign open = (ps == fifteen);
    end
endmodule
Example: vending machine (cont’d)

- Minimize number of states - reuse states whenever possible

<table>
<thead>
<tr>
<th>present state</th>
<th>inputs</th>
<th>next state</th>
<th>output open</th>
</tr>
</thead>
<tbody>
<tr>
<td>0¢</td>
<td>0 0</td>
<td>0¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>5¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>10¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>5¢</td>
<td>0 0</td>
<td>5¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>10¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>15¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>10¢</td>
<td>0 0</td>
<td>10¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>15¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>15¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>15¢</td>
<td>– –</td>
<td>15¢</td>
<td>1</td>
</tr>
</tbody>
</table>

symbolic state table
Example: vending machine (cont’d)

- Uniquely encode states then come up with logic for next state bits (D1 and D0) and the output signal, open

<table>
<thead>
<tr>
<th>present state</th>
<th>inputs</th>
<th>next state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 Q0</td>
<td>D N</td>
<td>D1 D0</td>
<td>open</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>0 1</td>
<td>0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>1 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
<td>0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 1</td>
<td>0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0</td>
<td>1 0</td>
<td>0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td>1 1</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
<td>0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>1 0</td>
<td>0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>1 1</td>
<td>1 1</td>
<td>0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td>1 1</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1</td>
<td>0 0</td>
</tr>
</tbody>
</table>
Implementation of vending machine

- Mapping to logic (how many logic blocks in our FPGA?)

\[
D_1 = Q_1 + D + Q_0 \cdot N \\
D_0 = Q_0' \cdot N + Q_0 \cdot N' + Q_1 \cdot N + Q_1 \cdot D \\
\text{OPEN} = Q_1 \cdot Q_0
\]
Vending machine: Moore to synch. Mealy

- OPEN = Q1Q0 creates a combinational delay after Q1 and Q0 change
- This can be corrected by retiming, i.e., move flip-flops and logic through each other to improve delay – pre-compute OPEN then store it in FF
- OPEN.d = (Q1 + D + Q0N)(Q0'N + Q0N' + Q1N + Q1D)
  = Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D
- Implementation now is completely synchronous: outputs change on clock
  - another reason programmable devices have FF at end of logic
The following types of functions can be realized in a single ALM:

- Two independent 4-input functions
- An independent 5-input function and an independent 3-input function
- A 5-input function and a 4-input function, if they share one input
- Two 5-input functions, if they share two inputs
- An independent 6-input function
- Two 6-input functions, if they share four inputs and share function
- Some 7-input functions

An ALM also has 4 bits of memory
- We’ll discuss later when we talk about sequential logic
Example: traffic light controller

- Highway/farm road intersection
Example: traffic light controller (cont’d)

- A busy highway is intersected by a little used farmroad
- Detectors C sense the presence of cars waiting on the farmroad
  - with no car on farmroad, light remain green in highway direction
  - if vehicle on farmroad, highway lights go from Green to Yellow to Red, allowing the farmroad lights to become green
  - these stay green only as long as a farmroad car is detected but never longer than a set interval
  - when these are met, farm lights transition from Green to Yellow to Red, allowing highway to return to green
  - even if farmroad vehicles are waiting, highway gets at least a set interval as green
- Assume you have an interval timer (a second state machine) that generates:
  - a short time pulse (TS) and
  - a long time pulse (TL),
  - in response to a set (ST) signal.
  - TS is to be used for timing yellow lights and TL for green lights
Example: traffic light controller (cont’d)

- Tabulation of inputs and outputs

<table>
<thead>
<tr>
<th>inputs</th>
<th>description</th>
<th>outputs</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reset</td>
<td>place FSM in initial state</td>
<td>HG, HY, HR</td>
<td>assert green/yellow/red highway lights</td>
</tr>
<tr>
<td>C</td>
<td>detect vehicle on the farm road</td>
<td>FG, FY, FR</td>
<td>assert green/yellow/red highway lights</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>inputs</th>
<th>description</th>
<th>outputs</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST</td>
<td>start timing a short or long interval</td>
<td>TS</td>
<td>short time interval expired</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TL</td>
<td>long time interval expired</td>
</tr>
</tbody>
</table>

- Tabulation of unique states – some light configurations imply others

<table>
<thead>
<tr>
<th>state</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HG</td>
<td>highway green (farm road red)</td>
</tr>
<tr>
<td>HY</td>
<td>highway yellow (farm road red)</td>
</tr>
<tr>
<td>FG</td>
<td>farm road green (highway red)</td>
</tr>
<tr>
<td>FY</td>
<td>farm road yellow (highway red)</td>
</tr>
</tbody>
</table>
Add inputs/outputs to arcs

- Inputs: C, TS, TL
- Output: state and ST
Example: traffic light controller (cont’d)

- Completed state diagram

![State Diagram]

- States:
  - HG
  - HY
  - FG
  - FY

- Transitions:
  - (TL•C)'
  - TL•C / ST
  - TS / ST
  - TS'
  - TS / ST
  - TL+C' / ST
  - (TL+C)'

- Reset
Example: traffic light controller (cont’)

- Generate state table with symbolic states
- Consider state assignments

<table>
<thead>
<tr>
<th>Inputs C TL TS</th>
<th>Present State HG</th>
<th>Next State HG</th>
<th>Outputs ST H F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – – –</td>
<td>HG</td>
<td>HG</td>
<td>0 Green Red</td>
</tr>
<tr>
<td>– 0 –</td>
<td>HG</td>
<td>HG</td>
<td>0 Green Red</td>
</tr>
<tr>
<td>1 1 –</td>
<td>HG</td>
<td>HY</td>
<td>1 Green Red</td>
</tr>
<tr>
<td>– – 0</td>
<td>HY</td>
<td>HY</td>
<td>0 Yellow Red</td>
</tr>
<tr>
<td>– – 1</td>
<td>HY</td>
<td>FG</td>
<td>1 Yellow Red</td>
</tr>
<tr>
<td>1 0 –</td>
<td>FG</td>
<td>FG</td>
<td>0 Red Green</td>
</tr>
<tr>
<td>0 – –</td>
<td>FG</td>
<td>FY</td>
<td>1 Red Green</td>
</tr>
<tr>
<td>– 1 –</td>
<td>FG</td>
<td>FY</td>
<td>1 Red Green</td>
</tr>
<tr>
<td>– – 0</td>
<td>FY</td>
<td>FY</td>
<td>0 Red Yellow</td>
</tr>
<tr>
<td>– – 1</td>
<td>FY</td>
<td>HG</td>
<td>1 Red Yellow</td>
</tr>
</tbody>
</table>

SA1: HG = 00  HY = 01  FG = 11  FY = 10  
SA2: HG = 00  HY = 10  FG = 01  FY = 11  
SA3: HG = 0001  HY = 0010  FG = 0100  FY = 1000  (one-hot)  
SA4: HG = 001100  HY = 010100  FG = 100001  FY = 100010  (output-oriented)

output encoding – similar problem to state assignment
(Green = 00, Yellow = 01, Red = 10)

Example: traffic light controller (cont’)

- Generate state table with symbolic states
- Consider state assignments

<table>
<thead>
<tr>
<th>Inputs C TL TS</th>
<th>Present State HG</th>
<th>Next State HG</th>
<th>Outputs ST H F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – – –</td>
<td>HG</td>
<td>HG</td>
<td>0 Green Red</td>
</tr>
<tr>
<td>– 0 –</td>
<td>HG</td>
<td>HG</td>
<td>0 Green Red</td>
</tr>
<tr>
<td>1 1 –</td>
<td>HG</td>
<td>HY</td>
<td>1 Green Red</td>
</tr>
<tr>
<td>– – 0</td>
<td>HY</td>
<td>HY</td>
<td>0 Yellow Red</td>
</tr>
<tr>
<td>– – 1</td>
<td>HY</td>
<td>FG</td>
<td>1 Yellow Red</td>
</tr>
<tr>
<td>1 0 –</td>
<td>FG</td>
<td>FG</td>
<td>0 Red Green</td>
</tr>
<tr>
<td>0 – –</td>
<td>FG</td>
<td>FY</td>
<td>1 Red Green</td>
</tr>
<tr>
<td>– 1 –</td>
<td>FG</td>
<td>FY</td>
<td>1 Red Green</td>
</tr>
<tr>
<td>– – 0</td>
<td>FY</td>
<td>FY</td>
<td>0 Red Yellow</td>
</tr>
<tr>
<td>– – 1</td>
<td>FY</td>
<td>HG</td>
<td>1 Red Yellow</td>
</tr>
</tbody>
</table>

SA1: HG = 00  HY = 01  FG = 11  FY = 10  
SA2: HG = 00  HY = 10  FG = 01  FY = 11  
SA3: HG = 0001  HY = 0010  FG = 0100  FY = 1000  (one-hot)  
SA4: HG = 001100  HY = 010100  FG = 100001  FY = 100010  (output-oriented)
Logic for different state assignments

- **SA1**
  
  \[NS1 = C \cdot TL \cdot PS1 \cdot PS0 + TS \cdot PS1' \cdot PS0 + TS \cdot PS1 \cdot PS0' + C' \cdot PS1 \cdot PS0 + TL \cdot PS1 \cdot PS0\]
  
  \[NS0 = C \cdot TL \cdot PS1' \cdot PS0' + C \cdot TL' \cdot PS1 \cdot PS0 + PS1' \cdot PS0\]
  
  \[ST = C \cdot TL \cdot PS1' \cdot PS0' + TS \cdot PS1' \cdot PS0 + TS \cdot PS1 \cdot PS0' + C' \cdot PS1 \cdot PS0 + TL \cdot PS1 \cdot PS0\]
  
  \[H1 = PS1\]
  
  \[F1 = PS1'\]
  
  \[H0 = PS1' \cdot PS0\]
  
  \[F0 = PS1 \cdot PS0'\]

- **SA2**
  
  \[NS1 = C \cdot TL \cdot PS1' + TS' \cdot PS1 + C' \cdot PS1' \cdot PS0\]
  
  \[NS0 = TS \cdot PS1 \cdot PS0' + PS1' \cdot PS0 + TS' \cdot PS1 \cdot PS0\]
  
  \[ST = C \cdot TL \cdot PS1' + C' \cdot PS1' \cdot PS0 + TS \cdot PS1\]
  
  \[H1 = PS0\]
  
  \[F1 = PS0'\]
  
  \[H0 = PS1 \cdot PS0'\]
  
  \[F0 = PS1 \cdot PS0\]

- **SA3**
  
  \[NS3 = C' \cdot PS2 + TL \cdot PS2 + TS' \cdot PS3\]
  
  \[NS2 = TS \cdot PS1 + C \cdot TL' \cdot PS2\]
  
  \[NS1 = C \cdot TL \cdot PS0 + TS' \cdot PS1\]
  
  \[NS0 = C' \cdot PS0 + TL' \cdot PS0 + TS \cdot PS3\]
  
  \[ST = C \cdot TL \cdot PS0 + TS \cdot PS1 + C' \cdot PS2 + TL \cdot PS2 + TS \cdot PS3\]
  
  \[H1 = PS3 + PS2\]
  
  \[H0 = PS1\]
  
  \[F1 = PS1 + PS0\]
  
  \[F0 = PS3\]

- **SA4**
  
  left as an exercise but this is the one we are going to go with
Traffic light controller
as two communicating FSMs

- Without separate timer
  - S0 would require 7 states
  - S1 would require 3 states
  - S2 would require 7 states
  - S3 would require 3 states
  - S1 and S3 have simple transformation
  - S0 and S2 would require many more arcs
    - C could change in any of seven states

- By factoring out timer
  - greatly reduce number of states
    - 4 instead of 20
  - counter only requires seven or eight states
    - 12 total instead of 20
Traffic light controller FSM

- Specification of inputs, outputs, and state elements

```verilog
module FSM(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, clk);
  output HR, HY, HG;
  output FR, FY, FG;
  output ST;
  input TS, TL, C;
  input reset;
  input clk;

  reg [6:1] state;
  reg ST;

  parameter highwaygreen = 6'b001100;
  parameter highwayyellow  = 6'b010100;
  parameter farmroadgreen  = 6'b100001;
  parameter farmroadyellow = 6'b100010;

  assign HR = state[6];
  assign HY = state[5];
  assign HG = state[4];
  assign FR = state[3];
  assign FY = state[2];
  assign FG = state[1];

specify state bits and codes for each state
as well as connections to outputs
```
Traffic light controller FSM (cont’d)

initial begin state = highwaygreen; ST = 0; end

always @(posedge clk)
begin
  if (reset)
    begin state = highwaygreen; ST = 1; end
  else
    begin
      ST = 0;
      case (state)
        highwaygreen:
          if (TL & C) begin state = highwayyellow; ST = 1; end
        highwayyellow:
          if (TS) begin state = farmroadgreen; ST = 1; end
        farmroadgreen:
          if (TL | !C) begin state = farmroadyellow; ST = 1; end
        farmroadyellow:
          if (TS) begin state = highwaygreen; ST = 1; end
      endcase
    end
end
endmodule

Case statement triggered by clock edge.
Timer for traffic light controller

- Another FSM

```verbatim
module Timer(TS, TL, ST, Clk);
    output   TS;
    output   TL;
    input    ST;
    input    Clk;
    integer  value;

    assign TS = (value >= 4); // 5 cycles after reset
    assign TL = (value >= 14); // 15 cycles after reset

    always @(posedge Clk)
        if (ST) value = 0; else value = value + 1;

endmodule
```
Complete traffic light controller

- Tying it all together (FSM + timer)
  - structural Verilog (same as a schematic drawing)

```verilog
module main(HR, HY, HG, FR, FY, FG, reset, C, clk);
  output HR, HY, HG, FR, FY, FG;
  input  reset, C, clk;

  Timer part1(TS, TL, ST, clk);
  FSM part2(HR, HY, HG, FR, FY, FG, ST, TS, TL, C, reset, clk);
endmodule
```
Communicating finite state machines

- One machine's output is another machine's input

---

4-cycle handshake

- CLK
- FSM1
- X
- FSM2
- Y

A
B
C
D

machines advance in lock step
initial inputs/outputs: \( X = 0, \ Y = 0 \)