EECS 150 - Components and Design Techniques for Digital Systems

Lec 06 – Using FSMs
9-13-07

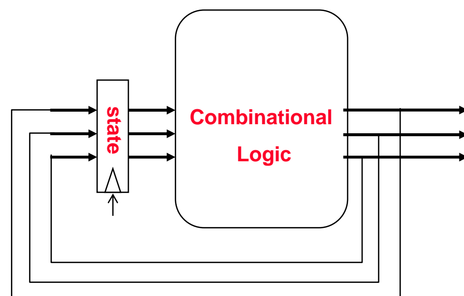
David Culler
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www.eecs.berkeley.edu/~culler
http:/inst.eecs.berkeley.edu/~cs150

Outline

- Review FSMs
- Mapping to FPGAs
- Typical uses of FSMs
- Synchronous Seq. Circuits – safe composition
- Timing
- FSMs in verilog

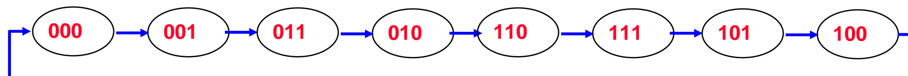
Review: Typical Controller: state



state(t+1) = F (state(t))

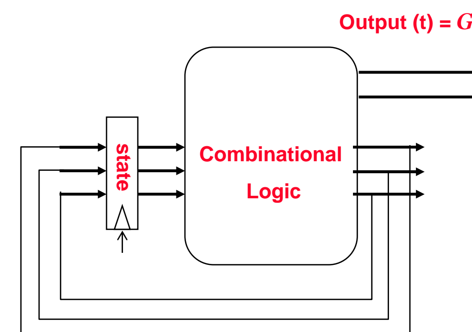
state			Next state		
i2	i1	i0	o2	o1	o0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	1	0
0	1	1	0	1	0
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	1	1
1	1	1	1	0	1

Example: Gray Code Sequence



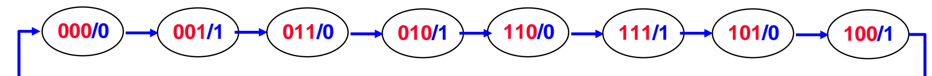
Typical Controller: state + output

Output (t) = G(state(t))

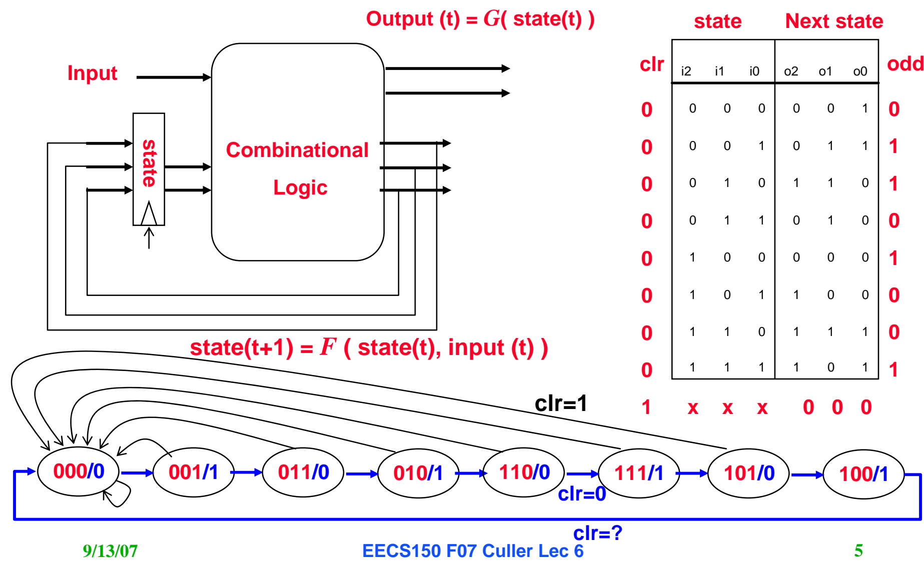


state(t+1) = F (state(t))

state			Next state			
i2	i1	i0	o2	o1	o0	odd
0	0	0	0	0	1	0
0	0	1	0	1	1	1
0	1	0	1	1	0	1
0	1	1	0	1	0	0
1	0	0	0	0	0	1
1	0	1	1	0	0	0
1	1	0	1	1	1	0
1	1	1	1	0	1	1



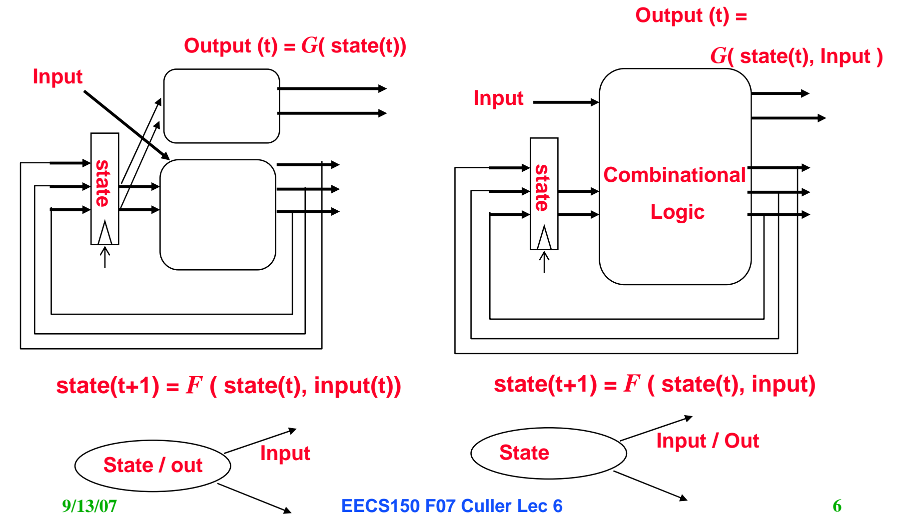
Typical Controller: state + output + input



clr	state i2	i1	i0	Next state o2	o1	o0	odd
0	0	0	0	0	0	1	0
0	0	0	1	0	1	1	1
0	0	1	0	1	1	0	1
0	0	1	1	0	1	0	0
0	1	0	0	0	0	0	1
0	1	0	1	1	0	0	0
0	1	1	0	1	1	1	0
0	1	1	1	1	0	1	1
1	x	x	x	0	0	0	

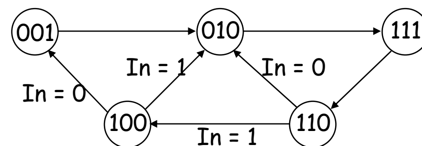
Review: Two Kinds of FSMs

- **Moore Machine vs Mealy Machine**



Review: Finite State Machine Representations

- States: determined by possible values in sequential storage elements
- Transitions: change of state
- Clock: controls when state can change by controlling storage elements



- Sequential Logic
  - Sequences through a series of states
  - Based on sequence of values on input signals
  - Clock period defines elements of sequence

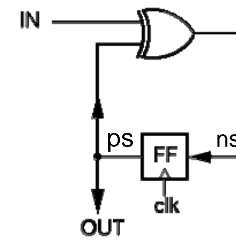
Review: Formal Design Process

Logic equations from table:

OUT = PS

NS = PS xor IN

- Circuit Diagram:



  - XOR gate for ns calculation
  - DFF to hold present state
  - no logic needed for output

- Review of Design Steps:

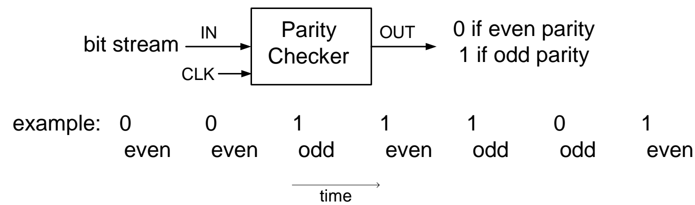
1. Circuit functional specification
2. State Transition Diagram
3. Symbolic State Transition Table
4. Encoded State Transition Table
5. Derive Logic Equations
6. Circuit Diagram

FFs for state

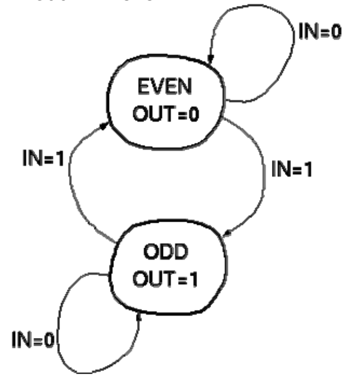
CL for NS and OUT

Take this seriously!

Formal Design Process



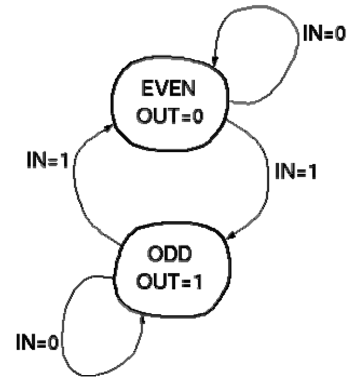
- **"State Transition Diagram"**
  - **circuit is in one of two states.**
  - **transition on each cycle with each new input, over exactly one arc (edge).**
  - **Output depends on which state the circuit is in.**



Formal Design Process

- **State Transition Table:**

present state	OUT	IN	next state
EVEN	0	0	EVEN
EVEN	0	1	ODD
ODD	1	0	ODD
ODD	1	1	EVEN



- **Invent a code to represent states:**

Let 0 = EVEN state, 1 = ODD state

present state (ps)	OUT	IN	next state (ns)
0	0	0	0
0	0	1	1
1	1	0	1
1	1	1	0

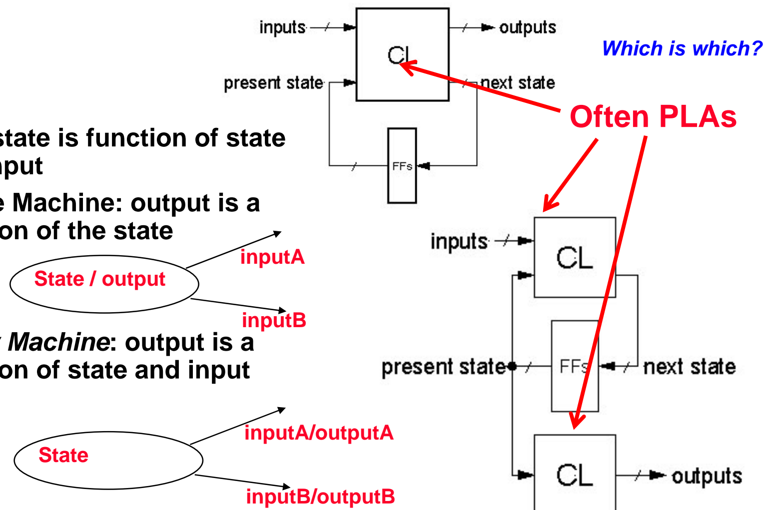
Derive logic equations from table (how?):

OUT = PS

NS = PS xor IN

Review: What's an FSM?

- **Next state is function of state and input**
- **Moore Machine: output is a function of the state**
- ***Mealy Machine*: output is a function of state and input**



How to quickly implement the State Transition Diagram?

One Answer: Xilinx 4000 CLB

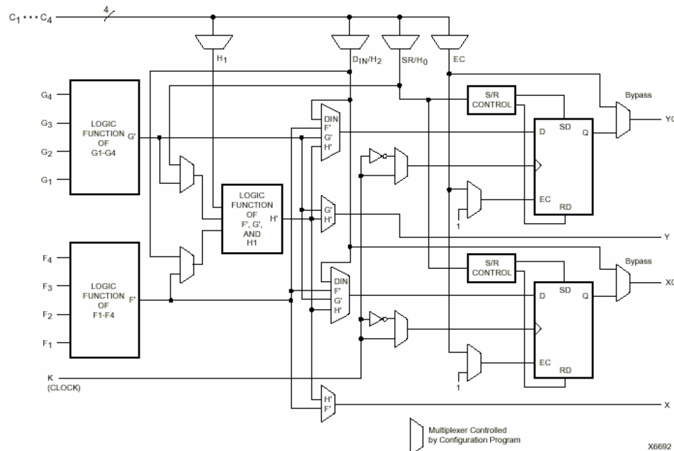


Figure 1: Simplified Block Diagram of XC4000 Series CLB (RAM and Carry Logic functions not shown)

Two 4-input functions, registered output

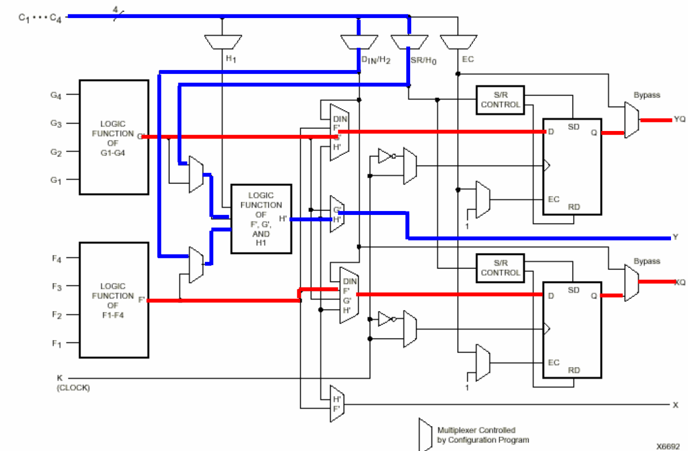


Figure 1: Simplified Block Diagram of XC4000 Series CLB (RAM and Carry Logic functions not shown)

5-input function, combinational output

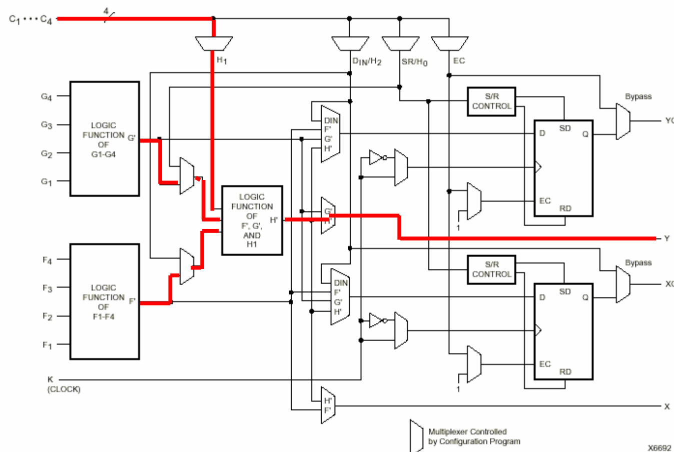
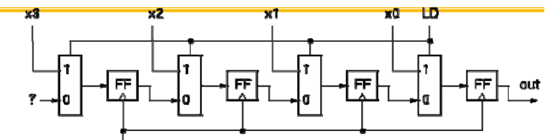


Figure 1: Simplified Block Diagram of XC4000 Series CLB (RAM and Carry Logic functions not shown)

Recall: Parallel to Serial Converter

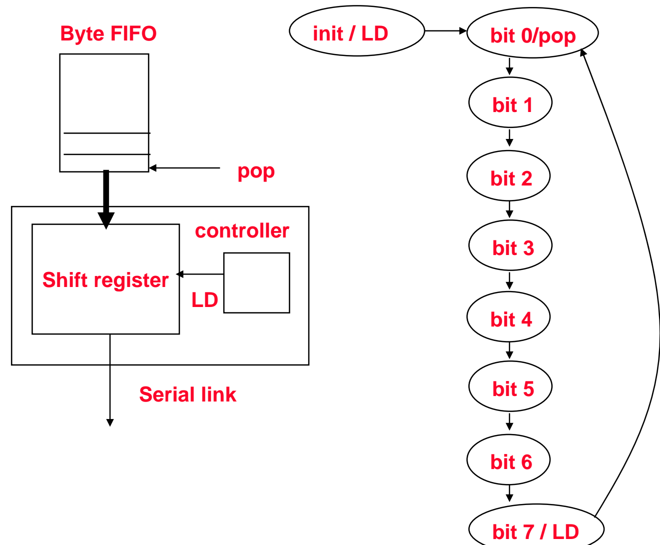



```
//Parallel to Serial converter
module ParToSer(LD, X, out, CLK);
  input [3:0] X;
  input       LD, CLK;
  output      out;
  reg         out;
  reg [3:0]   Q;
  assign out =      Q[0];
  always @ (posedge CLK) begin
      if (LD) Q <= X;
      else Q <=    {1'b0,Q[3:1]};
  end
endmodule // ParToSer
```

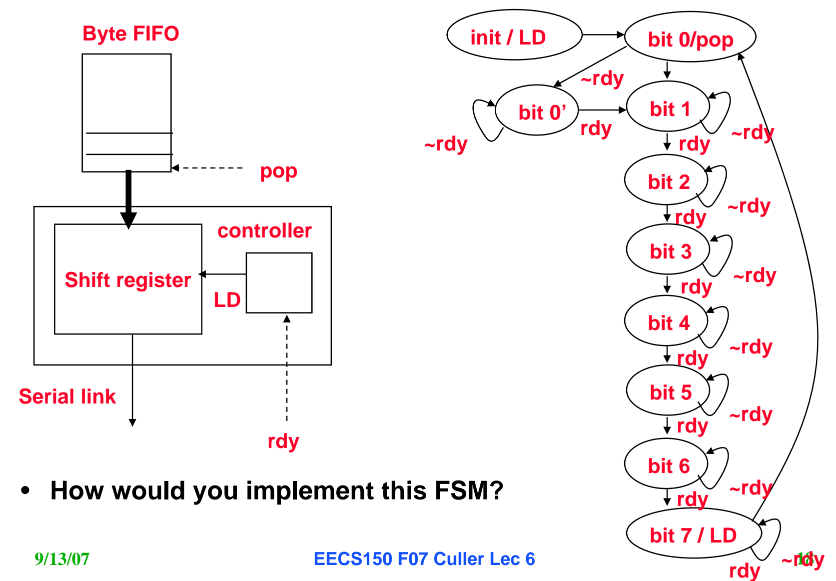
One common use of FSMs is in adapters from one subsystem to another.

- different data widths
- different bit rates
- different protocols, …

Example: Byte-bit stream



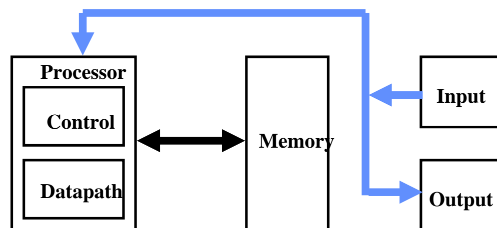
Byte-bit stream with Rate Matching



- How would you implement this FSM?

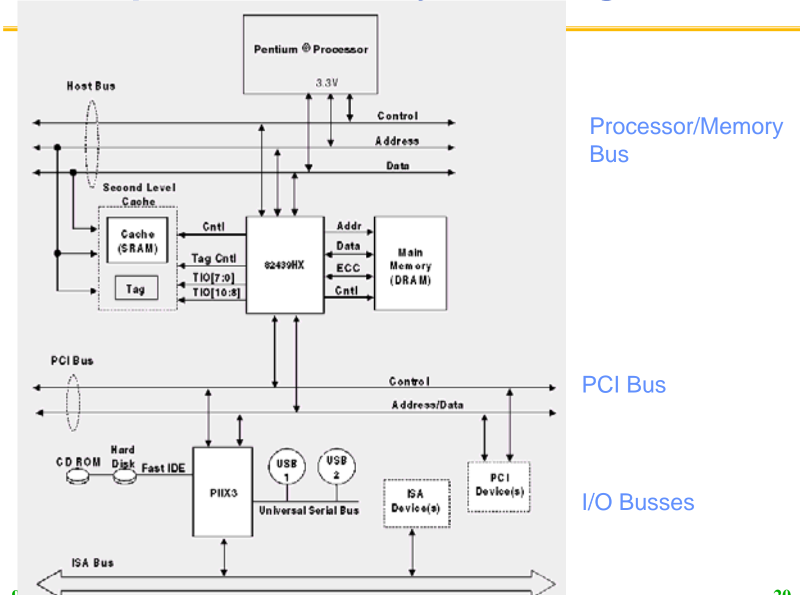
Another example: bus protocols

- A bus is:
  - shared communication link
  - single set of wires used to connect multiple subsystems



- A Bus is also a fundamental tool for composing large, complex systems (more later in the term)
  - systematic means of abstraction

Example: Pentium System Organization

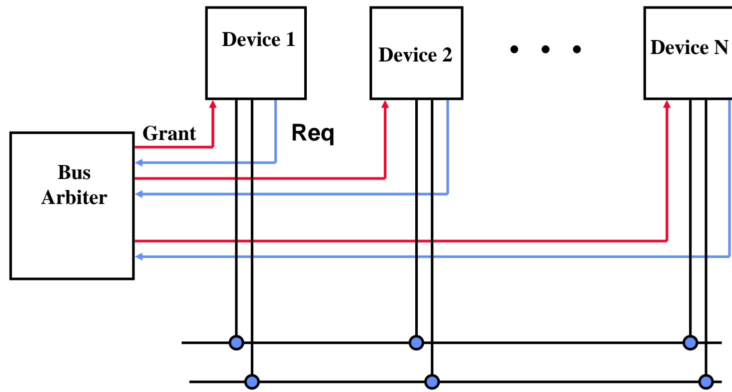


Processor/Memory Bus

PCI Bus

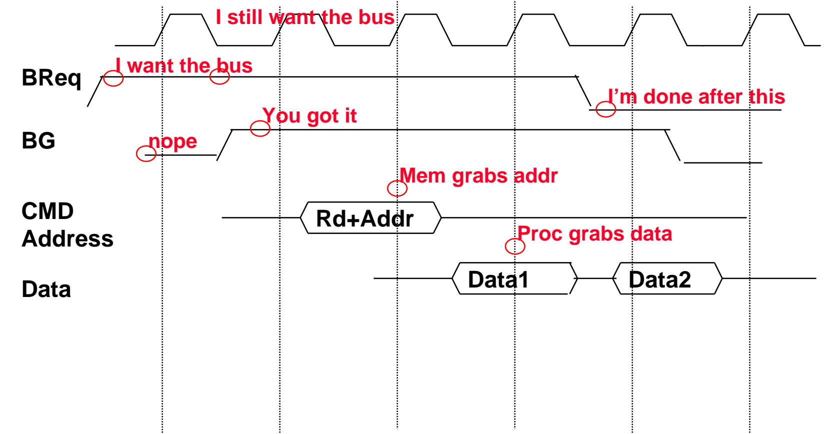
I/O Busses

Arbitration for the bus…



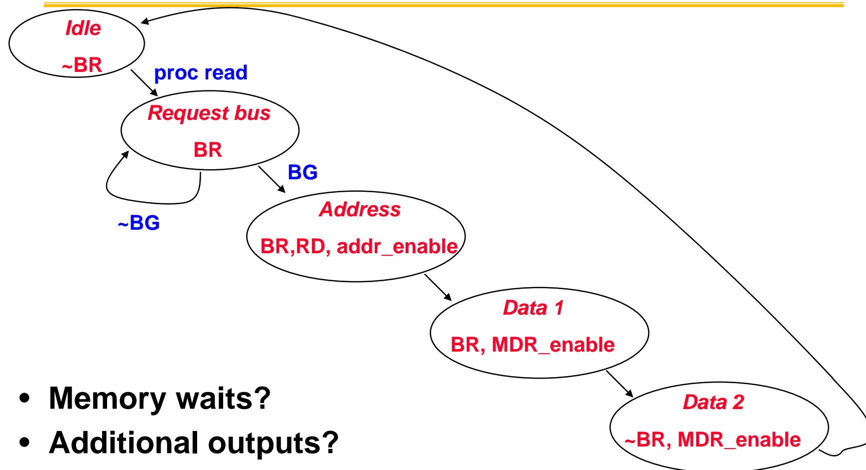
- Central arbitration shown here
  - Used in essentially all processor-memory busses and in high-speed I/O busses

Simple Synchronous Protocol



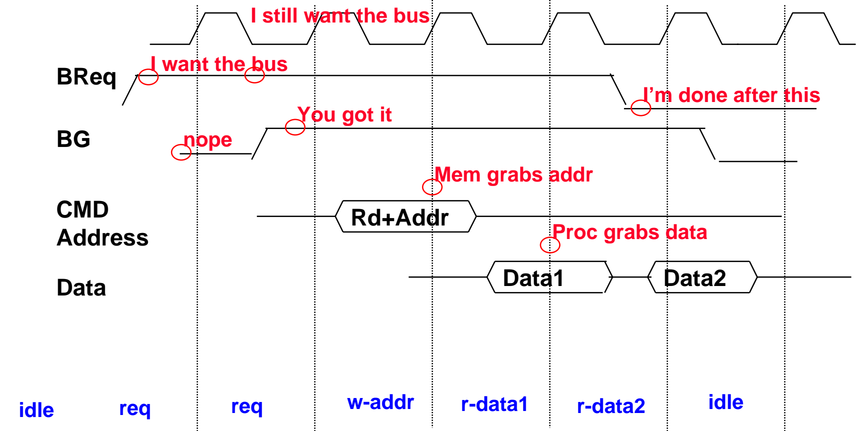
- Even memory busses are more complex than this
  - memory (slave) may take time to respond
  - it need to control data rate

Processor Side of Protocol - sketch



- Memory waits?
- Additional outputs?
- Memory side?

Simple Synchronous Protocol (cont)



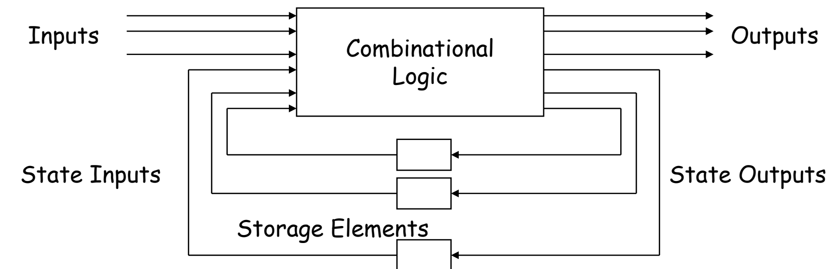
Announcements

- Reading 8.1-4 (slight change in ordering)
- HW 2 due tomorrow
- HW 3 will go out today
- Lab lecture on Verilog synthesis
- Next week feedback survey
- Input on discussion sections

- Technology in the News
  - iPhone "unlocked"
  - iPhone price drops by $200

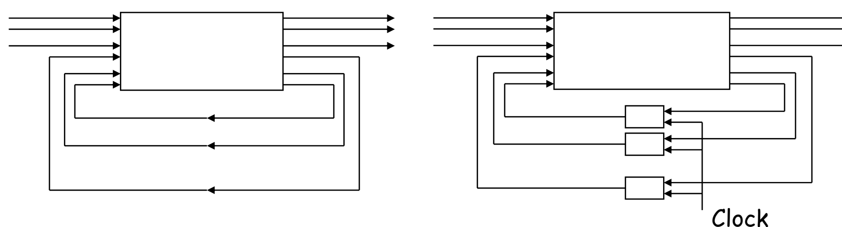
Fundamental Design Principle

- Divide circuit into combinational logic and state
- Localize feedback loops and make it easy to break cycles
- Implementation of storage elements leads to various forms of sequential logic

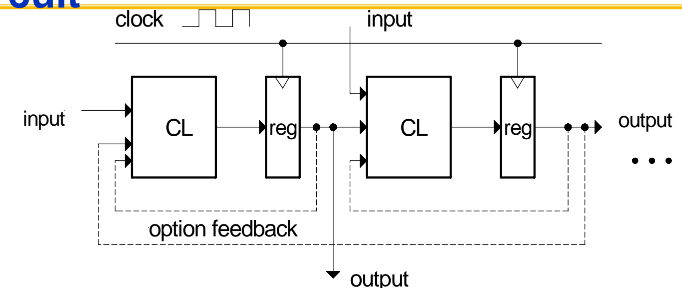


Forms of Sequential Logic

- Asynchronous sequential logic – "state" changes occur whenever state inputs change (elements may be simple wires or delay elements)
- Synchronous sequential logic – state changes occur in lock step across all storage elements (using a periodic waveform - the clock)



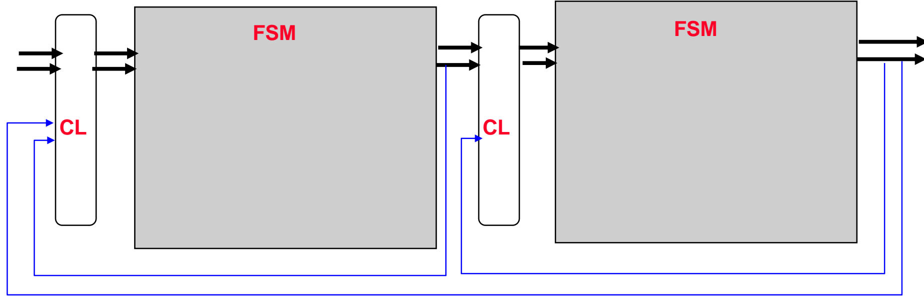
General Model of Synchronous Circuit



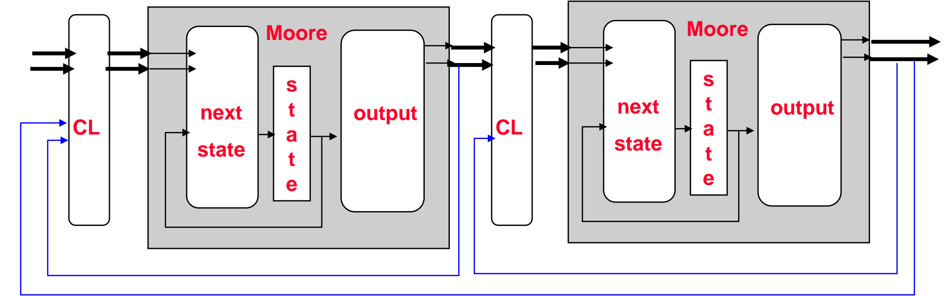
- All wires, except clock, may be multiple bits wide.
- Registers (reg)
  - collections of flip-flops
- clock
  - distributed to all flip-flops
  - typical rate?

- Combinational Logic Blocks (CL)
  - no internal state (no feedback)
  - output only a function of inputs
- Particular inputs/outputs are optional
- Optional Feedback
- ALL CYCLES GO THROUGH A REG!

Composing FSMs into larger designs

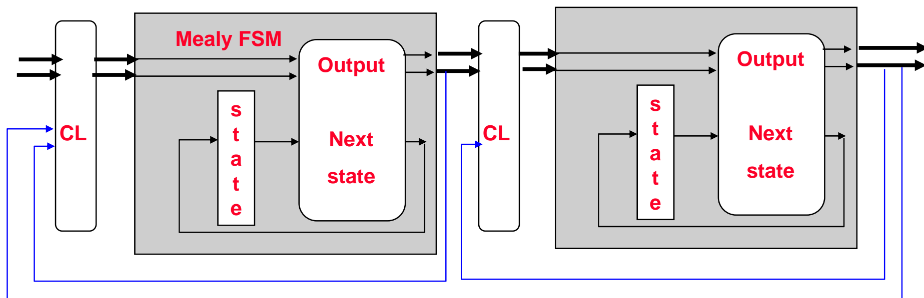


Composing Moore FSMs



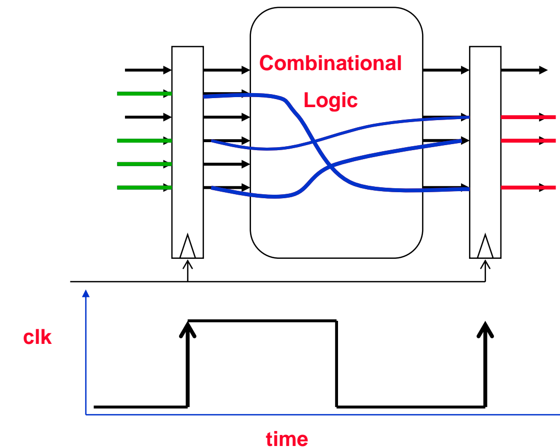
- Synchronous design methodology preserved

Composing Mealy FSMs …

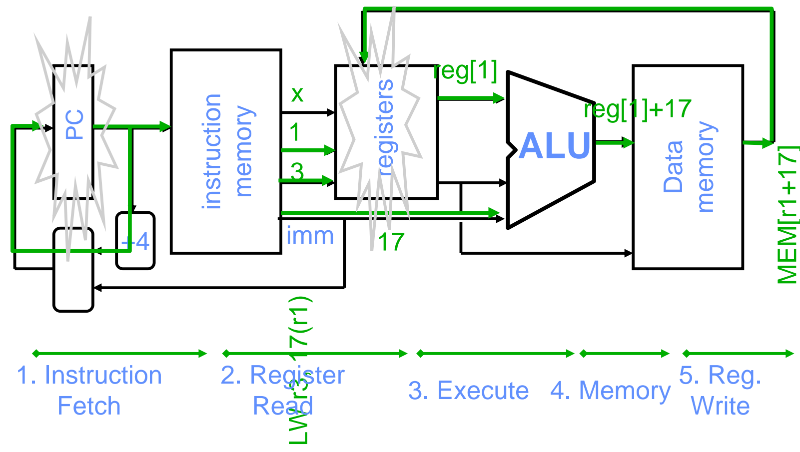


- Synchronous design methodology violated!!!
- Why do designers used them?
  - Few states, often more natural in isolation
  - Safe if latch all the outputs
    - » Looks like a mealy machine, but isn't really
    - » What happens to the timing?

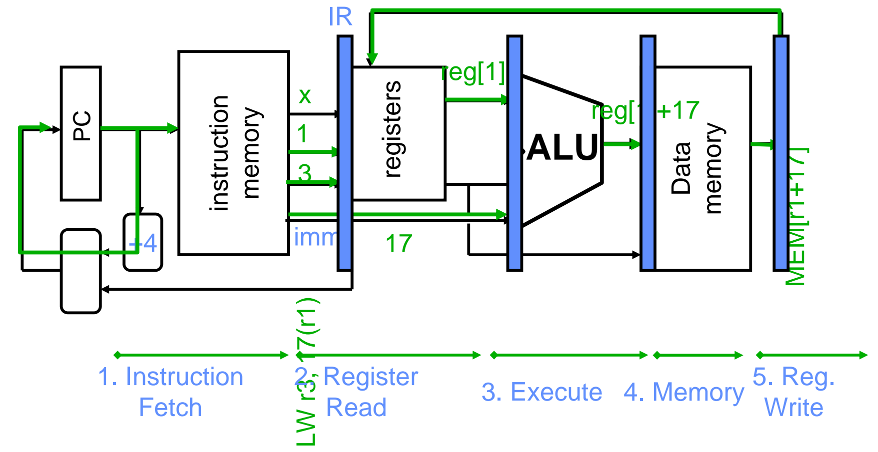
Recall: What makes Digital Systems tick?



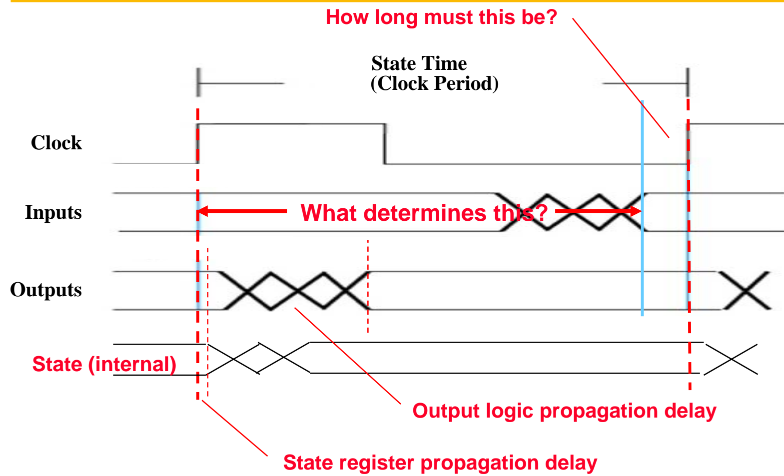
Recall 61C: Single-Cycle MIPS



Recall 61C: 5-cycle Datapath - pipeline

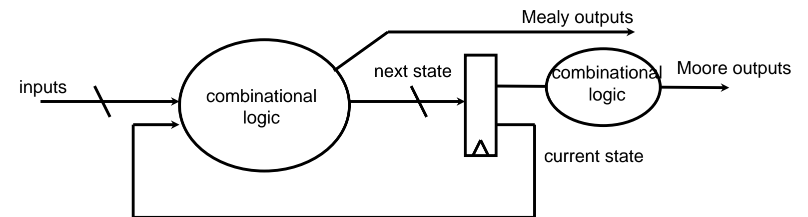


FSM timing



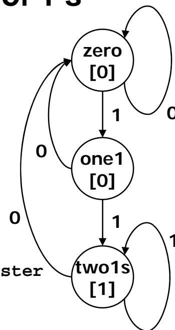
- **What determines min FSM cycle time (max clock rate)?**

Finite State Machines in Verilog



Verilog FSM - Reduce 1s example

- **Change the first 1 to 0 in each string of 1's**
  - **Example Moore machine implementation**



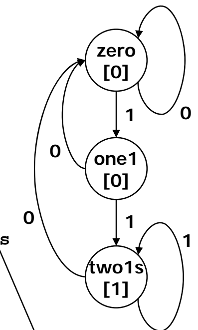

```
module Reduce(Out, Clock, Reset, In);
    output              Out;
    input               Clock, Reset, In;

    reg     Out;
    reg [1:0]           CurrentState;    // state register
    reg [1:0]           NextState;

    // State assignment
    localparam          STATE_Zero =     2'h0,
                        STATE_One1 =     2'h1,
                        STATE_Two1s =    2'h2,
                        STATE_X =        2'hX;
```


Moore Verilog FSM: combinational part

```
always @(In or CurrentState) begin
    NextState =                 CurrentState;
    Out =                       1'b0;
    case (CurrentState)
        STATE_Zero: begin // last input was a zero
            if (In) NextState =  STATE_One1;
        end
        STATE_One1: begin // we've seen one 1
            if (In) NextState =  STATE_Two1s;
            else NextState =     STATE_Zero;
        end
        STATE_Two1s: begin // we've seen at least 2 ones
            Out =                1;
            if (~In) NextState = STATE_Zero;
        end
        default: begin // in case we reach a bad state
            Out =                1'bx;
            NextState =          STATE_X;
        end
    endcase
end
```

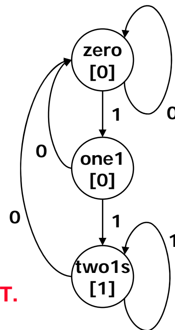


include all signals that are input to state and output equations

Compute: output = G(state)
next state = F(state, in)

Moore Verilog FSM: state part

```
// Implement the state register
always @ (posedge Clock) begin
    if (Reset) CurrentState <= STATE_Zero;
    else CurrentState <=       NextState;
end
endmodule
```



Note: posedge Clock requires NONBLOCKING ASSIGNMENT.

Blocking Assignment <-> Combinational Logic

Nonblocking Assignment <-> Sequential Logic (Registers)

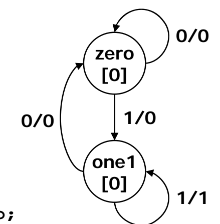
Mealy Verilog FSM for Reduce-1s example

```
module Reduce(Clock, Reset, In, Out);
    input    Clock, Reset, In;
    output   Out;
    reg      Out;
    reg      CurrentState;         // state register
    reg      NextState;

    localparam          STATE_Zero =       1'b0,
                        STATE_One =        1'b1;

    always @(posedge Clock) begin
        if (Reset) CurrentState <=         STATE_Zero;
        else CurrentState <=               NextState;
    end

    always @ (In or CurrentState) begin
        NextState =                        CurrentState;
        Out =                              1'b0;
        case (CurrentState)
            zero: if (In) NextState =      STATE_One;
            one: begin   // we've seen one 1
                      if (In) NextState = STATE_One;
                      else    NextState = STATE_Zero;
                      Out =              In;
            end
        endcase
    end
endmodule
```



Note: smaller state machine

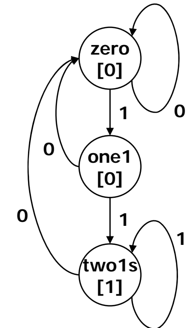
Output = G(state, input)

Restricted FSM Implementation Style

- **Mealy machine requires two always blocks**
  - Register needs posedge Clock block
  - Input to output needs combinational block
- **Moore machine can be done with one always block, but….**
  - E.g. simple counter
  - Very bad idea for general FSMs
    - » This will cost you hours of confusion, don't try it
    - » We will not accept labs with this style for general FSMs
  - Use two always blocks!
- **Moore outputs**
  - Share with state register, use suitable state encoding

Single-always Moore Machine (Not Allowed!)

```
module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;
  reg out;
  reg [1:0] state;        // state register
  parameter zero = 0, one1 = 1, two1s = 2;
```



Single-always Moore Machine (Not Allowed!)

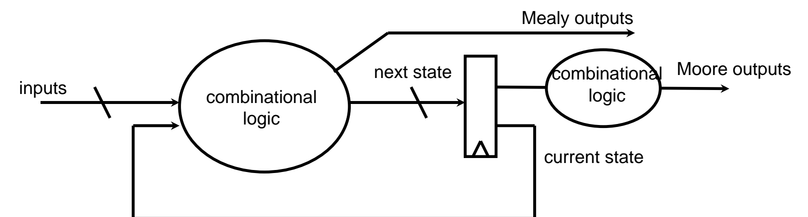
```
always @(posedge clk)
  case (state)
   zero: begin
        out <= 0;
        if (in) state <= one1;
        else    state <= zero;
      end
   one1:
      if (in) begin
          state <= two1s;
          out <= 1;
      end else begin
          state <= zero;
          out <= 0;
      end
   two1s:
      if (in) begin
         state <= two1s;
         out <= 1;
      end else begin
         state <= zero;
         out <= 0;
      end
    default: begin
        state <= zero;
        out <= 0;
      end
  endcase
endmodule
```

All outputs are registered

This is confusing: the output does not change until the *next* clock cycle

Finite State Machines

-



- **Recommended FSM Verilog implementation style**
  - Implement combinational logic using one always block
  - Implement an explicit state register using a second always block

Summary

- FSMs are critical tool in your design toolbox
  - Adapters, Protocols, Datapath Controllers, …
- They often interact with other FSMs
- Important to design each well and to make them work together well.
- Keep your verilog FSMs clean
  - Separate combinational part from state update
- Good state machine design is an iterative process