Problem 1 – Design a Verilog 16-bit adder module

```
module adder (A, B, sum);
  input [15:0]  A, B;
  output [15:0] sum;

  assign sum = A + B;
endmodule
```

Problem 1 – Design a Verilog 16-bit adder module

```
module adder (A, B, sum);
  input [15:0]  A, B;
  output [15:0] sum;
  reg     [15:0] sum;

  always @(A or B) begin
        sum = A + B;
  end
endmodule
```

Problem 2 – Design a Verilog 16-bit ALU

```
module alu (A, B, op, result);
  input [15:0]  A, B;       input [2:0] op;
  output [15:0] result;     reg     [15:0] result;
```

```
  always @(A or B or op) begin
        case (op)
        0:  result = A + B;
        . . .
        6:  result = B - A;
        default: result = 16'bX;
  end
endmodule
```

Problem 2 – Design a Verilog 16-bit ALU

```
module alu (A, B, op, result);
  input [15:0]  A, B;       input [2:0] op;
  output [15:0] result;
```

```
  assign result = (op==0)?(A+B):
                  . . .
                  (op==6)?(B-A): 16'bX;

endmodule
```

Problem 3 – Design a Verilog Register File (write)

```
module rfile (clk, reset AddrA, AddrB, AddrW, A, B, Din, write);
    reg [15:0] R0, R1, R2, R3;
```

```
 always @(posedge clk) begin
   case (AddrW)
         0:  R0 <= Din;
         1:  R1 <= Din;
         2:  R2 <= Din;
         3:  R3 <= Din;
   endcase
endmodule
```

Problem 3 – Design a Verilog Register File (read)

```
module rfile (clk, reset AddrA, AddrB, AddrW, A, B, Din, write);
    reg [15:0] R0, R1, R2, R3;
```

```
always @(AddrA) begin
   case (AddrA or R0 or R1 or R2 or R3)
         0:  A = R0;
         1:  A = R1;
         2:  A = R2;
         3:  A = R3;
   endcase
end
endmodule
```

+ The equivalent always block for B

Restricted FSM Implementation Style

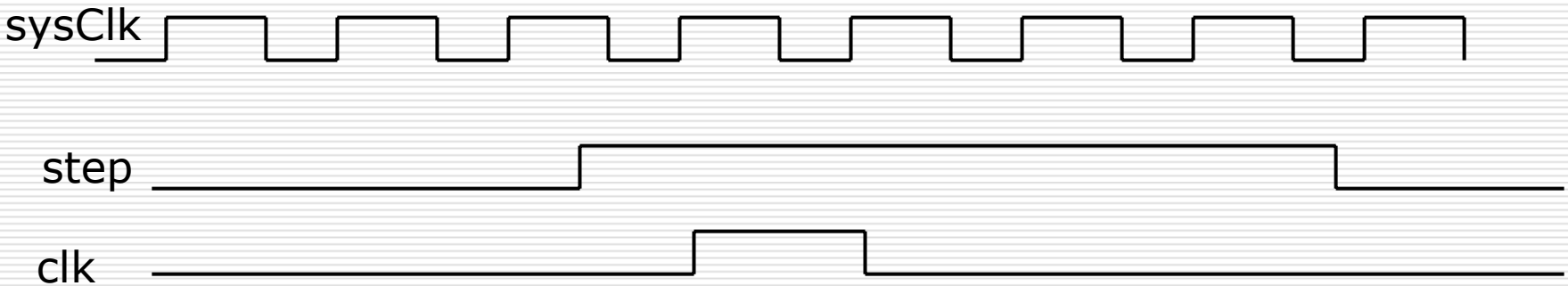
- Mealy machine requires two always blocks
  - register needs posedge CLK block
  - input to output needs combinational block
- Moore machine can be done with one always block
  - e.g. simple counter
  - ***Not a good idea for general FSMs***
    - Can be very confusing (see example)
- Moore outputs
  - Share with state register, use suitable state encoding

Problem 4 – Design a "Steppable" Clock

A clock generator:

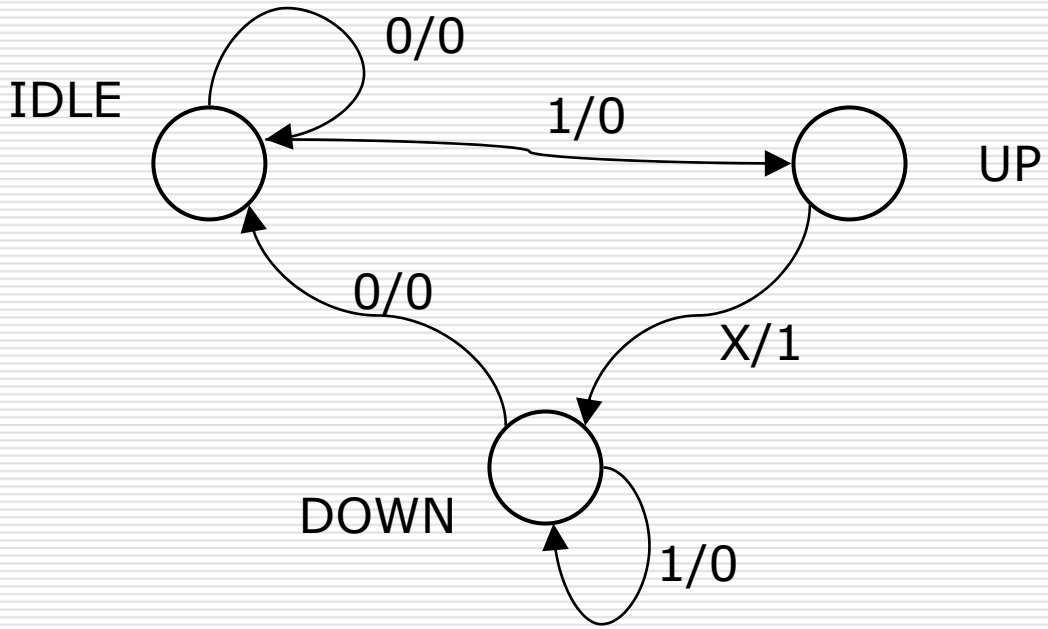
Mode 0 : clk is free-running, 1/2 sysClk frequency

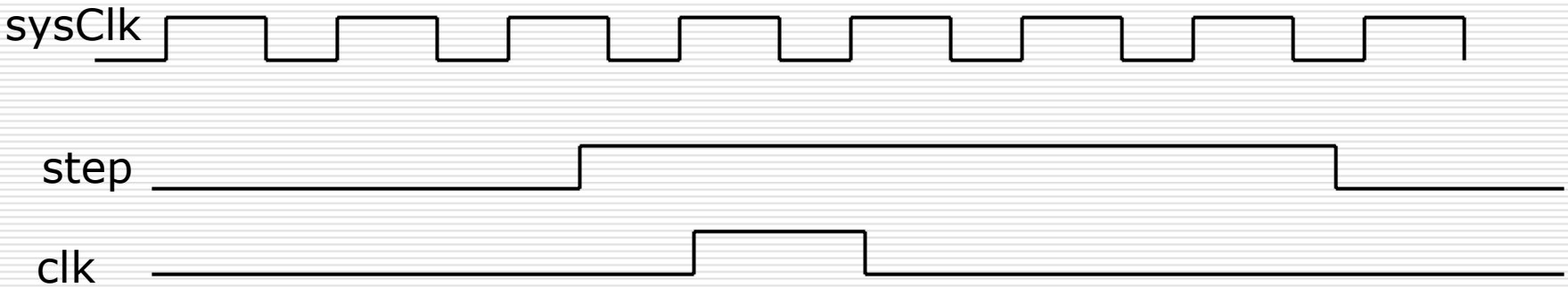
Mode 1 : clk is stopped, step causes one pulse on clk



Problem 4 – Design a "Steppable" Clock

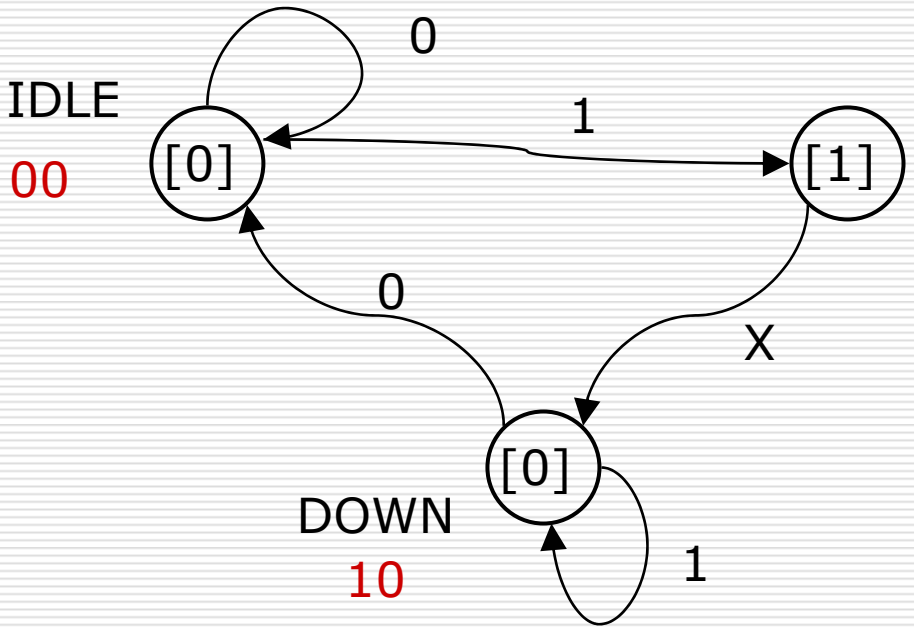
Draw a state diagram – start with Mode 1 (step mode)





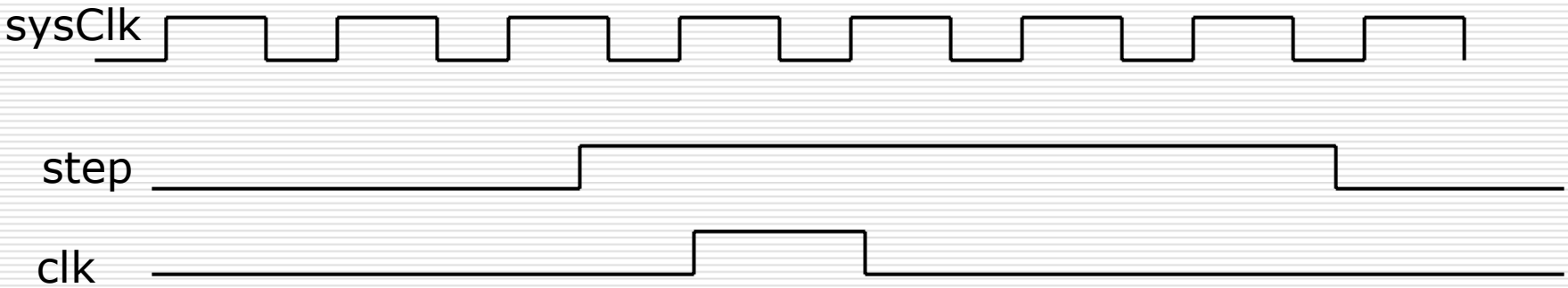
Problem 4 – Design a "Steppable" Clock

Draw a state diagram – start with Mode 1 (step mode)



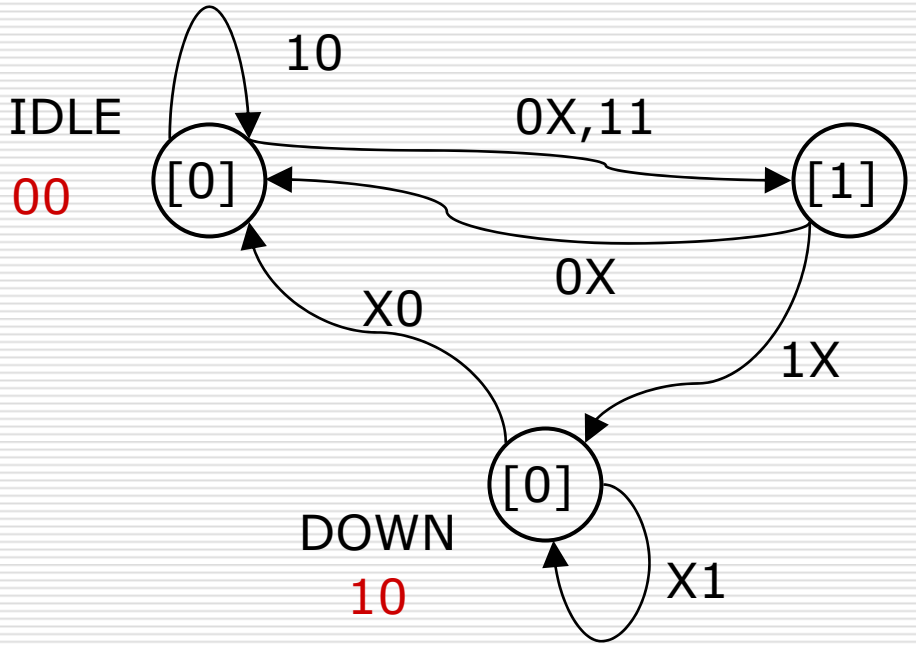
State Assignment?

Since clk is a Moore output, share it with a state bit



Problem 4 – Design a "Steppable" Clock

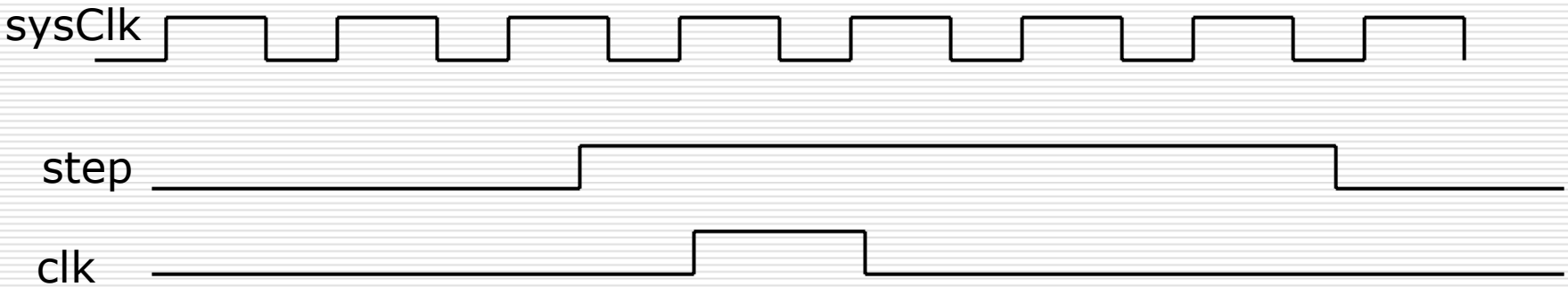
Draw a state diagram – start with Mode 1 (step mode)



inputs: mode,step

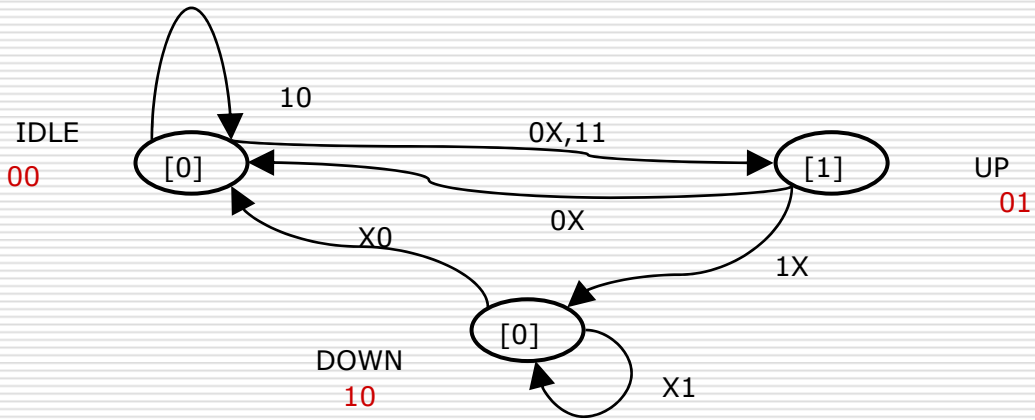
Now add the mode input

When mode = 0, clk is free-running



Verilog for "Steppable" Clock (state reg)

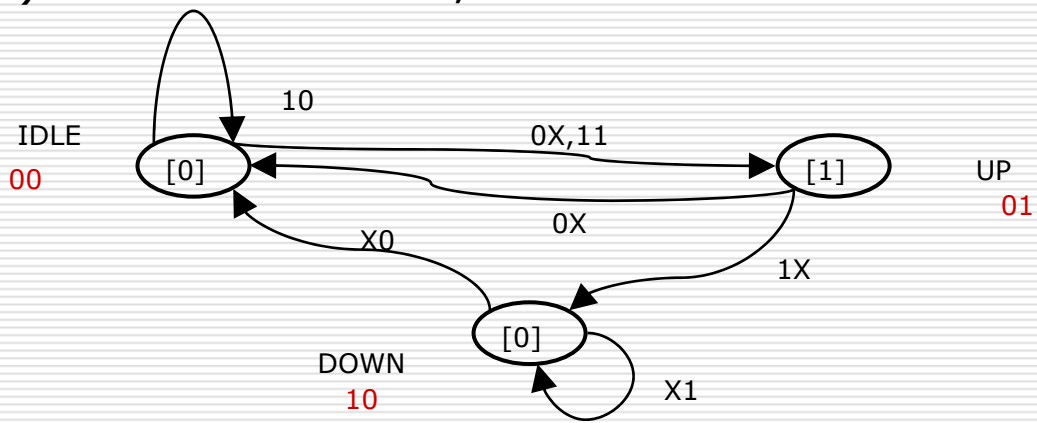
```
module stepClk (sysClk, mode, step, clk);
  input sysClk, mode, step;          output clk;

---

  parameter IDLE=0, UP=1, DOWN=2;          // Use names!
  reg [1:0]  state, nxtState;

  assign clk = state[0];          // clk output shared with state

  always @(posedge sysClk) begin
      state <= nxtState;
  end
```



Verilog for "Steppable" Clock (functions)

```
module stepClk (sysClk, mode, step, clk);
 input sysClk, mode, step;          output clk;
parameter IDLE=0, UP=1, DOWN=2;
```

```
 always @(state or mode or step) begin
   nxtState = state;              // Default (what if we leave out??)
   case (state)
        IDLE :  if (~mode | step) nxtState = UP;
        UP :    nxtState = DOWN;
        DOWN :  if (~mode) nxtState = UP;
                else if (~step) nxtState = IDLE;
   endcase
 end
```



Verilog for "Steppable" Clock (functions)

```
module stepClk (sysClk, mode, step, clk);
 input sysClk, mode, step;          output clk;
parameter IDLE=0, UP=1, DOWN=2;
```

```
 always @(state or mode or step) begin
   nxtState = state;              // Default (does this matter?)
   case (state)
        IDLE :  if (~mode | step) nxtState = UP;
        UP :    nxtState = DOWN;
        DOWN :  if (~mode) nxtState = UP;
                   else if (~step) nxtState = IDLE;
        default: nxtState = 2'bX;        // Does this matter?
   endcase
 end
```

Problem 5 : Data Switch

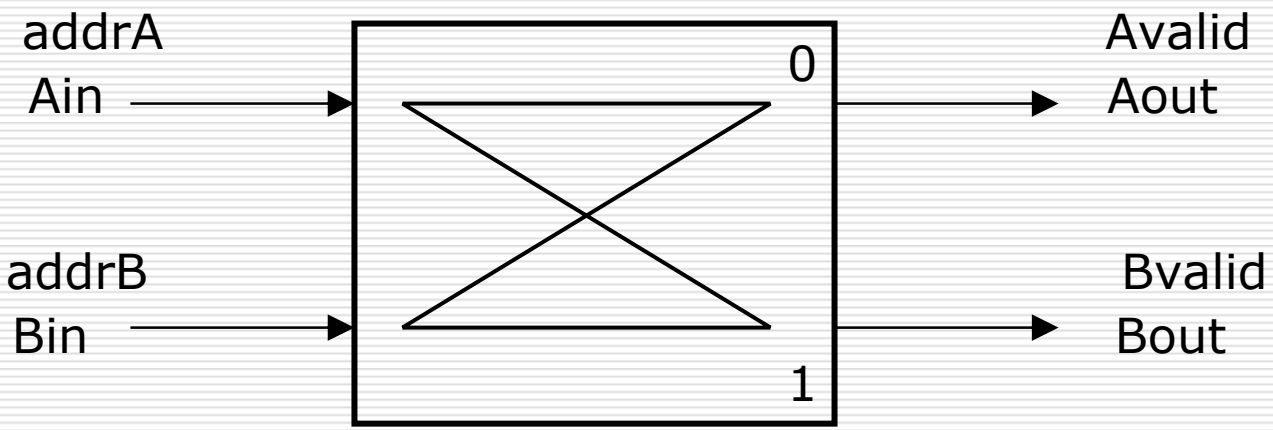
Two input data streams enter the switch, one item per clock

Each contains an address (addrA, addrB) which indicates which output port they want

The switch sends each to the desired output port

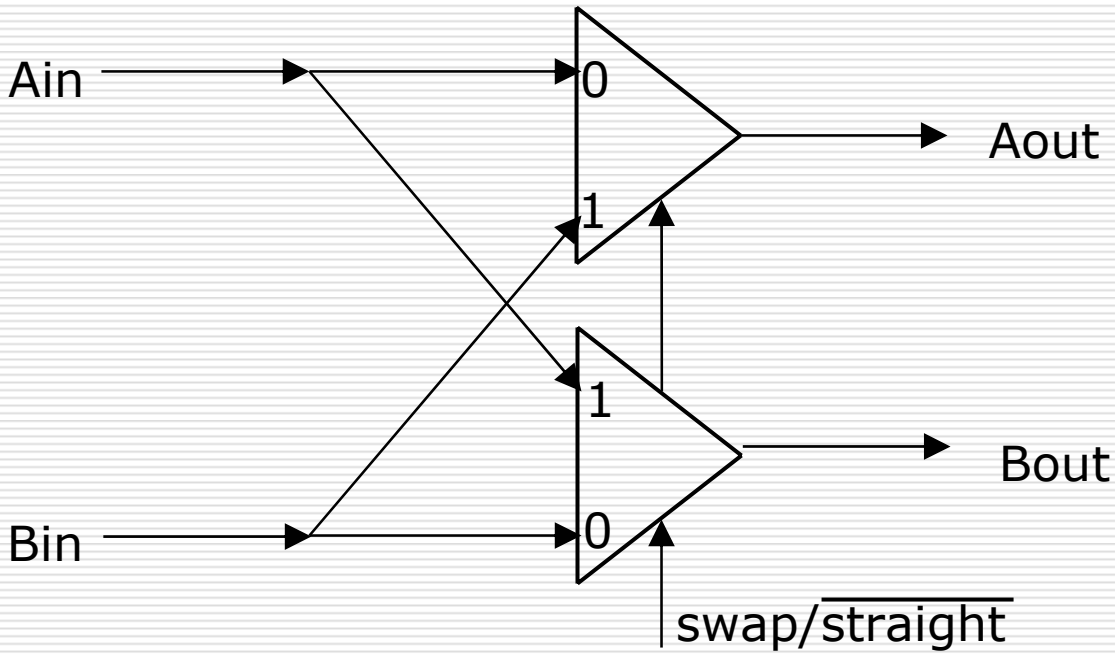
If they contend, then the switch treats them fairly

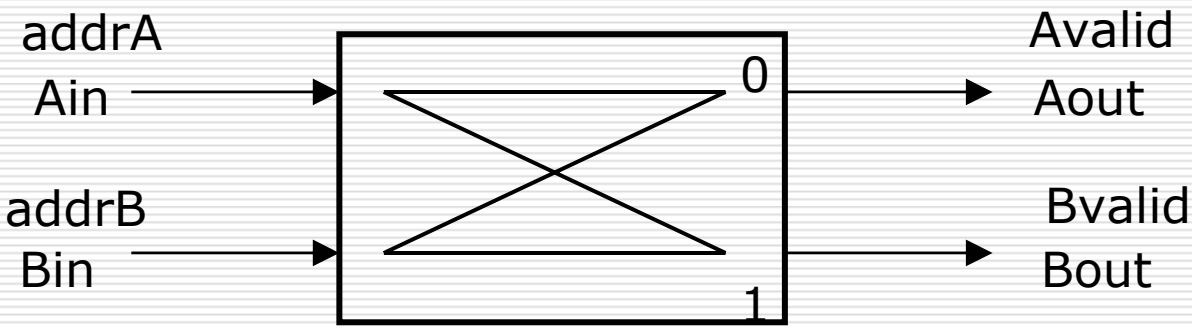
valid outputs indicate if there is a message for that output



Problem 5 : Data Switch

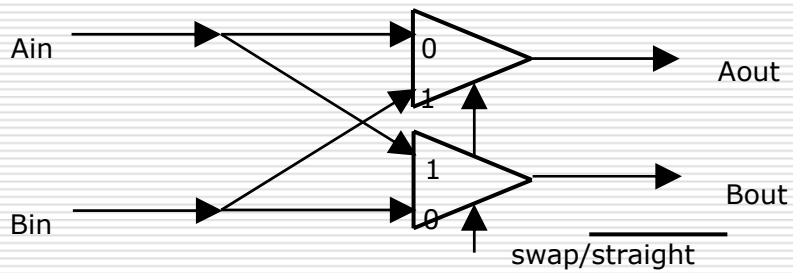
There is a control and data circuit.
Design the data circuit first.





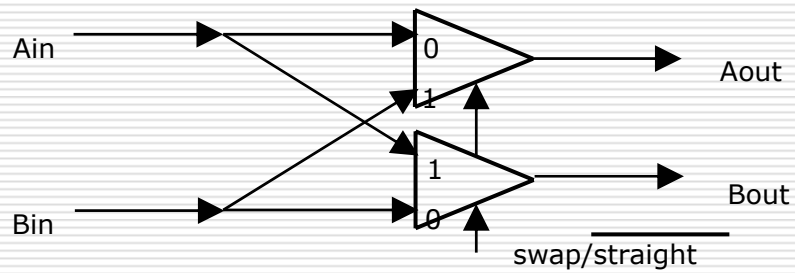
Problem 5 : Data Switch Verilog for data circuit:

```
input   [15:0] Ain, Bin;
output [15:0] Aout, Bout;
reg  swap;                    // Internal Control signal


assign Aout = swap ? Bin : Ain;

assign Bout = swap ? Ain : Bin;
```



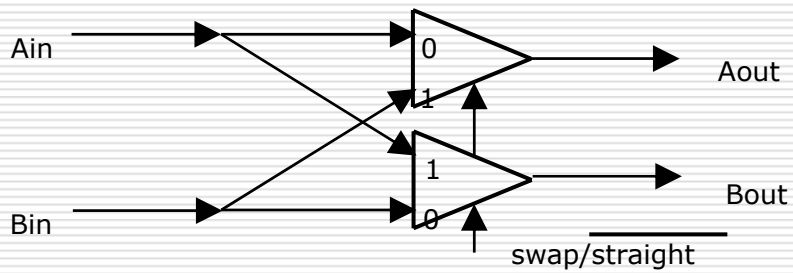
Problem 5 : Data Switch Verilog for data circuit:

```
input   [15:0] Ain, Bin;          // Add reg declaration
output [15:0] Aout, Bout;         // for always block!
reg  swap;                        // Internal Control signal


always @(Ain or Bin or swap) begin
   if (swap) begin
        Aout = Bin;  Bout = Ain;
   end else begin
        Aout = Ain;  Bout = Bin;
   end
end
```



Problem 5 : Data Switch Verilog for data circuit:

```
input   [15:0] Ain, Bin;
output [15:0] Aout, Bout;
reg  swap;                    // Internal Control signal


always @(Ain or Bin or swap) begin
   Aout = Ain;  Bout = Bin;    // Default
   if (swap) begin
        Aout = Bin;  Bout = Ain;
   end
end
```



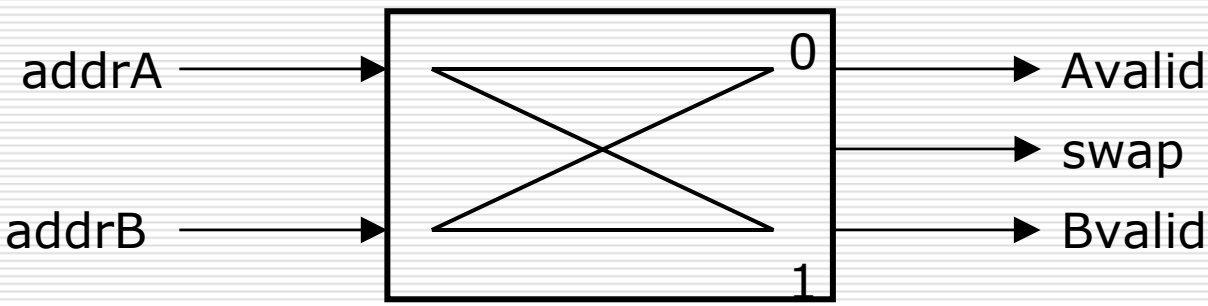
Problem 5 : Data Switch

AddrA,AddrB/swap

Now design the control

If we have addrA, addrB and swap, then we can compute Avalid and Bvalid:

```
assign Avalid = (addrA == swap);

assign Bvalid = (addrB != swap);
```

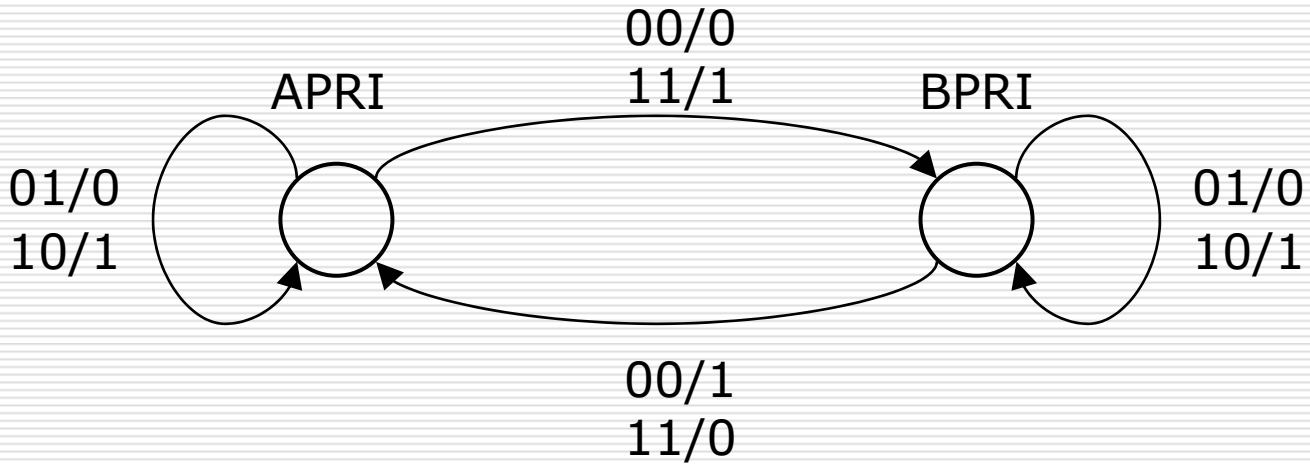


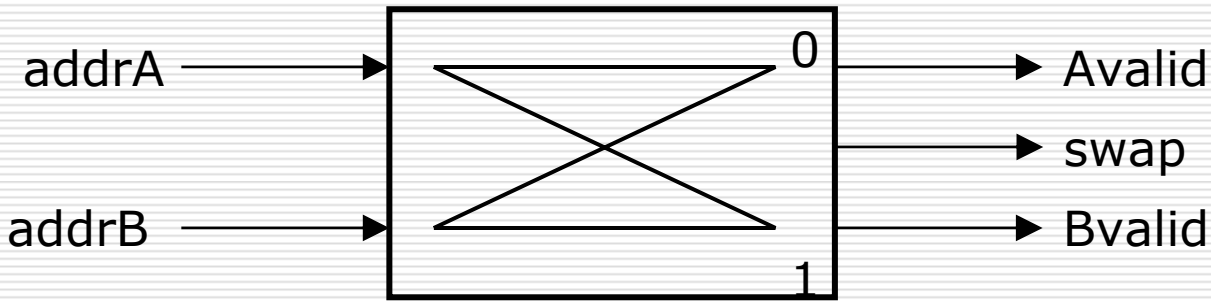
Problem 5 : Data Switch

Now design the control:

AddrA,AddrB/swap

Mealy! We forward the data ***this*** clock cycle





Problem 5 : Data Switch Verilog

```
module switch (clk, reset, Ain, addrA, Bin,
               addrB, Aout, Avalid, Bout, Bvalid);
  input clk, reset;
  input [15:0] Ain, Bin;
  input addrA, AddrB;
  output [15:0] Aout, Bout;
  output Avalid, Bvalid;
  reg  swap;                                  // Control signal
  assign Aout = swap ? Bin : Ain;
  assign Bout = swap ? Ain : Bin;
  assign Avalid = (addrA == swap);
  assign Bvalid = (addrB != swap);

  parameter APRI=0, BPRI=1;                   // State names
  reg  state, next_state;
  always @(posedge clk) begin
    if (reset) state <= APRI;
    else       state <= next_state;
  end

  always @(*) begin
    next_state = state;
    swap = 0;
    case (state)
      APRI: begin
        swap = addrA;
        if (addrA == addrB) nextState = BPRI;
      end
      BPRI: begin
        swap = ~addrB;
        if (addrA == addrB) nextState = APRI;
      end
    endcase
  end
endmodule
```