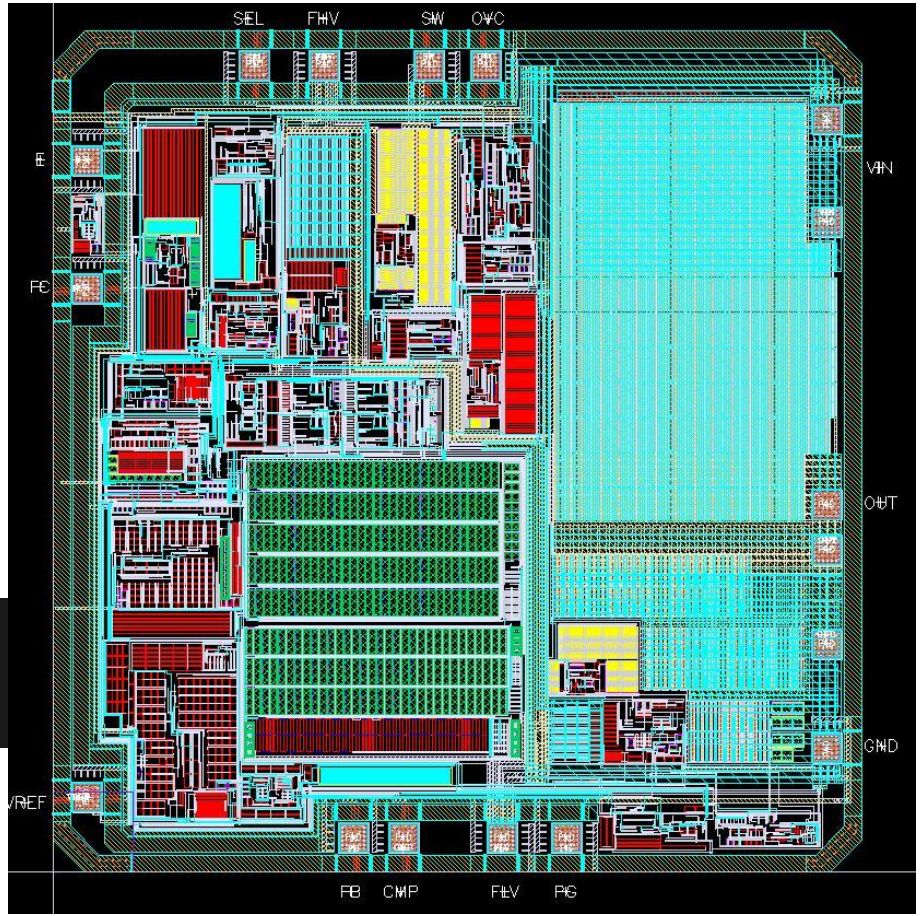
CHAPTER 12
CODING FOR SYNTHESIS

ECE4514

DIGITAL DESIGN 2



Topics

- Creating efficient decision trees
  - Trade-offs between priority and parallel structures
  - Dangers of multiple control branches
- Coding style traps
  - Usage of blocking and nonblocking assignments
  - Proper and improper usage of for-loops
  - Inference of combinatorial loops and latches
- Design partitioning and organization.
  - Organizing data path and control structures
  - Modular design
- Parameterizing a design for reuse

Reading

Nonblocking Assignments in Verilog Synthesis, Coding Styles That Kill!

and

Chapter 12

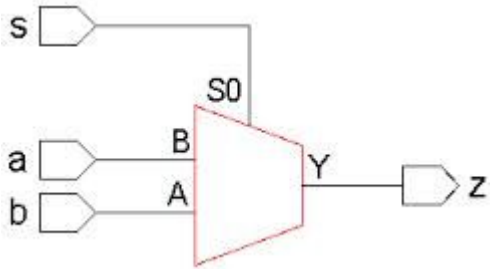
REQUIRED READING

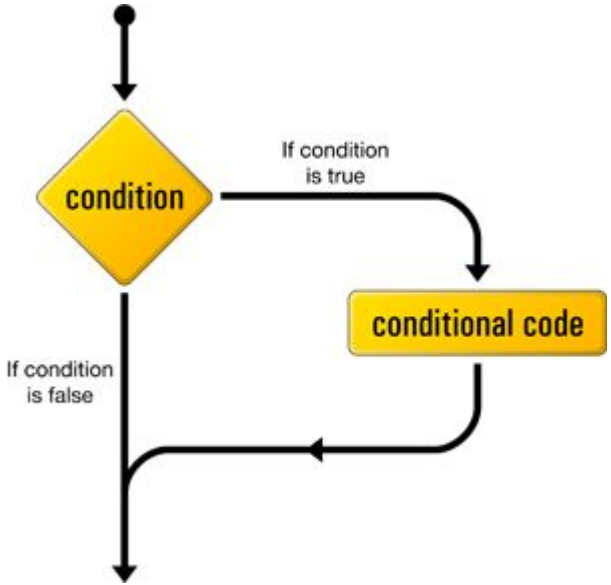
QUICK REVIEW

Decision Trees

REVIEW

IF

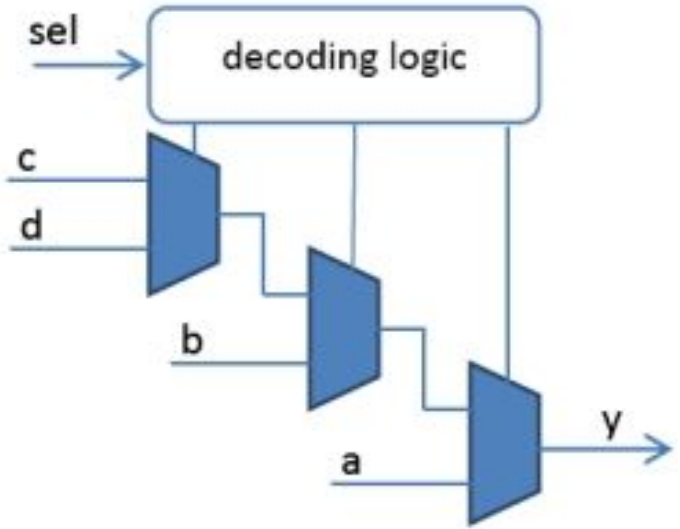




Decision Tree

REVIEW

```
always (sel or a or b or c or d)
  if (sel == 2'b00)
    y = a;
  else if (sel == 2'b01)
    y = b;
  else if (sel == 2'b10)
    y = c;
  else
    y = d;
```

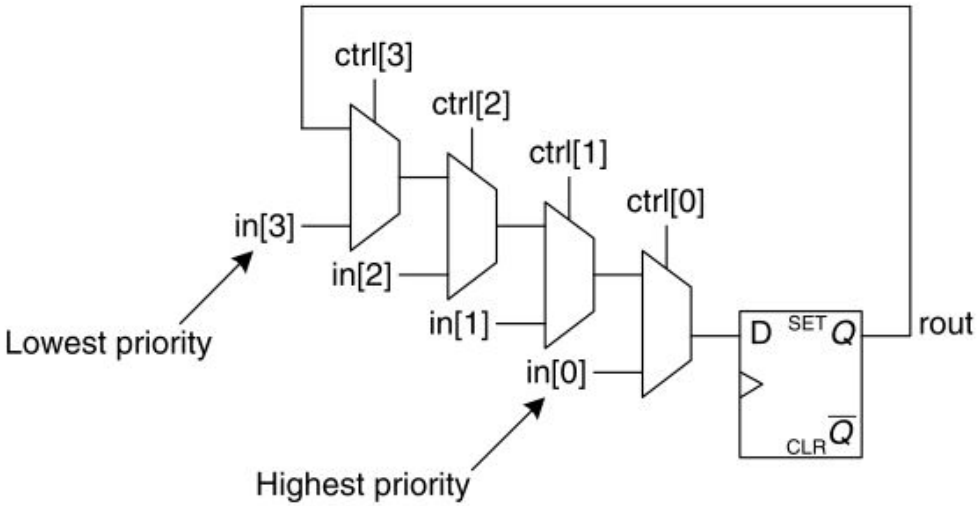


MUTUALLY EXCLUSIVE CLAUSES?

Decision Tree

REVIEW

```
module regwrite(
     output reg  rout,
     input       clk,
     input [3:0] in,
     input [3:0] ctrl);
always @(posedge clk)
     if(ctrl[0])      rout <= in[0];
     else if(ctrl[1]) rout <= in[1];
     else if(ctrl[2]) rout <= in[2];
     else if(ctrl[3]) rout <= in[3];
endmodule
```



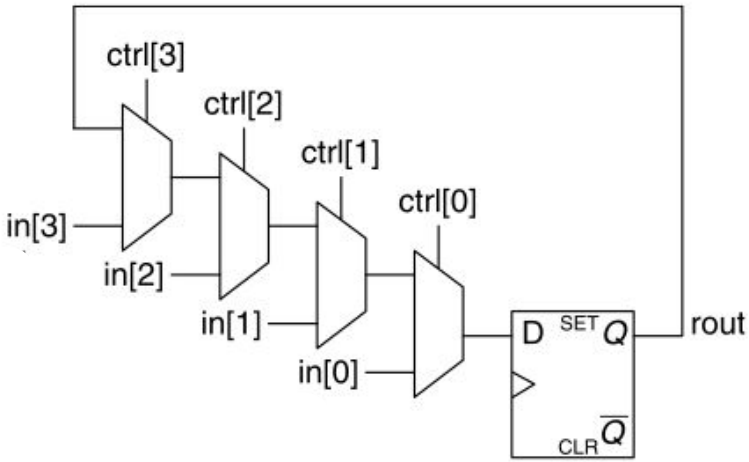
Decision Tree

REVIEW

Note that the propagation pathways are not equal for the *IN* and *CTRL* inputs.

What if we had more information on the arrival times of the *IN* inputs? Can we rewrite the code differently?

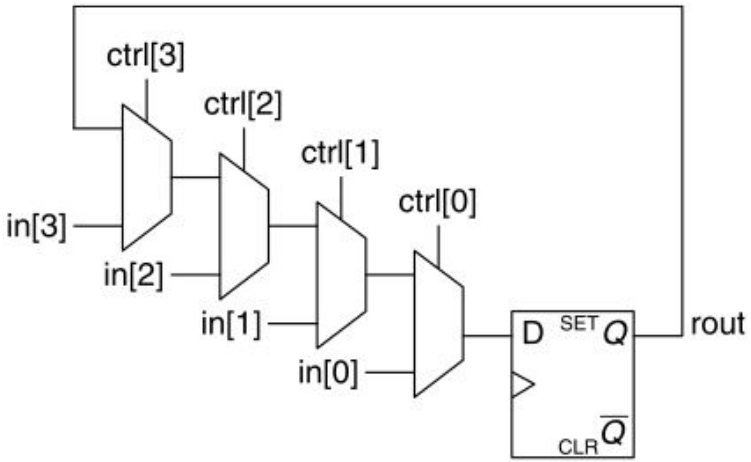
How about the *CTRL* inputs?



Decision Tree - Question

REVIEW

All CTRL signals arrive at T= 0 ps
Signal IN[0] arrives at T= 4 ps
Signal IN[1] arrives at T=13 ps
Signal IN[2] arrives at T= 8 ps
Signal IN[3] arrives at T= 8 ps



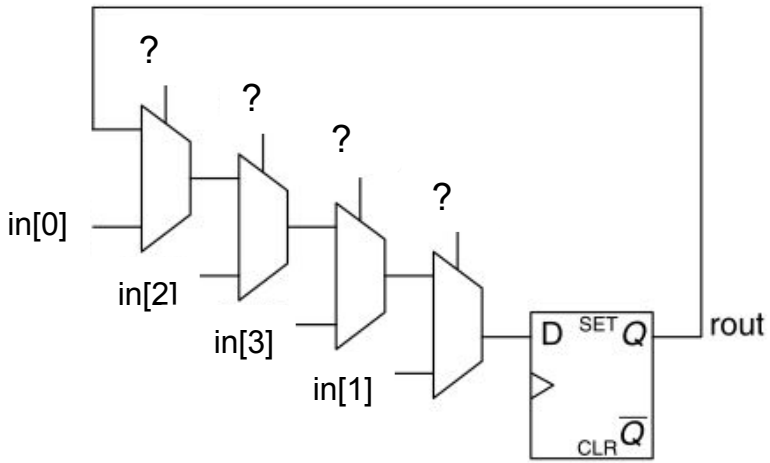
How would you redesign this to maximize the flip-flop set-up time?
Retain same functionality?

ORIGINAL

Decision Tree - Question

REVIEW

All CTRL signals arrive at T= 0 ps
Signal IN[0] arrives at T= 4 ps
Signal IN[1] arrives at T=13 ps
Signal IN[2] arrives at T= 8 ps
Signal IN[3] arrives at T= 8 ps



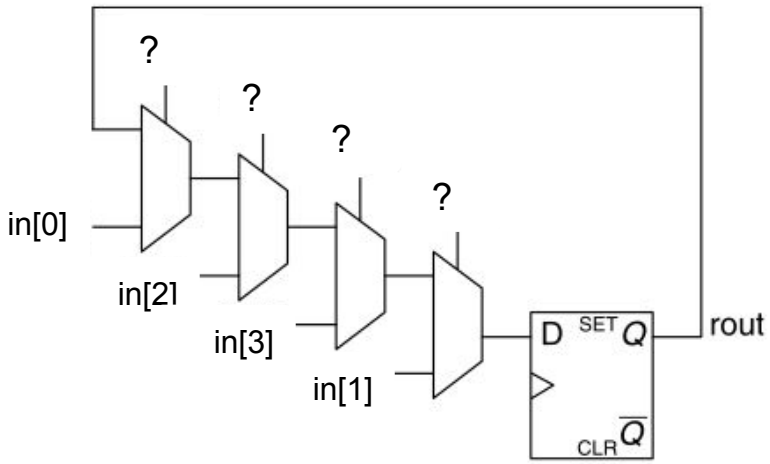
How would you redesign this to max the flip-flop set-up time?
Retain same functionality?

REVISED

Decision Tree -- Modified

REVIEW

```
module regwrite(
     ---------- );
always @(posedge clk)
     if(ctrl[1] & !ctrl[0])
          rout <= in[1];
     else if(ctrl[2]& !ctrl[0])
          rout <= in[2];
     else if(ctrl[3]& !ctrl[0])
          rout <= in[3];
     else if(ctrl[0])
          rout <= in[0];
endmodule
```

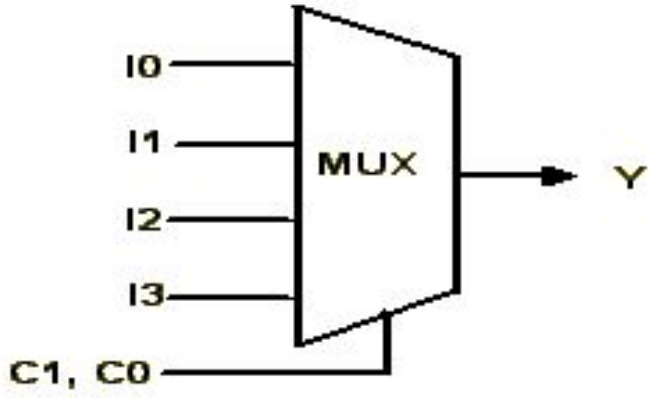


This is functionally identical, yet with different temporal properties.

Parallel Evaluation

REVIEW

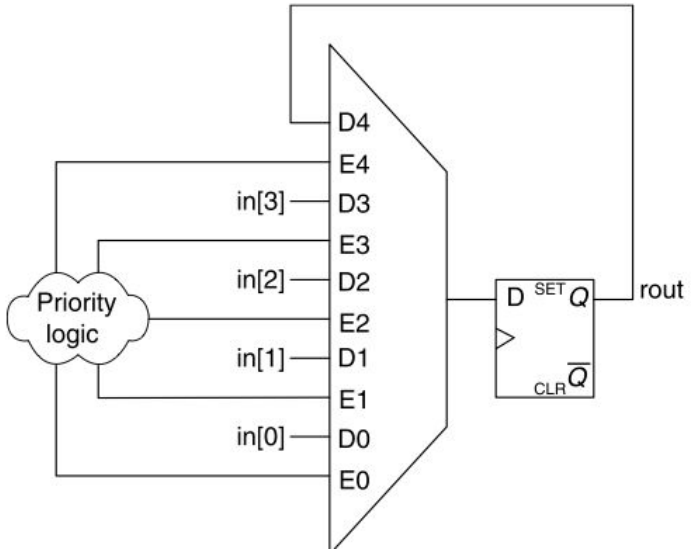
```
module
mux4_1_case(Y,I0,I1,I2,I3,C0,C1);
  input I0,I1,I2,I3,C0,C1;
  output Y;
  reg   Y;
  always@(I0 or I1 or I2 or I3 or
          C0 or C1) begin
    case ({C1,C0})
        2'b00 : Y = I0 ;
        2'b01 : Y = I1 ;
        2'b10 : Y = I2 ;
        2'b11 : Y = I3 ;
    endcase
  end
endmodule
```



All propagation pathways are the same length.
CASE items are mutually exclusive.
Decoder?
No *default* case?

Parallel Evaluation

```
case(1)
    ctrl[0]: rout <= in[0];
    ctrl[1]: rout <= in[1];
    ctrl[2]: rout <= in[2];
    ctrl[3]: rout <= in[3];
endcase
```



CASE items are **NOT** mutually exclusive
Priority encoding inferred

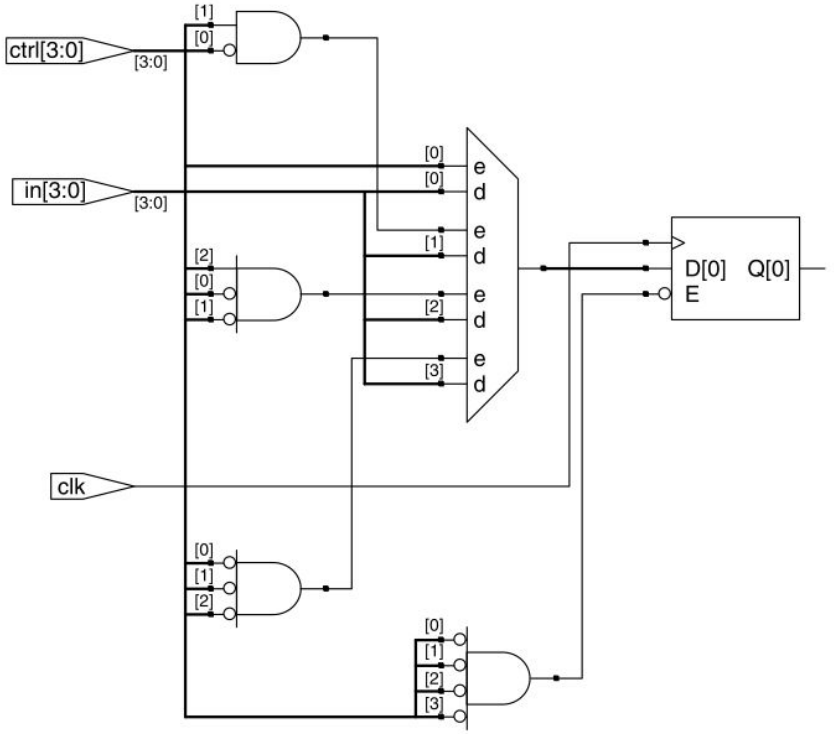
Parallel Evaluation

REVIEW

```
case(1)
    ctrl[0]: rout <= in[0];
    ctrl[1]: rout <= in[1];
    ctrl[2]: rout <= in[2];
    ctrl[3]: rout <= in[3];
endcase
```

CASE items are **NOT** mutually exclusive
Priority encoding inferred

Notice how lack of a default case is interpreted



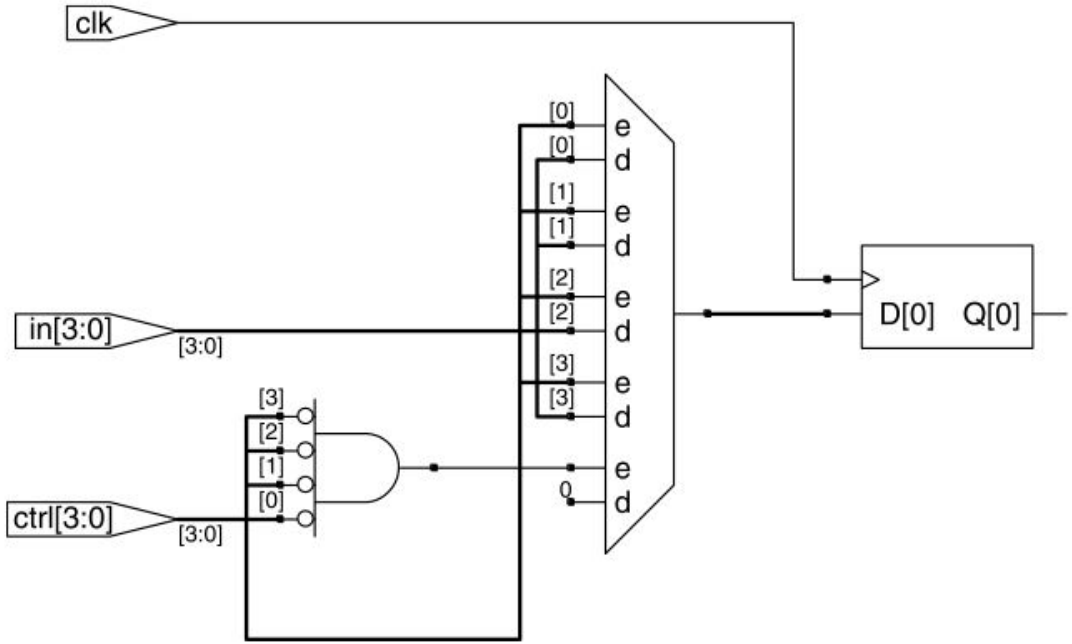
Parallel Evaluation

REVIEW

```
case(1)
    ctrl[0]: rout <= in[0];
    ctrl[1]: rout <= in[1];
    ctrl[2]: rout <= in[2];
    ctrl[3]: rout <= in[3];
    default: rout <= 0;
endcase
```

CASE items are **NOT** mutually exclusive, yet resolved with DEFAULT statement

No ‘holding’ of current state.



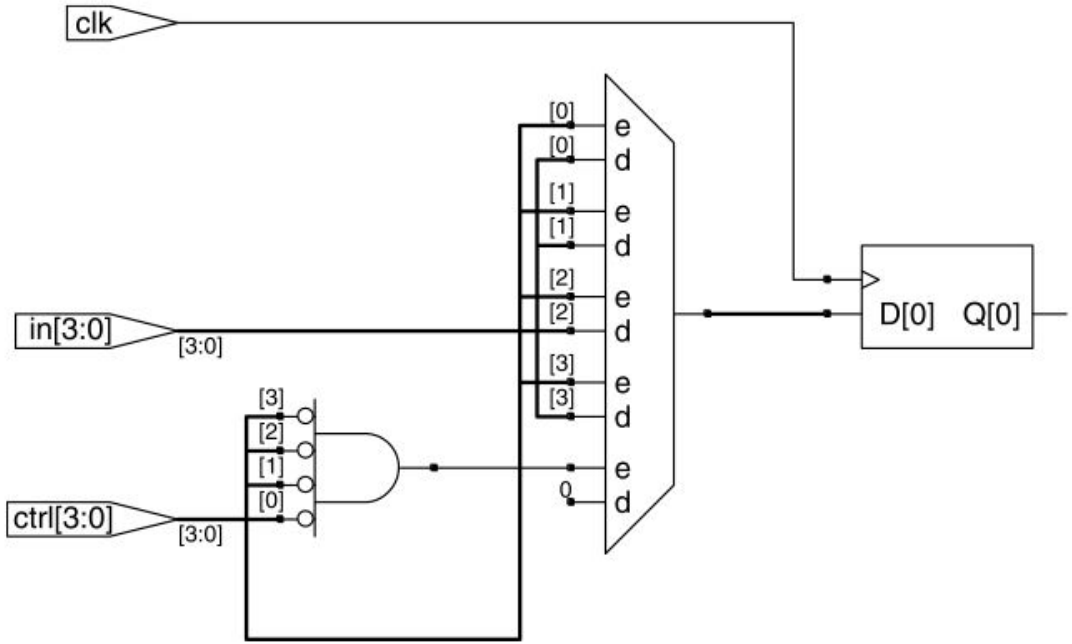
Parallel Evaluation

ALTERNATIVE EQUIVALENT

```
rout <= 0;
case(1)
    ctrl[0]: rout <= in[0];
    ctrl[1]: rout <= in[1];
    ctrl[2]: rout <= in[2];
    ctrl[3]: rout <= in[3];
endcase
```

CASE items are **NOT** mutually exclusive, yet resolved with DEFAULT statement

No ‘holding’ of current state.



CASE or IF/THEN/ELSE?

REVIEW

IF better if you know the timing relationships of inputs and control signals.

If you don’t know the timing relationships, or if you know all signals arrive at the same time, CASE may be better.

CASE or IF/THEN/ELSE?

REVIEW

If/else structures should be used when the decision tree has a priority encoding

Make sure CASE selectors are mutually exclusive and/or a default case is specified

Behavioral Modeling of Combinational Logic

Review ECE3544 rules:

`always@(`*<everything on the RHS of expressions, if conditionals, things read>*`) begin`

<all possible pathways must have at least one assigment to an ‘output’ reg>

<all assignments as blocking>

`end`

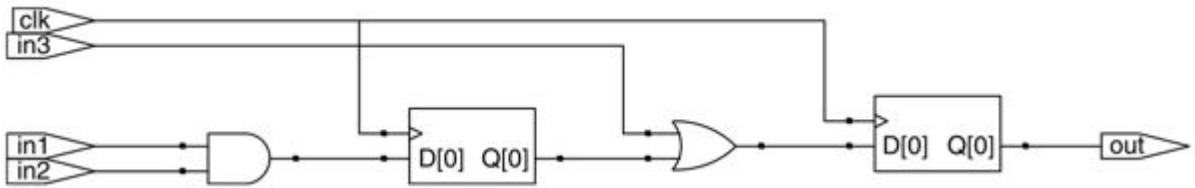
Behavioral Modeling of Sequential Logic

Review ECE3544 rules:

```
always@( posedge clk) begin
   if (reset) <do reset things>
   else begin
    <all assignments as non-blocking
      unless you know what you are doing>
   end
end
```

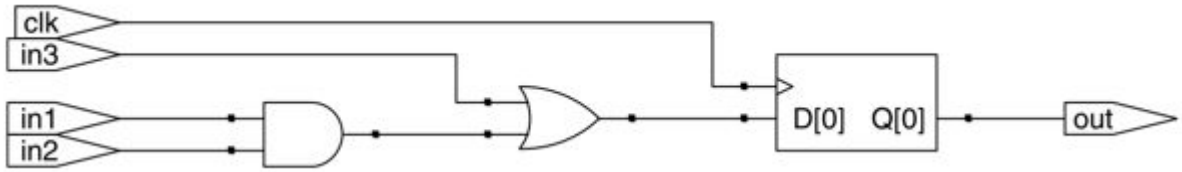
Good

```
module blockingnonblocking(
    output reg out,
    input      clk,
    input      in1, in2, in3);
    reg        logicfun;
  always @(posedge clk) begin
    logicfun <= in1 & in2;
    out      <= logicfun | in3;
  end
endmodule
```



Bad

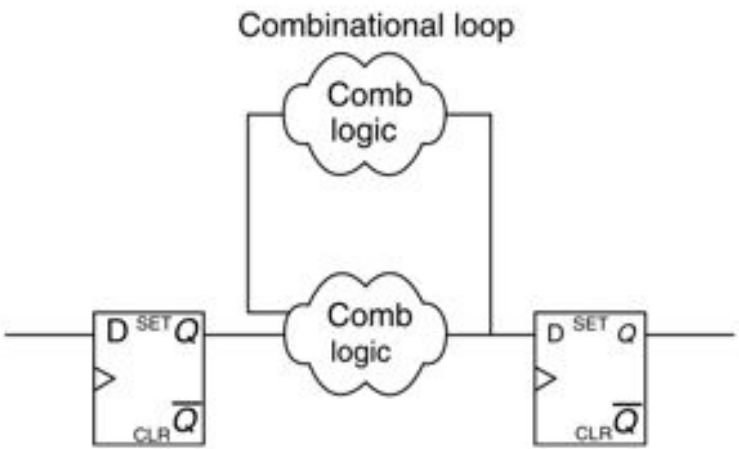
```
module blockingnonblocking(
    output reg out,
    input      clk,
    input      in1, in2, in3);
    reg        logicfun;
  always @(posedge clk) begin
    logicfun = in1 & in2;
    out      = logicfun | in3;
  end
endmodule
```



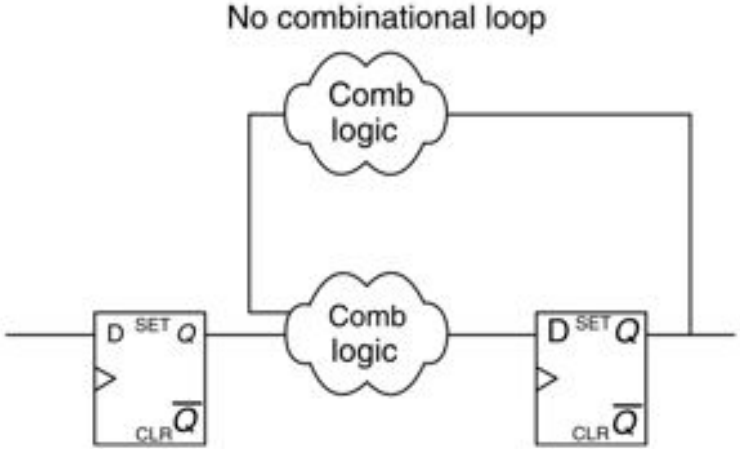
Compare Loops

```
reg [7:0] sum1=0, sum2=0;  // <Don't init like this
integer i, j;
always@(sum1) begin
   for (i=0; i<4; i=i+1) sum1 <= sum1 + 8'h1;
end

always@(sum2) begin
   for (j=0; j<4; j=j+1) sum2 = sum2 + 8'h1;
end
```

Combinational Loops

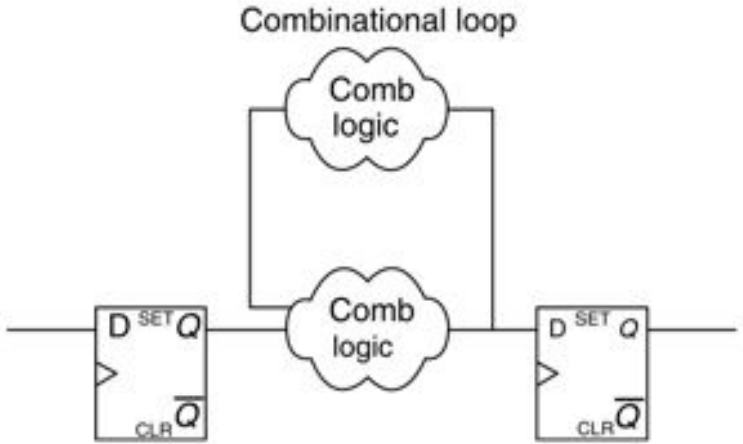


DANGEROUS



SAFE

Combinational Loops



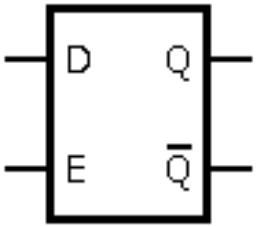
DANGEROUS

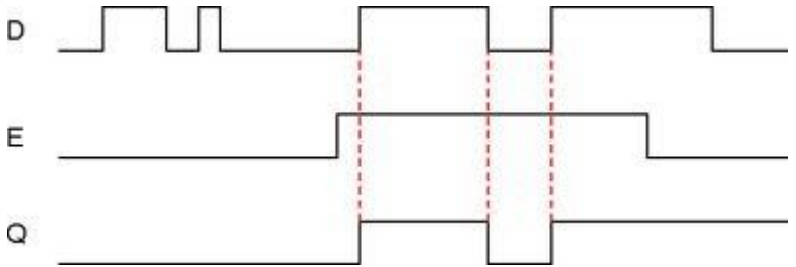
```
wire bad1, result;
assign bad1 = result ^ bad1;

wire bad2, a;
always @(a) b = bad2 ^ a;

wire bad3, bad4;
assign bad3 = bad4 | a;
assign bad4 = bad3 ^ b;
```

Inferred Latches

REVIEW

```
// latch inference
   module latch (
     input      iClk, iDat,
     output reg oDat);
     always @*
       if(iClk) oDat = iDat;
endmodule
```

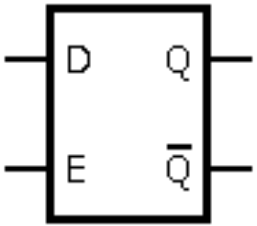


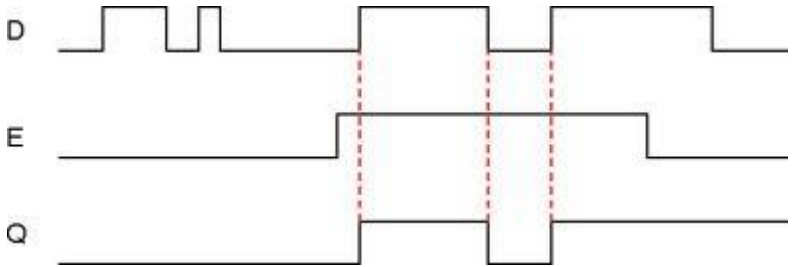


Inferred Latches

REVIEW

```
// BAD CODING STYLE
   assign O = C ? I: O;
```





Parameterization

```
`define CHIPID 8'hC9     // global chip ID
`define ONEMS 90000      // approximately 1ms with a 11ns clock
`define ULIMIT16 65535   // upper limit of an unsigned 16-bit word
```

Constants of global scope can be lumped all together in a “.vh” file.

At the top of all project files, add the include directive:

```
`include "my_global_constants.vh"
```

Style

Things that are not “parameters” and are not of global scope can be locally defined with DEFINE directives

```
// State Machine States
`define SM_IDLE       4'd1
`define SM_START      4'd2
`define SM_WAIT       4'd3
`define SM_DONE       4'd4
```

Parameters

True module parameters should be defined in the module heading.

```
module paramreg #(parameter WIDTH = 8) (
     output reg [WIDTH-1:0] rout,
     input                  clk,
     input      [WIDTH-1:0] rin,
     input                  rst);
  always @(posedge clk)
    if(!rst) rout <= 0;
    else     rout <= rin;
endmodule
```

Parameters

Overriding parameters can be done within the component instantiation

```
paramreg #(.WIDTH(22))
         r2(.clk(clk), .rin(rin), .rst(rst), .rout(rout));
```

or

```
defparam r2.WIDTH = 22;
paramreg r2(.clk(clk), .rin(rin), .rst(rst), .rout(rout));
```

Parameters

Use `localparam` to do math on parameter values:

```
// MIXED HEADER STYLE FOR LOCALPARAM
module multparam #(parameter WIDTH1 = 8, parameter WIDTH2 = 8)
                  (oDat, iDat1, iDat2);
    localparam             WIDTHOUT = WIDTH1 + WIDTH2;
    output [WIDTHOUT-1:0] oDat;
    input [WIDTH1-1:0]    iDat1;
    input [WIDTH2-1:0]    iDat2;
    assign oDat = iDat1 * iDat2;
  endmodule
```

Summary (1 of 2)

- If/else structures should be used when the decision tree has a priority encoding.
- It is good design practice to keep all register assignments inside one single control structure.
- Use blocking assignments to model combinational logic.
- Use nonblocking assignments to model sequential logic.
- Avoid mixing blocking and nonblocking assignments in one always block.

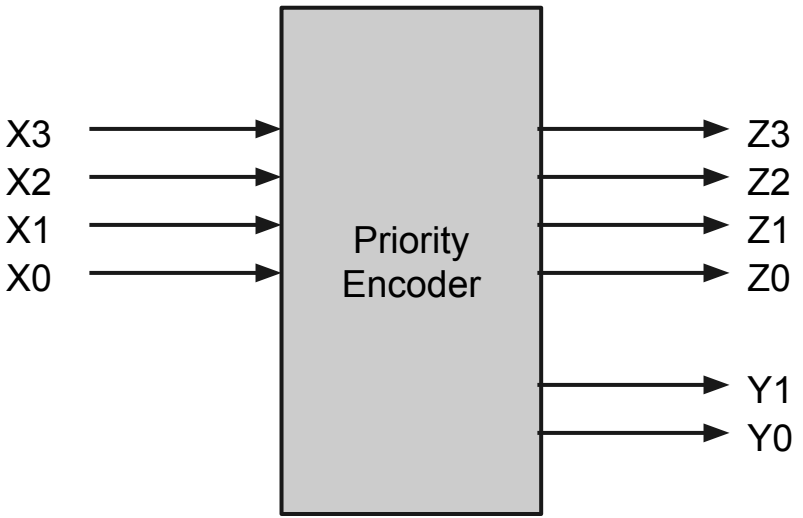
Summary (2 of 2)

- For-loops should not be used to implement software-like iterative algorithms.
- Data path and control blocks should be partitioned into different modules.
- It is good design practice to use only one clock and only one type of reset
- in each module.
- `define/ifdef` directives should be used for global definitions.
- Parameters should be used for local definitions that will change from module to module.
- Named parameter passing is superior to positional parameter passing or the defparam statement.

Priority Encoder

REVIEW

x_3	x_2	x_1	x_0	y_1	y_0	z_3	z_2	z_1	z_0
1	X	X	X	1	1	1	0	0	0
0	1	X	X	1	0	0	1	0	0
0	0	1	X	0	1	0	0	1	0
0	0	0	X	0	0	0	0	0	1



Data Flow vs. Control

DATAPATH or DATAFLOW:

- *Portion of the circuit that performs the logical / arithmetic operations on an input data point or stream.*
- *Typically the “pipe” that carries the data from the input of the design to the output and performs the necessary operations on the data.*

CONTROL STRUCTURE:

- *Responsible for producing the signals to steer or sequence the data through a circuit.*
- *FSM*