Design

- Draw data-flow diagrams
  - Algorithm → dataflow → datapath and control
    - *Then* instance Verilog modules into Xilinx schematics
- Draw state diagrams
  - And do the state encoding
    - One-hot
      - Simplifies combinational logic (reduces # of inputs)
      - More CLBs for state, but less for logic
    - Output based → use same bits for outputs and state
      - Don't need CLBs to decode output from state
- Modularize your designs

Synthesis

- Logic synthesis
  - Compiler generates gates from combinational logic
  - Optimizer minimizes logic
- Sequential synthesis
  - Compiler generates sequential logic
  - Optimizer may minimize states
- Register-transfer level (RTL) synthesis
  - Can share logic elements (e.g. adders) via data flow
    - You specify multiplexers to accomplish the share
- Behavioral or high-level synthesis
  - Tools figure out resource sharing (e.g. sharing adders)
  - Tools derive finite-state machines

Synthesis in CSE467

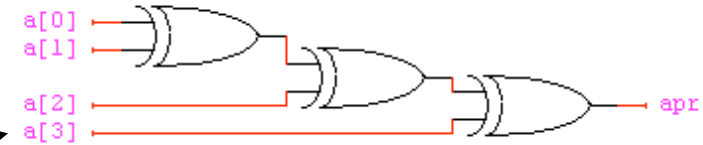
- SynplicityPro synthesizes modules to gates
  - Combinational logic
  - Sequential logic
- Xilinx ISE tools flatten and merge the gates
  - Minimizing combinational logic
  - Doesn't minimize states in encoded FSMs
- Xilinx ISE tools place and route the design
  - Assigning logic to CLBs
    - You get a timing report
    - You can apply constraints

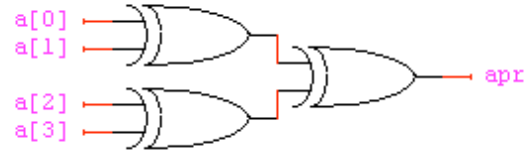
Logic synthesis

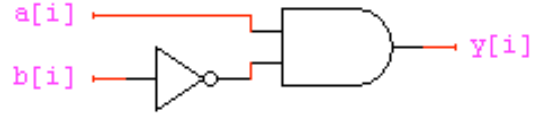
- Mappings
  - **`assign`** → Boolean equations
  - **`if`** and **`case`** → multiplexers
  - Arithmetic operators → adders, etc. from library
    - Parameterized to bit-width
- Optimizer
  - Looks for constants
  - Uses multilevel constructs to minimize logic

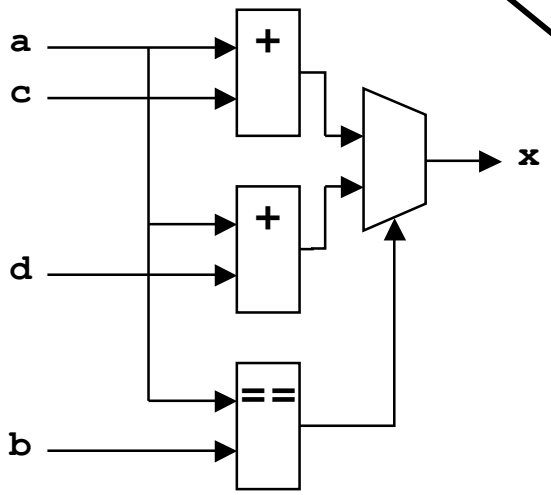
Synthesis examples

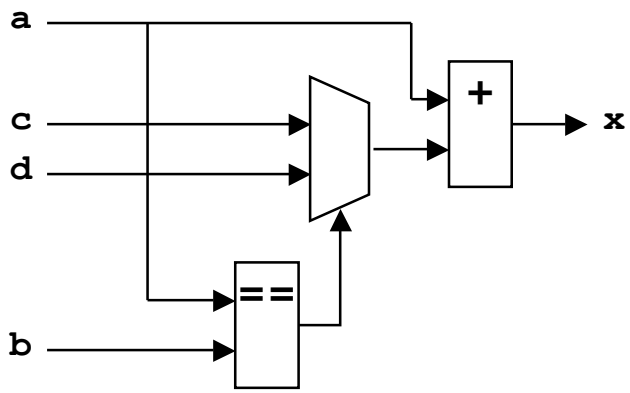
```
wire [3:0] x, y, a, b, c, d;

assign apr = ^a;
assign y = a & ~b;
assign x = (a == b) ?
           a + c : d + a;
```





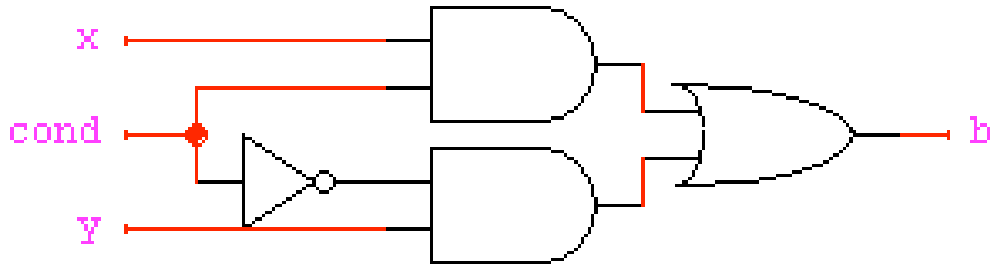






Conditional statements

```
case (cond)
  0 : b = y;
  1 : b = x;
endcase
```




```
if (cond) b = x; else b = y;
```

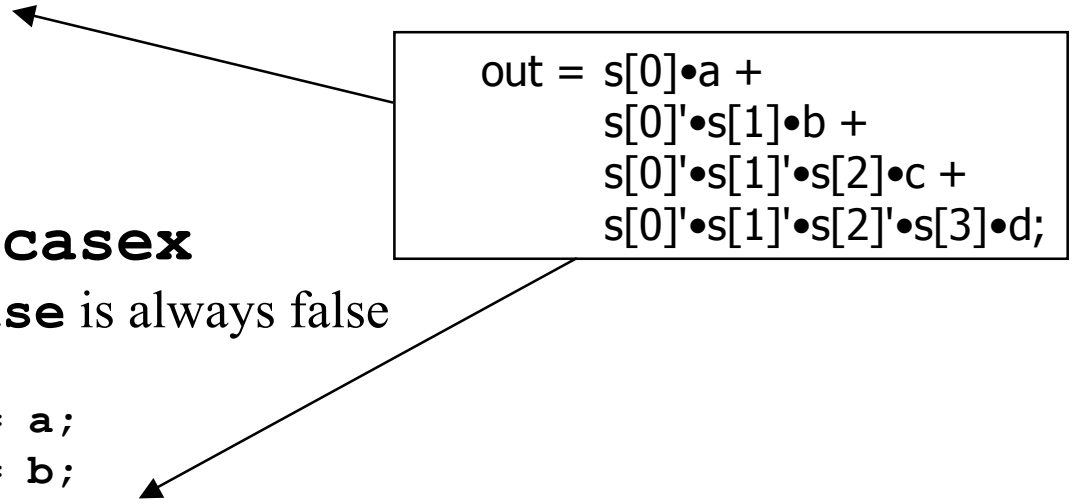
```
b = y;
if (cond) b = x;
```

Xilinx will synthesize this logic as a lookup table

`case` is a priority encoder

- Cases are evaluated sequentially

```
case (1'b1)
   s[0] : out = a;
   s[1] : out = b;
   s[2] : out = c;
   s[3] : out = d;
endcase
```



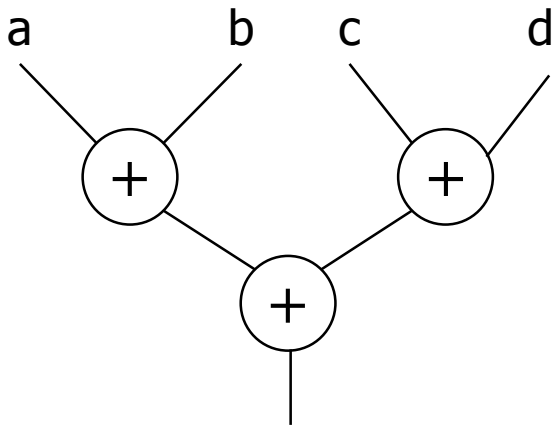
- Don't cares must use **`casex`**
  - Comparing to **`x`** in **`case`** is always false

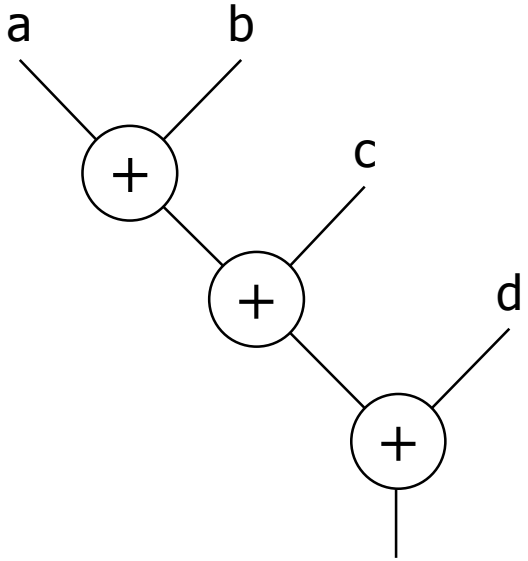
```
casex (s)
  4'b1xxx : out = a;
  4'b01xx : out = b;
  4'b001x : out = c;
  4'b0001 : out = d;
endcase
```

Arithmetic synthesis

- Parenthesis can indicate circuit structure
  - Synopsis will insert fast-carry adders
    - Won't minimize design after inserting adders

`(a + b) + (c + d)` versus `a + b + c + d`





Sequential synthesis and Xilinx

- Combinational logic is key to FPGA FSMs
  - Size of next-state functions
  - Critical path is combinational
- One-hot encoding
  - Minimizes # of inputs into next-state functions
  - Uses more registers
    - But simpler, faster combinational logic
- Output-based encoding
  - Register outputs are FSM outputs
    - No need for combinational logic after registers
    - Saves CLBs

Bit widths in arithmetic expressions

- Bit widths set by declarations
  - Intermediate values are inferred

```
reg [3:0] a, b, temp;
reg [5:0] c, z;

z = (a + b) + c; // intermediate value is 6-bit truncated

temp = a + b;    // result is 4-bit truncated
z = temp + c;    // result is 6-bit truncated
```

Memories

- Cannot directly reference a single bit

```
reg [7:0] byte;
reg [7:0] memory [0:255];

...
...
...
always...
  byte = memory[23];
  bit = byte[3];
end
```

Memory width

Memory depth

Tristate buffers

- *z* is a tristated value
- Example: Tristate driving BusOut

```
module tstate (EnA, EnB, BusA, BusB, BusOut);
  input EnA, EnB;
  input  [7:0] BusA, BusB;
  output [7:0] BusOut;

  assign BusOut = EnA ? BusA : 8'bZ;
  assign BusOut = EnB ? BusB : 8'bZ;
endmodule
```

Verilog: What we skipped...

- Simulation
  - from Verilog directly
  - with delays
- RTL synthesis
  - Functional logic specifies operations
  - Explicit resource sharing
- Behavioral synthesis
  - All logic inferred by compiler
  - Derived FSMs
    - Often more logic and/or slower than necessary