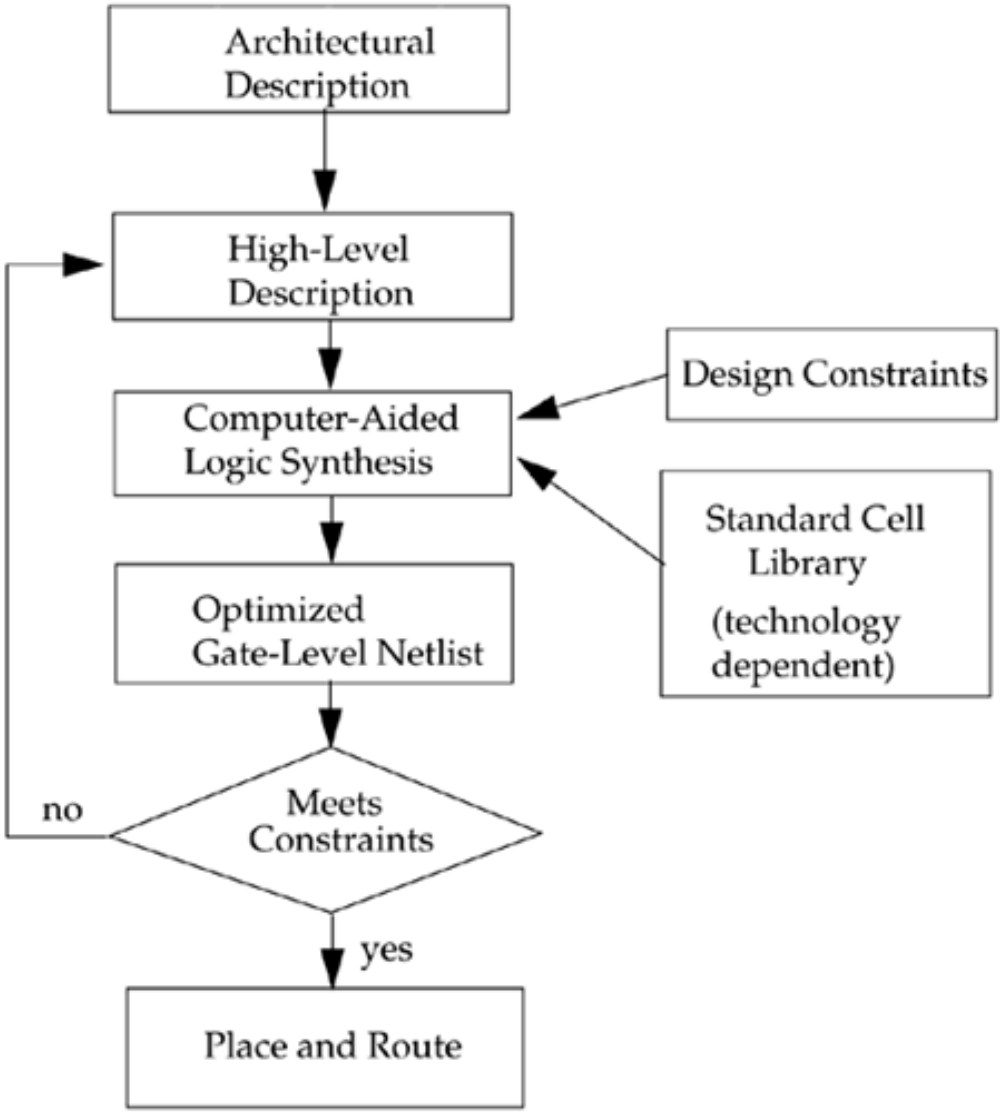
Logic Synthesis with Verilog HDL

Logic Synthesis

- *Logic synthesis* is the process of converting a high-level description of the design into an optimized gate-level representation, given a standard cell library and certain design constraints.

- A standard cell library or technology library can have simple cells, such as basic logic gates like **`and`**, **`or`**, and **`nor`**, or macro cells, such as adders, muxes, and special flipflops.

Computer-Aided Logic Synthesis



Verilog Constructs

Construct Type	Keyword or Description	Notes
ports	input, inout, output	
parameters	parameter	
module definition	module	
signals and variables	wire, reg, tri	Vectors are allowed
instantiation	module instances, primitive gate instances	E.g., mymux m1(out, i0, i1, s); E.g., nand (out, a, b);
functions and tasks	function, task	Timing constructs ignored
procedural	always, if, then, else, case, casex, casez	initial is not supported
procedural blocks	begin, end, named blocks, disable	Disabling of named blocks allowed
data flow	assign	Delay information is ignored
loops	for, while, forever,	while and forever loops must contain @(posedge clk) or @(negedge clk)

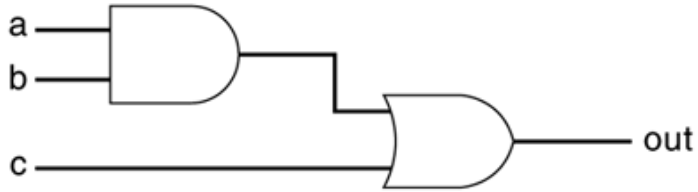
Verilog Operators

- All operators in Verilog are allowed for logic synthesis
- Only operators such as === and !== that are related to x and z are not allowed, as equality with x and z does not have much meaning in logic synthesis
- While writing expressions, it is recommended that you use parentheses to group logic

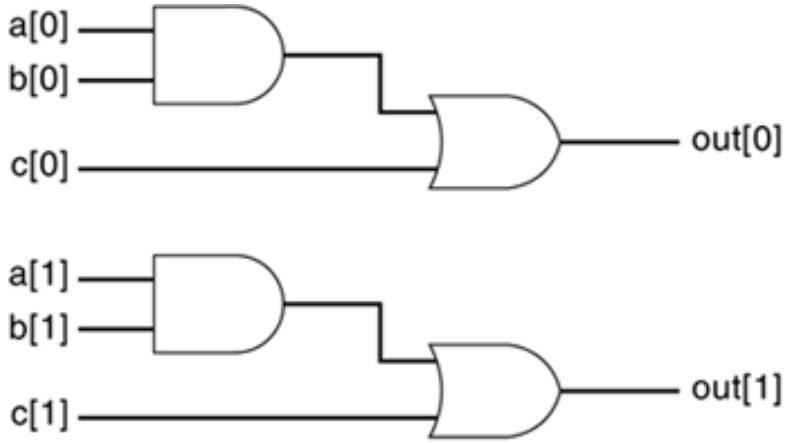
Interpretation of Verilog Constructs

`assign` statement

```
assign out = (a & b) | c;
```

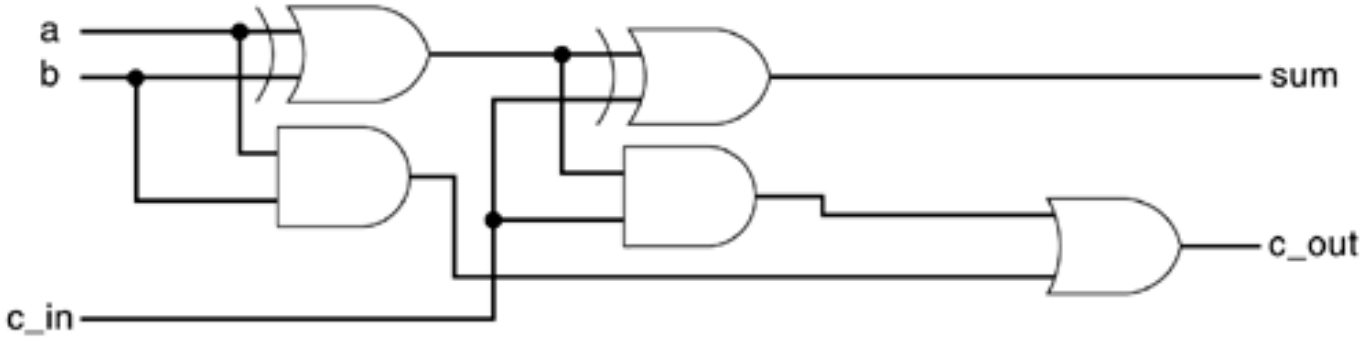


- If a, b, c, and out are 2-bit vectors [1:0], then the above assign statement will frequently translate to two identical circuits for each bit.



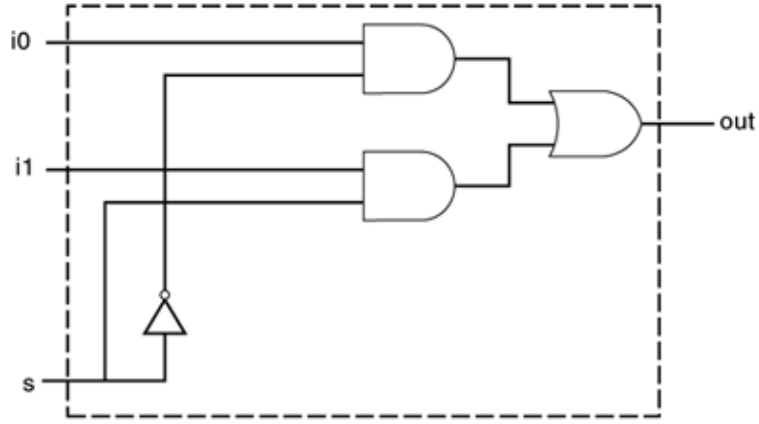
- If arithmetic operators are used, each arithmetic operator is implemented in terms of arithmetic hardware blocks available to the logic synthesis tool

```
assign {c_out, sum} = a + b + c_in;
```



- If a conditional operator ? is used, a multiplexer circuit is inferred.

```
assign out = (s) ? i1 : i0;
```



if-else statement

- Single if-else statements translate to multiplexers where the control signal is the signal or variable in the if clause.

```
if(s)
   out = i1;
else
   out = i0;
```

Case statement

- case statement also infer multiplexers

```
case (s)
     1'b0 : out = i0;
     1'b1 : out = i1;
endcase
```

for loops

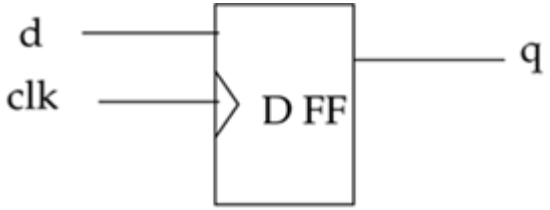
- used to build cascaded combinational logic

```
c = c_in;
for(i=0; i <=7; i = i + 1)
{c, sum[i]} = a[i] + b[i] + c;
                // builds an 8-bit ripple adder
c_out = c;
```

always statement

```
always @(posedge clk)
    q <= d;
```

- inferred as a positive edge-triggered D-flipflop



- level-sensitive latch

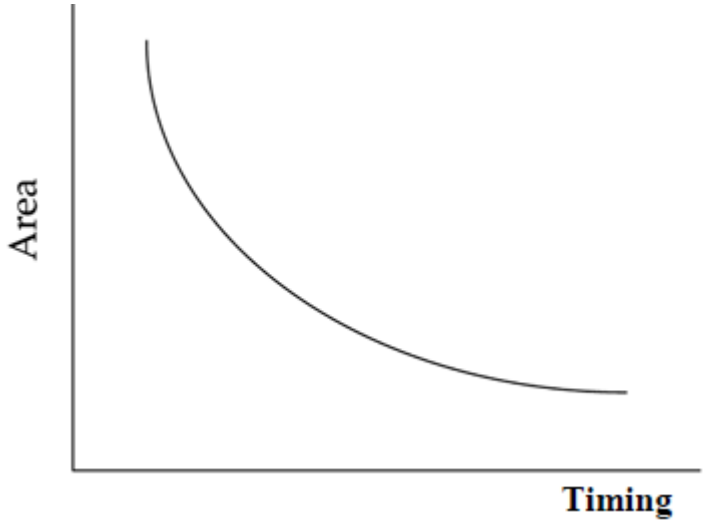
```
always @(clk or d)
    if (clk)
        q <= d;
```

Modeling Tips for Logic Synthesis

- Use meaningful names for signals and variables
- Avoid mixing positive and negative edge-triggered flipflops
  - introduce inverters and buffers into the clock tree
  - clock skews
- Use parentheses to optimize logic structure
- Be careful with multiple assignments to the same variable
- Define if-else or case statements explicitly
  - level-sensitive latches may be inferred

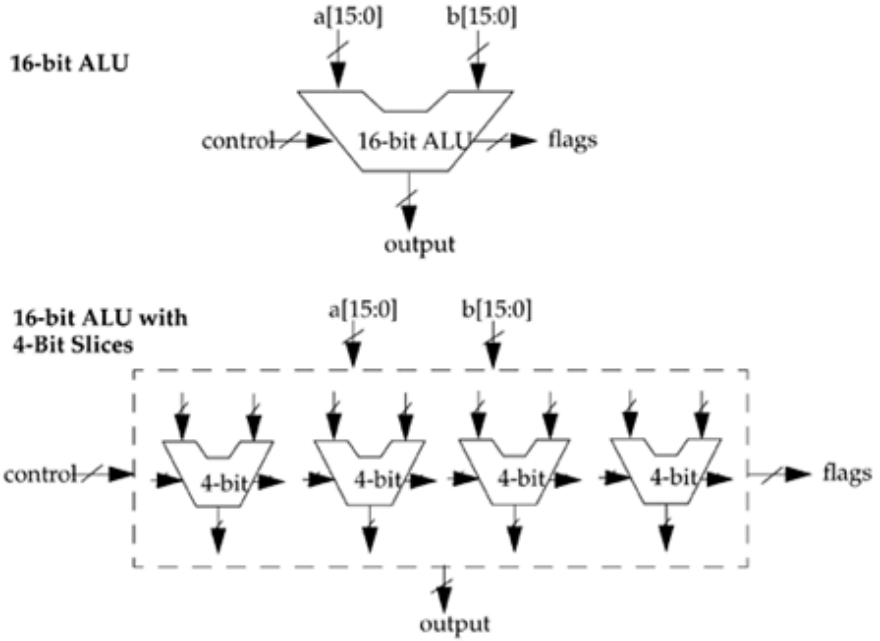
Design constraints

- Timing
- Area
- Power



Design Partitioning

- Horizontal partitioning



- Vertical partitioning