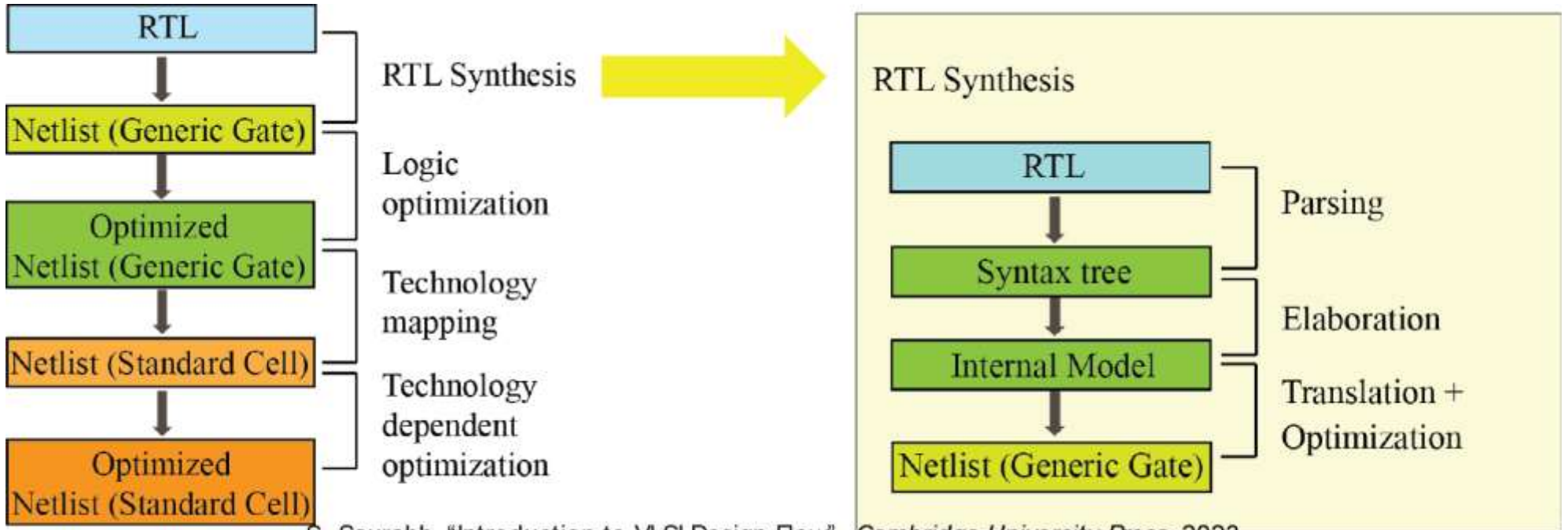
VLSI DESIGN FLOW: RTL TO GDS

Lecture 13
RTL Synthesis- Part II

Sneh Saurabh
Electronics and Communications Engineering
IIIT Delhi

Lecture Plan



S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

- Translation of some more Verilog Constructs to Circuit
- Optimization

RTL Synthesis

Verilog Constructs to Circuit

For Loop

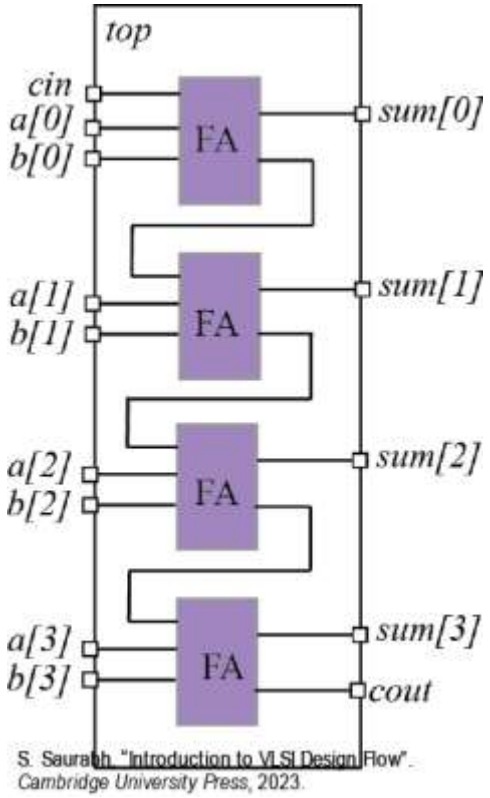
- A for loop can model repeated logic structure
- RTL synthesis tool can unroll (expand) the loop and instantiate circuit elements for each iteration.
  - Must know the number of iterations during synthesis (not during runtime)
  - Synthesizable for a fixed number of iterations

```
module top(a, b, cin, sum, cout);
      input [3:0]a, b; input cin;
      output [3:0]sum; output cout;
      reg [3:0]sum; reg cout; reg carry;
      reg [2:0]idx;

      always @(*) begin
         carry = cin;
         for(idx=0; idx < 4; idx=idx+1)
         begin
            {carry, sum[idx]} =
               a[idx] + b[idx] + carry;
         end
         cout = carry;
      end

endmodule
```

```
{carry, sum[0]}=a[0]+b[0]+cin;
{carry, sum[1]}=a[1]+b[1]+carry;
{carry, sum[2]}=a[2]+b[2]+carry;
{cout, sum[3]}=a[3]+b[3]+carry;
```



S. Saurabh, "Introduction to VLSI Design Flow", Cambridge University Press, 2023.

Functions

Function synthesizes to a combinational logic block with one output (scalar or vector).

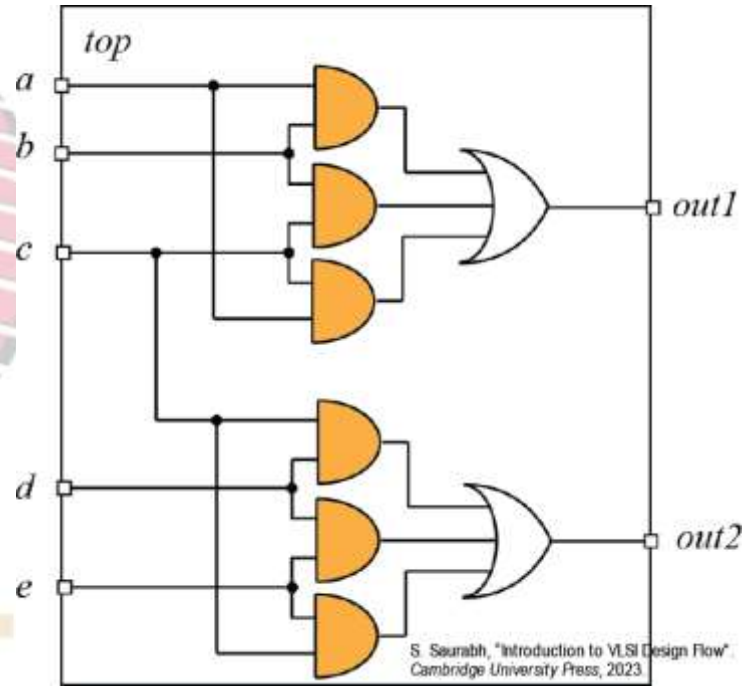
```
module top(a, b, c, d, e, out1, out2);
        input a, b, c, d, e;
        output out1, out2;
        reg out2;

        function MAJOR3;
                input A, B, C;
                begin
                        MAJOR3 = (A&B)|(B&C)|(C&A);
                end
        endfunction

        assign out1 = MAJOR3(a, b, c);

        always @(*) begin
                out2 = MAJOR3(c, d, e);
        end

endmodule
```



Operators: Synthesis Tasks

Operators:

- Verilog supports various kinds of operators
  - Logical: operators !, &&, || etc.
  - Bitwise operators: ~, !, |, ^, ~^ etc.
  - Arithmetic operators: +, -, * / etc.
  - Relational: <, >, == etc.

- Directly translate logical and bit-wise operators to their corresponding logic gates and optimize them subsequently.

Arithmetic and relational operators:

- Translate them to an internal representation or data structure
- Perform operator-level optimization
- Map operators to specific implementation

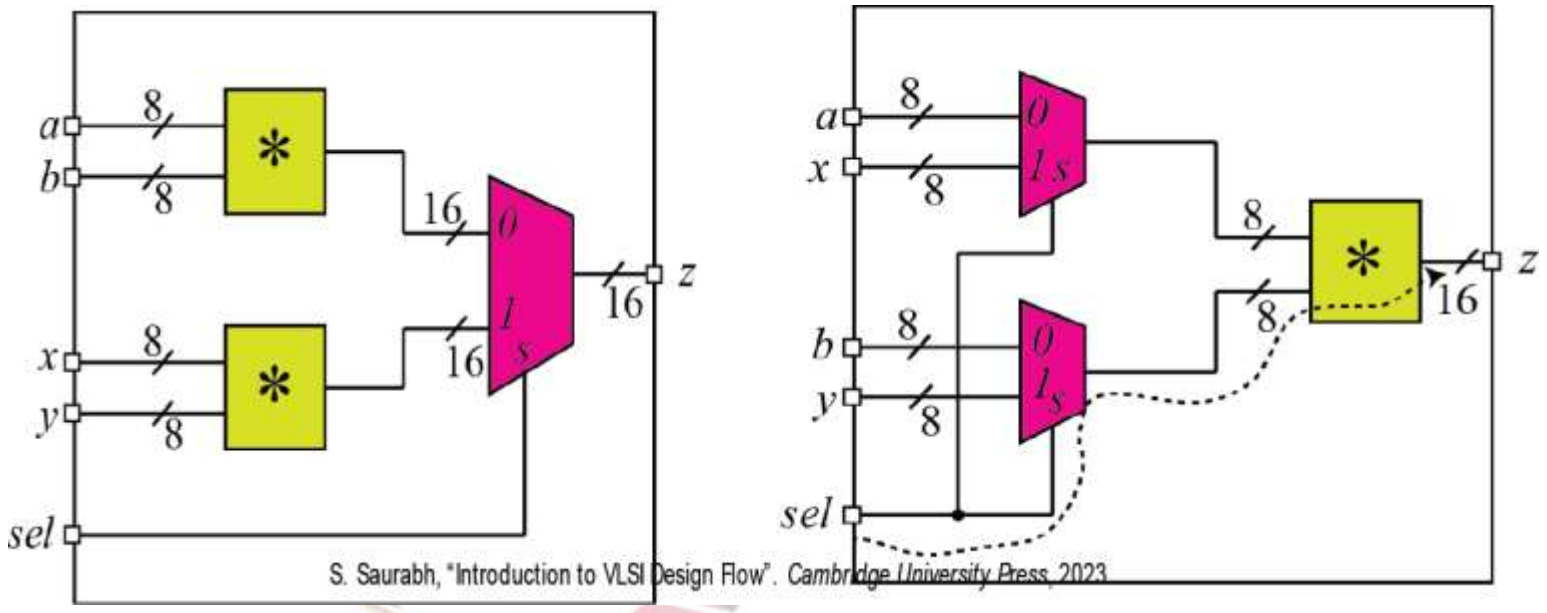
RTL Synthesis

Resource/Timing Optimization

Operators: Resource Sharing

```
if (sel == 1'b0)
        z = a*b;
else
        z = x*y;
```

Assume: *a, b, x, y* are of 8 bits and *z* is of 16 bits.



S. Saurabh, "Introduction to VLSI Design Flow", *Cambridge University Press*, 2023

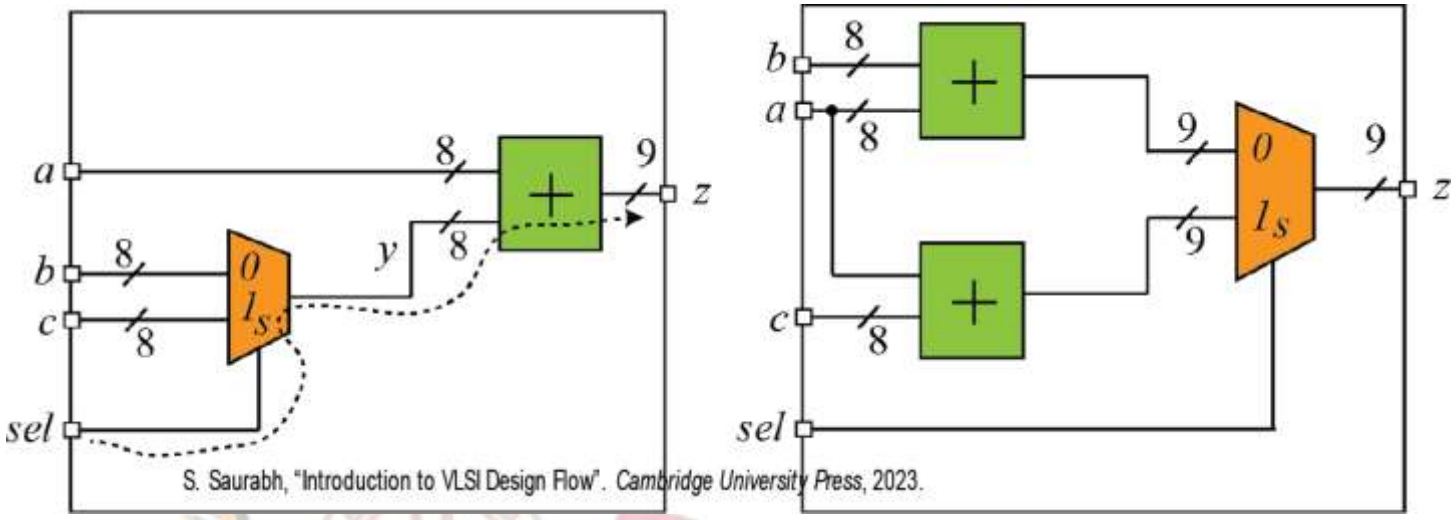
- Critical path can change
- Need to ensure that path through the select pin of the multiplexer does not violate timing constraint

Operators: Speculation (Resource Unsharing)

```
if (sel == 1'b0)
      y = b;
else
      y = c;

z = a+y;
```

Assume: *a, b, c, y* are of 8 bits.



- Assume that path through *sel* is critical.

- Decreases delay of the critical path by employing more resources
- Performs addition irrespective of the sum being needed
  - ➢ speculation or eager computation

RTL Synthesis

Implementation of Arithmetic Function

Operators: Implementation

- Tools map arithmetic operators to predefined modules implementing these operators.
- A set of internal parameterized modules implementing arithmetic operators (+, –, *, /) and relational operators (==, >, >=, <, <=).
- Instantiate these internal modules and choose the right set of parameters

Operators: Implementation Example

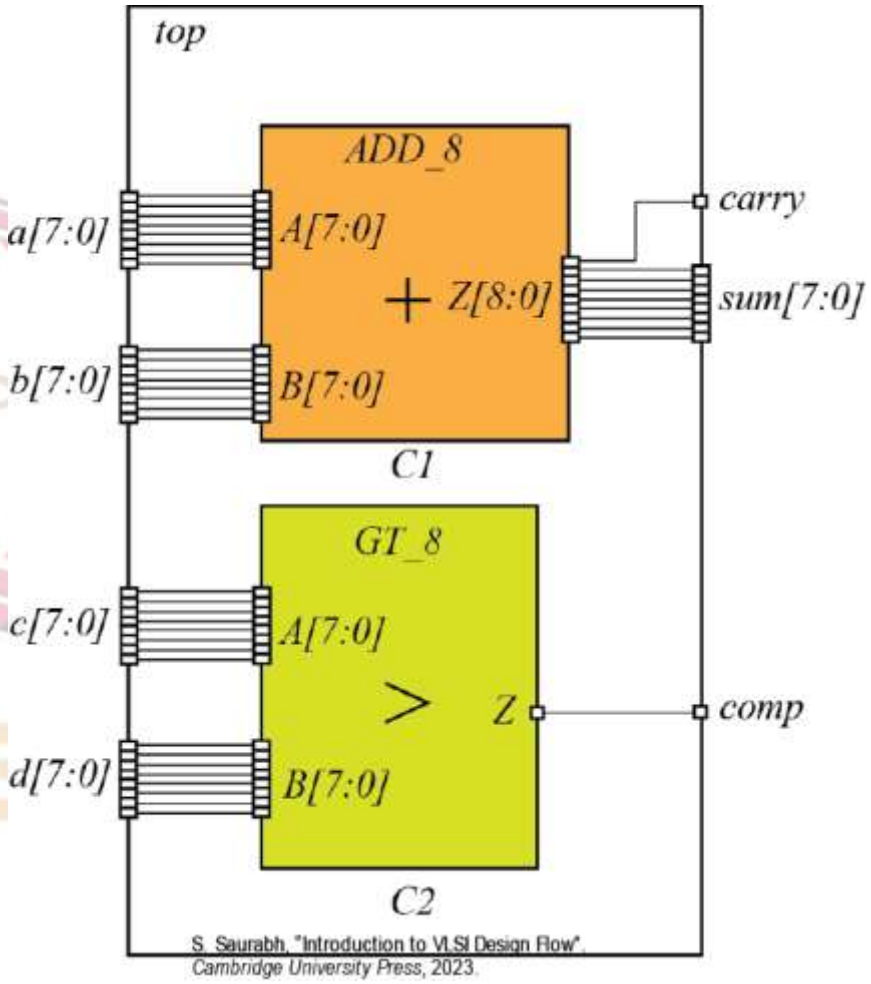
```
module top(a, b, c, d, sum, carry, comp);
        input [7:0]a, b, c, d;
        output [7:0]sum;
        output carry, comp;

        assign {carry,sum}= a + b;
        assign comp = (c > d);

endmodule
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

- Exact implementation tool-specific (logical function same)
- Architecture switching in later stages possible



RTL Synthesis

Compiler Optimizations

Compiler Optimizations

- Adapt compiler optimization techniques to RTL synthesis
- Can be applied to the parse tree or the internal model of the RTL
  - Typically, optimizations applied in passes
- Gives significant gains for arithmetic operators

Compiler Optimizations: Examples

Constant Propagation: replace expression with constant when possible

```
a = 8*8;
b = (a*1024)/32;
c = (b+32+b+32);
```

```
a = 64;
b = 2048;
c = 4160;
```

Common subexpression elimination: replace identical subexpression in multiple expressions with a single variable if it reduces the cost, such as area

```
x = p+a*b;
y = q+a*b;
```

```
c = a*b;
x = p+c;
y = q+c;
```

Strength reduction: replace an expensive arithmetic operation with an equivalent less expensive operation

```
x = a*64;
y = b/4;
z = c*17;
```

```
x = a<<6;
y = b>>2;
z = (c<<4)+c;
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

References

- G. D. Micheli. “Synthesis and Optimization of Digital Circuits”. *McGraw-Hill Higher Education*, 1994.
- S. Saurabh, “Introduction to VLSI Design Flow”. Cambridge: *Cambridge University Press*, 2023.