EEL 4783: HDL in Digital System Design

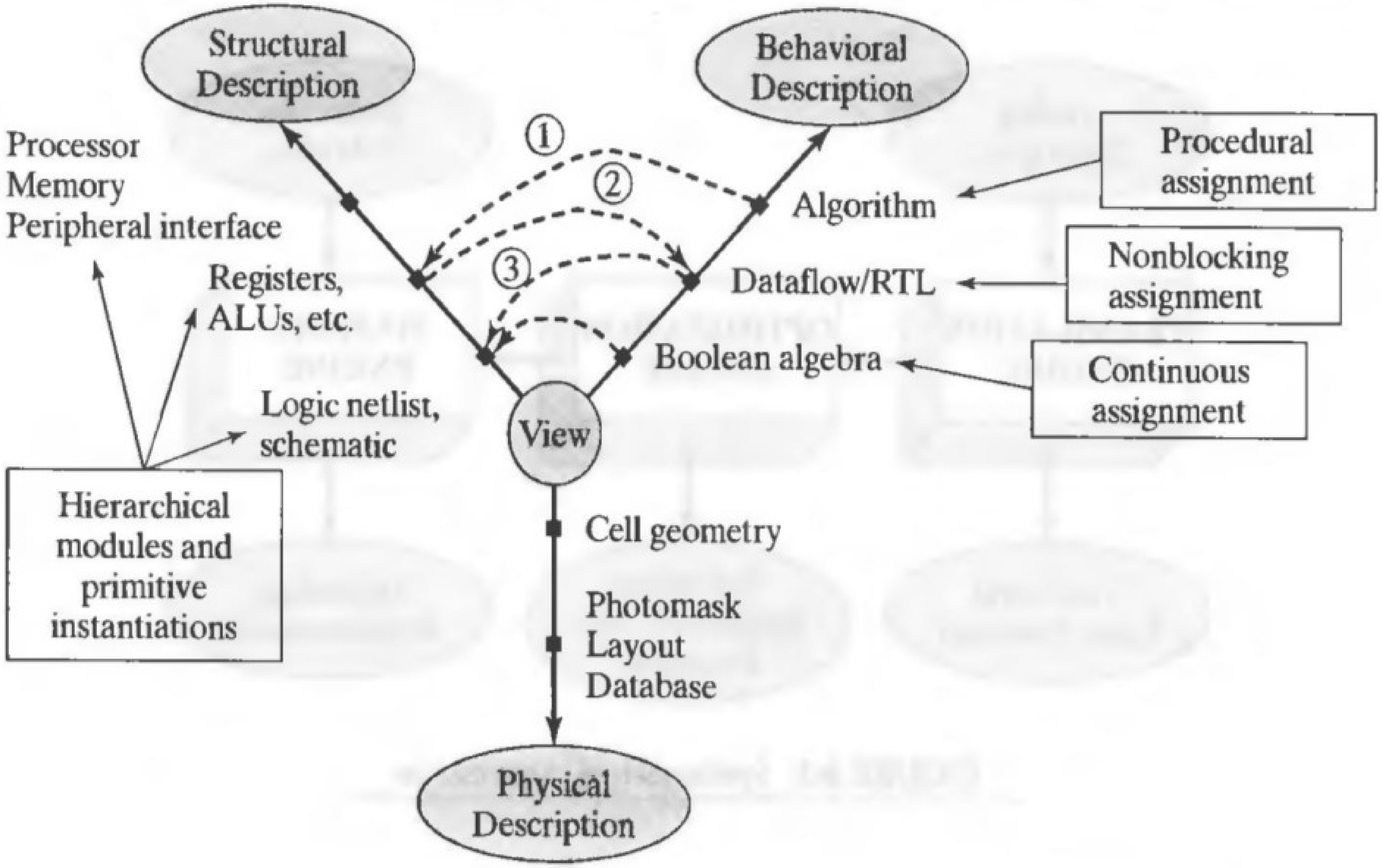
Lecture 7: HDL Programming for Logic Synthesis

Prof. Mingjie Lin

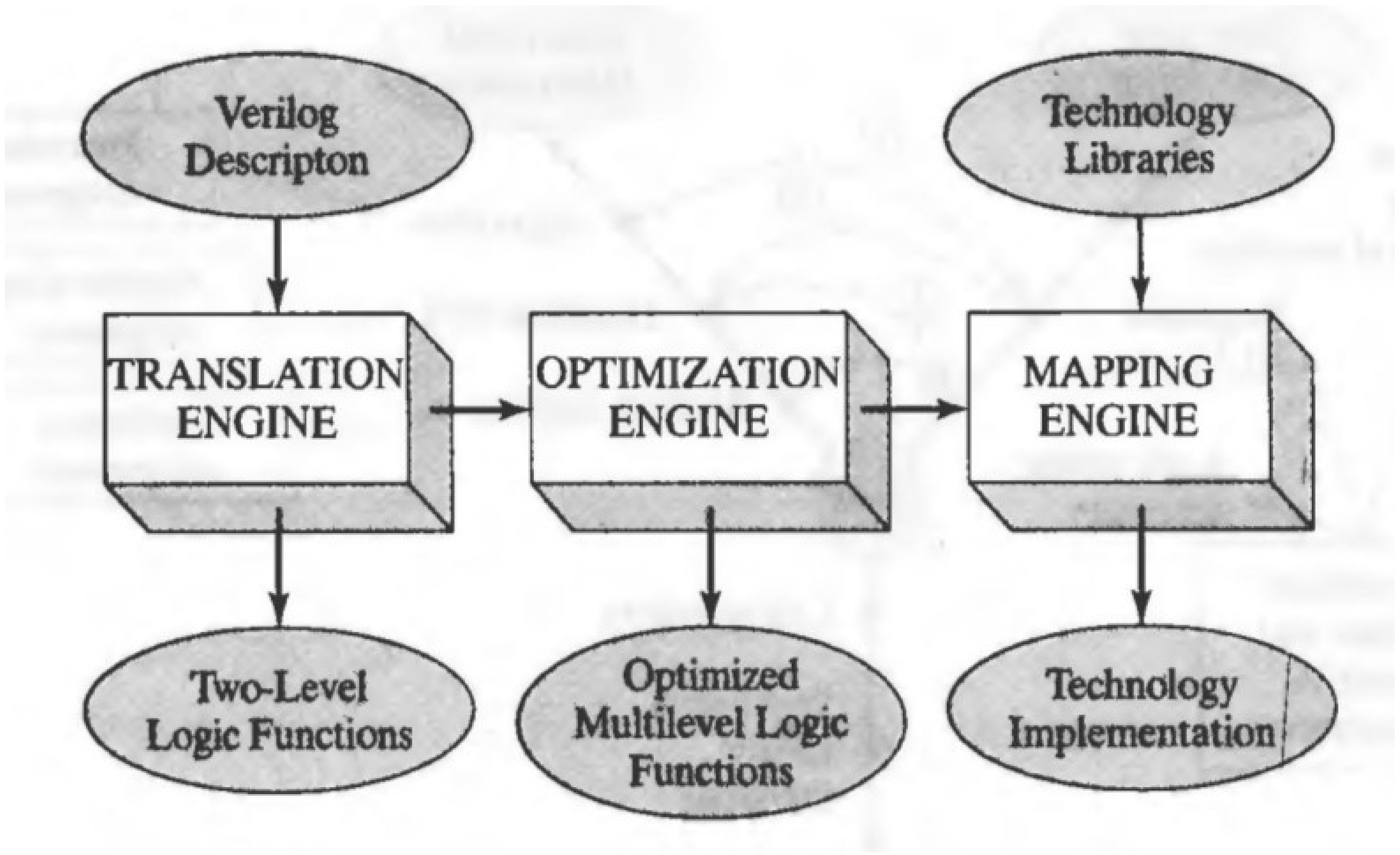
How to Write Synthesis-Friendly HDL Codes?

- Detect and eliminate redundant logic?
- Detect combinational feedback feedback loops
- Exploit Don't care conditions
- Detect unused states
- Detect and collapse equivalent states
- Synthesize optimized logic circuits

Behavioral, Structural, Physical Views

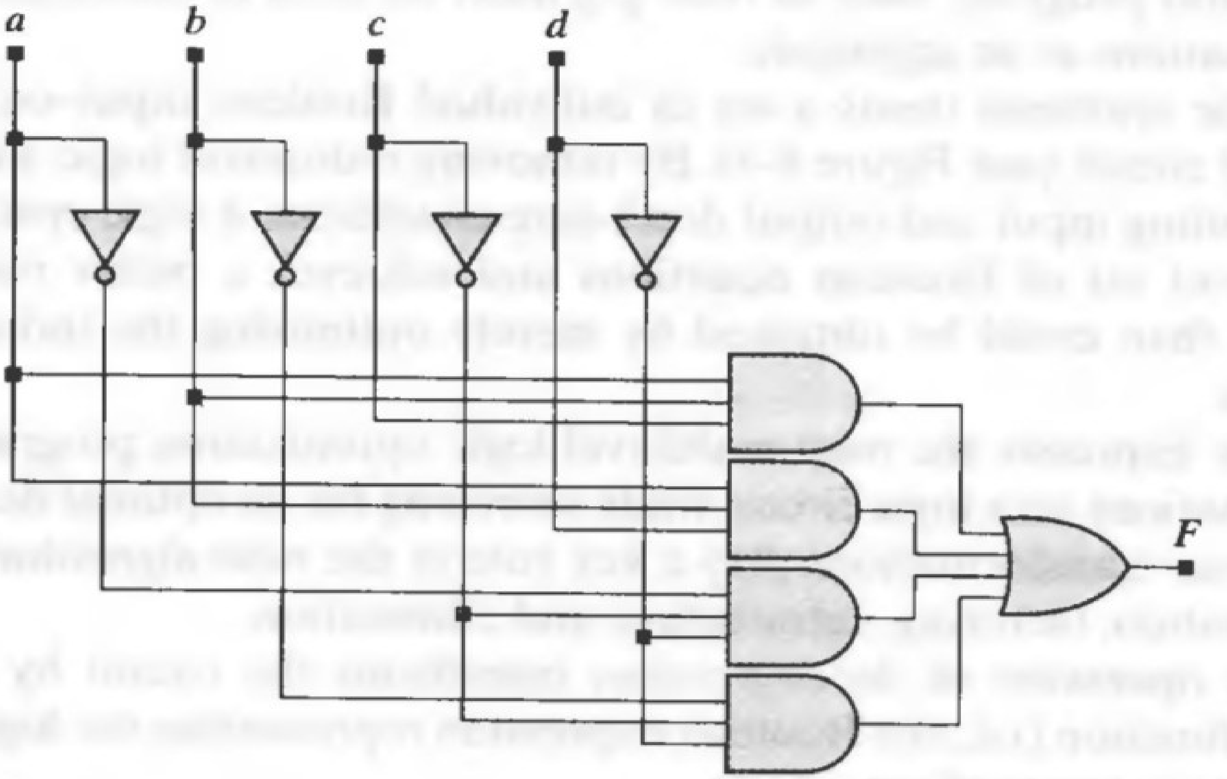


Synthesis Tools Organization



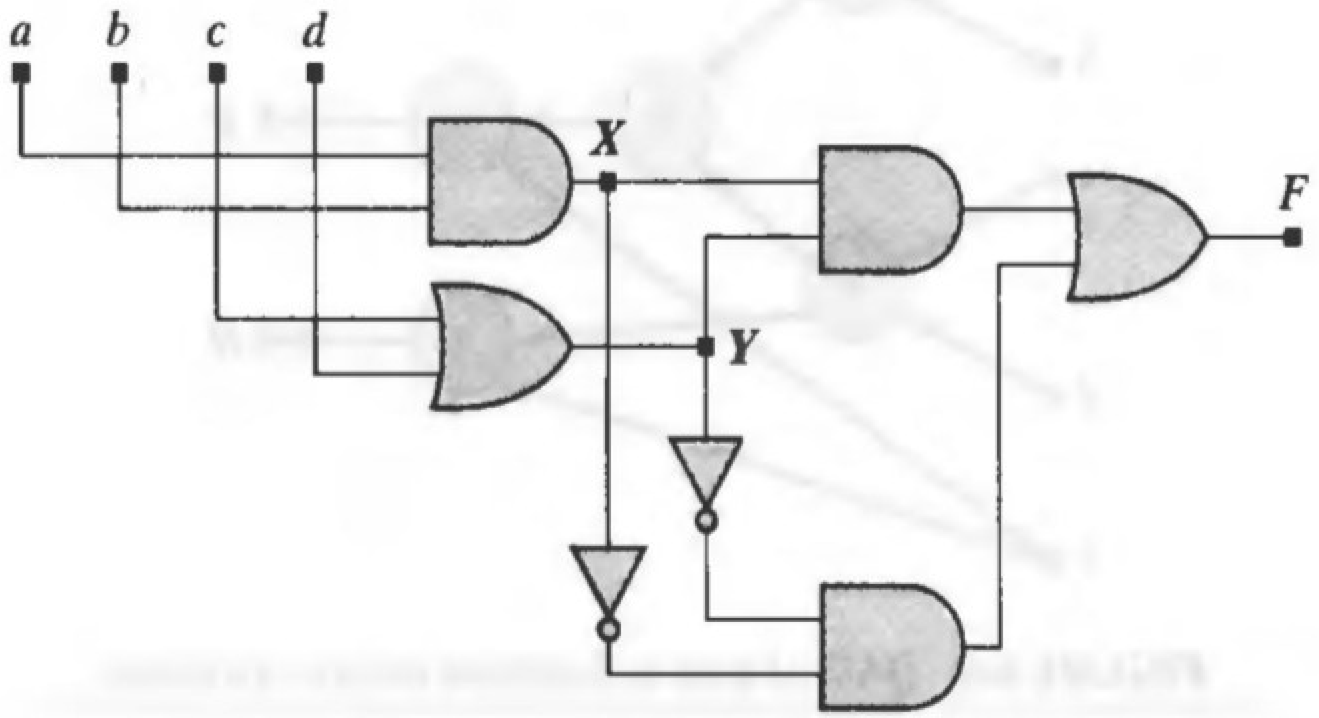
Sum-of-Products Implementations

$$F = abc + abd + a'b'c' + b'c'd'$$

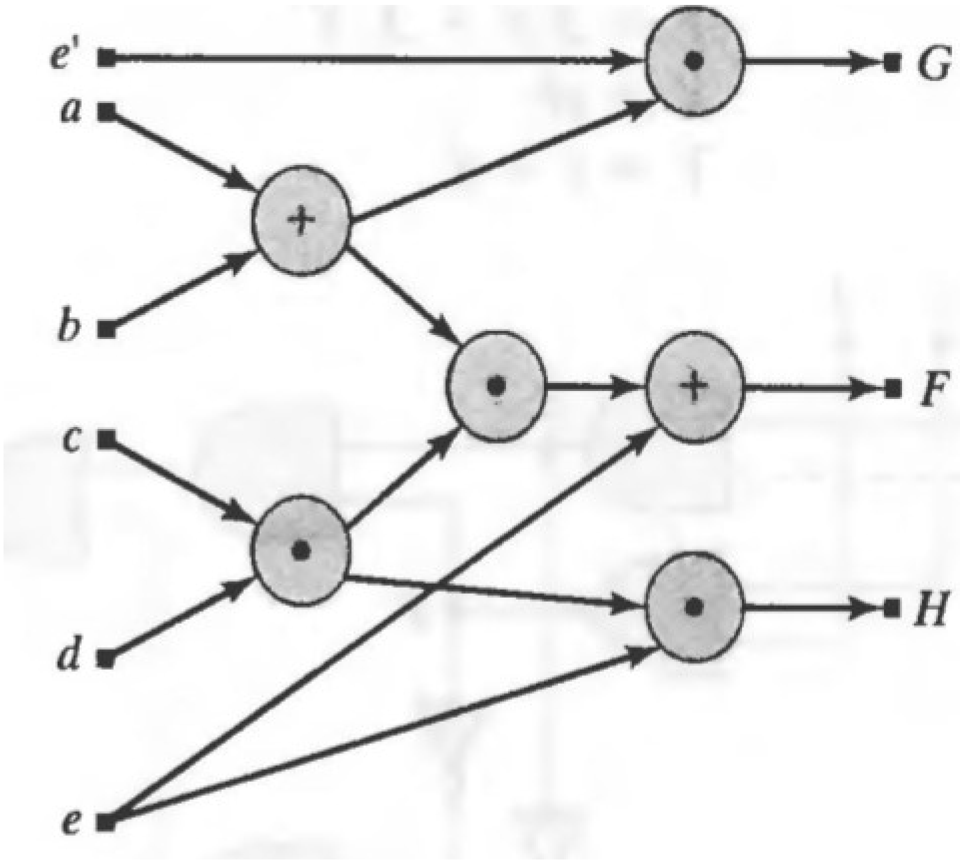


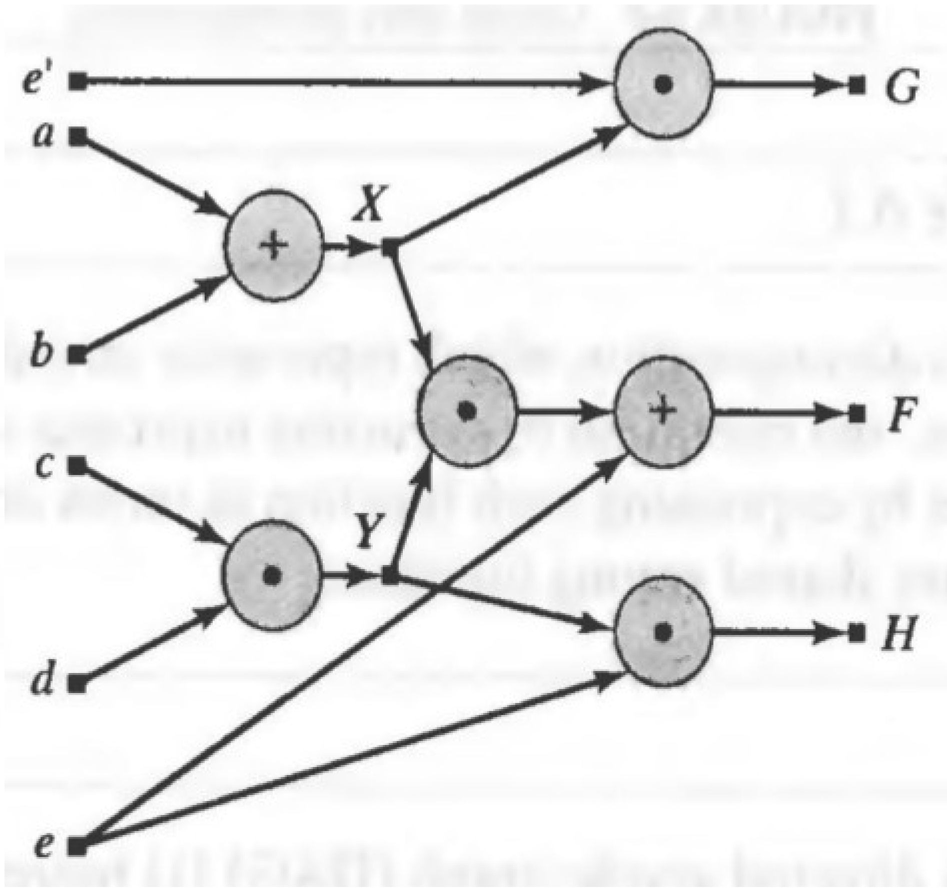
Optimized Design

$$F = XY + X'Y'$$
$$X = ab$$
$$Y = c + d$$

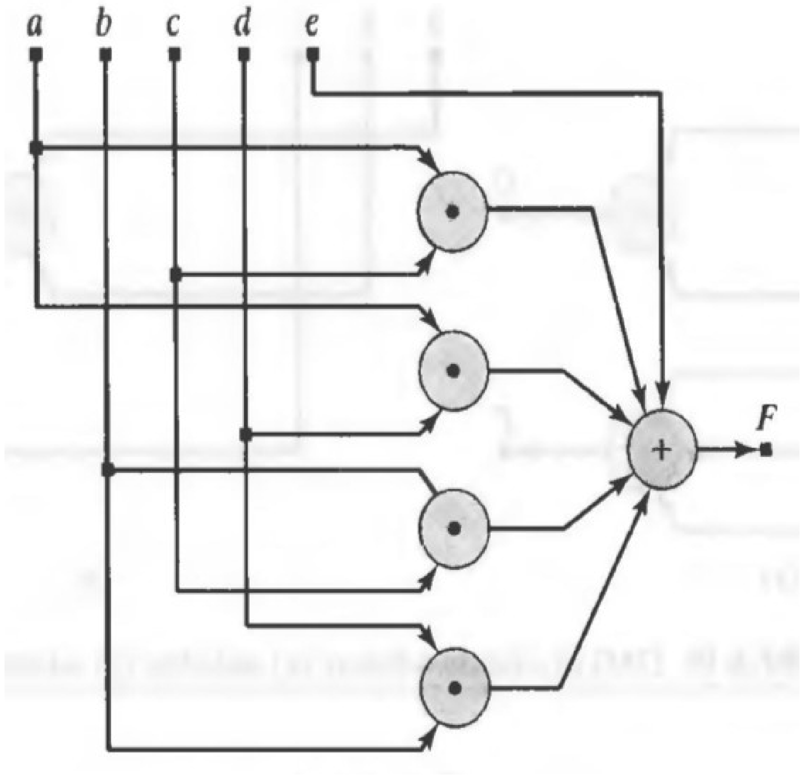


DAG Graph

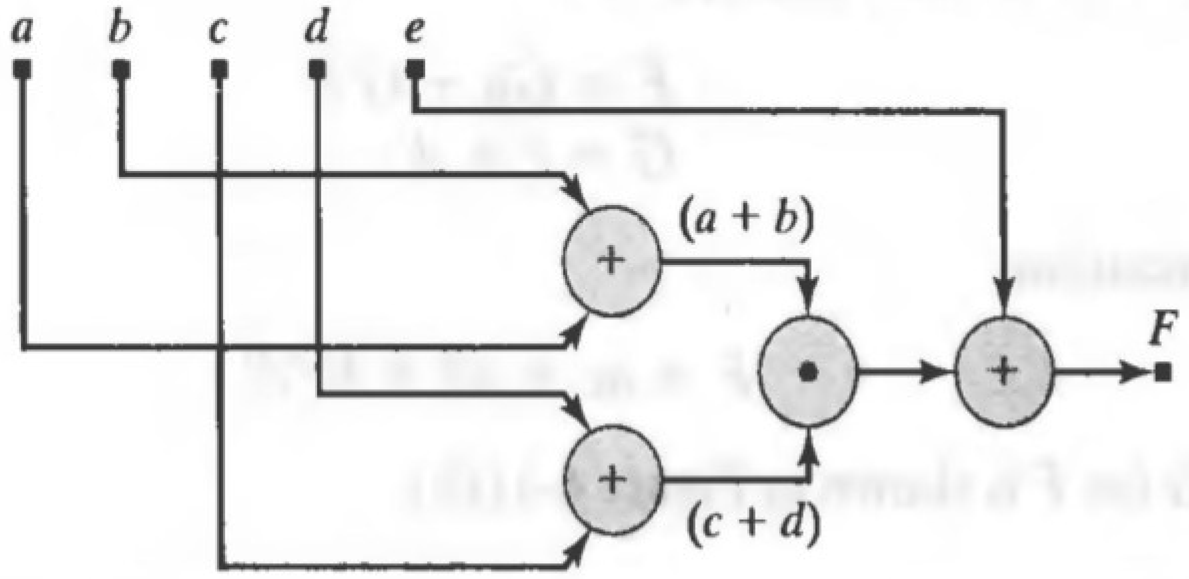




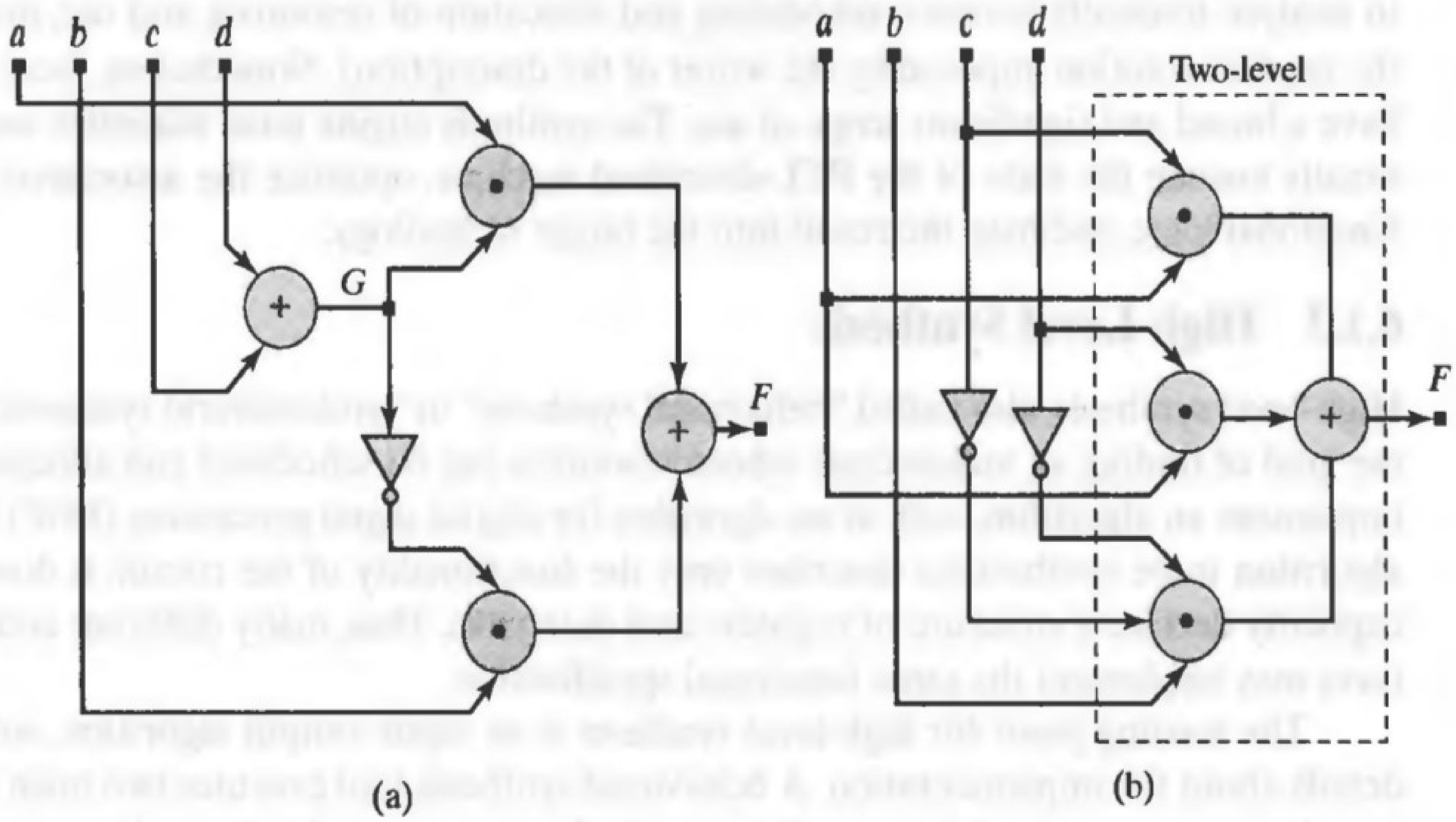
$$F = ac + ad + bc + bd + e$$



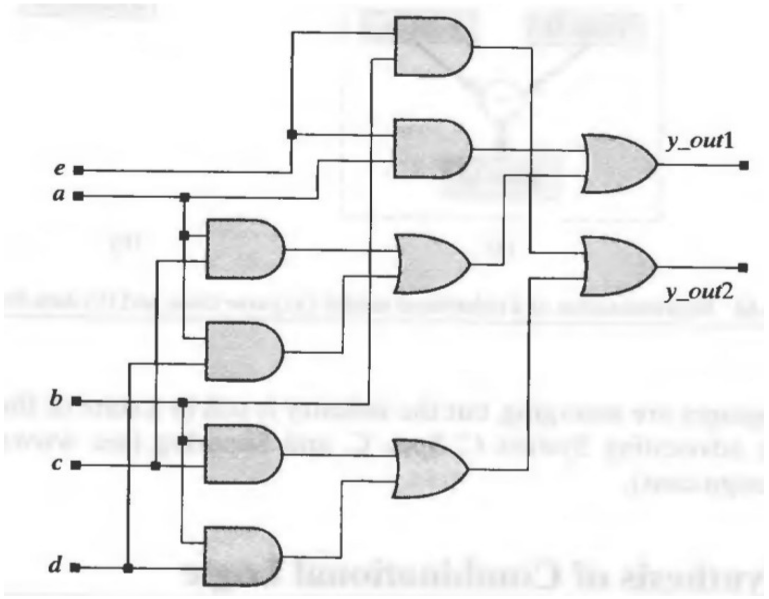
$$F = (a + b)(c + d) + e$$

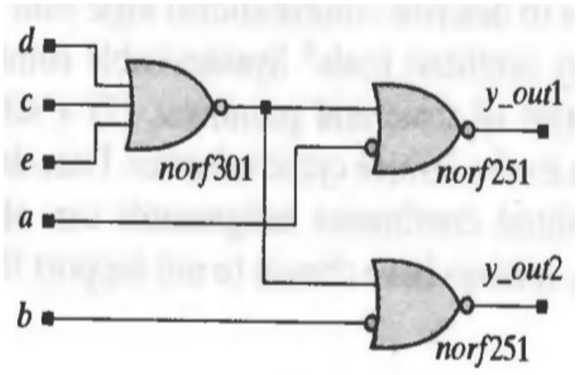


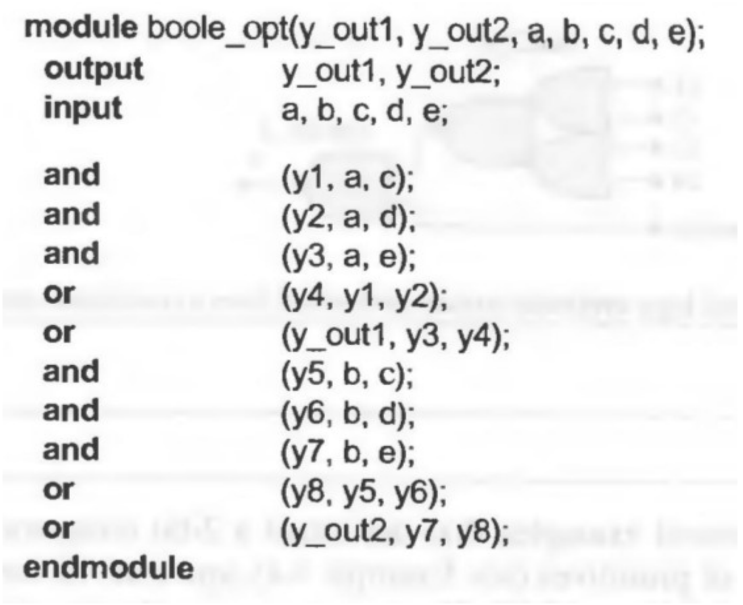
Logic Optimization



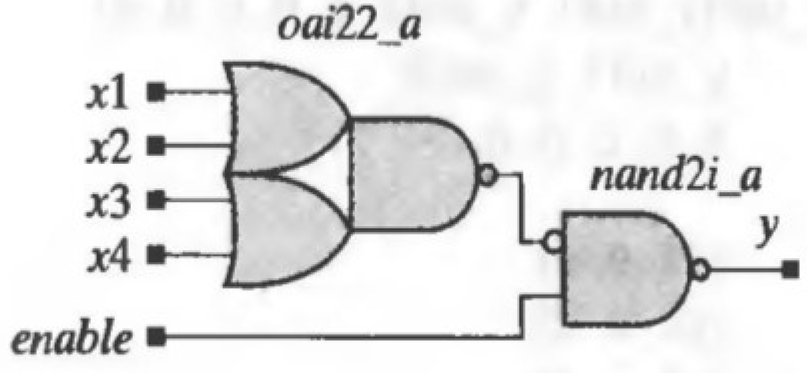
Synthesis of Comb. Circuits





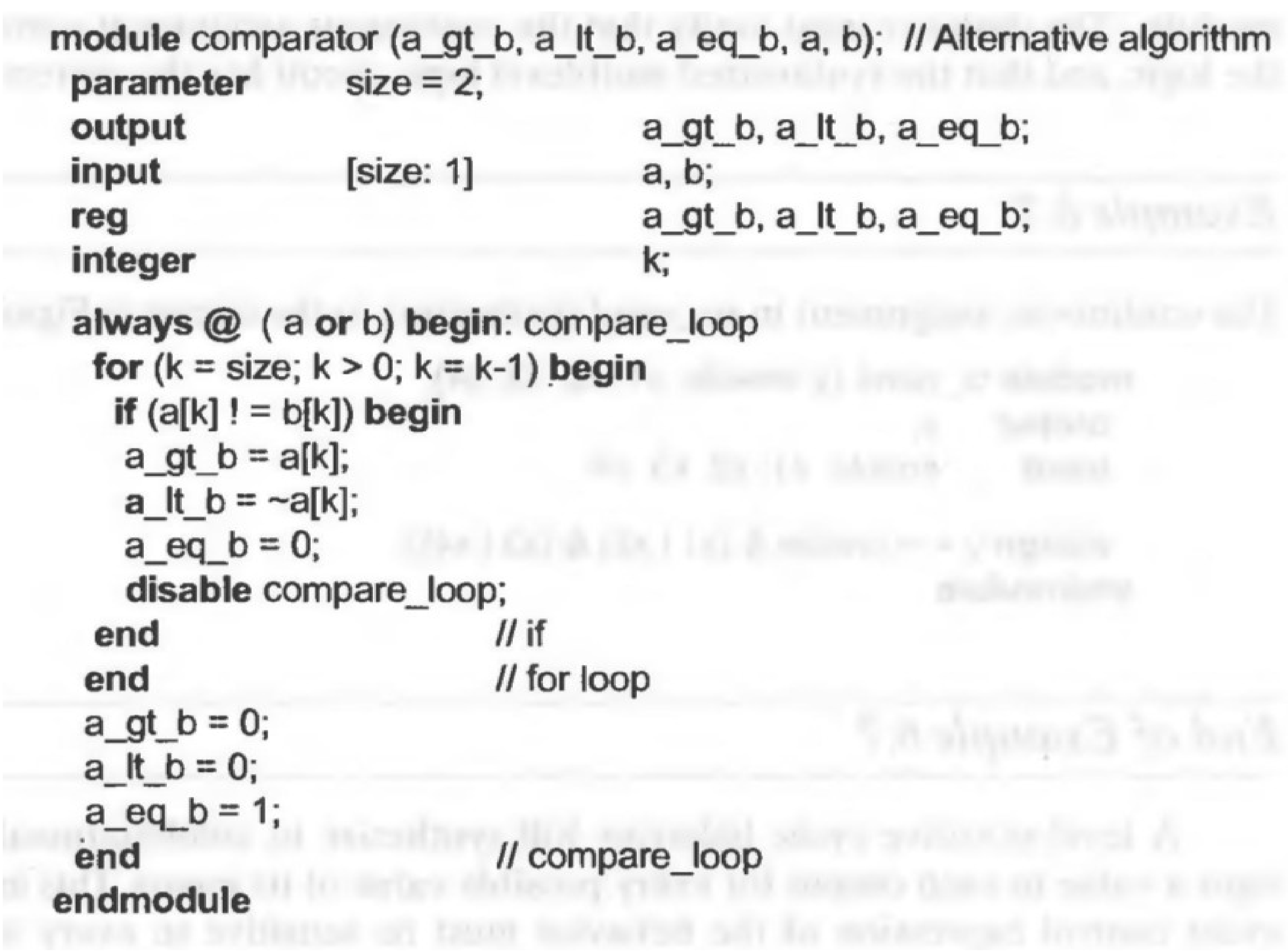

```
module boole_opt(y_out1, y_out2, a, b, c, d, e);
 output          y_out1, y_out2;
 input           a, b, c, d, e;

 and             (y1, a, c);
 and             (y2, a, d);
 and             (y3, a, e);
 or              (y4, y1, y2);
 or              (y_out1, y3, y4);
 and             (y5, b, c);
 and             (y6, b, d);
 and             (y7, b, e);
 or              (y8, y5, y6);
 or              (y_out2, y7, y8);
endmodule
```



```
module or_nand (y, enable, x1, x2, x3, x4);
  output    y;
  input     enable, x1, x2, x3, x4;

  assign y = ~(enable & (x1 | x2) & (x3 | x4));
endmodule
```

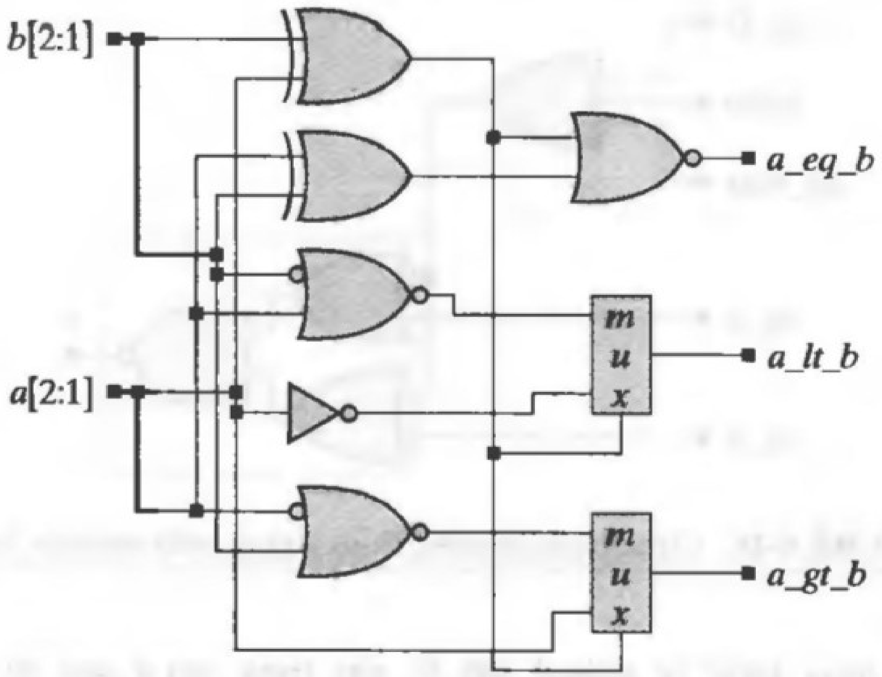


```
module comparator (a_gt_b, a_lt_b, a_eq_b, a, b);  // Alternative algorithm
 parameter       size = 2;
 output                        a_gt_b, a_lt_b, a_eq_b;
 input           [size: 1]     a, b;
 reg                           a_gt_b, a_lt_b, a_eq_b;
 integer                       k;

 always @  ( a or b) begin: compare_loop
  for (k = size; k > 0; k = k-1) begin
   if (a[k] ! = b[k]) begin
    a_gt_b = a[k];
    a_lt_b = ~a[k];
    a_eq_b = 0;
    disable compare_loop;
  end                    // if
  end                    // for loop
  a_gt_b = 0;
  a_lt_b = 0;
  a_eq_b = 1;
 end                     // compare_loop
endmodule
```

```
module comparator (a_gt_b, a_lt_b, a_eq_b, a, b);  // Alternative algorithm
  parameter        size = 2;
  output                           a_gt_b, a_lt_b, a_eq_b;
  input            [size: 1]       a, b;
  reg                              a_gt_b, a_lt_b, a_eq_b;
  integer                          k;

  always @  ( a or b) begin: compare_loop
   for (k = size; k > 0; k = k-1) begin
    if (a[k] ! = b[k]) begin
     a_gt_b = a[k];
     a_lt_b = ~a[k];
     a_eq_b = 0;
     disable compare_loop;
   end                       // if
  end                        // for loop
  a_gt_b = 0;
  a_lt_b = 0;
  a_eq_b = 1;
 end                         // compare_loop
endmodule
```



Final issues

- Please fill out the student info sheet before leaving
- Come by my office hours (right after class)
- Any questions or concerns?