EE 3755

Verilog Handout #4

```
/// LSU EE 3755 -- Fall 2001 -- Computer Organization
//
/// Verilog Notes 4 -- Synthesis

/// Under Construction
 // Time-stamp: <14 September 2001, 12:18:11 CDT, koppel@sol>

/// Contents
 // Synthesis Overview
 // Descriptive Styles
 // Implicit Structural Descriptive Style
 // Behavioral
 // Synthesis Steps
 // Synthesis of Simple Logic
 // Synthesizing Arithmetic
 // Synthesis of Conditional Operator
 // Synthesis of Delays
 // Adder Comparison

/// References

// :P:   Palnitkar, "Verilog HDL"
// :Q:   Qualis, "Verilog HDL Quick Reference Card Revision 1.0"
// :H:   Hyde, "Handbook on Verilog HDL"
// :LRM: IEEE, Verilog Language Reference Manual  (Hawaii Section Numbering)
// :PH:  Patterson & Hennessy, "Computer Organization & Design"


////////////////////////////////////////////////////////////////////////////////
/// Synthesis Overview


 /// Synthesis
//
//  The steps needed to convert a Verilog (or other HDL) description into
//  a form that can be manufactured or downloaded into an FPGA or other
//  programmable device.

 /// Synthesis Technology Targets
//
//  Target technology definition:
//   The type of semiconductor (or other) technology used to build the design.
//   Sometimes shortened to ``target''

 ///  Common Targets
//
//  ASIC
//   A fully custom chip.
//   Output of synthesis program drives machines fabricating chip.

//  FPGA
//   Programmable logic.
//   Output of synthesis program downloaded into pre-manufactured chip.

 /// What Synthesis Program Does
//
//  Reads Verilog description.
//  Writes design file that specifies (for ASICS):
//   The components that are needed. (and, mux, ram)
//   How they are connected.
//   The components' locations and how the wires are routed.

 /// Synthesis In a Perfect World
//
//  Prepare Verilog description.
//  Using simulation verify that description works as desired.
//  Click ``synthesize'' and enter credit card number.
//  Chips arrive, and perform exactly as simulated chips do.
```

```
/// Synthesis In the Early 21st Century
//
//  Prepare Verilog description following the synthesis program's /design rules/.
//  Using simulation verify that description works as desired.
//  Partially synthesize, redo design, etc.


////////////////////////////////////////////////////////////////////////////////
/// Descriptive Styles

 ///  Descriptive Style Definition
//
//   A set of rules for a Verilog description.

 ///  Three Major Styles
//
//    Explicit Structural:  Most restrictive.
//    Implicit Structural:  Still restrictive.
//    Behavioral:           Least restrictive. (Not yet covered.)
//
//  A Verilog description is called ``explicit structural'' if it
//   follows explicit structural design rules. (described below).
//   The same holds for the other styles.

 /// Explicit Structural Descriptive Style
//
//  The Verilog specifies every component and how they are wired.
//
//  Tedious to write but easy to synthesize.
//
//  Rules:
//   No operators (+,&,~, etc.)
//   No assign's (continuous assignments).
//   No initial/always blocks (covered later).

// :Example:
//
//  The module below is explicit structure and is neither implicit
//  structural nor behavioral.

module bfa_structural(sum,cout,a,b,cin);
   input a,b,cin;
   output sum,cout;

   wire   term001, term010, term100,term111;
   wire   ab, bc, ac;
   wire   na, nb, nc;

   or o1(sum,term001,term010,term100,term111);
   or o2(cout,ab,bc,ac);

   and a1(term001,na,nb,cin);
   and a2(term010,na,b,nc);
   and a3(term100,a,nb,nc);
   and a4(term111,a,b,cin);

   not n1(na,a);
   not n2(nb,b);
   not n3(nc,cin);

   and a10(ab,a,b);
   and a11(bc,b,cin);
   and a12(ac,a,cin);

endmodule
```

```
////////////////////////////////////////////////////////////////////////////////
/// Implicit Structural Descriptive Style

// Some components and wiring are specified using expressions.

// Less tedious to write, and still easy to synthesize.

// Rules:
//  No initial/always blocks (covered later).


// :Example:
//
//  The module below is implicit structure and is neither explicit
//  structural nor behavioral.

module bfa_implicit(sum,cout,a,b,cin);
   input a,b,cin;
   output sum,cout;

   assign sum =
          ~a & ~b &  cin |
          ~a &  b & ~cin |
           a & ~b & ~cin |
           a &  b &  cin;

   assign cout = a & b | b & cin | a & cin;

   // It can't be explicit structural because of the assign and operators.

endmodule

////////////////////////////////////////////////////////////////////////////////
/// Behavioral

// Easiest to write.

// Cannot be synthesized.
// (More restrictive behavioral styles can be synthesized, these
//  will be covered later.)

// Rules:
//  Any Verilog.

// :Example:
//
//  The module below is behavioral and is neither explicit nor
//  implicit structural.
//
//  This particular behavioral module could be synthesized, though not
//  all behavioral code can be.

module bfa_behavioral(sum,cout,a,b,cin);
   input a,b,cin;
   output sum,cout;

   // Material for this module will be covered later.
   // It's here to show what behavioral code looks like.

   reg    sum, cout;

   always @( a or b or cin )
     begin

        sum = ~a & ~b &  cin |
              ~a &  b & ~cin |
               a & ~b & ~cin |
               a &  b &  cin;
        cout = a & b | b & cin | a & cin;

     end

endmodule

////////////////////////////////////////////////////////////////////////////////
/// Synthesis Steps

 /// Synthesis Steps (Design Flow)
//  (1) Synthesize to RTL (Inference)
//  (2) Technology Mapping
//  (3) Optimization
//  (4) Place and Route


 /// Step 1:  Synthesize to RTL (Inference)
//
//  Input:  The Verilog description.
//  Output: An Explicit Structural Description

//  For implicit structural input, operators are replaced with pre-defined
//  modules or gates.

//  For example, & with and gate(s), ~ with inverter.

//  The handling of behavioral code covered later.

//  Synthesis programs have special modules for arithmetic and other
//  complex operations.

//  The output of this step consists of generic gates and modules
//  (see next step).


 /// Step 2:  Technology Mapping
//
//  The generic gates and modules are replaced with gates and
//  modules from the target technology library.

//    ASIC technology libraries have a ``normal'' set of gates and
//    modules: AND, OR, adders, etc.

//    FPGA technology libraries may consists of primarily
//    multiplexors or lookup tables.

//  Target gates don't always match generic ones, so changes made.

//    For example, a generic 3-input AND gate might be mapped to a
//    four-input AND gate in the technology library with one input
//    tied to logic 1.

//  One of several target gates might be chosen based on fanout needs.

//  Generic arithmetic modules replaced by technology modules, if available,
//  otherwise by gates.


 /// Step 3:  Optimization

//  The output of the last step is optimized to reduce the number of
//  gates and the critical path length.

//  Though not covered in class, this step relies on constraints provided
//  by the designer. (For example, ``keep this path delay below 10 ns.'')

 /// Step 4: Place and Route
```

```
//  Finally, locations are chosen for components in the
//  optimized design and routes are chosen for wires connecting them.

//  No further details will be given for this step.


////////////////////////////////////////////////////////////////////////////////
/// Synthesis of Simple Logic

// Verilog description to implement:  x = ab + ac + ad;
// Target: ASIC produced by Fab Fab Tech [tm] (Name made up.)

// :Example:
//
// Verilog code, written by human, to implement x = ab + ac + ad;

module logic(x,a,b,c,d);
   input a, b, c, d;
   output x;

   assign x = a & b  |  a & c  |  a & d;

endmodule


// :Example:
//
//  Simplified output of inference stage.
//
//  The Verilog below was hand written but designed to show what would
//  be produced by the synthesis steps.  The actual output is
//  less readable, see next example.

module logic_rtl(x,a,b,c,d);
   input a, b, c, d;
   output x;

   wire   ab, ac, ad;

   and a1(ab,a,b);
   and a2(ac,a,c);
   and a3(ad,a,d);

   or o1(x,ab,ac,ad);

endmodule


// :Example:
//
//  The real output of the inference stage.

module logic_rtl_real ( x, a, b, c, d ) ;

    output x ;
    input a ;
    input b ;
    input c ;
    input d ;

    wire nx0, nx2, nx4, nx6;

    and (nx0, a, b) ;
    and (nx2, a, c) ;
    or (nx4, nx0, nx2) ;
    and (nx6, a, d) ;
    or (x, nx4, nx6) ;
```

```
endmodule


// :Example:
//
// Simplified output of the technology mapping step.
//
// Here generic gates (and, or) are replaced with specific
// gates from the target technology, fab_fab_and_2 and fab_fab_or_4.

module logic_tech(x,a,b,c,d);
   input a, b, c, d;
   output x;

   wire   ab, ac, ad;

   fab_fab_and_2 a1(ab,a,b);
   fab_fab_and_2 a2(ac,a,c);
   fab_fab_and_2 a3(ad,a,d);

   fab_fab_or_4 o1(x,ab,ac,ad,1'b0);

endmodule


// :Example:
//
// Simplified output of the optimization step. Logic is
// simplified by optimization program.

module logic_opt(x,a,b,c,d);
   input a, b, c, d;
   output x;

   wire   ab, ac, ad;

   fab_fab_or_4 o1(bcd,b,c,d,1'b0);
   fab_fab_and_2 a1(x,a,bcd);

endmodule


// :Example:
//
// The real output of the optimization stage.  IV1NP is an inverter,
// NR2R1 is a two-input NOR gate, and NR3Q1 is a three-input NOR gate.

module logic_real_opt ( x, a, b, c, d ) ;

    output x ;
    input a ;
    input b ;
    input c ;
    input d ;

    wire nx80, nx81;

    IV1NP ix82 (.X (nx80), .A (a)) ;
    NR3Q1 ix83 (.X (nx81), .A1 (b), .A2 (c), .A3 (d)) ;
    NR2R1 x_rename_rename (.X (x), .A1 (nx80), .A2 (nx81)) ;
endmodule


////////////////////////////////////////////////////////////////////////////////
/// Synthesizing Arithmetic

// Design hardware with two 8-bit inputs, and 1-bit output that is
// true if sum of unsigned integers on inputs > 120 and false
// otherwise.

// :Example:
//
// Verilog description of hardware written by human.

module too_bit(x,a,b);
   input [7:0] a, b;
   output      x;

   assign      x = a + b > 120;

endmodule


// :Example:
//
// Simplified output of inference step.  Operators replaced with
// instantiation of generic (can be fitted to many technologies) adder
// and comparison units.

module too_bit_rtl(x,a,b);
   input [7:0] a,b;
   output      x;

   wire [8:0]  ab;

   generic_add_8 a1(ab,a,b);
   generic_compare_gt_9 gc1(x,ab,9'd120);

endmodule


// :Example:
//
// The actual output of the inference step.

module too_bit ( x, a, b ) ;

    output x ;
    input [7:0]a ;
    input [7:0]b ;

    wire GND, nx2, nx3, nx4, nx5, nx6, nx7, nx8, nx9, nx10, PWR;
    wire [0:0] \$dummy ;

    assign GND = 0 ;
    add_9u_9u_9u_0_0 x_add_0 (.cin (GND), .a ({GND,a[7],a[6],a[5],a[4],a[3],a[2]
                     ,a[1],a[0]}), .b ({GND,b[7],b[6],b[5],b[4],b[3],b[2],b[1],
                     b[0]}), .d ({nx2,nx3,nx4,nx5,nx6,nx7,nx8,nx9,nx10}), .cout (
                     \$dummy [0])) ;
    assign PWR = 1 ;
    gt_9u_9u x_gt_1 (.a ({nx2,nx3,nx4,nx5,nx6,nx7,nx8,nx9,nx10}), .b ({GND,GND,
             PWR,PWR,PWR,PWR,GND,GND,GND}), .d (x)) ;
endmodule


// :Example:
//
// Simplified output of technology mapping.
//
//  Synthesis program used individual gates rather than a comparison unit
//  because one input is a constant, and so the gates can be optimized.

module too_bit_tech(x,a,b);
   input [7:0] a,b;
   output      x;
   wire [8:0]  ab;

   fab_fab_add_8 a1(ab,a,b);

   // Gates implementing a comparison circuit. (Not shown.)

endmodule


// :Example:
//
// Simplified output of optimization step.
//
// Since one input to comparison is a constant comparison was
// optimized.

module too_bit_opt(x,a,b);
   input [7:0] a,b;
   output      x;

   wire [8:0]  ab;

   fab_fab_add_8 a1(ab,a,b);

   // Gates implementing a comparison circuit, simplified because
   // one operand is a constant.

endmodule


// :Example:
//
// The actual output of optimization.

module too_bit ( x, a, b ) ;
    output x ;
    input [7:0]a ;
    input [7:0]b ;

    wire nx8, nx10, nx12, nx18, nx30, nx38, nx44, nx98, nx102, nx125, nx127,
         nx129, nx131, nx134, nx136, nx138, nx141, nx158, nx161, nx163, nx165,
         nx168, nx183, nx187, nx199, nx200, nx201, nx202, nx203, nx204, nx205,
         nx206, nx207, nx208, nx209, nx174, nx210, nx211, nx212, nx213, nx214,
         nx215, nx216, nx217, nx218, nx219, nx220, nx221, nx222, nx223, nx88,
         nx224, nx225, nx226, nx227, nx228, nx229, nx230, nx231, nx232, nx233;

    OAI2N0 ix45 (.X (nx44), .A1 (nx125), .A2 (nx127), .B1 (nx129), .B2 (nx131)
          ) ;
    IV1N0 ix126 (.X (nx125), .A (b[3])) ;
    IV1N0 ix128 (.X (nx127), .A (a[3])) ;
    XN2R0 ix130 (.X (nx129), .A1 (b[3]), .A2 (a[3])) ;
    AO2I0 ix132 (.X (nx131), .A1 (b[2]), .A2 (a[2]), .B1 (nx10), .B2 (nx134)) ;
    XR2T0 ix11 (.X (nx10), .A1 (b[2]), .A2 (a[2])) ;
    OAI2N0 ix135 (.X (nx134), .A1 (nx136), .A2 (nx138), .B1 (nx18), .B2 (nx141)
          ) ;
    IV1N0 ix137 (.X (nx136), .A (a[1])) ;
    IV1N0 ix139 (.X (nx138), .A (b[1])) ;
    ND2N0 ix19 (.X (nx18), .A1 (a[0]), .A2 (b[0])) ;
    XN2R0 ix142 (.X (nx141), .A1 (b[1]), .A2 (a[1])) ;
    XR2T0 ix9 (.X (nx8), .A1 (b[3]), .A2 (a[3])) ;
    AO2I0 ix159 (.X (nx158), .A1 (b[3]), .A2 (a[3]), .B1 (nx8), .B2 (nx38)) ;
    OAI2N0 ix39 (.X (nx38), .A1 (nx161), .A2 (nx163), .B1 (nx165), .B2 (nx30)) ;
    IV1N0 ix162 (.X (nx161), .A (b[2])) ;
    IV1N0 ix164 (.X (nx163), .A (a[2])) ;
    XN2R0 ix166 (.X (nx165), .A1 (b[2]), .A2 (a[2])) ;
    AO2I0 ix31 (.X (nx30), .A1 (a[1]), .A2 (b[1]), .B1 (nx168), .B2 (nx12)) ;
    IV1N0 ix169 (.X (nx168), .A (nx18)) ;
    XR2T0 ix13 (.X (nx12), .A1 (b[1]), .A2 (a[1])) ;
    NR2R0 ix184 (.X (nx183), .A1 (nx102), .A2 (nx98)) ;
    XR2T0 ix103 (.X (nx102), .A1 (a[0]), .A2 (b[0])) ;
    XN2R0 ix99 (.X (nx98), .A1 (nx12), .A2 (nx18)) ;
    XR2T0 ix188 (.X (nx187), .A1 (nx10), .A2 (nx30)) ;
    IV1NP ix234 (.X (nx199), .A (b[6])) ;
    IV1NP ix235 (.X (nx200), .A (a[6])) ;
    AO2I1 ix236 (.X (nx201), .A1 (a[6]), .A2 (nx199), .B1 (b[6]), .B2 (nx200)) ;
    OR2T0 ix237 (.X (nx202), .A1 (b[5]), .A2 (a[5])) ;
    IV1NP ix238 (.X (nx203), .A (a[4])) ;
    IV1NP ix239 (.X (nx204), .A (b[4])) ;
    NR2Q1 ix240 (.X (nx205), .A1 (a[4]), .A2 (b[4])) ;
    IV1NP ix241 (.X (nx206), .A (b[5])) ;
    IV1NP ix242 (.X (nx207), .A (a[5])) ;
    OAI5N1 ix243 (.X (nx208), .A1 (nx203), .A2 (nx204), .B1 (nx205), .B2 (nx232)
           , .C1 (nx206), .C2 (nx207)) ;
    AN2T0 ix244 (.X (nx209), .A1 (b[5]), .A2 (a[5])) ;
    OAI2N1 nx174_rename (.X (nx174), .A1 (nx203), .A2 (nx204), .B1 (nx205), .B2 (
           nx232)) ;
    OAOI1 ix245 (.X (nx210), .A1 (nx209), .A2 (nx174), .B (nx202), .C (nx201)) ;
    AO3I1 ix246 (.X (nx211), .A1 (nx201), .A2 (nx202), .A3 (nx208), .B (nx210)
          ) ;
    NR2Q1 ix247 (.X (nx212), .A1 (b[5]), .A2 (a[5])) ;
    AN2T0 ix248 (.X (nx213), .A1 (a[4]), .A2 (b[4])) ;
    NR2R1 ix249 (.X (nx214), .A1 (nx213), .A2 (nx233)) ;
    OAI2N1 ix250 (.X (nx215), .A1 (nx212), .A2 (nx209), .B1 (nx214), .B2 (nx205)
           ) ;
    NR2Q1 ix251 (.X (nx216), .A1 (nx206), .A2 (a[5])) ;
    NR2Q1 ix252 (.X (nx217), .A1 (nx207), .A2 (b[5])) ;
    OAI5N1 ix253 (.X (nx218), .A1 (a[4]), .A2 (b[4]), .B1 (nx213), .B2 (nx233),
           .C1 (nx216), .C2 (nx217)) ;
    IV1N2 ix254 (.X (nx219), .A (nx232)) ;
    IV1NP ix255 (.X (nx220), .A (a[4])) ;
    IV1NP ix256 (.X (nx221), .A (b[4])) ;
    AO2I1 ix257 (.X (nx222), .A1 (b[4]), .A2 (nx220), .B1 (a[4]), .B2 (nx221)) ;
    AO1I1 ix258 (.X (nx223), .A1 (a[4]), .A2 (b[4]), .B (nx205)) ;
    OAI2N1 nx88_rename (.X (nx88), .A1 (nx219), .A2 (nx222), .B1 (nx223), .B2 (
           nx232)) ;
    OR2T0 ix259 (.X (nx224), .A1 (nx8), .A2 (nx131)) ;
    ND2N1 ix260 (.X (nx225), .A1 (nx8), .A2 (nx131)) ;
    AO2I1 ix261 (.X (nx226), .A1 (nx224), .A2 (nx225), .B1 (nx183), .B2 (nx187)
          ) ;
    ND4N0 ix262 (.X (nx227), .A1 (nx215), .A2 (nx218), .A3 (nx88), .A4 (nx226)
          ) ;
    AO2LP ix263 (.X (nx228), .A1 (a[6]), .A2 (b[6]), .B1 (b[7]), .B2 (a[7])) ;
    OAI5N1 ix264 (.X (nx229), .A1 (a[4]), .A2 (b[4]), .B1 (nx213), .B2 (nx233),
           .C1 (b[5]), .C2 (a[5])) ;
    ND2N1 ix265 (.X (nx230), .A1 (b[5]), .A2 (a[5])) ;
    AO1A0 ix266 (.X (nx231), .A1 (nx229), .A2 (nx230), .B (nx201)) ;
    OAI3N1 x_rename_rename (.X (x), .A1 (nx211), .A2 (nx227), .B1 (nx228), .B2 (
           nx231)) ;
    WGT1 ix267 (.X (nx232), .CK (nx158)) ;
    WGT1 ix268 (.X (nx233), .CK (nx44)) ;
endmodule


////////////////////////////////////////////////////////////////////////////////
/// Synthesis of Conditional Operator

// The conditional operator is synthesized as a multiplexor.
//
//
// :Sample: assign x = s ? a : b;
// Two-input mux, ~~x~~ s is control, a and b are data inputs,

// The following code containing chained conditional operators
// usually synthesizes to a single multiple-input multiplexor.
//
```

```
// :Sample: assign x =  s==0 ? foo : s==1 ? bar : s==3 ? foobar : other;


//
// Verilog description for ALU module.
//    a + b if input op = 0
//    a - b if input op = 1.
//    a     if input op = 2
//    b     otherwise.


// :Example:
//
// Human-written code for an ALU module.

module alu(x,a,b,op);
   input [7:0] a,b;
   input [1:0] op;
   output [8:0] x;

   assign       x =
                op == 0 ? a + b :
                op == 1 ? a - b :
                op == 2 ? a     : b;

endmodule


// :Example:
//
// Simplified output of inference stage.
//
// Note that conditional operator replaced with a multiplexor.
// (Leonardo uses the conditional operator in its inferred code.)

module alu_inf_simple(x,a,b,op);
   input [7:0] a,b;
   input [1:0] op;
   output [8:0] x;

   wire [8:0]   aplusb,aminusb;

   generic_sum_8 gs1(aplusb,a,b);
   generic_diff_8 gd1(aminusb,a,b);
   generic_mux_9_4 mux(x,op,aplusb,aminusb,a,b);

endmodule


// :Example:
//
// The actual output of the inference stage.  Leonardo still uses a
// conditional operator in places and uses a data selector with one
// bit of control for each data input rather than a mux with a binary
// control input

module alu_inf_real ( x, a, b, op ) ;

    output [8:0]x ;
    input [7:0]a ;
    input [7:0]b ;
    input [1:0]op ;

    wire GND, nx5, nx6, nx7, nx8, nx9, nx10, nx11, nx12, nx13, PWR, nx19, nx20,
         nx21, nx22, nx23, nx24, nx25, nx26, nx27, nx46, nx120, nx121, nx122,
         nx123, nx132, nx134, NOT_nx134;
    wire [2:0] \$dummy ;
    assign GND = 0 ;
    eq_2u_2u x_eq_0 (.a ({op[1],op[0]}), .b ({GND,GND}), .d (nx120)) ;
    add_9u_9u_9u_0_0 x_add_1 (.cin (GND), .a ({GND,a[7],a[6],a[5],a[4],a[3],a[2]
                      ,a[1],a[0]}), .b ({GND,b[7],b[6],b[5],b[4],b[3],b[2],b[1],
                      b[0]}), .d ({nx5,nx6,nx7,nx8,nx9,nx10,nx11,nx12,nx13}), .cout (
                      \$dummy [0])) ;
    assign PWR = 1 ;
    eq_2u_2u x_eq_2 (.a ({op[1],op[0]}), .b ({GND,PWR}), .d (nx121)) ;
    sub_10s_10s_10s_0 x_sub_3 (.cin (PWR), .a ({GND,GND,a[7],a[6],a[5],a[4],a[3]
                       ,a[2],a[1],a[0]}), .b ({GND,GND,b[7],b[6],b[5],b[4],b[3],
                       b[2],b[1],b[0]}), .d ({\$dummy [1],nx19,nx20,nx21,nx22,
                       nx23,nx24,nx25,nx26,nx27}), .cout (\$dummy [2])) ;
    eq_2u_2u x_eq_4 (.a ({op[1],op[0]}), .b ({PWR,GND}), .d (nx122)) ;
    assign x[8] = nx120 ? nx5 : nx46 ;
    and (nx46, nx19, nx121) ;
    select_4_4 modgen_select_6 (.a ({nx120,nx121,nx122,nx123}), .b ({nx6,nx20,
               a[7],b[7]}), .d (x[7])) ;
    select_4_4 modgen_select_7 (.a ({nx120,nx121,nx122,nx123}), .b ({nx7,nx21,
               a[6],b[6]}), .d (x[6])) ;
    select_4_4 modgen_select_8 (.a ({nx120,nx121,nx122,nx123}), .b ({nx8,nx22,
               a[5],b[5]}), .d (x[5])) ;
    select_4_4 modgen_select_9 (.a ({nx120,nx121,nx122,nx123}), .b ({nx9,nx23,
               a[4],b[4]}), .d (x[4])) ;
    select_4_4 modgen_select_10 (.a ({nx120,nx121,nx122,nx123}), .b ({nx10,nx24,
               a[3],b[3]}), .d (x[3])) ;
    select_4_4 modgen_select_11 (.a ({nx120,nx121,nx122,nx123}), .b ({nx11,nx25,
               a[2],b[2]}), .d (x[2])) ;
    select_4_4 modgen_select_12 (.a ({nx120,nx121,nx122,nx123}), .b ({nx12,nx26,
               a[1],b[1]}), .d (x[1])) ;
    select_4_4 modgen_select_13 (.a ({nx120,nx121,nx122,nx123}), .b ({nx13,nx27,
               a[0],b[0]}), .d (x[0])) ;
    or (nx132, nx120, nx121) ;
    or (nx134, nx132, nx122) ;
    assign NOT_nx134 = ~nx134 ;
    and (nx123, NOT_nx134, PWR) ;
endmodule


// :Example:
//
// Simplified output of technology mapping.
//
// Unfortunately, Fab Fab does not have a 9-bit multiplexor in their
// technology library and so a 12-bit mux is used instead.

module alu_tech(x,a,b,op);
   input [7:0] a,b;
   input [1:0] op;
   output [8:0] x;

   wire [8:0]   aplusb,aminusb;
   wire [2:0]   dummy;

   fab_fab_sum_8 gs1(aplusb,a,b);
   fab_fab_diff_8 gd1(aminusb,a,b);
   fab_fab_mux_12_4 mux({dummy,x},op,{3'b0,aminusb},{3'b0,aplusb},
                        {3'b0,a},{3'b0,b});


endmodule


// :Example:
//
// Simplified output of optimization step:
//
// In this case the (fictional) optimizer is clever. Instead of using
// an adder and a subtractor, it just negates b and uses a mux to
// select the negated b or the regular b.

module alu_opt(x,a,b,op);
   input [7:0] a,b;
   input [1:0] op;
   output [8:0] x;

   wire [7:0]   minusb, adder_input_1, adder_input_2;
   wire         zip;

   fab_fab_neg_8 neg(minusb,b);
   fab_fab_and_2 and1(zip,op[0],op[1]);
   fab_fab_mux_8_2 mux1(adder_input_1,zip,a,8'b0);
   fab_fab_mux_8_4 mux2(adder_input_2,op,b,minusb,8'b0,b);
   fab_fab_sum_8 gs1(x,adder_input_1,adder_input_2);

endmodule

// :Example:
//
// The actual optimized description.  The synthesis program decided
// to break everything down into gates and optimize that.  There
// is no need to trace through the module below to see how it works.

module alu ( x, a, b, op ) ;

    output [8:0]x ;
    input [7:0]a ;
    input [7:0]b ;
    input [1:0]op ;

    wire nx2, nx4, nx10, nx12, nx14, nx16, nx18, nx20, nx26, nx32, nx40, nx48,
         nx60, nx74, nx80, nx90, nx122, nx152, nx178, nx186, nx208, nx234, nx254,
         nx274, nx288, nx308, nx257, nx261, nx263, nx265, nx269, nx275, nx277,
         nx279, nx281, nx285, nx287, nx297, nx299, nx301, nx303, nx307, nx311,
         nx313, nx315, nx325, nx327, nx331, nx333, nx335, nx337, nx339, nx345,
         nx350, nx352, nx355, nx357, nx360, nx362, nx364, nx366, nx369, nx371,
         nx373, nx375, nx378, nx381, nx383, nx386, nx388, nx391, nx395, nx397,
         nx401, nx403, nx407, nx409, nx432, nx434, nx436, nx440, nx454, nx455,
         nx456, nx457, nx458, nx459, nx460, nx461, nx462, nx427, nx463, nx464,
         nx465, nx466, nx467, nx468, nx160, nx414, nx293, nx469, nx470, nx471,
         nx472, nx473, nx474, nx475, nx476, nx273, nx393, nx477, nx478, nx479;

    OAI3N0 ix207 (.X (x[0]), .A1 (nx32), .A2 (op[1]), .B1 (nx261), .B2 (nx269)
           ) ;
    ND2N0 ix33 (.X (nx32), .A1 (b[0]), .A2 (nx257)) ;
    IV1N0 ix258 (.X (nx257), .A (a[0])) ;
    AOA4I0 ix262 (.X (nx261), .A1 (nx263), .A2 (nx265), .B (nx478), .C (a[0])) ;
    IV1N0 ix264 (.X (nx263), .A (b[0])) ;
    IV1N0 ix266 (.X (nx265), .A (op[1])) ;
    NR2R1 ix179 (.X (nx178), .A1 (nx265), .A2 (op[0])) ;
    ND3N0 ix270 (.X (nx269), .A1 (b[0]), .A2 (op[0]), .A3 (op[1])) ;
    OAI3N0 ix227 (.X (x[1]), .A1 (nx273), .A2 (nx275), .B1 (nx281), .B2 (nx287)
           ) ;
    XN2R0 ix276 (.X (nx275), .A1 (nx277), .A2 (nx279)) ;
    XR2T0 ix278 (.X (nx277), .A1 (a[1]), .A2 (b[1])) ;
    NR2R0 ix280 (.X (nx279), .A1 (nx263), .A2 (a[0])) ;
    AO2I0 ix282 (.X (nx281), .A1 (b[1]), .A2 (nx477), .B1 (a[1]), .B2 (nx478)) ;
    IV1N0 ix187 (.X (nx186), .A (nx285)) ;
    ND2N0 ix286 (.X (nx285), .A1 (op[0]), .A2 (op[1])) ;
    ND2N0 ix288 (.X (nx287), .A1 (nx208), .A2 (nx293)) ;
    XN2R0 ix209 (.X (nx208), .A1 (nx26), .A2 (nx277)) ;
    ND2N0 ix27 (.X (nx26), .A1 (b[0]), .A2 (a[0])) ;
    OAI3N0 ix247 (.X (x[2]), .A1 (nx297), .A2 (nx4), .B1 (nx313), .B2 (nx315)) ;
    XN2R0 ix298 (.X (nx297), .A1 (nx299), .A2 (nx307)) ;
    OAI2N0 ix300 (.X (nx299), .A1 (nx301), .A2 (nx303), .B1 (nx20), .B2 (nx26)
           ) ;
    IV1N0 ix302 (.X (nx301), .A (b[1])) ;
```

```
IV1N0 ix304 (.X (nx303), .A (a[1])) ;
XN2R0 ix21 (.X (nx20), .A1 (a[1]), .A2 (b[1])) ;
XR2T0 ix308 (.X (nx307), .A1 (a[2]), .A2 (b[2])) ;
ND2N0 ix5 (.X (nx4), .A1 (nx311), .A2 (nx265)) ;
IV1N0 ix312 (.X (nx311), .A (op[0])) ;
AO2I0 ix314 (.X (nx313), .A1 (b[2]), .A2 (nx477), .B1 (a[2]), .B2 (nx478)) ;
ND2N0 ix316 (.X (nx315), .A1 (nx2), .A2 (nx234)) ;
NR2R0 ix3 (.X (nx2), .A1 (op[1]), .A2 (nx311)) ;
XN2R0 ix235 (.X (nx234), .A1 (nx307), .A2 (nx40)) ;
OAI2N1 ix41 (.X (nx40), .A1 (nx303), .A2 (b[1]), .B1 (nx277), .B2 (nx279)) ;
OAI3N0 ix267 (.X (x[3]), .A1 (nx325), .A2 (nx4), .B1 (nx337), .B2 (nx339)) ;
XN2R0 ix326 (.X (nx325), .A1 (nx327), .A2 (nx122)) ;
XR2T0 ix328 (.X (nx327), .A1 (a[3]), .A2 (b[3])) ;
OAI1A0 ix123 (.X (nx122), .A1 (nx331), .A2 (nx333), .B (nx335)) ;
IV1N0 ix332 (.X (nx331), .A (a[2])) ;
IV1N0 ix334 (.X (nx333), .A (b[2])) ;
ND2N0 ix336 (.X (nx335), .A1 (nx299), .A2 (nx307)) ;
AO2I0 ix338 (.X (nx337), .A1 (b[3]), .A2 (nx477), .B1 (a[3]), .B2 (nx478)) ;
ND2N0 ix340 (.X (nx339), .A1 (nx2), .A2 (nx254)) ;
XN2R0 ix255 (.X (nx254), .A1 (nx327), .A2 (nx48)) ;
OAI1A0 ix49 (.X (nx48), .A1 (nx331), .A2 (b[2]), .B (nx345)) ;
ND2N0 ix346 (.X (nx345), .A1 (nx18), .A2 (nx40)) ;
XN2R0 ix19 (.X (nx18), .A1 (a[2]), .A2 (b[2])) ;
OAI3N0 ix287 (.X (x[4]), .A1 (nx350), .A2 (nx4), .B1 (nx355), .B2 (nx357)) ;
XN2R0 ix351 (.X (nx350), .A1 (nx352), .A2 (nx14)) ;
AO2I0 ix353 (.X (nx352), .A1 (a[3]), .A2 (b[3]), .B1 (nx327), .B2 (nx122)) ;
XN2R0 ix15 (.X (nx14), .A1 (a[4]), .A2 (b[4])) ;
AO2I0 ix356 (.X (nx355), .A1 (b[4]), .A2 (nx477), .B1 (a[4]), .B2 (nx478)) ;
ND2N0 ix358 (.X (nx357), .A1 (nx2), .A2 (nx274)) ;
XN2R0 ix275 (.X (nx274), .A1 (nx360), .A2 (nx362)) ;
XR2T0 ix361 (.X (nx360), .A1 (a[4]), .A2 (b[4])) ;
OAI2N0 ix363 (.X (nx362), .A1 (b[3]), .A2 (nx364), .B1 (nx366), .B2 (nx327)
       ) ;
IV1N0 ix365 (.X (nx364), .A (a[3])) ;
AO2I0 ix367 (.X (nx366), .A1 (a[2]), .A2 (nx333), .B1 (nx18), .B2 (nx40)) ;
OAI3N0 ix307 (.X (x[5]), .A1 (nx273), .A2 (nx369), .B1 (nx381), .B2 (nx383)
       ) ;
XR2T0 ix370 (.X (nx369), .A1 (nx371), .A2 (nx373)) ;
XR2T0 ix372 (.X (nx371), .A1 (a[5]), .A2 (b[5])) ;
OAI2N1 ix374 (.X (nx373), .A1 (b[4]), .A2 (nx375), .B1 (nx60), .B2 (nx360)
       ) ;
IV1N0 ix376 (.X (nx375), .A (a[4])) ;
AO2I0 ix61 (.X (nx60), .A1 (nx378), .A2 (a[3]), .B1 (nx48), .B2 (nx16)) ;
IV1N0 ix379 (.X (nx378), .A (b[3])) ;
XN2R0 ix17 (.X (nx16), .A1 (a[3]), .A2 (b[3])) ;
AO2I0 ix382 (.X (nx381), .A1 (b[5]), .A2 (nx477), .B1 (a[5]), .B2 (nx478)) ;
ND2N0 ix384 (.X (nx383), .A1 (nx288), .A2 (nx293)) ;
XR2T0 ix289 (.X (nx288), .A1 (nx371), .A2 (nx386)) ;
OAI2N0 ix387 (.X (nx386), .A1 (nx388), .A2 (nx375), .B1 (nx14), .B2 (nx352)
       ) ;
IV1N0 ix389 (.X (nx388), .A (b[4])) ;
OAI3N0 ix327 (.X (x[6]), .A1 (nx273), .A2 (nx391), .B1 (nx401), .B2 (nx403)
       ) ;
XR2T0 ix392 (.X (nx391), .A1 (nx393), .A2 (nx395)) ;
OAI1A0 ix396 (.X (nx395), .A1 (b[5]), .A2 (nx397), .B (nx74)) ;
IV1N0 ix398 (.X (nx397), .A (a[5])) ;
ND2N0 ix75 (.X (nx74), .A1 (nx373), .A2 (nx12)) ;
XN2R0 ix13 (.X (nx12), .A1 (a[5]), .A2 (b[5])) ;
AO2I0 ix402 (.X (nx401), .A1 (b[6]), .A2 (nx477), .B1 (a[6]), .B2 (nx478)) ;
ND2N0 ix404 (.X (nx403), .A1 (nx308), .A2 (nx293)) ;
XR2T0 ix309 (.X (nx308), .A1 (nx393), .A2 (nx479)) ;
OAI1A0 ix153 (.X (nx152), .A1 (nx397), .A2 (nx407), .B (nx409)) ;
IV1N0 ix408 (.X (nx407), .A (b[5])) ;
ND2N0 ix410 (.X (nx409), .A1 (nx371), .A2 (nx386)) ;
AO2I0 ix81 (.X (nx80), .A1 (nx407), .A2 (a[5]), .B1 (nx373), .B2 (nx12)) ;
OAI2N0 ix175 (.X (x[8]), .A1 (nx432), .A2 (nx434), .B1 (nx4), .B2 (nx440)) ;
ND2N0 ix433 (.X (nx432), .A1 (nx265), .A2 (op[0])) ;
OAI2N0 ix435 (.X (nx434), .A1 (b[7]), .A2 (nx436), .B1 (nx90), .B2 (nx414)
        ) ;
    IV1N0 ix437 (.X (nx436), .A (a[7])) ;
    AO2I0 ix91 (.X (nx90), .A1 (nx427), .A2 (a[6]), .B1 (nx395), .B2 (nx10)) ;
    XN2R0 ix11 (.X (nx10), .A1 (a[6]), .A2 (b[6])) ;
    AO2I0 ix441 (.X (nx440), .A1 (a[7]), .A2 (b[7]), .B1 (nx414), .B2 (nx160)) ;
    IV1NP ix480 (.X (nx454), .A (b[7])) ;
    IV1N2 ix481 (.X (nx455), .A (nx477)) ;
    IV1NP ix482 (.X (nx456), .A (a[7])) ;
    IV1N2 ix483 (.X (nx457), .A (nx478)) ;
    IV1NP ix484 (.X (nx458), .A (a[7])) ;
    IV1NP ix485 (.X (nx459), .A (b[7])) ;
    AO2I1 ix486 (.X (nx460), .A1 (b[7]), .A2 (nx458), .B1 (a[7]), .B2 (nx459)) ;
    IV1NP ix487 (.X (nx461), .A (a[6])) ;
    NR2Q1 ix488 (.X (nx462), .A1 (nx461), .A2 (b[6])) ;
    IV1NP nx427_rename (.X (nx427), .A (b[6])) ;
    NR2Q1 ix489 (.X (nx463), .A1 (nx427), .A2 (a[6])) ;
    NR2R1 ix490 (.X (nx464), .A1 (nx463), .A2 (nx80)) ;
    AN2T0 ix491 (.X (nx465), .A1 (nx265), .A2 (op[0])) ;
    OAI1A1 ix492 (.X (nx466), .A1 (nx462), .A2 (nx464), .B (nx465)) ;
    OR2T0 ix493 (.X (nx467), .A1 (b[6]), .A2 (a[6])) ;
    AN2T0 ix494 (.X (nx468), .A1 (b[6]), .A2 (a[6])) ;
    AO1A0 nx160_rename (.X (nx160), .A1 (nx479), .A2 (nx467), .B (nx468)) ;
    OAI2N1 nx414_rename (.X (nx414), .A1 (nx454), .A2 (a[7]), .B1 (nx456), .B2 (
           b[7])) ;
    NR2Q1 nx293_rename (.X (nx293), .A1 (op[0]), .A2 (op[1])) ;
    ND2N1 ix495 (.X (nx469), .A1 (nx414), .A2 (nx293)) ;
    IV1NP ix496 (.X (nx470), .A (b[6])) ;
    ND2Q1 ix497 (.X (nx471), .A1 (a[6]), .A2 (nx470)) ;
    OAI3N1 ix498 (.X (nx472), .A1 (nx463), .A2 (nx80), .B1 (nx465), .B2 (nx471)
           ) ;
    AO1I1 ix499 (.X (nx473), .A1 (nx479), .A2 (nx467), .B (nx468)) ;
    IV1NP ix500 (.X (nx474), .A (op[0])) ;
    IV1NP ix501 (.X (nx475), .A (op[1])) ;
    ND3N1 ix502 (.X (nx476), .A1 (nx460), .A2 (nx474), .A3 (nx475)) ;
    AOA6I0 x_7__rename_rename (.X (x[7]), .A1 (nx454), .A2 (nx455), .B1 (nx456)
           , .B2 (nx457), .C1 (nx460), .C2 (nx466), .D1 (nx160), .D2 (nx469), .E1 (
           nx414), .E2 (nx472), .F1 (nx473), .F2 (nx476)) ;
    ND2Q1 nx273_rename (.X (nx273), .A1 (nx265), .A2 (op[0])) ;
    OAI2N1 nx393_rename (.X (nx393), .A1 (nx427), .A2 (a[6]), .B1 (nx461), .B2 (
           b[6])) ;
    WGT1 ix503 (.X (nx477), .CK (nx186)) ;
    WGT1 ix504 (.X (nx478), .CK (nx178)) ;
    WGT1 ix505 (.X (nx479), .CK (nx152)) ;
endmodule


////////////////////////////////////////////////////////////////////////////////
/// Synthesis of Delays

 /// Delays

//  Synthesis programs ignore delays.  (Do not use delays in Verilog
//  intended for synthesis.)


// :Example:
//
//  Human-written Verilog code to implement an AND gate with a 3 cycle delay.

module delayed_and(x,a,b);
   input a, b;
   output x;

   assign #3 x = a & b;

endmodule
```

```
// :Example:
//
// Simplified output of inference step.
//
// Note that the delay is gone.  The synthesized logic will very
// likely not have a delay of three.  The delay is determined by the
// target technology, fanout, routing, etc; the human-inserted delay
// is ignored.

module delayed_and_rtl(x,a,b);
   input a, b;
   output x;

   and a1(x,a,b);

endmodule

//  Further steps not shown for the delayed_and.


////////////////////////////////////////////////////////////////////////////////
/// Adder Comparison


//  Adders from last set (l03.v).

//  If synthesis is the goal, use ``another_adder'' because synthesis
//  program will choose a good adder design for the target technology.

//  If ripple_4 is synthesized, the logic will be optimized but it
//  still might not perform as fast as another_adder, though in
//  example below it does.

//  A synthesized version of ripple_4_d will be identical to ripple_4
//  because delays are ignored.

//  Note:
//  Gate-level designs of arithmetic units are shown in class to show
//  how they are built.


// :Example:
//
// Cost and speed of adder described using four BFA's.
//
// Data collected 14 September 2001
// Cost: (Sample ASIC technology)  139 gates.
// Delay:                          1.34 ns

module ripple_4(sum,cout,a,b);
   input [3:0] a, b;
   output [3:0] sum;
   output       cout;

   wire         c0, c1, c2;

   bfa_implicit bfa0(sum[0],c0,a[0],b[0],1'b0);
   bfa_implicit bfa1(sum[1],c1,a[1],b[1],c0);
   bfa_implicit bfa2(sum[2],c2,a[2],b[2],c1);
   bfa_implicit bfa3(sum[3],cout,a[3],b[3],c2);

endmodule


// :Example:
//
// Cost and speed of adder described using the add operator.
```

```
//
// Data collected 14 September 2001
// Cost: (Sample ASIC technology)  139 gates.
// Delay:                          1.55 ns

module another_adder(sum,cout,a,b);
   input [3:0] a, b;
   output [3:0] sum;
   output       cout;

   wire [4:0]   sum_with_carry = a + b;
   assign       sum = sum_with_carry[3:0];
   assign       cout = sum_with_carry[4];

endmodule
```