
Verilog – Combinational Logic

Verilog for Synthesis

Verilog – logic and numbers

- Four-value logic system
 - 0 – logic zero, or false condition
 - 1 – logic 1, or true condition
 - x, X – unknown logic value
 - z, Z - high-impedance state
- Number formats
 - b, B binary
 - d, D decimal (default)
 - h, H hexadecimal
 - o, O octal
- 16'H789A – 16-bit number in hex format
- 1'b0 – 1-bit

Verilog types

- Constants

- `parameter` `DIME = 10;`
- `parameter` `width = 32, nickel = 5;`
- `parameter` `quarter = 8'b0010_0101;`

- Nets

- `wire` `clock, reset_n;`
- `wire[7:0]` `a_bus;`

- Registers

- `reg` `clock, reset_n;`
- `reg[7:0]` `a_bus;`

- Integer

- only for use as general purpose variables in loops
- `integer` `n;`

Operators

- Bitwise

– ~ negation	Verilog	VHDL
– & and	y = a & b;	y = a AND b;
– inclusive or	y = a b;	y = a OR b;
– ^ exclusive or	y = a ^ b;	y = a XOR b;
–	y = ~(a & b);	y = a NAND b;
–	y = ~ a;	y = NOT a;

- Reduction (no direct equivalent in VHDL)

- Accept single bus and return single bit result
 - & and y = & a_bus;
 - ~& nand
 - | or y = | a_bus;
 - ^ exclusive or

Operators (cont'd)

- Relational (return 1 for true, 0 for false)

- < less than, <=
- > greater than >=

- Equality

- == logical equality
- != logical inequality

- Logical Comparison Operators

- ! logical negation
- && logical and
- || logical or

- Arithmetic Operators

- +
- -
- *

Operators (cont'd)

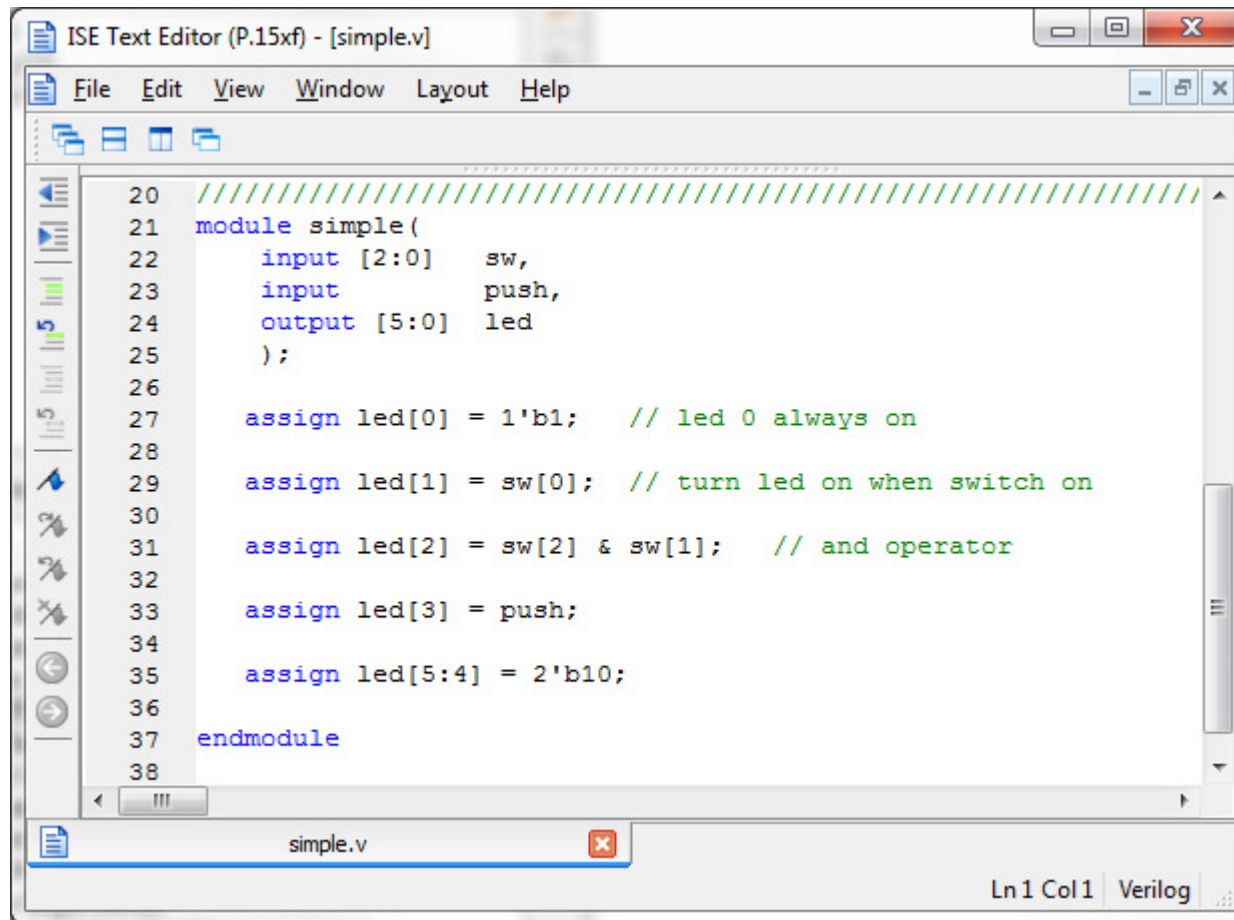
- Shift

- << logical shift left, (<<< arithmetic)
- >> logical shift right (>>> arithmetic)

- Conditional

- Only in Verilog - selects one of pair expressions
- ? :
- Logical expression before ? is evaluated
- If true, the expression before : is assigned to output
- If false, expression after : is assigned to output
 - `Y = (A > B) ? 1 : 0`
 - `Y = (A == B) ? A + B : A - B`

Simple Combinational Example

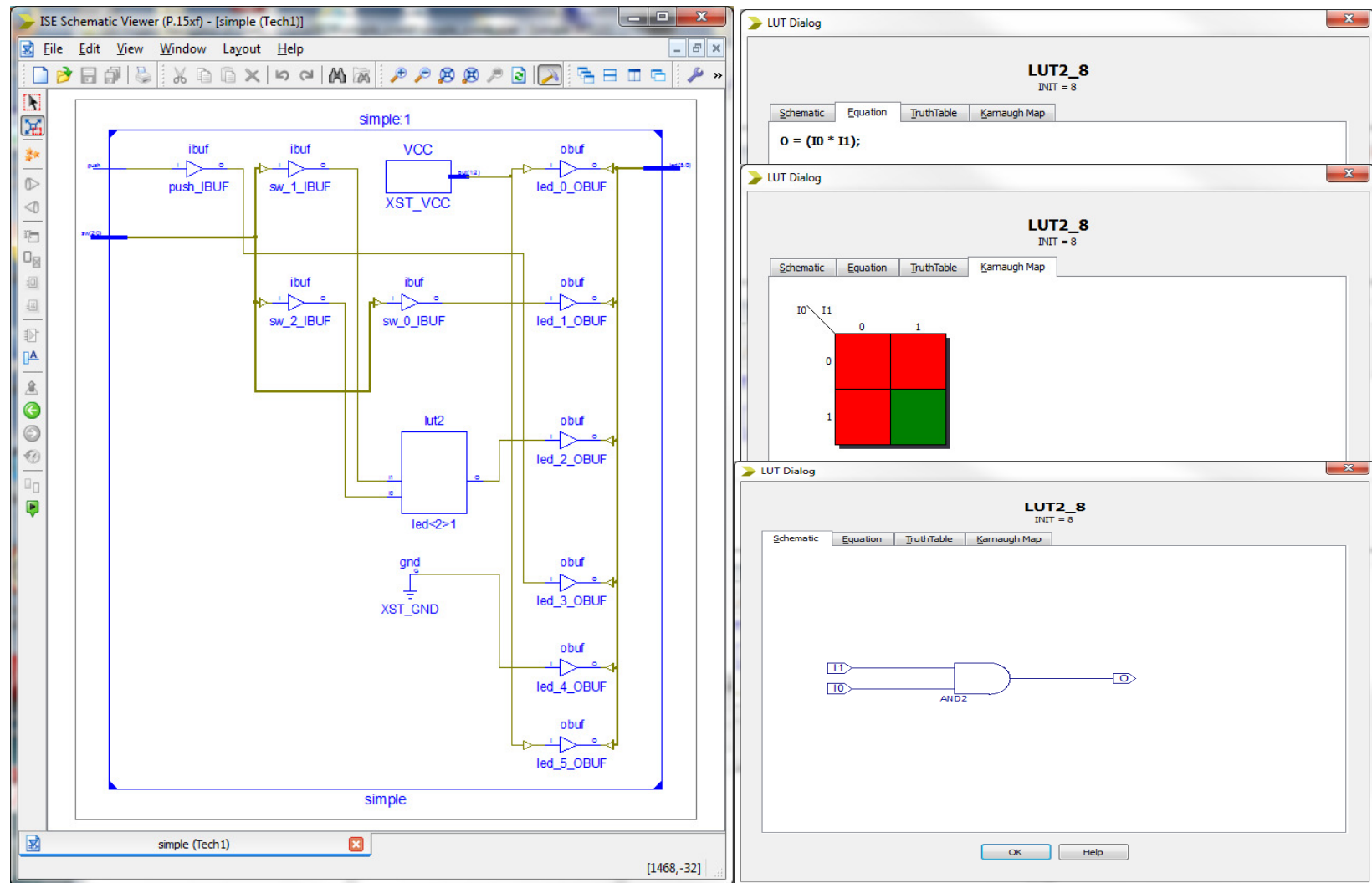


The screenshot shows the ISE Text Editor window titled "ISE Text Editor (P.15xf) - [simple.v]". The window contains a Verilog module named "simple". The code is as follows:

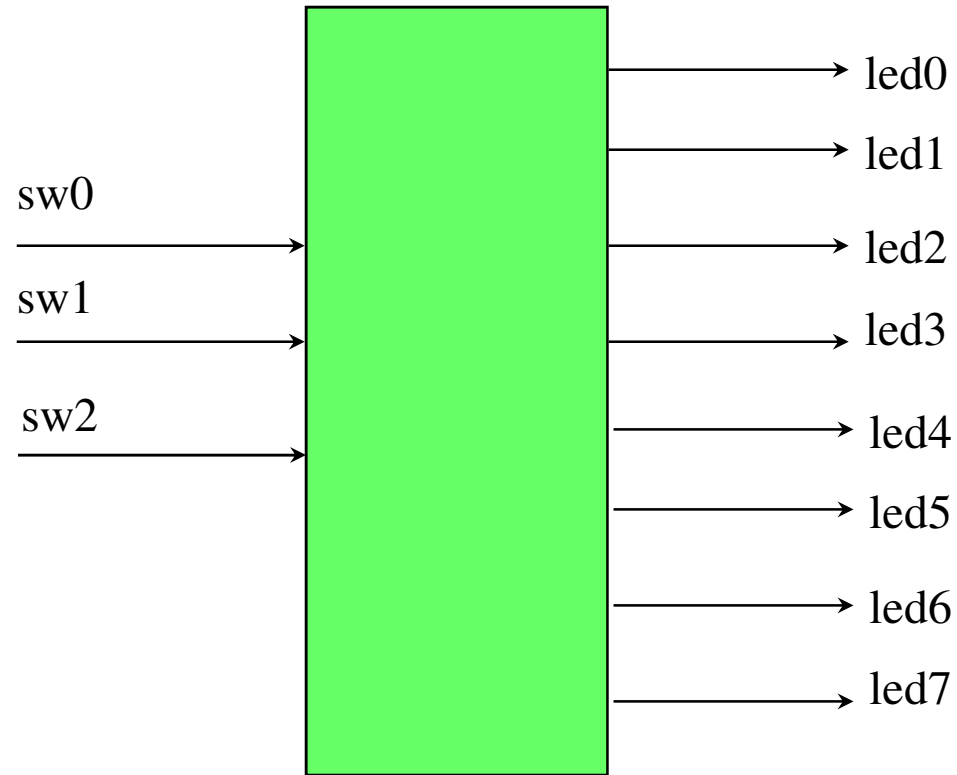
```
20 //////////////////////////////////////////////////
21 module simple(
22     input [2:0]    sw,
23     input          push,
24     output [5:0]   led
25 );
26
27     assign led[0] = 1'b1;    // led 0 always on
28
29     assign led[1] = sw[0];   // turn led on when switch on
30
31     assign led[2] = sw[2] & sw[1]; // and operator
32
33     assign led[3] = push;
34
35     assign led[5:4] = 2'b10;
36
37 endmodule
38
```

The status bar at the bottom indicates "Ln 1 Col 1 Verilog".

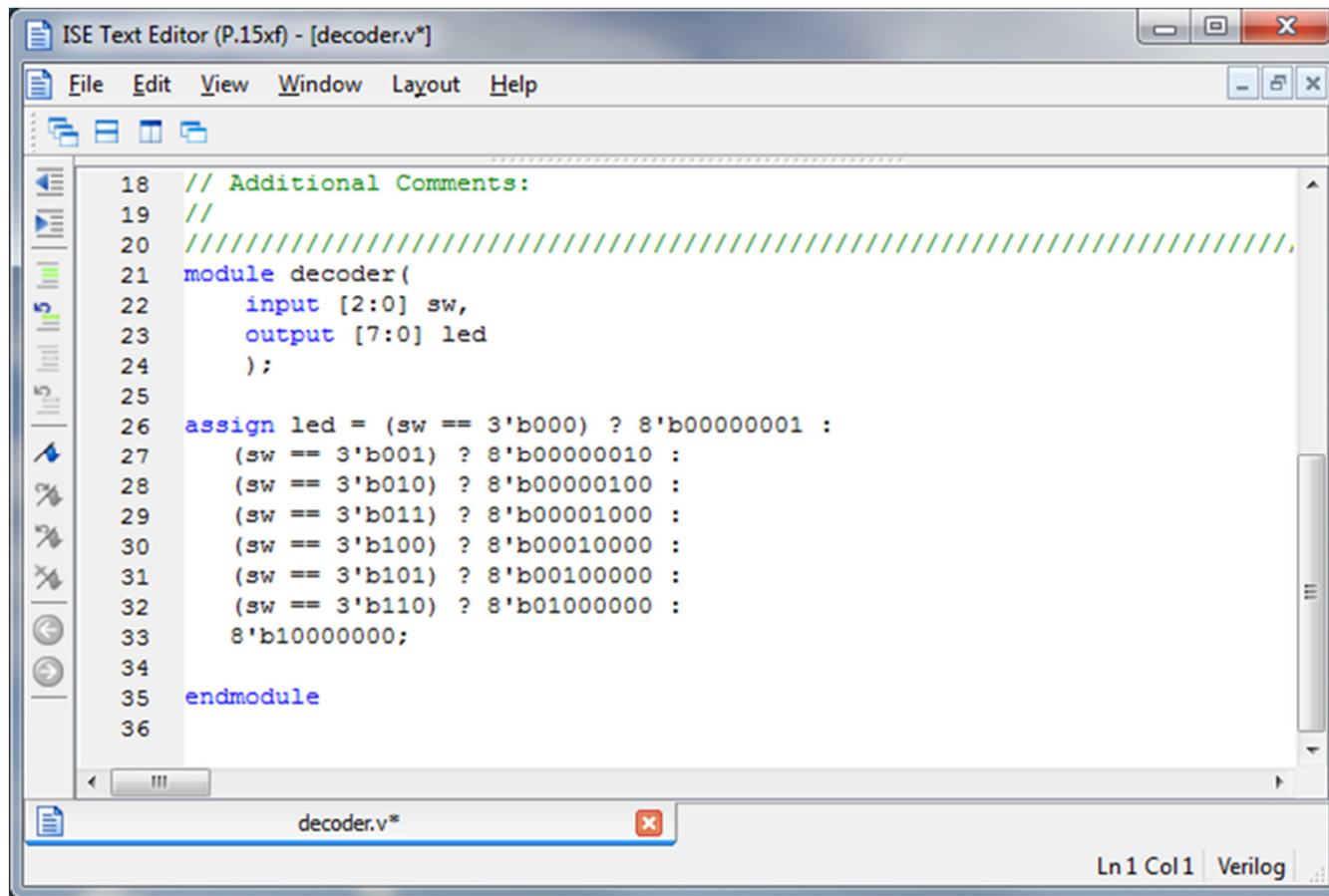
View Technology Schematic



Decoder Tutorial Demo Example



Verilog Source Code



The screenshot shows the ISE Text Editor window with the file name [decoder.v*]. The menu bar includes File, Edit, View, Window, Layout, and Help. The code is as follows:

```
18 // Additional Comments:
19 //
20 ///////////////////////////////////////////////////////////////////,
21 module decoder(
22     input [2:0] sw,
23     output [7:0] led
24 );
25
26 assign led = (sw == 3'b000) ? 8'b00000001 :
27     (sw == 3'b001) ? 8'b00000010 :
28     (sw == 3'b010) ? 8'b00000100 :
29     (sw == 3'b011) ? 8'b00001000 :
30     (sw == 3'b100) ? 8'b00010000 :
31     (sw == 3'b101) ? 8'b00100000 :
32     (sw == 3'b110) ? 8'b01000000 :
33     8'b10000000;
34
35 endmodule
36
```

The status bar at the bottom indicates 'Ln 1 Col 1 Verilog'.

Concurrent statements

- VHDL
 - Process
 - Signal assignments
- Verilog
 - `always` statement
 - Continuous assignment - `assign`

Verilog wire and register data objects

- Wire – net, connects two signals together
 - `wire` `clk`, `en`;
 - `wire` `[15:0]` `a_bus`;
- Reg – register, holds its value from one procedural assignment statement to the next
 - Does not imply a physical register – depends on use
 - `reg` `[7:0]` `b_bus`;

Index and Slice

- VHDL

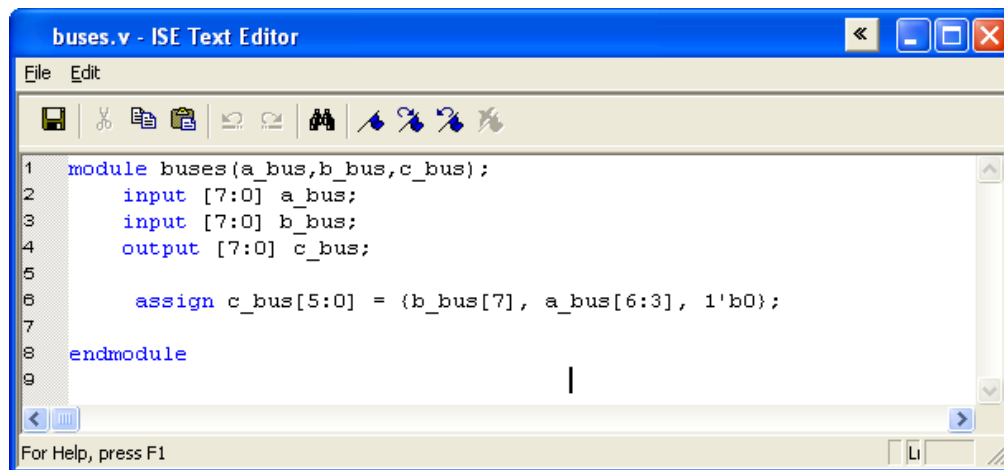
- Use `to` and `downto` to specify slice
- Concatenation `&`

- `c_bus(3 downto 0) <= b_bus(7 downto 4);`
- `c_bus(5 downto 0) <= b_bus(7) & a_bus(6 downto 3) & '0';`

- Verilog

- Use colon `:`
- Concatenation `{,}`

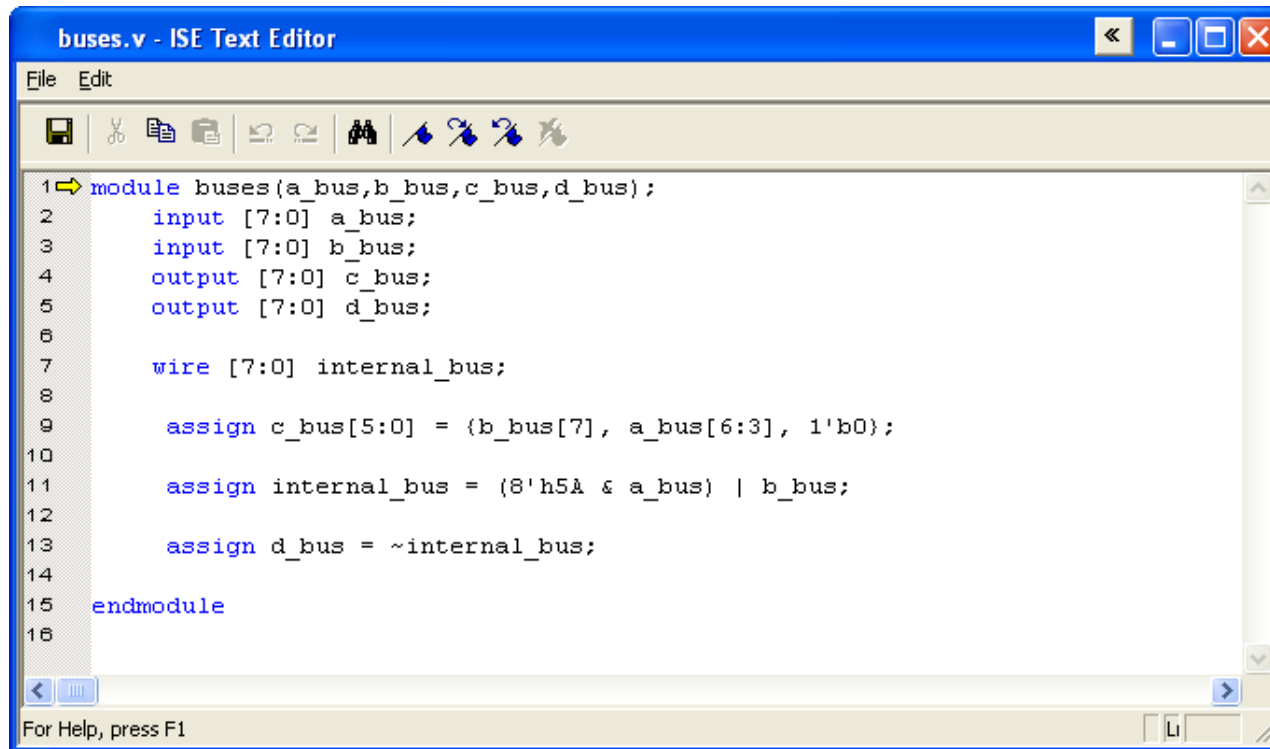
- `assign c_bus[3:0] = b_bus[7:4];`
- `assign c_bus[5:0] = {b_bus[7], a_bus[6:3], 1'b0};`



```
buses.v - ISE Text Editor
File Edit
[Icons]
1 module buses(a_bus,b_bus,c_bus);
2     input [7:0] a_bus;
3     input [7:0] b_bus;
4     output [7:0] c_bus;
5
6     assign c_bus[5:0] = {b_bus[7], a_bus[6:3], 1'b0};
7
8 endmodule
9
For Help, press F1
```

Internal wires

- Declare internal wires:



```
1 module buses(a_bus,b_bus,c_bus,d_bus);
2     input [7:0] a_bus;
3     input [7:0] b_bus;
4     output [7:0] c_bus;
5     output [7:0] d_bus;
6
7     wire [7:0] internal_bus;
8
9     assign c_bus[5:0] = {b_bus[7], a_bus[6:3], 1'b0};
10
11     assign internal_bus = (8'h5A & a_bus) | b_bus;
12
13     assign d_bus = ~internal_bus;
14
15 endmodule
16
```

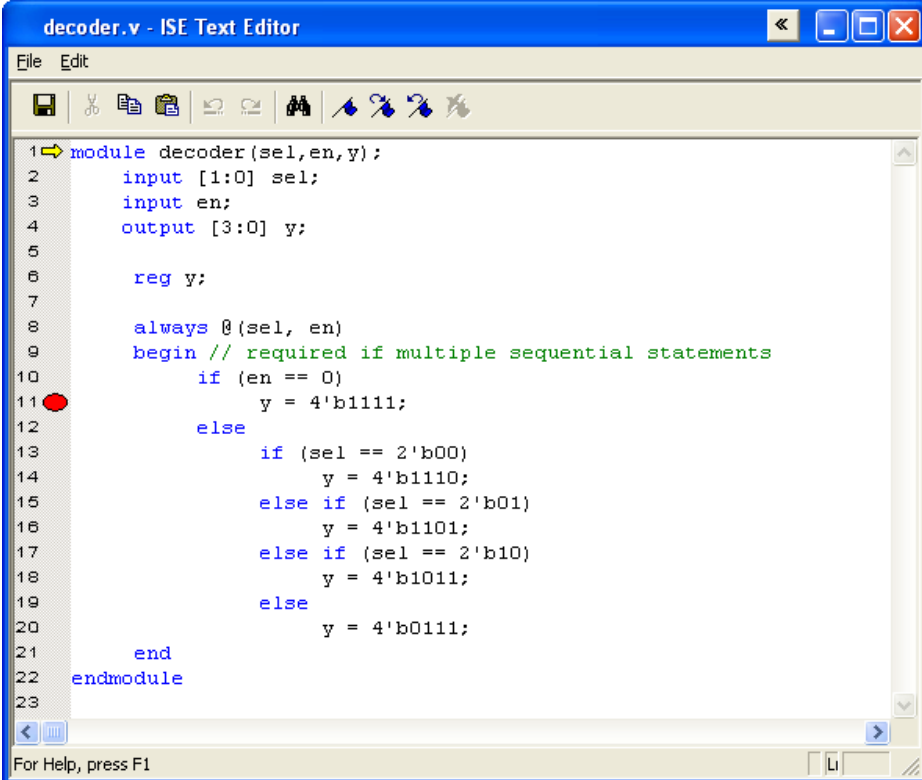
For Help, press F1

Sequential Statements

- VHDL
 - reside in process statement
- Verilog
 - reside in an `always` statement
 - `if` statements (no `endif`)
 - `case` statements (`endcase`)
 - `for`, `repeat while` loop statements
 - Note: use `begin` and `end` to block sequential statements

Decoder – always statement

- 2 to 4 decoder with enable
- Combinational logic using **always** statement with sensitivity list
 - similar to VHDL **process** – for cyclic behavior
 - (@) event control operator
 - **begin .. end** block statement
 - note **reg** for y



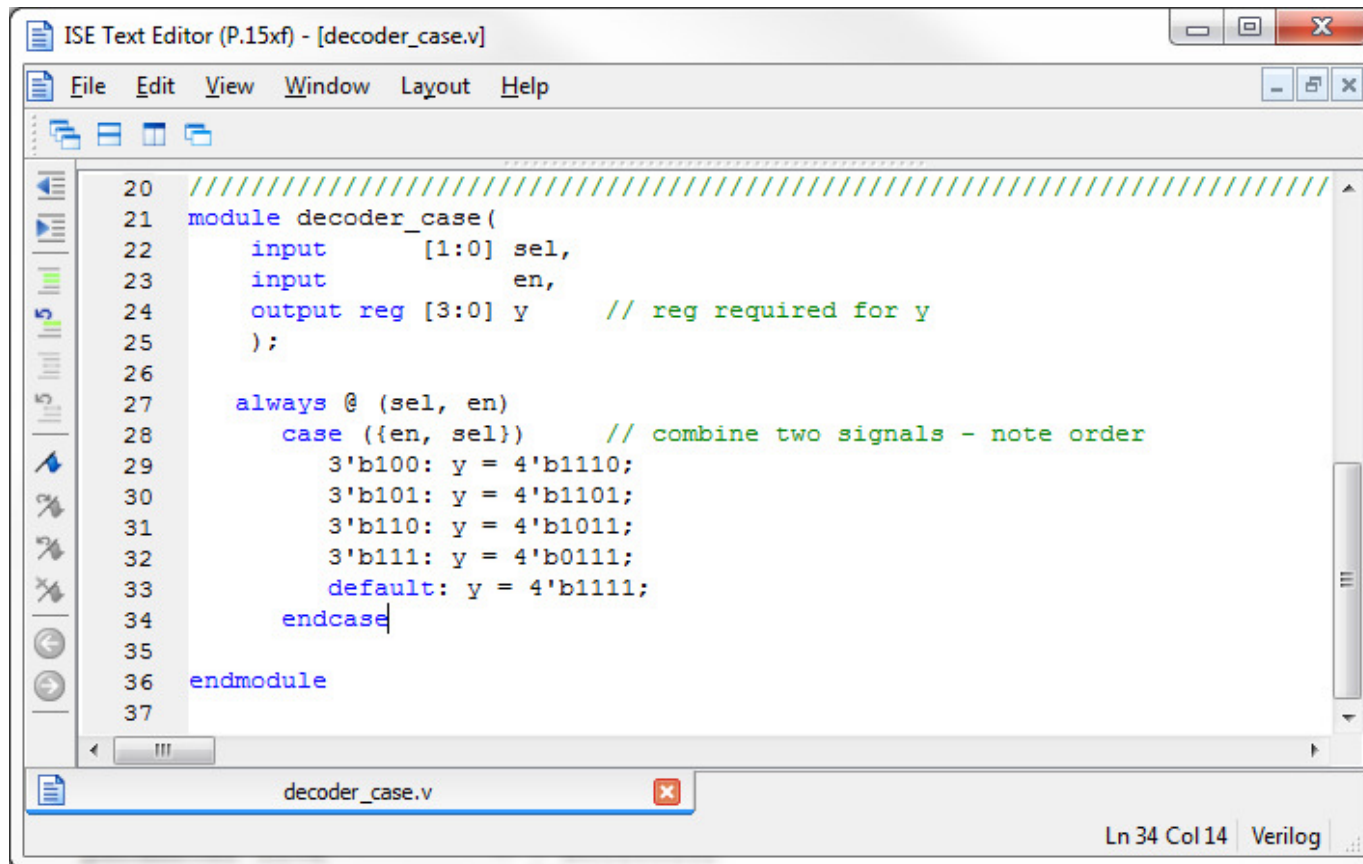
```
1 module decoder(sel,en,y);
2     input [1:0] sel;
3     input en;
4     output [3:0] y;
5
6     reg y;
7
8     always @(sel, en)
9     begin // required if multiple sequential statements
10         if (en == 0)
11             y = 4'b1111;
12         else
13             if (sel == 2'b00)
14                 y = 4'b1110;
15             else if (sel == 2'b01)
16                 y = 4'b1101;
17             else if (sel == 2'b10)
18                 y = 4'b1011;
19             else
20                 y = 4'b0111;
21         end
22     endmodule
23
```


Decoder (cont'd)

- Combinational logic using `always` statement with sensitivity list
 - similar to VHDL `process` – for cyclic behavior
 - (`@`) event control operator
 - `begin .. end` block statement
 - Statements execute sequentially
 - `if` statement
 - `case` statement
 - Note: `case` expression can concatenate signals (`{,}`)
 - Sensitivity list
 - (`a or b or c`)
 - Verilog 2001 allows comma-separated list (`a, b, c`)

Decoder – CASE statement

- CASE is better for this type of design - no priority
 - Exactly same logic produced

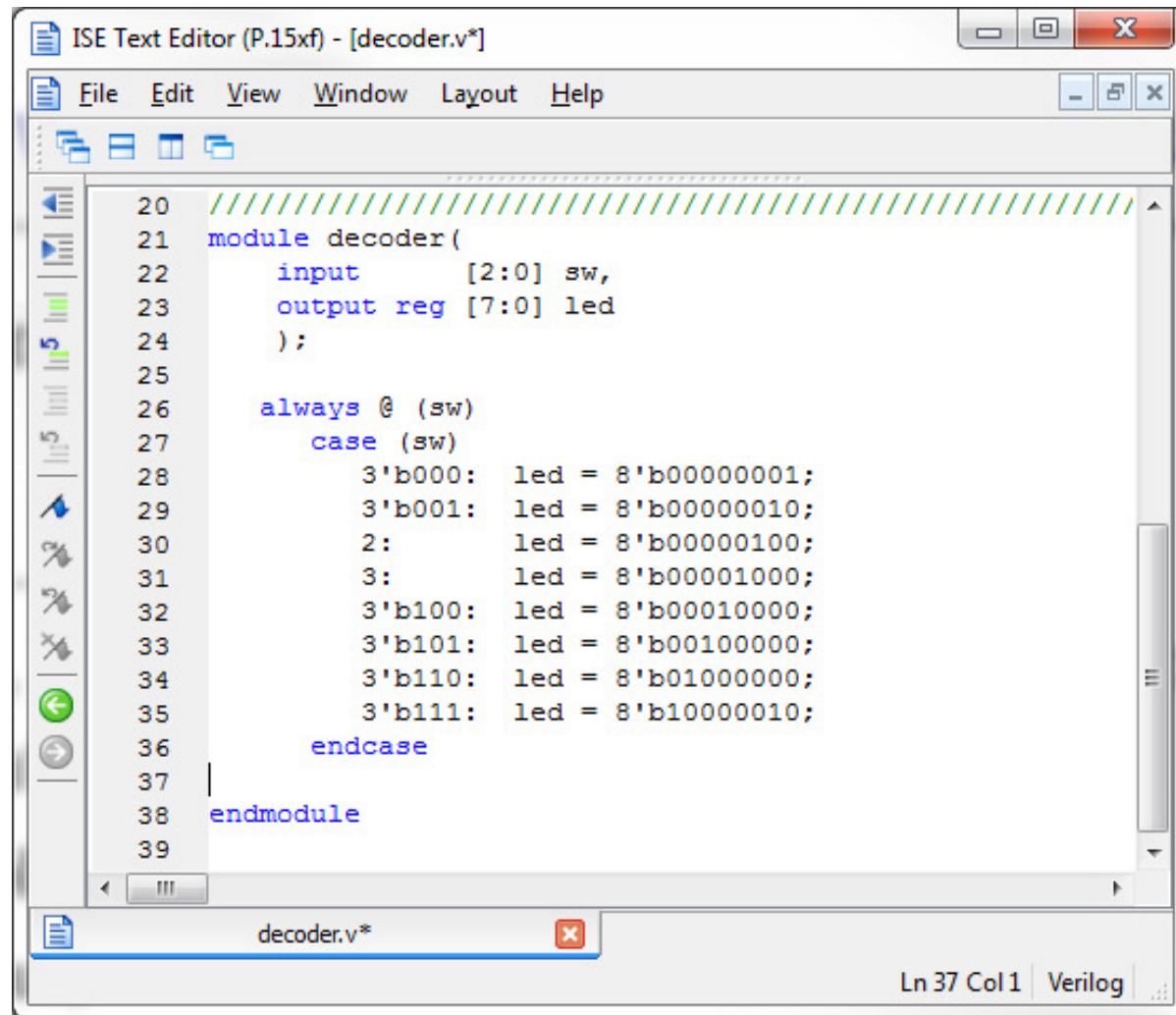


The screenshot shows the ISE Text Editor window titled "ISE Text Editor (P.15xf) - [decoder_case.v]". The editor displays Verilog code for a module named `decoder_case`. The code includes two inputs, `sel` (1-bit) and `en` (1-bit), and one output, `y` (4-bit), which is a register. The logic is implemented using an `always` block triggered by `sel` and `en`, containing a `case` statement that combines these two signals. The case statement has four entries: `3'b100` maps to `4'b1110`, `3'b101` maps to `4'b1101`, `3'b110` maps to `4'b1011`, and `3'b111` maps to `4'b0111`. A default case maps to `4'b1111`. The code is as follows:

```
20 //////////////////////////////////////////////////
21 module decoder_case(
22     input      [1:0] sel,
23     input      en,
24     output reg [3:0] y    // reg required for y
25 );
26
27 always @ (sel, en)
28     case ({en, sel})      // combine two signals - note order
29         3'b100: y = 4'b1110;
30         3'b101: y = 4'b1101;
31         3'b110: y = 4'b1011;
32         3'b111: y = 4'b0111;
33         default: y = 4'b1111;
34     endcase
35
36 endmodule
37
```

The status bar at the bottom indicates "Ln 34 Col 14 Verilog".

Decoder – 3 to 8 with CASE

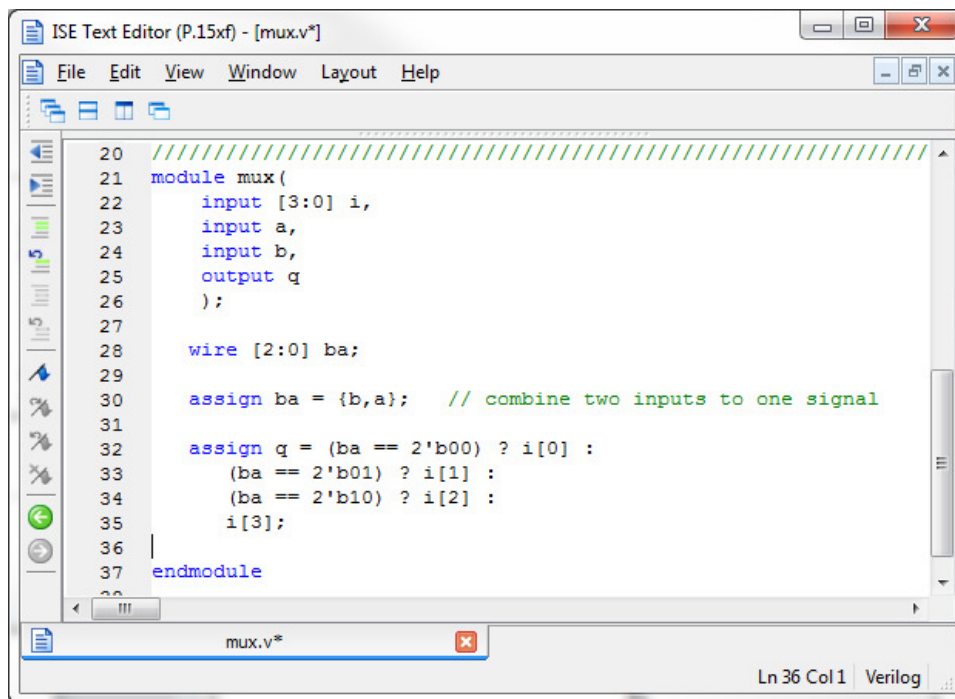


The screenshot shows the ISE Text Editor window titled "ISE Text Editor (P.15xf) - [decoder.v*]". The window contains a Verilog module named "decoder". The code defines an input "sw" of type [2:0] and an output reg "led" of type [7:0]. It uses an "always" block triggered by "sw" to implement a 3-to-8 decoder using a "case" statement. The case statement maps 8 input values (3'b000 to 3'b111) to 8-bit binary outputs (8'b00000001 to 8'b10000010). The status bar at the bottom indicates "Ln 37 Col 1 Verilog".

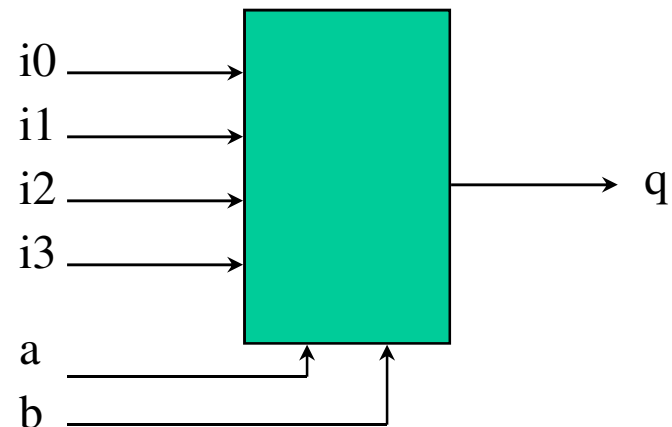
```
20 ////////////////////////////////////////////
21 module decoder(
22     input    [2:0] sw,
23     output reg [7:0] led
24 );
25
26     always @ (sw)
27         case (sw)
28             3'b000: led = 8'b00000001;
29             3'b001: led = 8'b00000010;
30             2:     led = 8'b00000100;
31             3:     led = 8'b00001000;
32             3'b100: led = 8'b00010000;
33             3'b101: led = 8'b00100000;
34             3'b110: led = 8'b01000000;
35             3'b111: led = 8'b10000010;
36         endcase
37
38     endmodule
39
```

MUX example

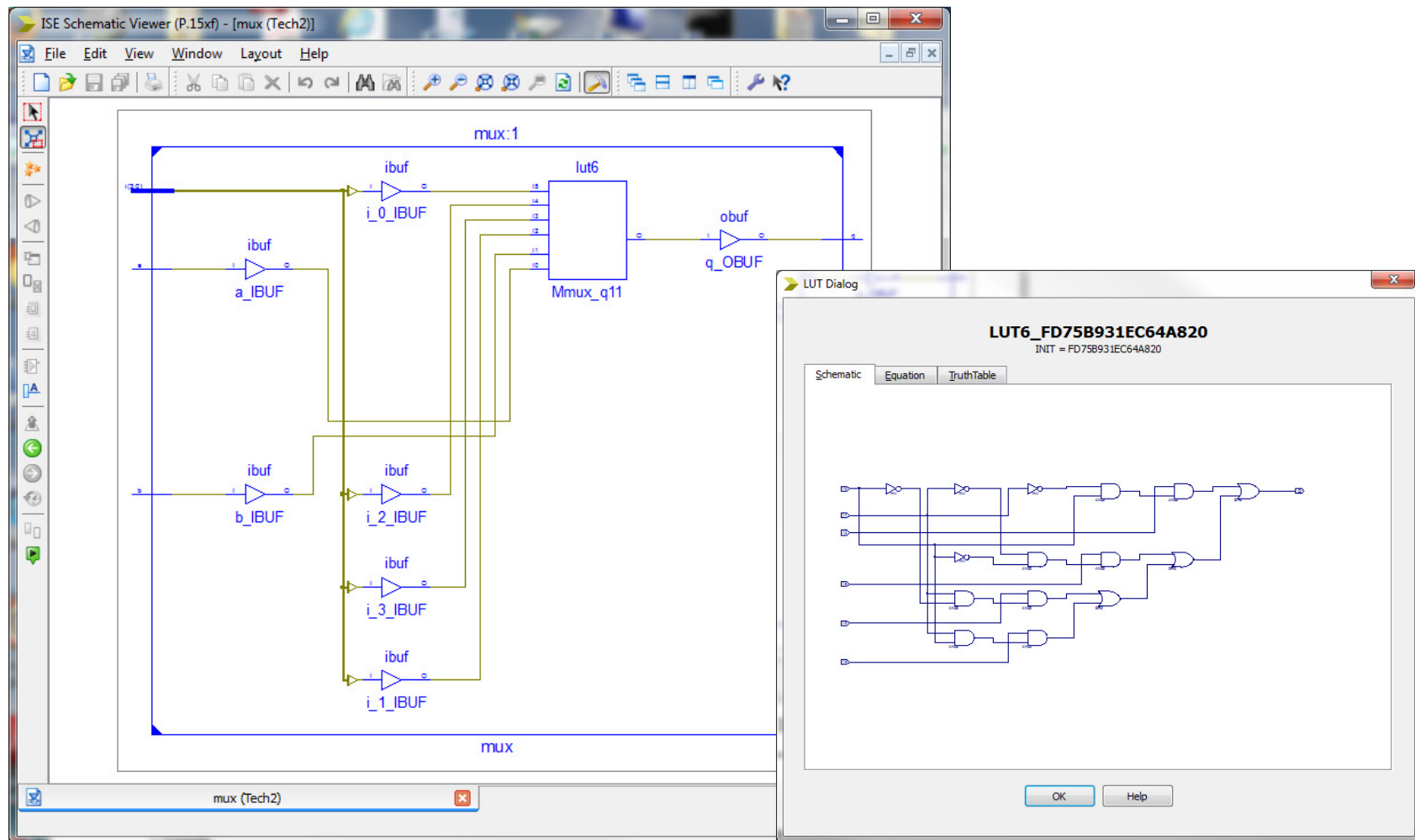
- Example multiplexer with conditional operator
- Selects different values for the target signal
 - priority associated with series of conditions
 - (similar to an IF statement)



```
20 //////////////////////////////////////////////////
21 module mux(
22     input [3:0] i,
23     input a,
24     input b,
25     output q
26 );
27
28     wire [2:0] ba;
29
30     assign ba = {b,a}; // combine two inputs to one signal
31
32     assign q = (ba == 2'b00) ? i[0] :
33               (ba == 2'b01) ? i[1] :
34               (ba == 2'b10) ? i[2] :
35               i[3];
36
37 endmodule
```



Synthesis Results – Technology Schematic



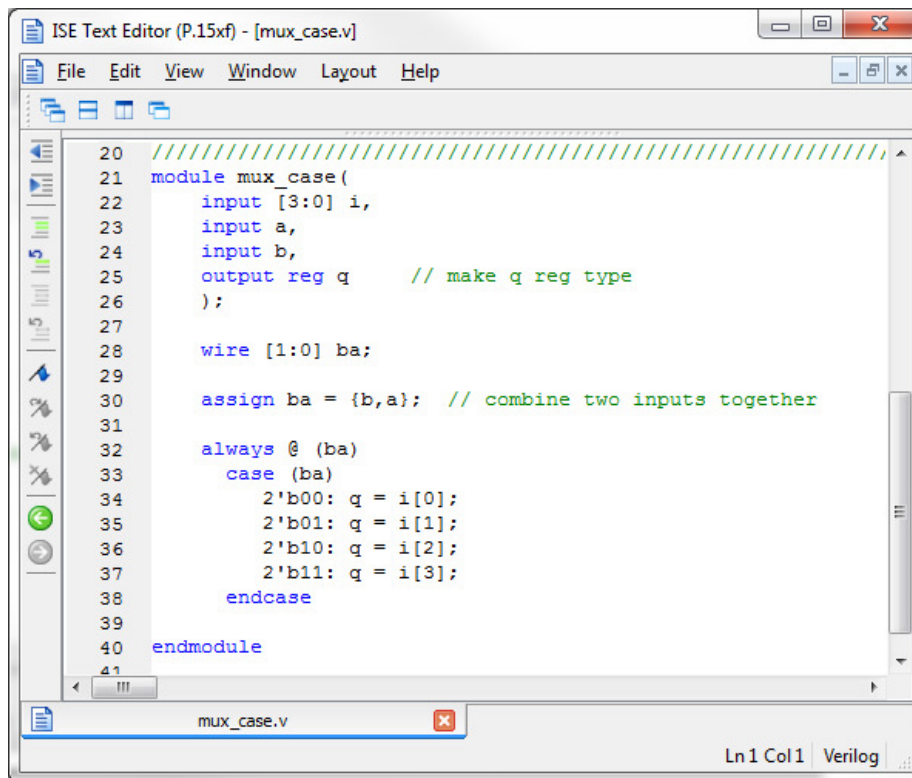
$$O = ((I0 * I1 * I3) + (!I0 * I1 * I4) + (!I0 * !I1 * I5) + (I0 * !I1 * I2));$$

Mux – with CASE statement

- Include all inputs on sensitivity list

Elaborating module <mux_case>.

WARNING:HDLCompiler:91 - "C:\ece3829\mux_case\mux_case.v" Line 34: Signal <i> missing in the sensitivity list is added for synthesis purposes. HDL and post-synthesis simulations may differ as a result.



```
20 //////////////////////////////////////////////////
21 module mux_case(
22     input [3:0] i,
23     input a,
24     input b,
25     output reg q    // make q reg type
26 );
27
28     wire [1:0] ba;
29
30     assign ba = {b,a}; // combine two inputs together
31
32     always @ (ba)
33         case (ba)
34             2'b00: q = i[0];
35             2'b01: q = i[1];
36             2'b10: q = i[2];
37             2'b11: q = i[3];
38         endcase
39
40 endmodule
41
```

Mux – fixed sensitivity list

- Exact same logic produced as using conditional operator

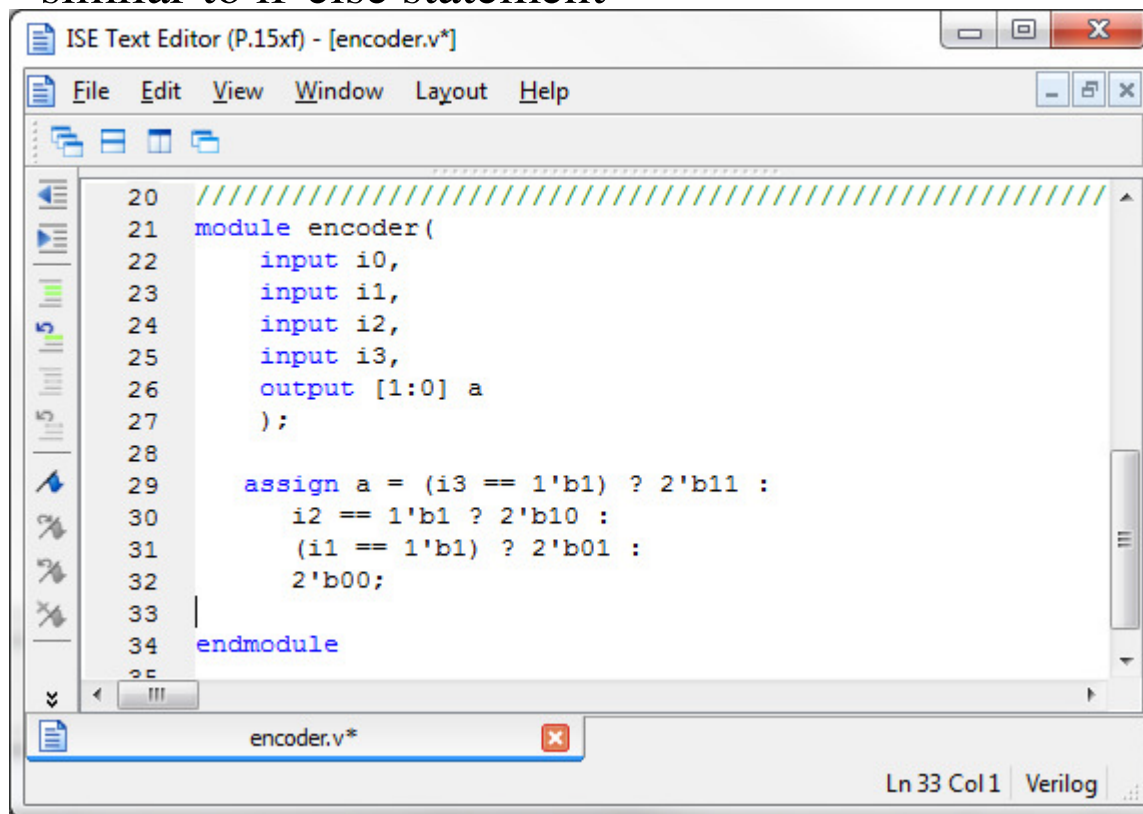
The screenshot displays the ISE Text Editor window with the file `ISE Text Editor (P.15xf) - [mux_case.v*]`. The Verilog code defines a module `mux_case` with inputs `i` (4-bit), `a`, `b`, and output `q` (reg type). It combines `a` and `b` into a 2-bit bus `ba` and uses a case statement to select the output `q` based on `ba`. The sensitivity list is `(ba, i)`.

```
20 //////////////////////////////////////////////////
21 module mux_case(
22     input [3:0] i,
23     input a,
24     input b,
25     output reg q    // make q reg type
26 );
27
28     wire [1:0] ba;
29
30     assign ba = {b,a}; // combine two inputs together
31
32     always @ (ba, i)    // all inputs required on sensitivity
33     case (ba)
34         2'b00: q = i[0];
35         2'b01: q = i[1];
36         2'b10: q = i[2];
37         2'b11: q = i[3];
38     endcase
39
40 endmodule
```

The LUT Dialog window shows the schematic for `LUT6_FD75B931EC64A820`. The schematic is a complex logic diagram representing the 4-to-1 multiplexer logic, showing inputs, internal logic gates, and the output `q`.

Priority Encoder

- Priority Encoder using conditional operator
- Priority order determined by sequence
 - similar to if-else statement



The screenshot shows the ISE Text Editor window titled "ISE Text Editor (P.15xf) - [encoder.v*]". The menu bar includes File, Edit, View, Window, Layout, and Help. The code is written in Verilog and defines a module named 'encoder'. It has four input signals (i0, i1, i2, i3) and one output signal (a, a 2-bit bus [1:0]). The logic is implemented using a conditional operator (?:) to assign the value of 'a' based on the priority of the inputs. The priority is determined by the sequence of inputs: i3 has the highest priority, followed by i2, i1, and i0. The code is as follows:

```
20 //////////////////////////////////////////////////
21 module encoder(
22     input i0,
23     input i1,
24     input i2,
25     input i3,
26     output [1:0] a
27 );
28
29     assign a = (i3 == 1'b1) ? 2'b11 :
30         i2 == 1'b1 ? 2'b10 :
31         (i1 == 1'b1) ? 2'b01 :
32         2'b00;
33
34 endmodule
```

The status bar at the bottom indicates "Ln 33 Col 1 Verilog".

Encoder – Technology Schematic

=====

* HDL Synthesis *

=====

Synthesizing Unit <encoder>.

Related source file is "C:\ece3829\encoder\encoder.v".

WARNING:Xst:647 - Input <i0> is never used. This port will be preserved and left unconnected if it belongs to a top-level block or it belongs to a sub-block and the hierarchy of this sub-block is preserved.

Summary:

inferred 2 Multiplexer(s).

Unit <encoder> synthesized.

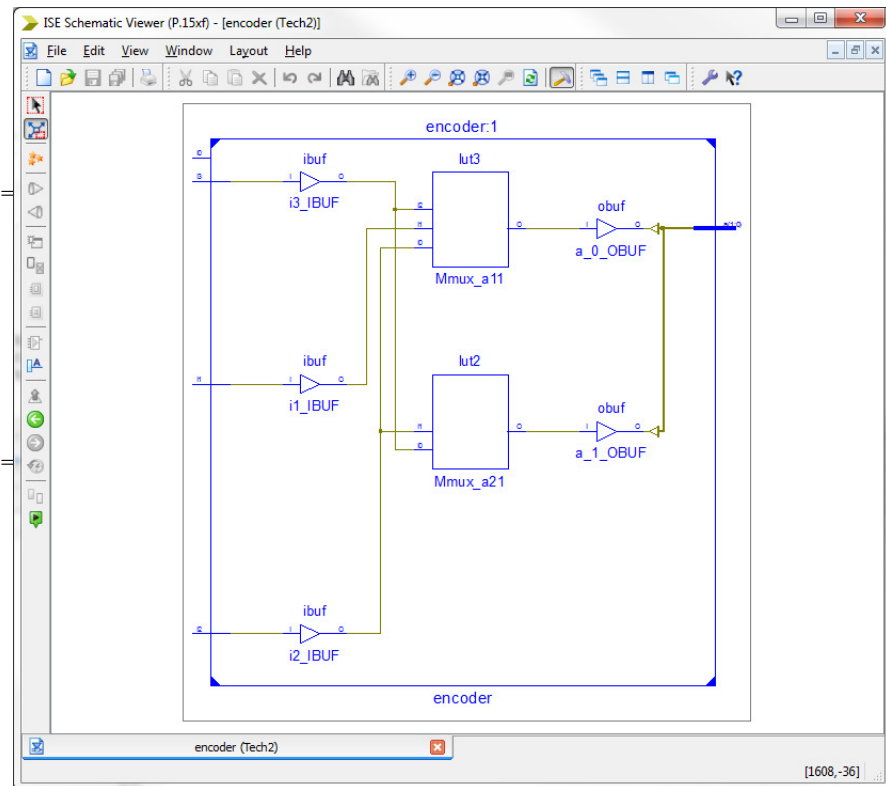
=====

HDL Synthesis Report

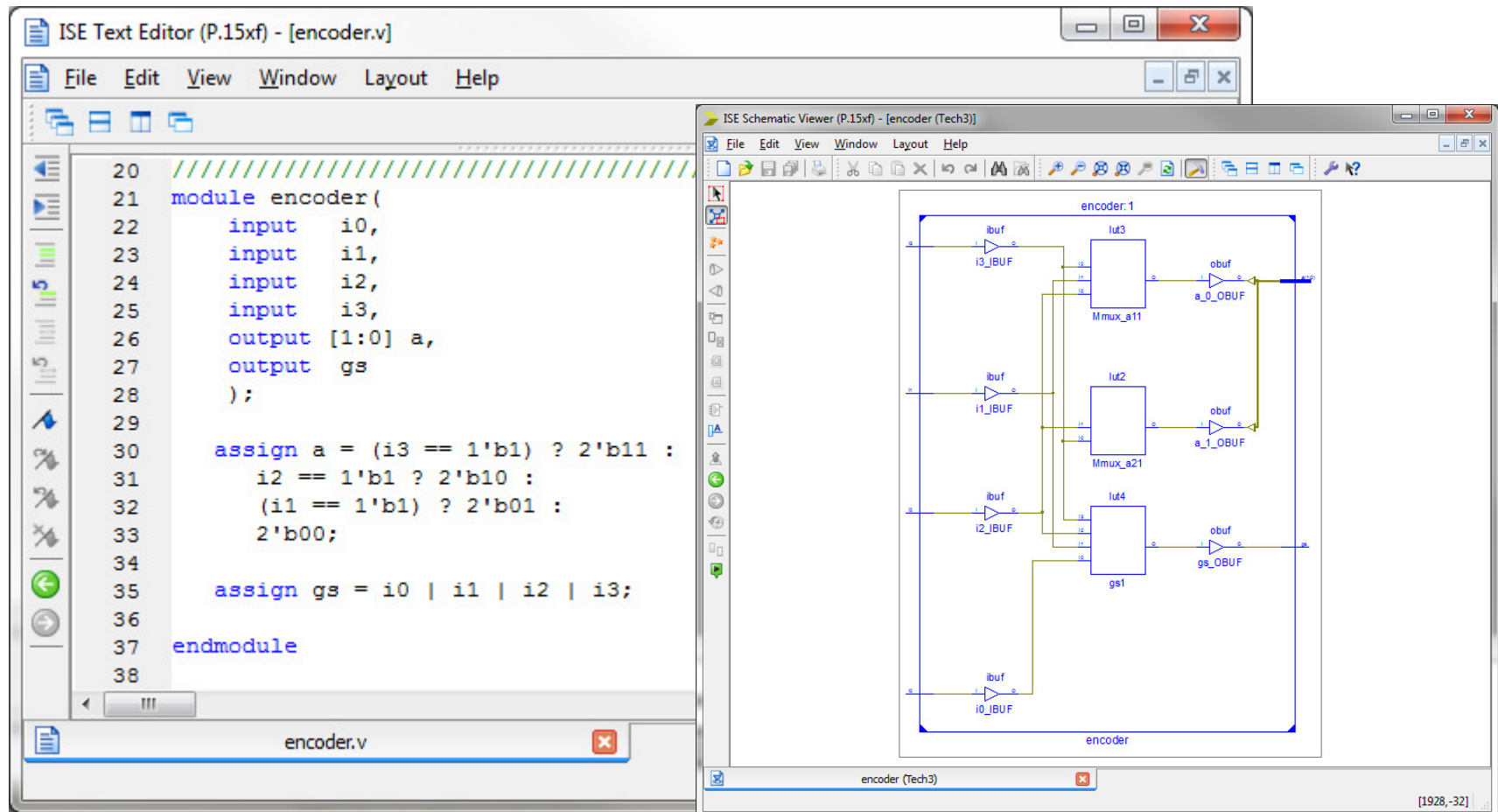
Macro Statistics

# Multiplexers	: 2
2-bit 2-to-1 multiplexer	: 2

=====



Add 'gs' output



Synthesize - Design Summary

```
=====
*                               Design Summary                               *
=====
```

Clock Information:

No clock signals found in this design

Asynchronous Control Signals Information:

No asynchronous control signals found in this design

Timing Summary:

Speed Grade: -3

Minimum period: No path found

Minimum input arrival time before clock: No path found

Maximum output required time after clock: No path found

Maximum combinational path delay: 5.456ns

```
=====
```

Implement Design

Device Utilization Summary:

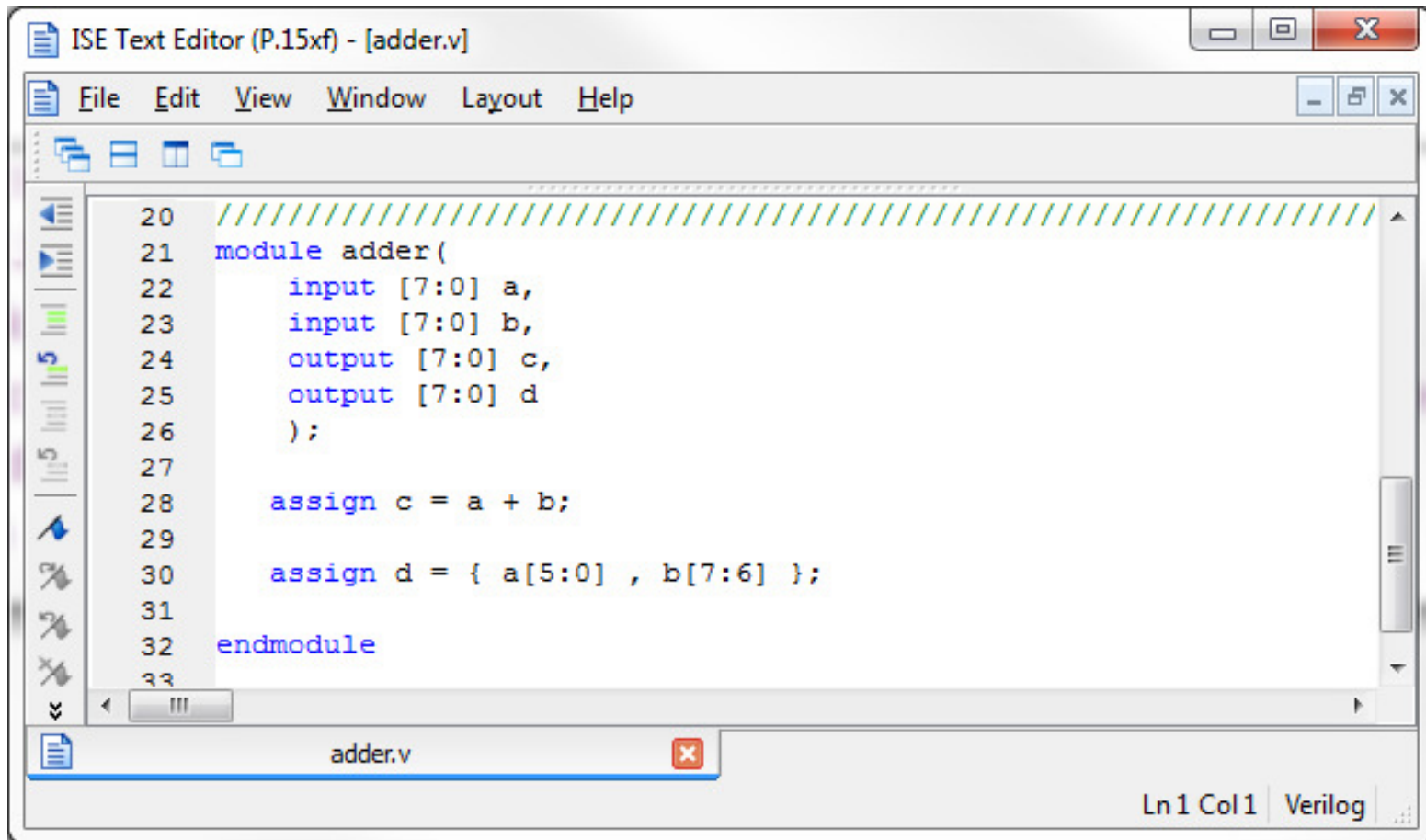
Slice Logic Utilization:

Number of Slice Registers:	0 out of	18,224	0%
Number of Slice LUTs:	2 out of	9,112	1%
Number used as logic:	2 out of	9,112	1%
Number using O6 output only:	1		
Number using O5 output only:	0		
Number using O5 and O6:	1		
Number used as ROM:	0		
Number used as Memory:	0 out of	2,176	0%

Slice Logic Distribution:

Number of occupied Slices:	2 out of	2,278	1%
Number of MUXCYs used:	0 out of	4,556	0%
Number of LUT Flip Flop pairs used:	2		
Number with an unused Flip Flop:	2 out of	2	100%
Number with an unused LUT:	0 out of	2	0%
Number of fully used LUT-FF pairs:	0 out of	2	0%
Number of slice register sites lost to control set restrictions:	0 out of	18,224	0%

Creating adder – using LUTs

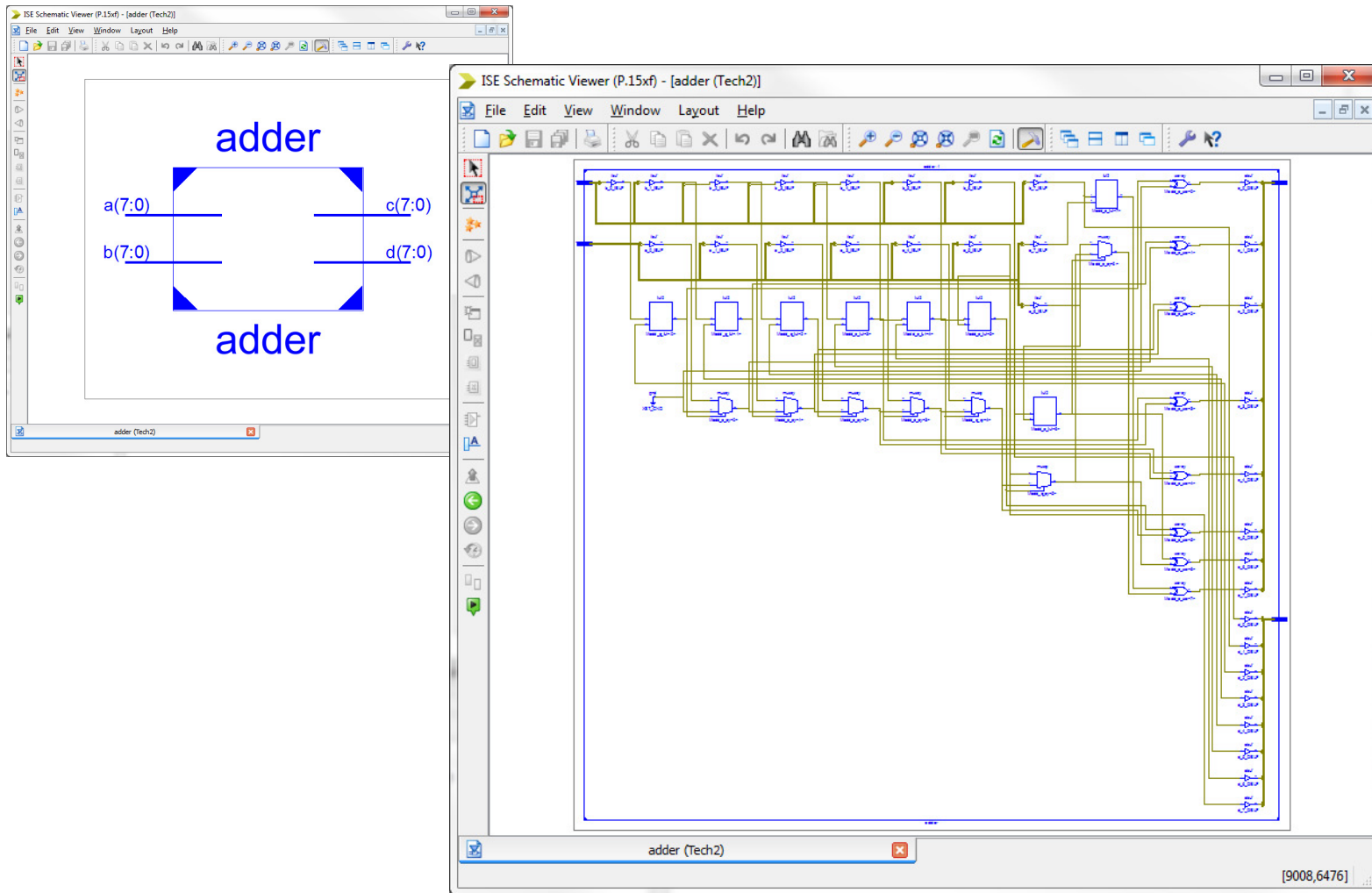


The screenshot shows the ISE Text Editor window titled "ISE Text Editor (P.15xf) - [adder.v]". The window contains a Verilog module definition for an adder. The code is as follows:

```
20 //////////////////////////////////////////////////
21 module adder(
22     input [7:0] a,
23     input [7:0] b,
24     output [7:0] c,
25     output [7:0] d
26 );
27
28     assign c = a + b;
29
30     assign d = { a[5:0] , b[7:6] };
31
32 endmodule
33
```

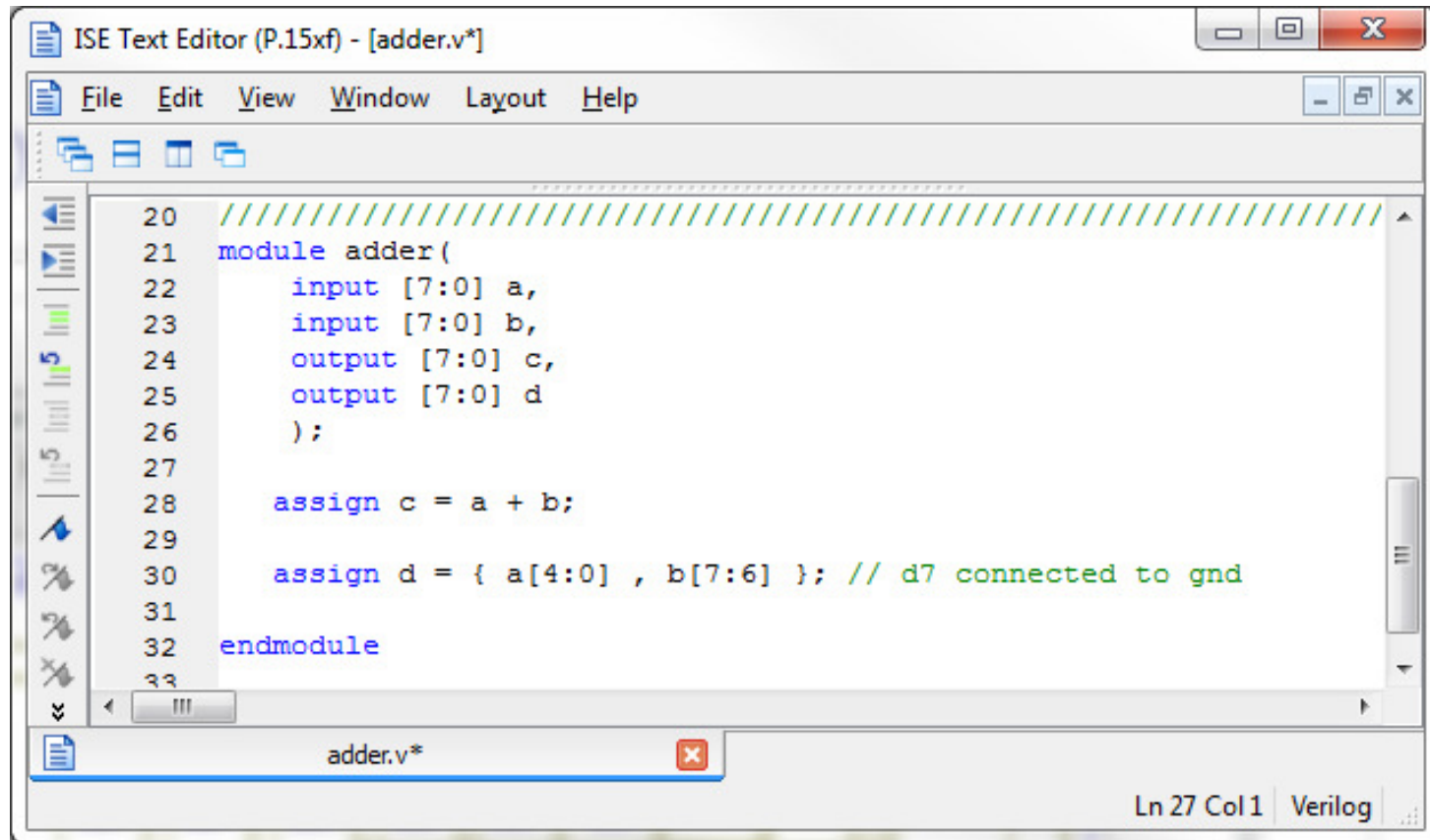
The status bar at the bottom right indicates "Ln 1 Col 1 Verilog".

Technology Schematic



Example of simple mistake

- No errors or warnings!



```
20 //////////////////////////////////////////////////
21 module adder(
22     input [7:0] a,
23     input [7:0] b,
24     output [7:0] c,
25     output [7:0] d
26 );
27
28     assign c = a + b;
29
30     assign d = { a[4:0] , b[7:6] }; // d7 connected to gnd
31
32 endmodule
33
```

Top-Down Design Hierarchy

- Instantiate module (counter example with decoder)

```
module decoder(  
    input  [3:0] count,  
    output [6:0] seven_seg  
);  
  
// instantiate decoder module in counter  
// using position of ports  
decoder d1 (count_val, seven_seg_val);  
  
// or using formal and actual names  
decoder d1 (.count(count_val), .seven_seg(seven_seg_val));
```


Tri-state example

- Using conditional operator in continuous assignment

