Maximize ispMACH 5000VG and ispLSI 5000VE Functionality with Dual-OR Macrocells

July 2001 Technical Note TN1006

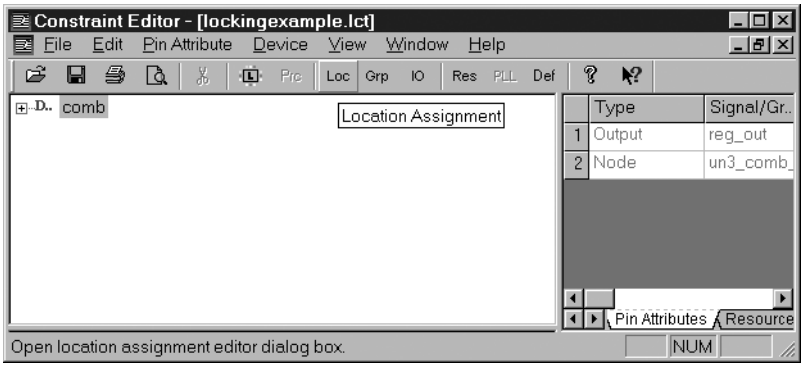
Introduction

Coming close to the macrocell limit? Why not double the number of functions used by your dual-OR output capability of the macrocells. This applications note shows two ways to implement the dual-OR function in the I/O architecture. The first method locks pins and signals to the same macrocell, and the second method groups signals together and lets the compiler assign the dual-OR functions for you.

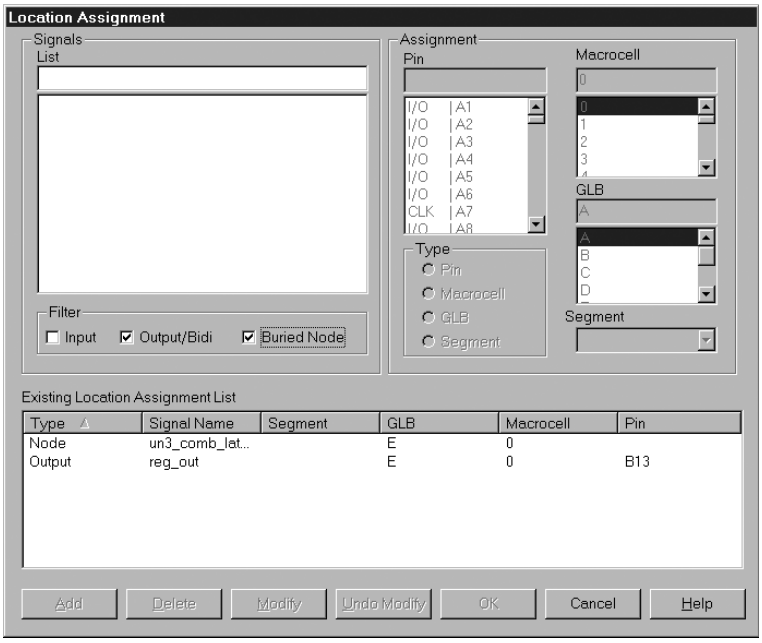
Method 1: Locking Pins

This method forces two functions into a macrocell by locking pins and nodes. In the example found in Appendix A, the output “reg_out” is locked to Pin B13. The node “un3_comb_latch_out” is locked to Macrocell E0, the same macrocell as Pin B13 uses. To lock pins:

1) Open the Constraint Editor as shown below and click the Loc (Location Assignment) button

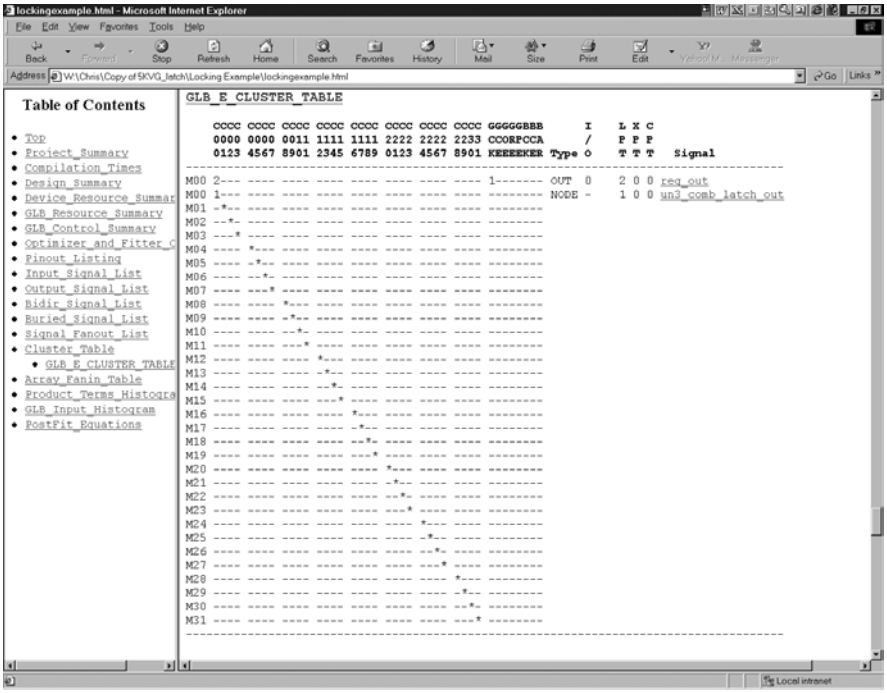


2) The Location Assignment Window will open, as shown below. Select Output/Bidi and Buried Node. It is usually easiest to lock pins first. Once a pin assignment has been chosen, click the Add button to lock the pin. Following this, the buried node can be locked to the same macrocell using the same procedure.



3) Save the changes and close the Constraint Editor.

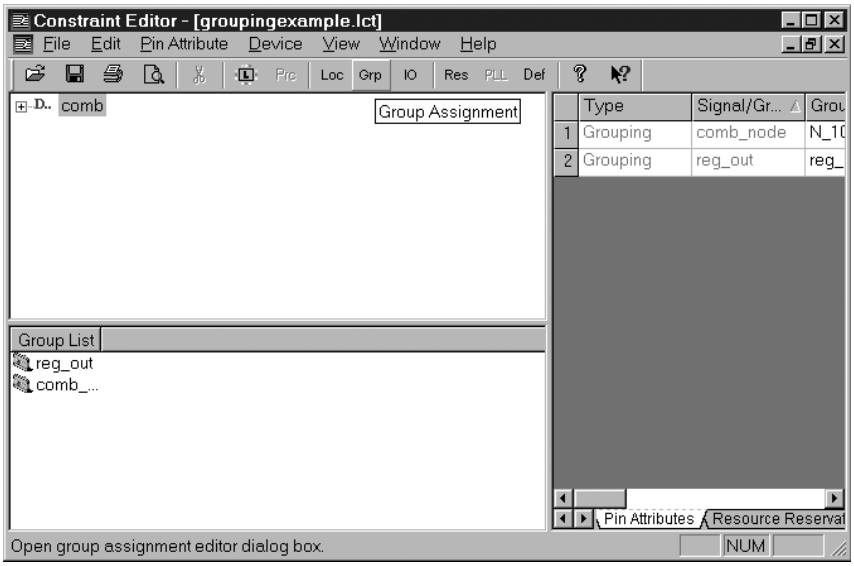
4) To verify both functions are using the same macrocell, look at the number of two-function macrocells used in the section Device Resource Summary in the report file. Also in the report file you can look at the GLB_E_CLUSTER_TABLE, as shown below. The number '2' in the cluster table entry shows that the macrocell is implementing two functions, and the number 1 shows the macrocell implementing a single function.



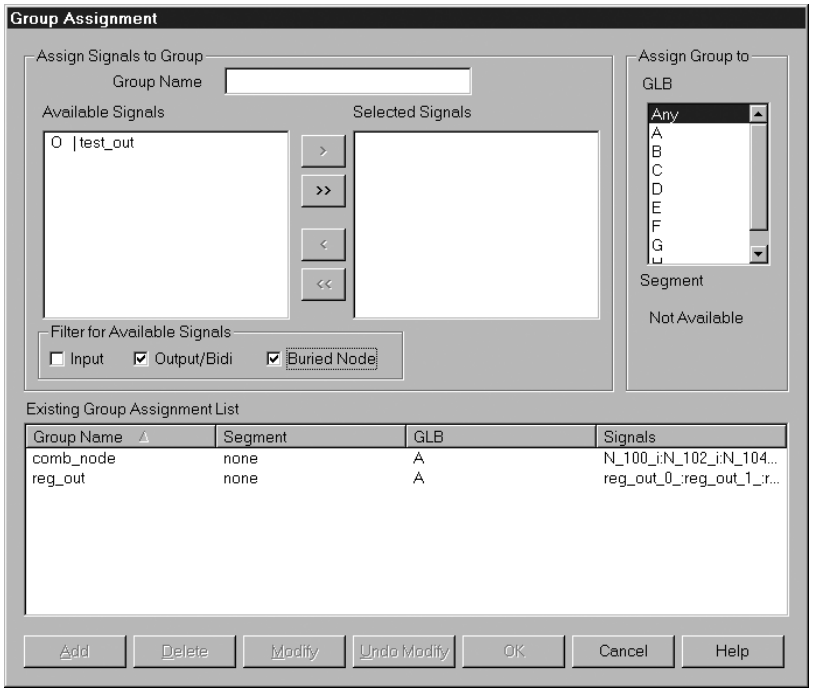
Method 2: Grouping Pins

The second method to implement the dual-OR function within a macrocell is by grouping more than 32 signals into same GLB. This forces the compiler to implement two functions in macrocells. In the following example, found in Appendix B, signals are grouped into GLB A. To group pins:

1) Open the Constraint Editor as shown below and click the Grp (Group Assignment) button.

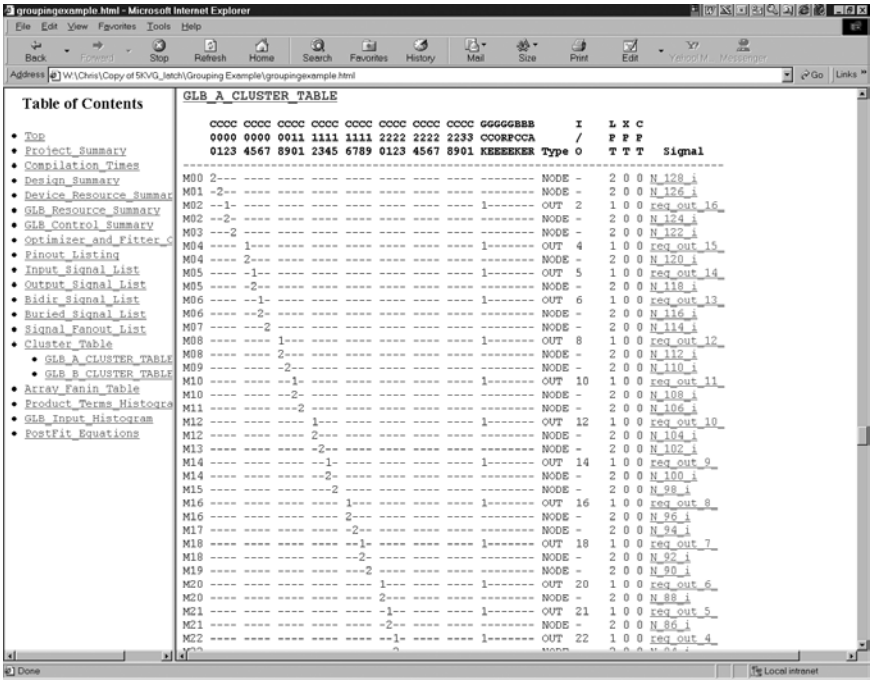


2) The Group Assignment Window will open, as shown below. Select Output/Bidi and Buried Node. Select a group name for the output pins. Then select the GLB to group it to. Followed by clicking the Add button. The same procedure can be used for buried nodes.



3 Save your changes and close the Constraint Editor.

4) To verify both functions are using the same macrocell, look at the number of two-function macrocells used in the section Device Resource Summary in the report file. Also in the report file you can look at the GLB_E_CLUSTER_TABLE, as shown below. The number '2' in the cluster table entry shows that the macrocell is implementing two functions, and the number '1' shows the macrocell implementing a single function.



Summary

By taking advantage of the ispMACH 5000VG and ispLSI 5000V family dual-OR output of the macrocells, Lattice's ispLEVER™ Development System software assigns up to two logic functions for a given macrocell. The first example simply locks the pin function and the macrocell feedback function to the same macrocell. The second example lets the software assign dual function as needed when the user forces more than 32 functions in a GLB.

The design examples show a latch function implemented with the combinatorial path of the dual OR and a register function with the remaining dual OR path. Although this capability can greatly increase the logic functions, the functions must be of small product term (PT) usage. The general recommendation is to selectively use these features where a few simple registers or latches must be added to already packed design. The software provides the maximum flexibility by allowing the user to implement any two (register or combinatorial) functions within the macrocell.

Technical Support Assistance

Hotline: 1-800-LATTICE (Domestic)
1-408-826-6002 (International)
e-mail: techsupport@latticesemi.com

Appendix A Locking VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
  port          (reg_in: in std_logic;
                 comb_latch_din, comb_le: in std_logic;
                 sysclk: in std_logic;
                 reg_out: out std_logic);
end comb;

architecture comb_list of comb is
begin

process (sysclk) -- clock name is in sensitivity list
                 variable comb_latch_out: std_logic;

begin
  if rising_edge(sysclk) then
                 reg_out <= reg_in or comb_latch_out;
  end if;

  comb_latch_out := (comb_latch_din and (not(comb_le))) or (comb_latch_out and comb_le);

end process;

end comb_list;
```

Appendix B Grouping VHDL Code

```
library ieee;
use ieee.std_logic_1164.all;

entity comb is
  port          (reg_in: in std_logic;
                 comb_latch_din, comb_le: in std_logic;
                 sysclk: in std_logic;
                 reg_out: out std_logic_vector(16 downto 0);
                 test_out: out std_logic);
end comb;

architecture comb_list of comb is
begin

process (sysclk) -- clock name is in sensitivity list
  variable comb_latch_out : std_logic_vector(31 downto 0);

begin
  if rising_edge(sysclk) then
                 reg_out(0) <= reg_in or comb_latch_out(0);
                 reg_out(1) <= reg_in or comb_latch_out(1);
                 reg_out(2) <= reg_in or comb_latch_out(2);
                 reg_out(3) <= reg_in or comb_latch_out(3);
                 reg_out(4) <= reg_in or comb_latch_out(4);
                 reg_out(5) <= reg_in or comb_latch_out(5);
                 reg_out(6) <= reg_in or comb_latch_out(6);
                 reg_out(7) <= reg_in or comb_latch_out(7);
                 reg_out(8) <= reg_in or comb_latch_out(8);
                 reg_out(9) <= reg_in or comb_latch_out(9);
                 reg_out(10) <= reg_in or comb_latch_out(10);
                 reg_out(11) <= reg_in or comb_latch_out(11);
                 reg_out(12) <= reg_in or comb_latch_out(12);
                 reg_out(13) <= reg_in or comb_latch_out(13);
                 reg_out(14) <= reg_in or comb_latch_out(14);
                 reg_out(15) <= reg_in or comb_latch_out(15);
                 reg_out(16) <= reg_in or comb_latch_out(16);

  end if;


  test_out <= comb_latch_out(0) and comb_latch_out(1) and comb_latch_out(2) and comb_latch_out(3) and
comb_latch_out(4) and comb_latch_out(5) and comb_latch_out(6) and comb_latch_out(7) and comb_latch_out(8)
and comb_latch_out(9) and comb_latch_out(10) and comb_latch_out(11) and comb_latch_out(12) and
comb_latch_out(13) and comb_latch_out(14) and comb_latch_out(15) and comb_latch_out(16) and
comb_latch_out(17) and comb_latch_out(18) and comb_latch_out(19) and comb_latch_out(20) and
comb_latch_out(21) and comb_latch_out(22) and comb_latch_out(23) and comb_latch_out(24) and
comb_latch_out(25) and comb_latch_out(26) and comb_latch_out(27) and comb_latch_out(28) and
comb_latch_out(29) and comb_latch_out(30) ;
```

```
  comb_latch_out(0) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(0) and comb_le);
  comb_latch_out(1) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(1) and comb_le);
  comb_latch_out(2) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(2) and comb_le);
  comb_latch_out(3) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(3) and comb_le);

  comb_latch_out(4) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(4) and comb_le);
  comb_latch_out(5) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(5) and comb_le);
  comb_latch_out(6) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(6) and comb_le);
  comb_latch_out(7) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(7) and comb_le);
  comb_latch_out(8) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(8) and comb_le);
  comb_latch_out(9) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(9) and comb_le);
  comb_latch_out(10) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(10) and comb_le);
  comb_latch_out(11) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(11) and comb_le);
  comb_latch_out(12) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(12) and comb_le);
  comb_latch_out(13) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(13) and comb_le);
  comb_latch_out(14) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(14) and comb_le);
  comb_latch_out(15) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(15) and comb_le);
  comb_latch_out(16) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(16) and comb_le);
  comb_latch_out(17) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(17) and comb_le);
  comb_latch_out(18) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(18) and comb_le);
  comb_latch_out(19) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(19) and comb_le);
  comb_latch_out(20) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(20) and comb_le);
  comb_latch_out(21) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(21) and comb_le);
  comb_latch_out(22) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(22) and comb_le);
  comb_latch_out(23) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(23) and comb_le);
  comb_latch_out(24) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(24) and comb_le);
  comb_latch_out(25) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(25) and comb_le);
  comb_latch_out(26) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(26) and comb_le);
  comb_latch_out(27) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(27) and comb_le);
  comb_latch_out(28) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(28) and comb_le);
  comb_latch_out(29) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(29) and comb_le);
  comb_latch_out(30) := (comb_latch_din and (not(comb_le))) or (comb_latch_out(30) and comb_le);

end process;

end comb_list ;
```