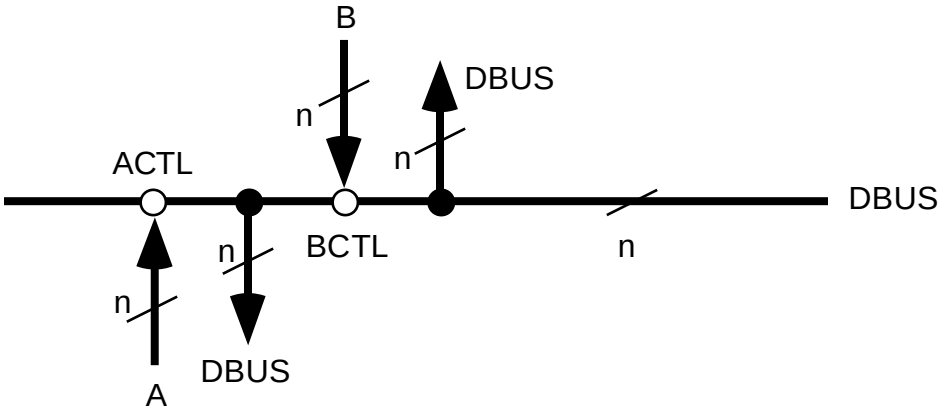
Filename="ch8_2

8.0 Resolving Multi-Signal Drivers

8.1 Buses



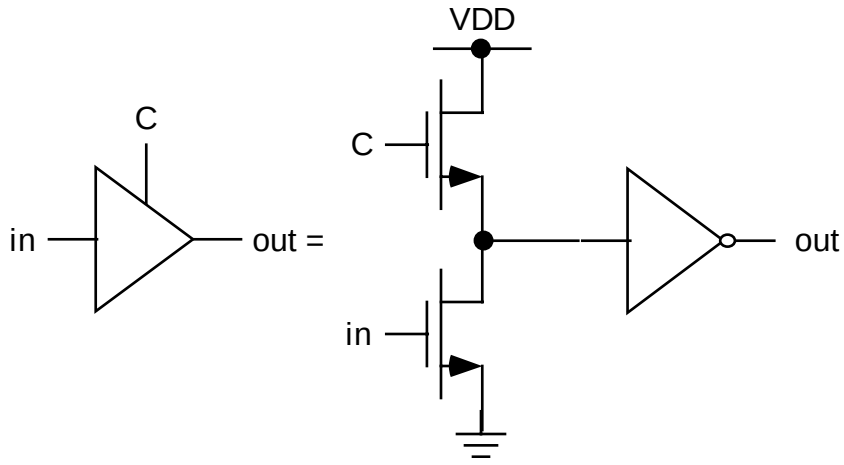
Open circle denotes an input connection
Solid dot denotes an output connection

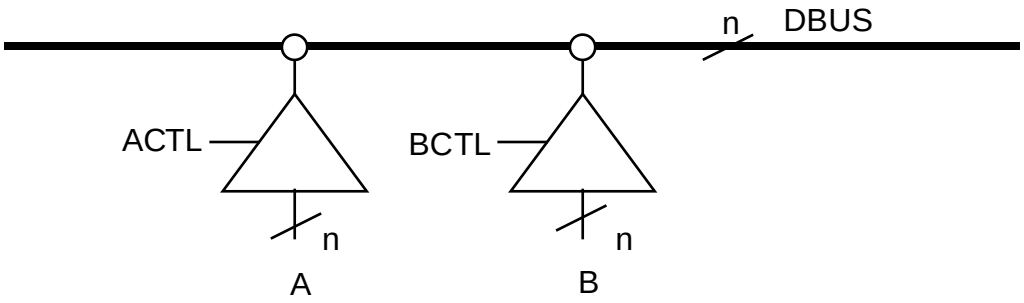
The open circles at the bus inputs respresent switched connections

ACTL	BCTL	DBUS
0	0	impementation dependent (no input activated)
0	1	B
1	0	A
1	1	implementation dependent (both inputs activated)

8.2 Bus Implementation

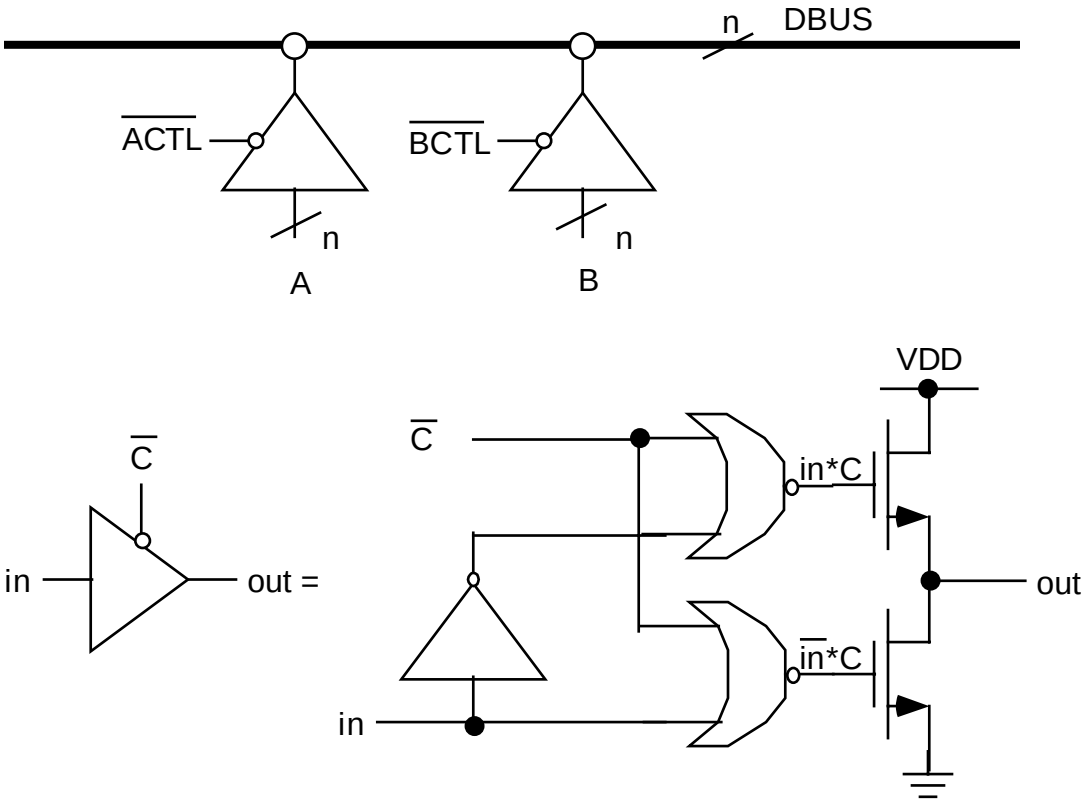
8.2.1 NMOS Bus Driver





ACTL	BCTL	DBUS
0	0	Floating
0	1	B
1	0	A
1	1	Undefined (Avoid)

8.2.2 Tri-State Bus Driver



$\overline{\text{ACTL}}$	$\overline{\text{BCTL}}$	DBUS
0	0	Undefined (Avoid)
0	1	A
1	0	B
1	1	Floating

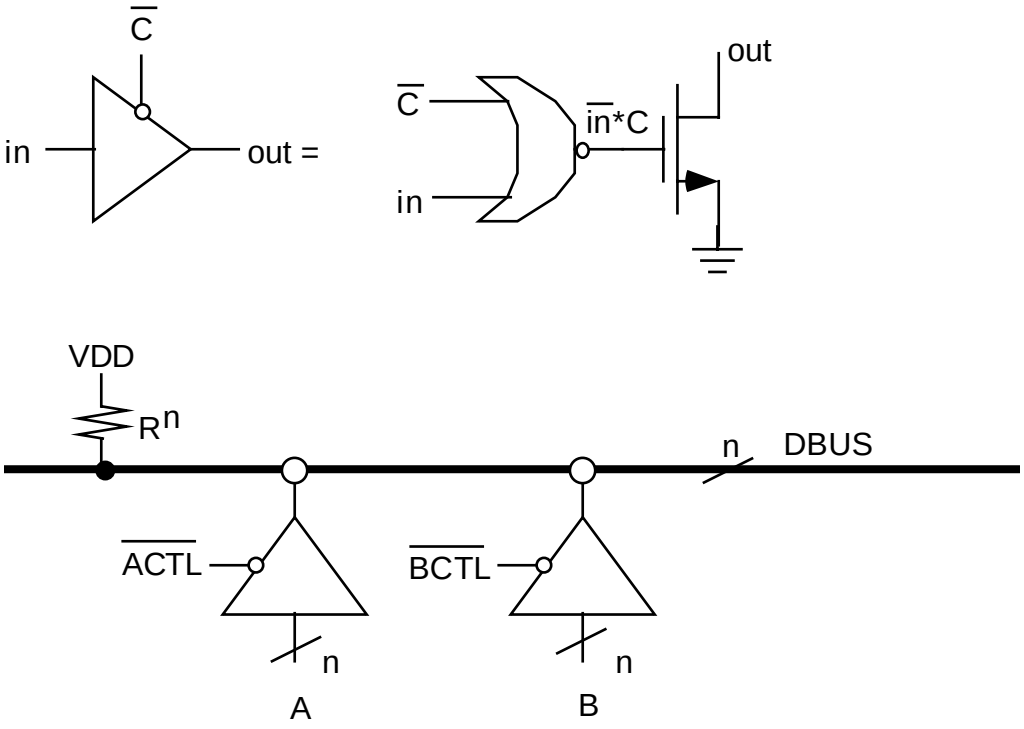
When

C=1 both out transistors are turned off and the output out is left floating

C=0 one of the output transtor is enabled and the other disabled, as selected by the input data in, out <= in.

There are situations where bus control inputs are generated by devices operating independently and asynchronously. Interrupt request (IRQ) lines are a common example. Multiple drivers can be enabled simultaneously during normal operation, and hence **tri-state bus should not be used**.

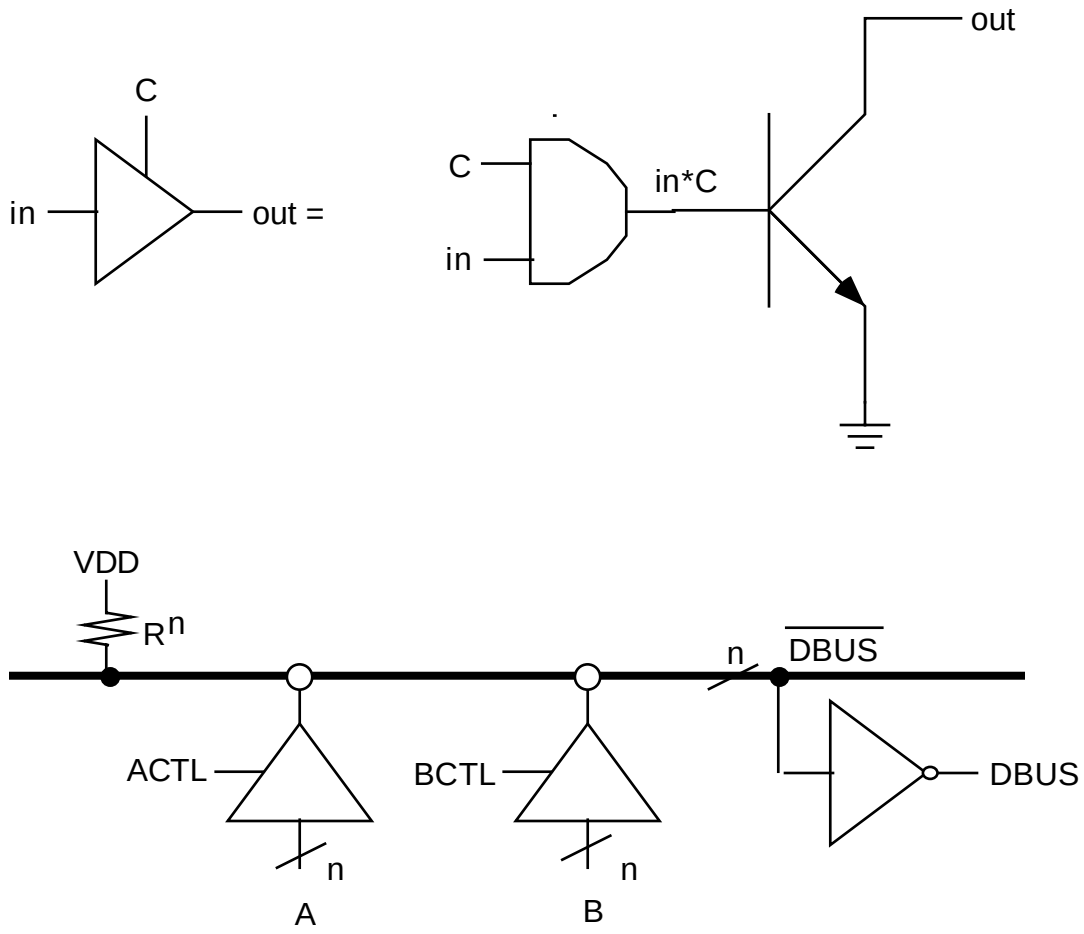
8.2.3 Open-Drain Bus Driver



The driver is a NOR gate followed by an inverter with the pull-up resistor removed. A pull-up resistor is provided externally to hold the bus at 1 when no driver is pulling it low. When no driver is enabled, DBUS = 1^n. If multiple drivers are enabled, then the bus will be pulled low if any enabled inputs is low, hence DBUS is the AND of the enabled inputs.

$\overline{\text{ACTL}}$	$\overline{\text{BCTL}}$	DBUS
0	0	A and B
0	1	A
1	0	B
1	1	1^n

8.2.4 Open-Collector Bus Driver



TTL version of the open-drain OR-gate is the open-collector NAND-gate. It is formed from the basic TTL gate by removing the pull-up from the output. The DBUS is the OR of the enabled inputs.

ACTL	BCTL	$\overline{\text{DBUS}}$	DBUS
0	0	1^n	0^n
0	1	$\overline{B}$	B
1	0	$\overline{A}$	A
1	1	$\overline{A}$ and $\overline{B}$	A + B

8.3 Resolving Multi-Signal Drivers

Simulators originally used a two-state value system, representing logic 1 and logic 0. This is declared as type BIT:

```
TYPE BIT IS ('0', '1');
```

For a single source system this is adequate. When two or more souces are driving a signal, if one source is driving a value of '1' and the other driving a '0', the result depend on the technologies used, it may be design error in some and unknown in others. With two-state system, this state can not be represented.

To represent conditions of unknown state and floating state, a minimum of four-value system is required. This unresolved 4 logic state system (Autologic II , old standard) is declared as:

```
TYPE qsim_state IS (
        'X',    --Forcing Unknown
        '0',    --Forcing 0
        '1',    --Forcing 1
        'Z'     --High Impedance
          );
```

To use this standard logic system, VHDL code must include the following lines at the beginning.

```
LIBRARY MGC_PORTABLE;
USE MGC_PORTABLE.QSIM_LOGIC.ALL;
USE MGC_PORTABLE.QSIM_RELATIONS.ALL;
```

For more complex circuit simulation, the simulator must be able to represent signals that do not swing to the power rails of '1' (VDD), and '0' (GND) such as TTL logic level where 'H' means >=2.5 and 'L' <= 0.8V. The new IEEE STANDARD LOGIC 1164 (Autologic II, current standard) is a 9 value system. The unresolved 9 logic state system is declared as follows:

```
TYPE std_ulogic IS (
        'U',    --Uninitialized
        'X',    --Forcing Unknown
        '0',    --Forcing 0
        '1',    --Forcing 1
        'Z',    --High Impedance
        'W',    --Weak Unknown
        'L',    --Weak 0
        'H',    --Weak 1
        '-'     --Don't care
         );
```

To use this standard logic system, VHDL code must include the following lines at the beginning.

```
LIBRARY IEEE, ARITHMETIC;
USE IEEE.STD_LOGIC_1164.ALL;
USE ARITHMETIC.STD_LOGIC_ARITH.ALL;
```

8.3.1 Resolving a Signal Value When Driven by Multiple Assignments

Every signal assignment statement assigns a projected waveform to a driver. It is possible (and probable in hardware designs) that your model contains more than one signal assignment statement (each with its own driver) that attempts to assign different values to the same signal at the same time. When this happens, your

model must provide a resolution function that specifies how to resolve the assignment. Each signal that requires a resolution function makes reference to the appropriate function in the signal declaration.

Example:

```
ENTITY mult_driver IS
        PORT(a,b,c,d:IN qsim_state;z:OUT qsim_state);
END mult_driver;

ARCHITECTURE wired_and OF mult_driver IS
        FUNCTION anding(drivers:qsim_state_vector)RETURN qsim_state IS
                VARIABLE temp:qsim_state:='1';
        BEGIN
                FOR i IN drivers'RANGE LOOP
                        temp:=temp AND drivers(i);          --see the AND table
                END LOOP;
                RETURN temp;
        END anding;

        SIGNAL line_and: anding qsim_state;
BEGIN
        line_and <= a;
        line_and <= b;
        line_and <= c;
        line_and <= d;
        z<= line_and;
END wired_and;
```

Qsim AND Table

AND	**0**	**1**	**X**	**Z**
0	0	0	0	0
1	0	1	X	X
X	0	X	X	X
Z	0	X	X	X

Std_ulogic AND TABLE

AND	**U**	**X**	**0**	**1**	**Z**	**W**	**L**	**H**	**-**
U	U	U	0	U	U	U	0	U	U
X	U	X	0	X	X	X	0	X	X
0	0	0	0	0	0	0	0	0	0
1	U	X	0	1	X	X	0	1	X
Z	U	X	0	X	X	X	0	X	X
W	U	X	0	X	X	X	0	X	X
L	0	0	0	0	0	0	0	0	0
H	U	X	0	1	X	X	0	1	X
-	U	X	0	X	X	X	0	X	X

If the bus is wired or type replace the function to oring:

```
FUNCTION oring(drivers:qsim_state_vector)RETURN qsim_state IS
        VARIABLE temp:qsim_state:='0';
BEGIN
        FOR i IN drivers'RANGE LOOP
                temp:=temp OR drivers(i);            --see the OR table
        END LOOP
        RETURN temp;
END oring;;

 SIGNAL line_or: oring qsim_state;
```

Qsim OR Table

OR	**0**	**1**	**X**	**Z**
0	0	1	X	X
1	1	1	1	1
X	X	1	X	X
Z	X	1	X	X

Std_ulogic OR Table

OR	**U**	**X**	**0**	**1**	**Z**	**W**	**L**	**H**	**-**
U	U	U	U	1	U	U	U	1	U
X	U	X	X	1	X	X	X	1	X
0	U	X	0	1	X	X	0	1	X
1	1	1	1	1	1	1	1	1	1
Z	U	X	X	1	X	X	X	1	X
W	U	X	X	1	X	X	X	1	X
L	U	X	0	1	X	X	0	1	X
H	1	1	1	1	1	1	1	1	1
-	U	X	X	1	X	X	X	1	X

If the bus is neither wired_and nor wired_or, such as wiring several signals into a common node. A new two operand function called wire is defined:

```
TYPE qsim_2D IS ARRAY(qsim_state,qsim_state) OF qsim_state
FUNCTION wire (a,b:qsim_state) RETURN qsim_state IS
        CONSTANT wire_table:qsim_2D:=(
                        (`0','X','X','0'),
                        (`X','1','X','1'),
                        (`X','X','X','X'),
                        (`0','1','X','Z'));
BEGIN
        RETURN wire_table(a,b);
END wire;
```

Qsim WIRE Table

WIRE	**0**	**1**	**X**	**Z**
0	0	X	X	0
1	X	1	X	1
X	X	X	X	X
Z	0	1	X	Z

Std_ulogic WIRE Table

	U	**X**	**0**	**1**	**Z**	**W**	**L**	**H**	**-**
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
-	U	X	X	X	**X**	X	X	X	X

NOTE: this is the same as resolution_table in IEEE package

Interpretation of wire:

1. IF the two inputs are equal, the wire value will be the same as the inputs.
2. wire(a,'Z')=a , wire(`Z',b)=b: Value `Z' on either of the inputs is absorbed by a stronger value (`0','1', or`X').
3. wire(a,b)=X ,if a<>b and a<>'Z' and b<>'Z': conflicting non `Z' values on the inputs result in `X' value.

```
FUNCTION wiring(drivers:qsim_state_vector) RETURN qsim_state IS
        VARIABLE temp: qsim_state:='Z';
BEGIN
        FOR i IN drivers'RANGE LOOP
                temp:=wire(temp,drivers(i));
        END LOOP;
        RETURN temp;
END wiring;

 SIGNAL bus_wire:wiring qsim_state;
```

The corresponding resolving function of wiring in the 9-state logic system is given below:

```
FUNCTION resolved (s ; std_ulogic_vector) RETURN std_ulogic IS
        VARIABLE result : std_ulogic := ‘Z” ; --weakest state default
BEGIN
        --the test for a single driver is essential otherwise the
        --loop would return ‘X’ for a single driver of ‘-‘ and that
        --would conflict with the value of a single driver unresolved signal.
        IF (s’LENGTH = 1) THEN RETURN s(s’LOW);
        ELSE
                FOR i IN s’RANGE LOOP
                        result := resolution_table(result, s(i));
                END LOOP;
        END IF;
        RETURN result;
END resolved;


SUBTYPE std_logic IS resolved std_ulogic;
TYPE std_logic_vector IS ARRAY (NATURAL RANGE <>) OF std_logic;
```

The above resolution function are best put in a design package inorder that they be available to all the design entities. In addition, subtypes and types can be declared to handle line or bus contention that will facilitate signal declaration that requires resolution function.

```
PACKAGE design_package IS
  FUNCTION anding(drivers:qsim_state_vector)RETURN qsim_state;
  SUBTYPE anded_qsim_state IS anding qsim_state; -- for resolving line drivers
  TYPE anded_qsim_state_vector IS ARRAY(NATURAL RANGE<>) OF anded_qsim_state;
   -- for resolving bus drivers
  FUNCTION oring(drivers:qsim_state_vector)RETURN qsim_state;
  SUBTYPE ored_qsim_state IS oring qsim_state;
  TYPE ored_qsim_state_vector IS ARRAY(NATURAL RANGE<>) OF ored_qsim_state;

  FUNCTION wiring(drivers:qsim_state_vector)RETURN qsim_state;
  SUBTYPE wired_qsim_state IS wiring qsim_state;
  TYPE wired_qsim_state_vector IS ARRAY(NATURAL RANGE<>) OF
            wired_qsim_state ;
END design_package;

PACKAGE BODY design_package IS

  FUNCTION anding(drivers:qsim_state_vector)RETURN qsim_state IS
          VARIABLE temp:qsim_state:='1';
  BEGIN
          FOR i IN drivers'RANGE LOOP
                  temp:=temp AND drivers(i);
          END LOOP;
          RETURN temp;
  END anding;

   FUNCTION oring(drivers:qsim_state_vector)RETURN qsim_state IS
          VARIABLE temp:qsim_state:='0';
   BEGIN
          FOR i IN drivers'RANGE LOOP
                  temp:=temp OR drivers(i);
          END LOOP
          RETURN temp;
   END oring;

   TYPE qsim_2D IS ARRAY(qsim_state,qsim_state) OF qsim_state
   FUNCTION wire (a,b:qsim_state) RETURN qsim_state IS
          CONSTANT wire_table:qsim_2D:=(
                          (`0','X','X','0'),
                          (`X','1','X','1'),
                          (`X','X','X','X'),
                          (`0','1','X','Z'));
   BEGIN
          RETURN wire_table(a,b);
   END wire;

   FUNCTION wiring(drivers:qsim_state_vector) RETURN qsim_state IS
          VARIABLE temp: qsim_state:='Z';
   BEGIN
          FOR i IN drivers'RANGE LOOP
                  temp:=wire(temp,drivers(i));
          END LOOP;
```

```
        RETURN temp;
  END wiring;
END design_package;
```

Depending on the type of bus, they are declared accordingly:

```
SIGNAL bus_and : anded_qsim_state          --for wired_and bus
SIGNAL bus_or : ored_qsim_state  --for wired_or bus
SIGNAL bus_wire: wired_qsim_state          --for wired bus
```

8.3.2 WIRED_AND BUS - OPEN COLLECTOR GATES

The function of resolution functions and resolved signals is very useful for modeling various bus forms. A bussing structure formed by connecting the outputs of open collector gates is very common. The VHDL description of an open collector two-input NAND gate is given as follows:

```
ENTITY nand2 IS
        GENERIC(tplh:TIME:=10 ns;tphl:TIME:=15 ns);
        PORT(x,y:IN qsim_state;z:OUT qsim_state);
END nand2;

ARCHITECTURE open_collector OF nand2 IS
BEGIN
        z<='0' AFTER tphl WHEN (x AND y)='1' ELSE
           `Z' AFTER tplh WHEN (x AND y)='0' ELSE
           `X' AFTER tphl;
END open_collector;
```

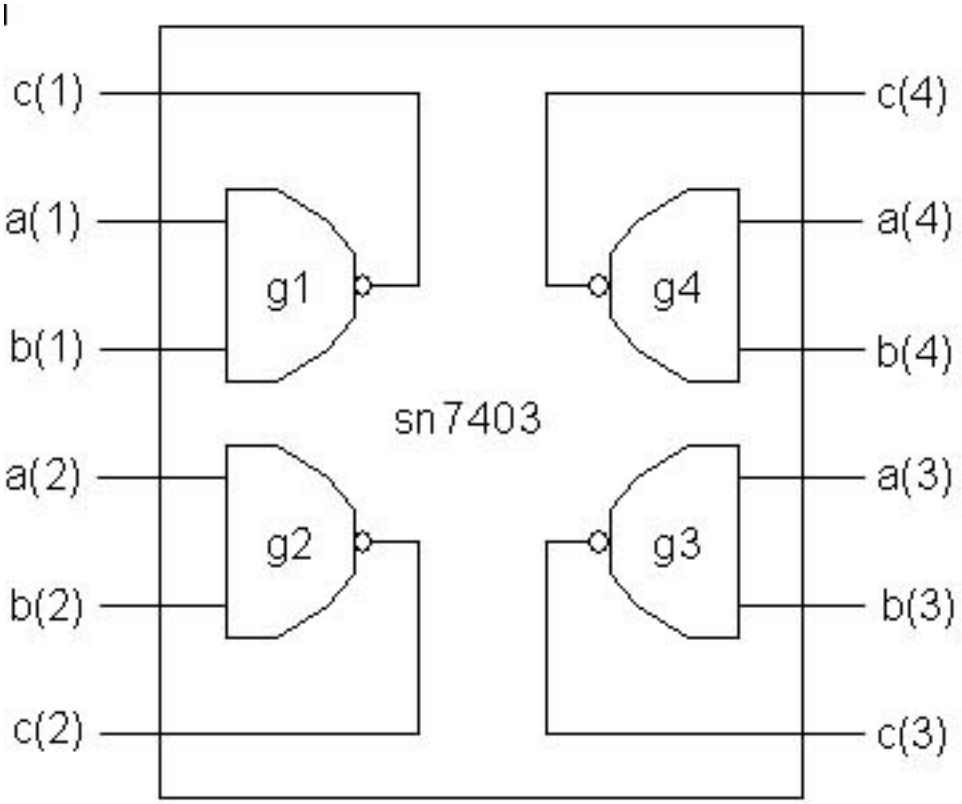
The output of an open_collector gate must be connected to a pull-up,resistor to the power supply, several such outputs can be connected to a common pull-up resistor. In general, the function of a pull-up resistor is to produce a `1' if none of its drivers is `0', and to produce a `0' if at least one driver is `0'. For qsim_state type this is identical to type anded_qsim_state. Therefore, a line or bus connected to a pull-up resistor to a power supply is declared or modeled as follows:

```
SIGNAL pull_up: anded_qsim_state;
```

Associating this signal with an output port of an open collector gate is equivalent to connecting that output to a pull-up resistor in hardware.

9.3.2.1 VHDL Implementation of 7403

```
ENTITY sn7403 IS
        PORT(a,b:IN qsim_state_vector(1 TO 4);c:OUT qsim_state_vector(1 TO 4));
END sn7403;
ARCHITECTURE struct OF sn7403 IS
        COMPONENT nand2
                PORT(a,b:IN qsim_state;c:OUT qsim_state);
        END COMPONENT;
        FOR ALL:nand2 USE ENTITY WORK.nand2(open_collector);
BEGIN
        g1: :nand2 PORT MAP(a(1),b(1),c(1));
        g2: :nand2 PORT MAP(a(2),b(2),c(2));
        g3: :nand2 PORT MAP(a(3),b(3),c(3));
        g4: :nand2 PORT MAP(a(4),b(4),c(4));
END struct;
```

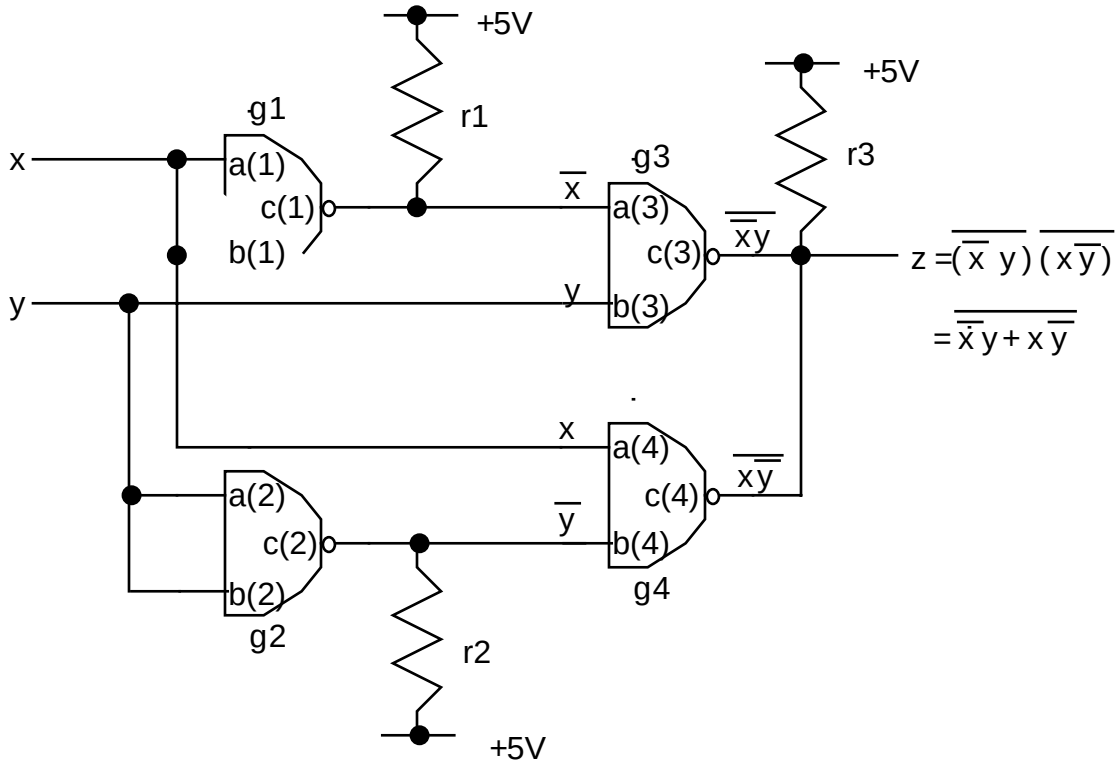


8.3.2.2 X_NOR Implementation using sn7403

```
ENTITY xnor IS
  PORT(x,y:IN qsim_state;z:OUT qsim_state);
END xnor;

ARCHITECTURE struct OF xnor IS
  COMPONENT sn7403
          PORT(a,b:IN qsim_state_vector(1 TO 4);c:OUT qsim_state_vector(1 TO 4));
  END COMPONENT;
  FOR ALL sn7403 USE ENTITY WORK.sn7403(struct);
  SIGNAL r1, r2, r3:anded_qsim_state:='Z';
BEGIN
          U1:sn7403(x,y,r1, x,                  --a(1), a(2), a(3), a(4)
                    x,y,y,r2,                   --b(1), b(2), b(3), b(4)
                    r1, r2, r3, r3);            --c(1), c(2), c(3), c(4)
END struct;
```

NOTE:
r1, r2 are wired_and bus with one driver. r3 is a wired_and bus with two drivers.

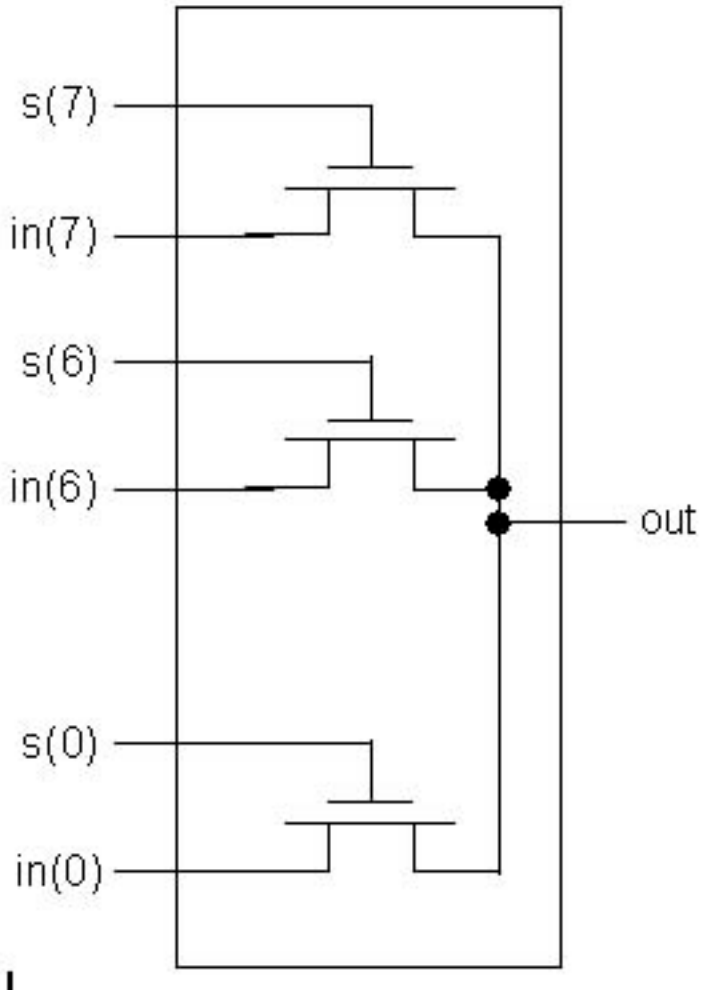


8.3.3 WIRED BUS - MOS Multiplexer

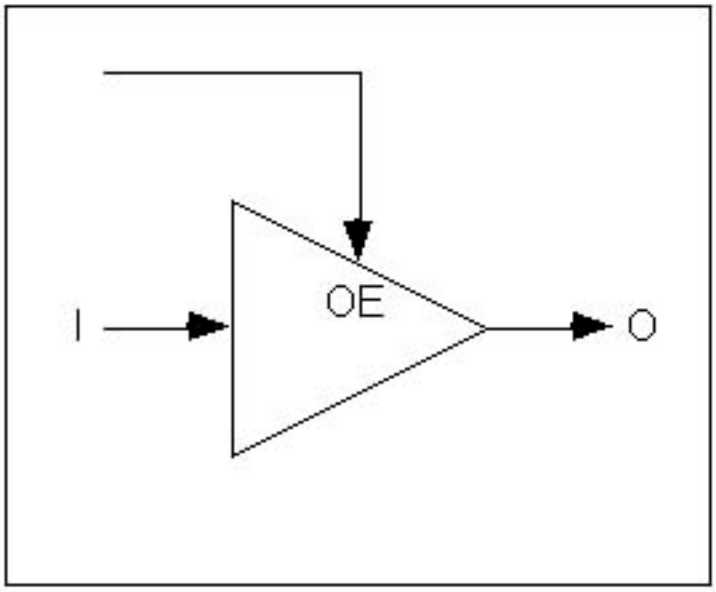
```
ENTITY mux_8_1 IS
        PORT(in,s :IN qsim_state_vector(7 DOWNTO 0); out:OUT qsim_state);
END mux_8_1;

ARCHITECTURE mult_guard OF mux_8_1 IS
        SIGNAL temp : wired_qsim_state BUS;
BEGIN
        b7: BLOCK(s(7)='1')BEGIN temp<=GUARDED in(7);END BLOCK
        b6: BLOCK(s(6)='1')BEGIN temp<=GUARDED in(6);END BLOCK
        b5: BLOCK(s(5)='1')BEGIN temp<=GUARDED in(5);END BLOCK
        b4: BLOCK(s(4)='1')BEGIN temp<=GUARDED in(4);END BLOCK
        b3: BLOCK(s(3)='1')BEGIN temp<=GUARDED in(3);END BLOCK
        b2: BLOCK(s(2)='1')BEGIN temp<=GUARDED in(2);END BLOCK
        b1: BLOCK(s(1)='1')BEGIN temp<=GUARDED in(1);END BLOCK
        b0: BLOCK(s(0)='1')BEGIN temp<=GUARDED in(0);END BLOCK
        out <= temp;
END mult_guard;
```

An interesting situation arises when all the drivers are disconnected from it, and that is all eight implied GUARD signals are FALSE. In this case, because of the BUS keyword is used in the declaration of temp, the wiring resolution function is called with a NULL input parameter. The definition of the wiring function specifies that the initial value of the accumulate variable (`Z') is returned as the function value if the entire loop statement in the statement part of this function is skipped due to a NULL range.



9.3.4 Tri-State Buffers



The block statement can be used to define tri-state logic. To synthesize tri-state devices, Autologic VHDL utilizes the notions of disconnect, guarded assignments, and resolved bus signals. When the guard expression of a bus signal is not true, the driver is disconnected from the target signal. This means that other signals can drive the target, which requires a resolution function to resolve the value of the target. The driving signal must be of the STD_LOGIC type. The resolved signal must be of signal kind bus.

The following illustrates the Autologic VHDL implementation of a single tri-state buffer:

```
1       LIBRARY IEEE;
2       USE IEEE.STD_LOGIC_1164.ALL;
3
4       ENTITY triout IS
5               PORT( oe : IN BIT; i : IN STD_LOGIC;
6                       o : OUT STD_LOGIC BUS)
7       END triout;
8
9       ARCHITECTURE arch_triout IS
10      tri_out:  BLOCK (oe ='1')
11              BEGIN
12                      o <= GUARDED i;
13              END BLOCK;
14      END arch_triout;
```

The following illustrates the CADENCE VHDL implementation of a single tri-state buffer:

```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

ENTITY triout IS
PORT( oe : IN bit; i : IN STD_LOGIC; o : OUT STD_LOGIC BUS);
END triout;
```

```
ARCHITECTURE arch_triout of triout IS
        BEGIN
                process(oe,i)
                begin
                        if oe='1' then
                                o<= i ;
                        else
                                o<= 'Z' ;
                        end if;
                end process;
END arch_triout;
```

The following is the Autologic VHDL implementation of a bank of four tri-state bufferes:

```
1       LIBRARY IEEE;
2       USE IEEE.STD_LOGIC_1164.ALL;
3
4       ENTITY tri_buf4 IS
5               PORT (oe : IN STD_LOGIC;
6                       i : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
7                       o : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) BUS);
8       END tri_buf4;
9
10      ARCHITECTURE arch_tri_buf4 OF tri_buf4 IS
11      BEGIN
12       tri_out: BLOCK (oe = '1')
13                      o <= GUARDED i;
14              END BLOCK tri_out;
END arch_tri_buf4;
```

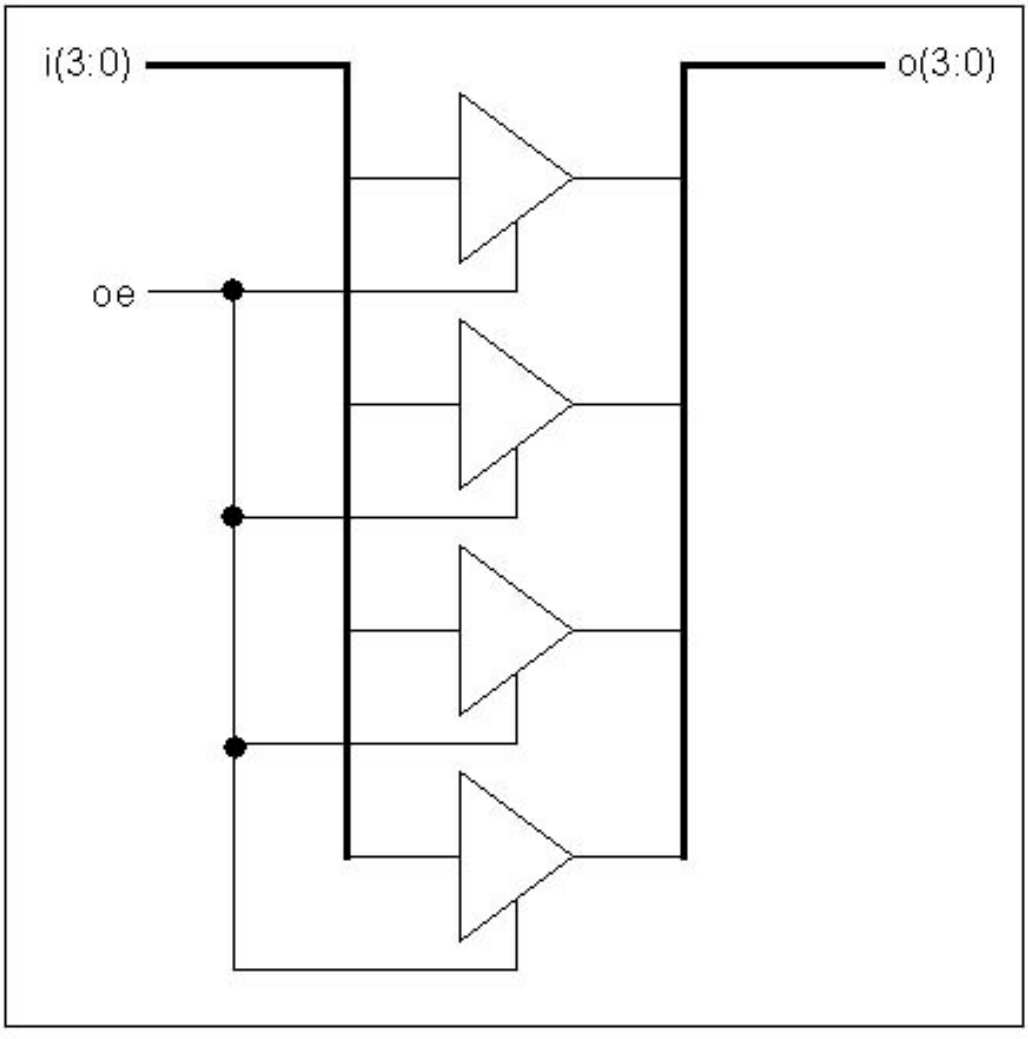
The following is the CADENCE VHDL implementation of a bank of four tri-state bufferes:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY tri_buf4 IS
PORT( oe : IN bit; i : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        o : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) BUS);
END tri_buf4;

ARCHITECTURE arch_tri_buf4 of tri_buf4 IS
        BEGIN
                process(oe,i)
                begin
                if oe='1' then
                        o<= i ;
                else
                        o<= "ZZZZ" ;
                end if;
                end process;
END arch_tri_buf4;
```

If the target signal o was not defined as of kind **BUS**, a bank of four transparent latches is implemented rather than a bank of four tri-state buffers.



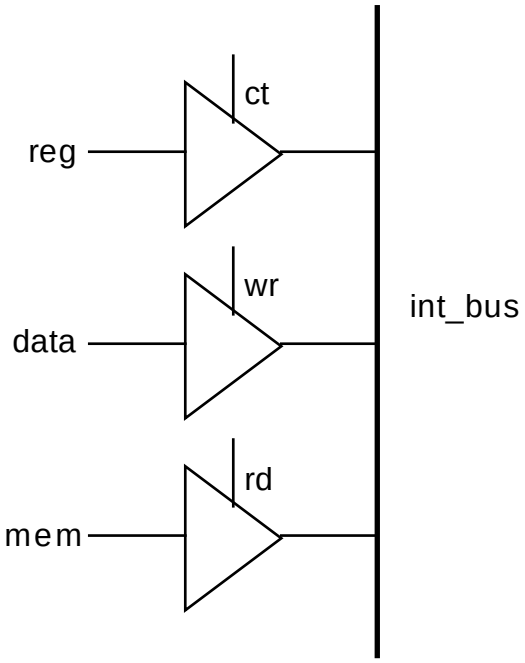
8.3.5 BUS AND REGISTER SIGNALS

The methods for guarding signals is to specify REGISTER or BUS as the signal kind in a signal declaration. Registers and buses are automatically guarded signals. Signals of the kind register, retain the last output value when all drivers are disconnected. Signals of the kind bus, re-evaluate the output value when all drivers are disconnected.

Events and transactions on the BUS and REGISTER kind of signals are exactly the same as long as at least one driver is turned on. If an event turns off the last active driver of a guarded signal, the resolution function is called for BUS signals with a NULL parameter; however, it will not be called if the kind is REGISTER. After all the drivers are turned off, the latter kind of signals retain their lasr driven value.

Example:

```
ENTITY Buses IS
        GENERIC(Delay:TIME:=0 ns);
        PORT(ct, wr, rd:IN STD_LOGIC; dbus:OUT STD_LOGIC BUS);
END Buses;
ARCHITECTURE mult_driv OF Buses IS
        SIGNAL Int_bus: STD_LOGIC BUS;
BEGIN
    b5:  BLOCK(ct='1' )
         BEGIN
                Int_bus <= GUARDED reg AFTER Delay ns;
         END BLOCK;
    b6:  BLOCK(wr='1')
         BEGIN
                Int_bus <= GUARDED data AFTER Delay ns;
         END BLOCK;
   b10:  BLOCK(rd='1')
         BEGIN
                Int_bus <= GUARDED mem AFTER Delay ns;
         END BLOCK
         dbus<= Int_bus;
 END mult_driv;
```



8.3.6 DISCONNECTION SPECIFICATION (NOT supported by Autologic)

In the guarded signal assignment

s<=GUARDED data AFTER 5 ns;

If data changes value while the guard expression is TRUE, the new value of data will be assigned to s after 5 ns. That is, the 5 ns delay applies only when a driver is connected, or being connected, and does not apply when a driver is disconnected from a guarded signal.

A disconnection specification statement can be used to specify the disconnection delay for a guarded signal within a guarded signal assignment. Such a statement contains the name of the signal, its type, and a time expression that specifies the disconnection delay value. To delay the disconnection of the Int_bus drivers, the following statement should be added to the declarative part of this architecture:

DISCONNECT Int_bus: STD_LOGIC AFTER 3 ns;

With the inclusion of this statement, if a Int_bus driver is turned off because its guard expression becomes FALSE, the effect of this driver remains on Int_bus for 3 ns after it has been turned off. The overall effect of this is that the output changes to `Z' 3 ns after the last source has been turned off.

Format:
DISCONNECT signal_name {,signal_name}:signal_name_type AFTER time_expression;
DISCONNECT OTHERS : signal_name_type AFTER time_expression;
DISCONNECT ALL: signal_name_type AFTER time_expression;

- The ALL keyword used for the signal list implies that the disconnection specification applies to all signals of the type specified.
- If OTHERS is used in place of the signal list, the disconnection specification applies to signals of the specified type for which disconnection has not been specified in the statements above this statement.

Example :

```
SIGNAL t: STD_LOGIC BUS; -- guarded signal declaration
DISCONNECT t: STD_LOGIC AFTER 10 ns; disconnection specification
```

There is a disconnection specification for every guarded signal, whether you explicitly define one or you elect to use the implicit default. The default specification is:

```
DISCONNECT guarded_sig_name:guarded_sig_type AFTER 0 ns;
```