

Forcing Signal Errors with VHDL

Ben Cohen

VhdlCohen Publishing
<http://www.vhdlcohen.com/>
vhdlcohen@aol.com
January 3, 2001

ABSTRACT

This paper presents a technique, which uses the user-defined resolution function feature of VHDL, to selectively control from VHDL the assertion of errors imposed on testbench signals of type *Std_Logic*. This technique allows the testbench environment to selectively inject errors at specific times and with specific values onto signals to verify the design-under-test responses to interface errors.

1 Introduction

Many designs require the verification of the corrective actions when an interface error occurs on a port of any mode (i.e., *in*, *out*, or *inout*). Examples include detection and flagging of parity/framing errors, error detection and correction, redundancy voting, and subsystem switching. However, the forcing of those bit errors must be performed without modifying or disturbing the design interfaces. This presents a problem in VHDL for signals of type *Std_Logic* because of the inherent resolution function defined in *Std_Logic_1164* package. This problem is demonstrated in Figure 1, where two subblocks are interfaced with a signal *DATA* of value "1101". If the desired error is attempted by having a MASTER ERROR INJECTOR assert "0010" to force the bus to a value of "0010", then the bus will resolve to "XXXX" instead of the desired error value.

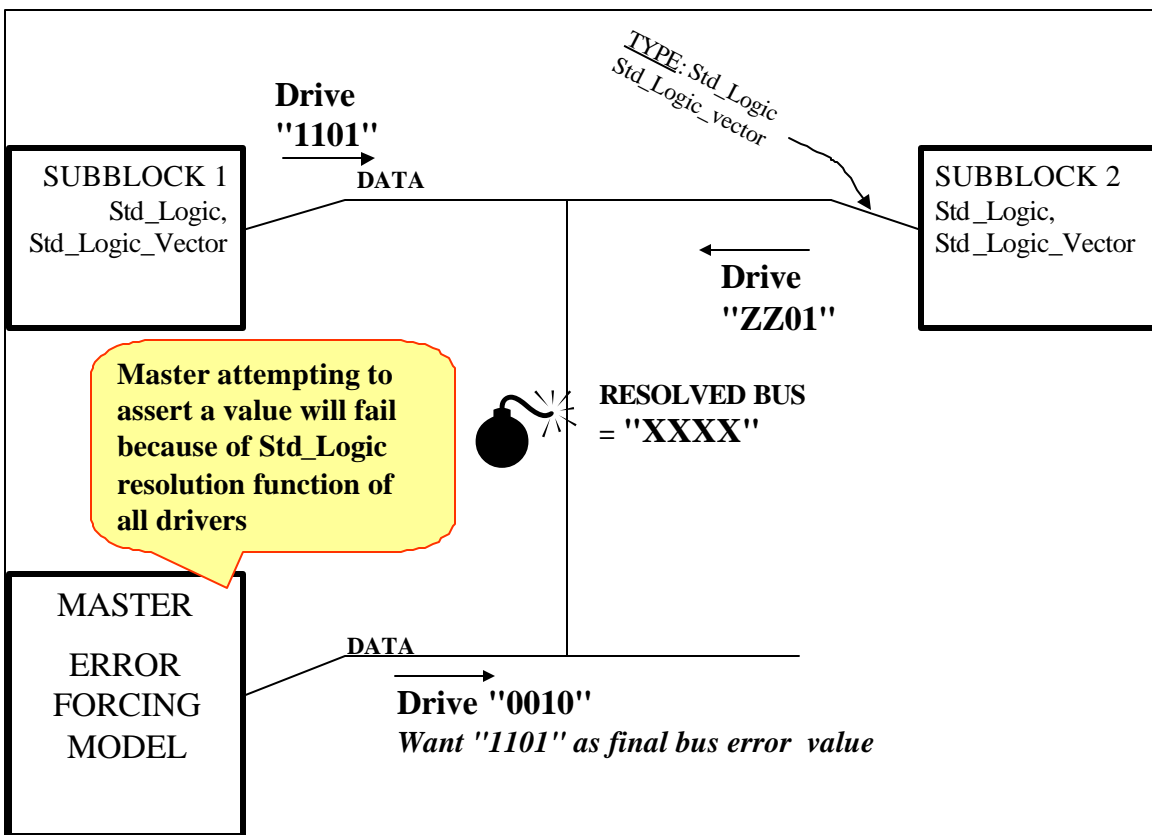


Figure 1 Incorrectly Attempting to Force a Value onto a Bus

Another alternative in forcing the desired value is to use *TCL* language or Programming Language Interface (PLI), or the native simulator simulation control language. The issues with such an approach include non-portability and complexity.

2 Use of User-Defined Resolution Function

A viable all VHDL alternative is to define a user-defined resolution function that operates on a signal of record type, and that resolves all signals driving the bus (or wire) to a value that depends upon the status of the desired error, as defined by the MASTER error injector. The record type that is operated by this special resolution function includes three fields:

1. **ID field.** It is of enumerated data type SLAVE or MASTER
2. **CONTROL field.** The control field has significance ONLY if the ID field for the driver is a MASTER. The control field is of enumerated data type NONE, BIT_FLIP, or VALUE.
 * **NONE** implies a normal, NO error condition operation
 * **BIT_FLIP** implies an INVERT of the value computed as normal (i.e., in the no error state) provided the DATA field bit is equal to '1'. Otherwise, there is no bit flip.
 * **VALUE** implies forcing the signal to the value defined in the DATA field.
3. **DATA field.** For the MASTER, this field represents the bit to be flipped if the control is BIT_FLIP and the data = '1', or the final value of the resolved data field if the control is VALUE. For a SLAVE, this field represents the port data.

There can only be one MASTER error injector in the model. All other interfaces driving this signal must be of SLAVE ID. Figure 2 represents two subblocks interconnected with a data signal of this resolved record type.

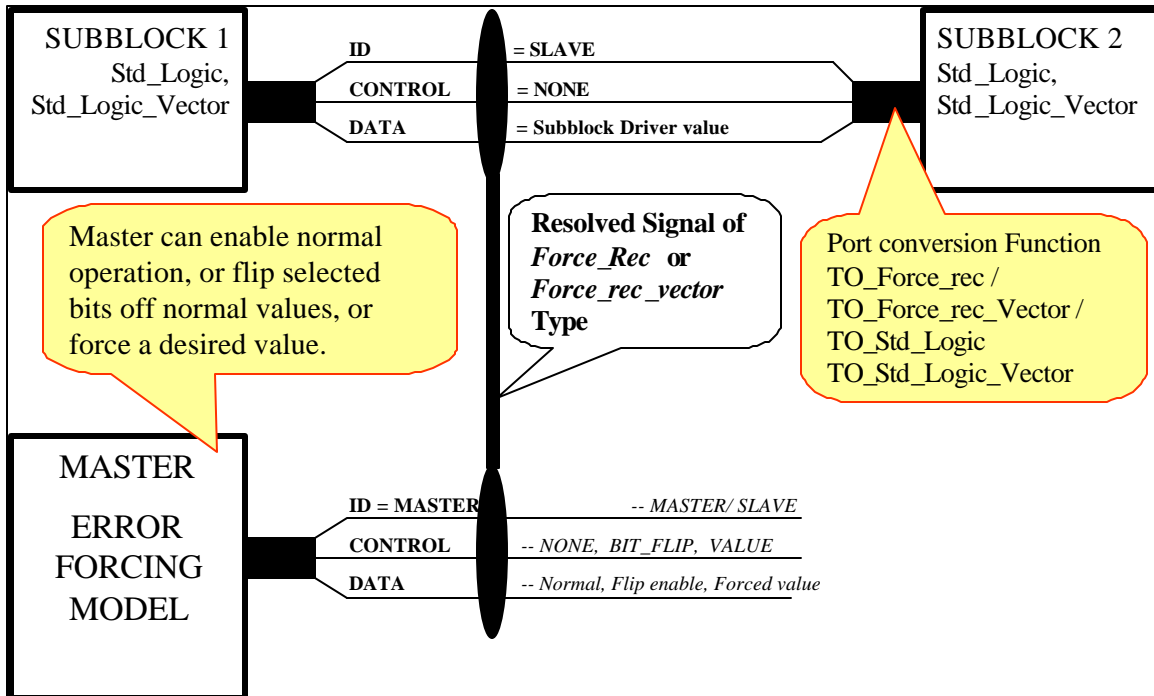


Figure 2 Forcing a Value onto a Bus with Resolved Record Type

2.1 Method of Operation

The resolution function and port conversion functions necessary to support this concept are provided in a VHDL package called *Force_Pkg* (all code available at <http://www.vhdlcohen.com/>). This package defines the record type as shown below:

```
type ID_Typ is (SLAVE, MASTER); -- Master can force signal to desired value
type Control_Typ is (NONE, BIT_FLIP, VALUE);
type Force_uRec_Typ is record
  ID    : ID_Typ; -- Who is master
  Control : Control_Typ; -- Type of Master control
    -- None: causes data to be as defined by std_logic_1164
    -- BIT_FLIP: causes a bit flip of resolved value if data = '1' and MASTER
    -- else, no bit flip
    -- VALUE : causes resolution function to resolve to VALUE of m master
  Data    : std_logic; -- Data value if Control = VALUE
    -- '1' = invert bit if Control = BIT_FLIP, or SLAVE data
end record Force_uRec_Typ;
```

The *Force_Pkg* package also defines a resolution function, handled by the simulator and transparent to the user, that defines how signals of the resolved force record type are resolved. The resolution function first computes the resolved data value for all VHDL SLAVE drivers (i.e., ID = SLAVE). It then examines the actions directed by the VHDL MASTER driver. If the CONTROL instruction is NONE, then the resolved value is the one computed for all the SLAVES. If the CONTROL instruction is BIT_FLIP, then for every bit of the MASTER's data field that is '1', the resolved data field for that bit position is the invert of the computed data field. If it is a '0', then the resolved value is the one computed for all the SLAVES for that bit. If the CONTROL instruction is VALUE, then the resolved value is a copy of the MASTER's data field.

The resolved bus type is of type *Force_Rec* or *Force_Rec_Vector* type. To support the type conversions for the SLAVE ports in the port association to and from *Std_Logic*, *Std_Logic_Vector* and *Force_Rec*, *Force_Rec_Vector* the package provides the functions shown below.

```
-- Conversions to std_logic and std_logic_vector
-- Used in ports for type conversions. Example:
-- TO_Force_rec_Vector(Yout32) => TO_Std_Logic_vector(Yout32)); -- [inout]
function TO_Std_Logic (S : Force_rec) return std_Logic;
function TO_Std_Logic_vector (S : Force_Rec_vector) return std_logic_vector;

-- conversions to Force_Rec and Force_Rec_vector. Examples
-- TO_Force_rec(Yout)      => Yout,          -- [out]
-- TO_Force_rec_Vector(Yout32) => TO_Std_Logic_vector(Yout32)); -- [inout]
function TO_Force_rec (S : std_Logic) return Force_rec;      -- ID = SLAVE
function TO_Force_rec_Vector (S : std_logic_vector) return Force_Rec_vector;
```

To support the MASTER interface to ports of the *Force_Rec* or *Force_Rec_Vector* types, the package provides the functions shown below.

```

-- Sets up an object of Force_Rec to a MASTER with FlipBit control
-- if S = '1', then bit is flipped
-- EXamples
-- Yout <= FlipBit('1');
-- Yout32 <= FlipBit(X"0000FF01");
function FlipBit (S : std_logic) return Force_Rec; -- ID = MASTER

-- is S(I) = '1', then bit is flipped
function FlipBit (S : std_logic_vector) return Force_Rec_vector; -- ID = MASTER
--
-----
-- Sets up an object of Force_Rec to a MASTER with VALUE control
-- EXamples
-- Yout <= SetValue('0');
-- Yout32 <= SetValue(X"0000FF01"); -- resolved to X"0000FF01"
function SetValue (S : std_logic) return Force_Rec; -- ID = MASTER

function SetValue (S : std_logic_vector) return Force_Rec_vector; -- ID = MASTER

-----
-- Sets up an object to a MASTER with NONE control select
-- EXamples
-- Yout <= NoError('0');
-- Yout32 <= NoError(s32);
function NoError (S : std_logic) return Force_Rec; -- ID = MASTER
-- S IS USED FOR SIZING
function NoError (S : std_logic_vector) return Force_Rec_vector;

```

2.2 Application Example

Figure 2.2-1 defines the device under test (DUT) with component name A.

```

library ieee;
use ieee.std_logic_1164.all;
entity A is
port (
    OutEnb : in std_Logic;
    Xin32 : in std_logic_vector(31 downto 0);
    Yout : out std_Logic;
    Yout32 : inout std_logic_vector(31 downto 0));
end entity A;

```

```

architecture RTL of A is
begin -- architecture RTL
  Yout32 <= Xin32 when OutEnb = '1' else
    (others => 'Z');
  Yout <= not Xin32(0) when OutEnb = '1' else
    'Z';
end architecture RTL;

```

Figure 2.2-1 Test Component

The testbench is shown in Figure 2.2-2.

```

library ieee;
  use ieee.std_logic_1164.all;
library Work;
  use Work.Force_Pkg.all;
entity TestForceTop is
end entity TestForceTop;
-----
architecture Beh of TestForceTop is
  signal OutEnb0 : std_logic := '1';
  signal OutEnb1 : std_logic := '0';
  signal Xin32   : std_logic_vector(31 downto 0) := X"AB32D4F6";
  signal Yout    : Force_rec;
  signal Yout32  : Force_Rec_vector(31 downto 0);
  signal s32     : std_logic_vector(31 downto 0);
  signal s1      : std_logic;
  signal Clk     : std_logic := '1';
begin -- architecture TestForceTop
  -- Testbench clock
  Clk <= not Clk after 50 ns;

  -- One instantiation of DUT
  A_0 : entity work.A
  port map (
    OutEnb          => OutEnb0,          -- [in]
    Xin32           => Xin32,           -- [in]
    TO_Force_rec(Yout) => Yout,          -- [out]
    TO_Force_rec_Vector(Yout32) => TO_Std_Logic_vector(Yout32)); -- [inout]

  -- Another instantiation of DUT
  A_1 : entity work.A
  port map (
    OutEnb          => OutEnb1,          -- [in]
    Xin32           => Xin32,           -- [in]
    TO_Force_rec(Yout) => Yout,          -- [out]
    TO_Force_rec_Vector(Yout32) => TO_Std_Logic_vector(Yout32)); -- [inout]

```

```

-- Test driver
-- purpose: Drives component A. NO FAULTS
Drive_Proc: process is
begin -- process Drive_Proc
  wait until Clk = '1';
  OutEnb0 <= not OutEnb0 after 1 ns;
  OutEnb1 <= not OutEnb1 after 1 ns;
  Xin32 <= not Xin32 after 1 ns;
end process Drive_Proc;

s1 <= Yout.Data;          -- FOR MONITORING

-- ERROR INJECTION using a separate concurrent statement.
-- A process is shown, but a component could have also been used.
ErrorInjection_Proc: process is
begin -- process ErrorInjection
  -- NO ERRORS
  Yout <= NoError('0');
  Yout32 <= NoError(s32);
  -- update signals
  wait for 2 ns;
  S32 <= TO_Std_Logic_vector(Yout32); -- for monitoring
  wait until Clk = '1';

-- SETTING A FORCED VALUE
  Yout <= SetValue('0');
  Yout32 <= SetValue(X"0000FF01");
  -- update signals
  wait for 2 ns;
  S32 <= TO_Std_Logic_vector(Yout32); -- for monitoring

  wait until clk = '1';
  -- SETTING FLIP BITS
  Yout <= FlipBit('1');
  Yout32 <= FlipBit(X"0000FF01");
  -- update signals
  wait for 2 ns;
  S32 <= TO_Std_Logic_vector(Yout32); -- for monitoring
  wait until Clk = '1';
end process ErrorInjection_Proc;
end architecture Beh;

```

Figure 2.2-2 Testbench with Error Injection

Figure 2.2-3 represents waveform simulation of the model. S32 represents a copy of the resolved value of the *Yout32* signal. Note that the error injector operates as expected under the controls of NONE, BIT-FLIP, and VALUE from the MASTER error injector.

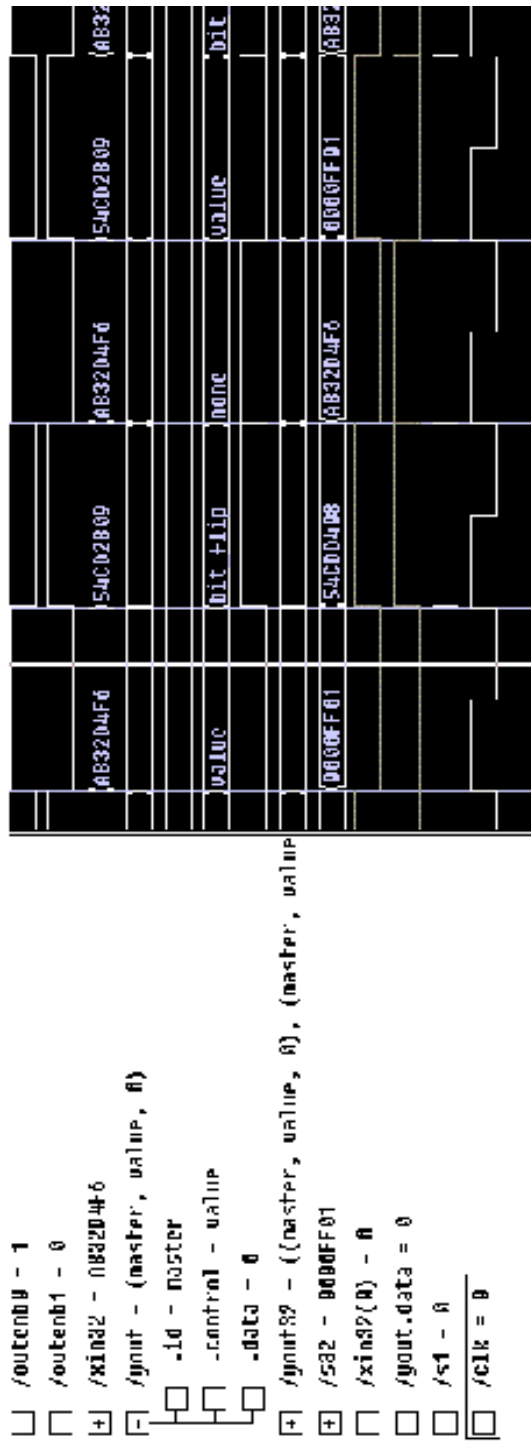


Figure 2.2-3 Simulation Results