CPE/EE 422/522
Advanced Logic Design
L12

Electrical and Computer Engineering
University of Alabama in Huntsville

Outline

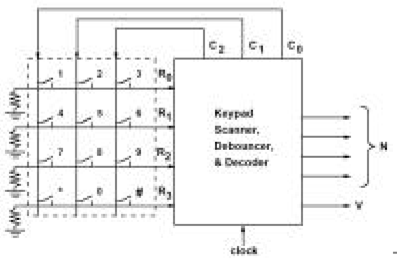
- What we know
  - How to model Combinational Networks in VHDL
    - Structural, Dataflow, Behavioral
  - How to model Flip-flops in VHDL
  - Processes
  - Delays (delta, transport, inertial)
  - How to model FSM in VHDL
  - Wait statements
  - Variables, Signals, Arrays
- What we do not know
  - VHDL Operators
  - Procedures, Functions
  - Packages, Libraries
  - Additional Topics (if time)

LAB 4: Keypad Scanner

- Lab4 preparation material
- Telephone keypad scanner
  - Section 3.5 in the textbook
  - Implemented using PLD (not relevant for you)

LAB 4: Block Diagram

- Keypad is wired in matrix form
  - switches are at the intersections of rows and columns
- Assumption: only one key is pressed at time



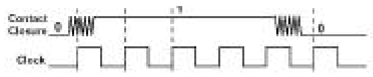
- N=N3N2N1N0
  - 0 – 0000
  - ...
  - 9 – 1001
  - * - 1010
  - # - 1011
- V=1: when valid key is detected it is active for one clock cycle time

LAB 4: Scan Procedure

1. Apply logic 1s to columns C0, C1, C2 and wait
2. If any key is pressed
   a 1 will appear on R0, R1, R2, or R3
3. Apply 1 to column C0 only;
   if any of Ri's is 1, a valid key is detected;
   set V=1 and corresponding N
4. If no key is detected in column C0 apply 1 on C1;
   Repeat the same for C2
5. When a valid key is detected, apply 1s
   to C0, C1, C2 and wait until no key is pressed
   - ensure that only one valid signal is generated each time a key is pressed

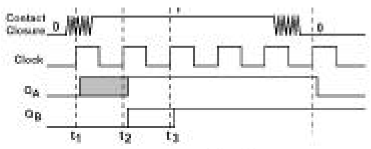
LAB 4: Debouncing

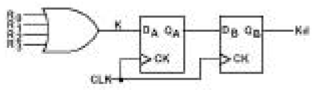
- Problem: with mechanical switch the contact will bounce causing noise in the switch output
  - contact may bounce for several milliseconds



- Solution: after a switch closure has been detected, wait for bounce to settle down before reading the key

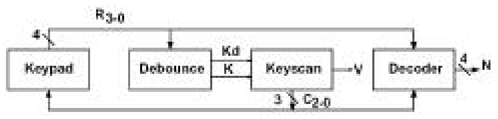
LAB 4: Debouncing and Synchronization Circuit



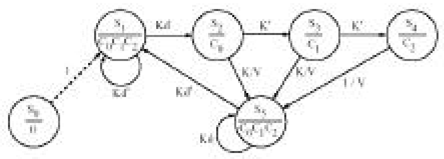


- Proposed debouncing circuit
- Important:
  <u>clock cycle time must be greater than the bounce time</u>

LAB 4: Scanner Modules

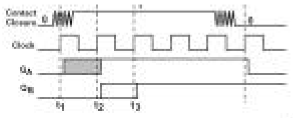


Scanner

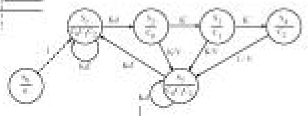


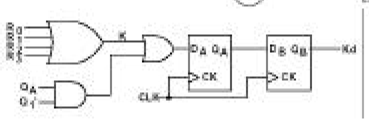
LAB 4: Scanner

- Problem: what is Kd in S5 if we have a key pressed in column C2?

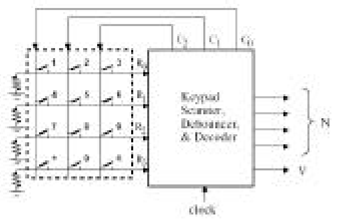


- Solution





LAB 4: Decoder



R_3	R_2	R_1	R_0	C_0	C_1	C_2	N_3	N_2	N_1	N_0	
0	0	0	1	1	0	0	0	0	0	1	
0	0	0	1	0	1	0	0	0	1	0	
0	0	0	1	0	0	1	0	0	1	1	
0	0	1	0	1	0	0	0	1	0	0	
0	0	1	0	0	1	0	0	1	0	1	
0	0	1	0	0	0	1	0	1	1	0	
0	1	0	0	1	0	0	0	1	1	1	
0	1	0	0	0	1	0	1	0	0	0	
0	1	0	0	0	0	1	1	0	0	1	
1	0	0	0	1	0	0	1	0	1	0	(*)
1	0	0	0	0	1	0	0	0	0	0	
1	0	0	0	0	0	1	1	0	1	1	(#)

Logic equations for decoder:

```
N3 = R2 C0' + R3 C1'
N2 = R1 + R2 C0
N1 = R0 C0' + R2'C2 + R1'R0'C0
N0 = R1 C1 + R1'C2 + R3'R1'C1'
```

Review: VHDL Functions

- Functions execute a sequential algorithm and return a single value to calling program

```
function rotate_right (reg : bit_vector)
    return bit_vector is
begin
    return  reg ror 1;
end rotate_right;
```

- A = “10010101”

```
B <= rotate_right(A);
```

- General form

```
function function-name (formal-parameter-list)
    return return-type is
    [declarations]
begin
    sequential statements  -- must include return return-value;
end function-name;
```

Review: For Loops

General form of a for loop:

```
[loop-label:] for loop-index in range loop
        sequential statements
end loop [loop-label];
```

Exit statement has the form:

```
exit;                    -- or
exit when condition;
```

For Loop Example:

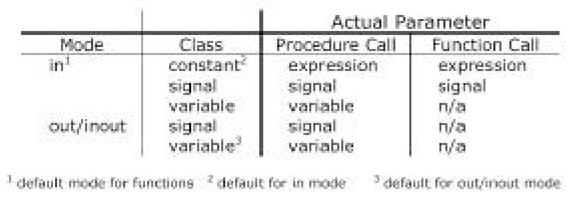
```
-- compare two 8-character strings and return TRUE if equal
function comp_string(string1, string2: string(1 to 8))
    return boolean is
variable B: boolean;
begin
    loopex: for j in 1 to 8 loop
        B := string1(j) = string2(j);
        exit when B=FALSE;
    end loop loopex;
    return B;
end comp_string;
```

Review: VHDL Procedures

- Facilitate decomposition of VHDL code into modules
- Procedures can return any number of values using output parameters
- General form

```
procedure procedure_name (formal-parameter-list) is
   [declarations]
   begin
     Sequential-statements
   end procedure_name;


procedure_name (actual-parameter-list);
```

Review: Parameters for Subprogram Calls

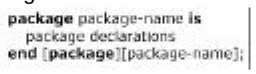


		Actual Parameter	
Mode	Class	Procedure Call	Function Call
in[1]	constant[2]	expression	expression
	signal	signal	signal
	variable	variable	n/a
out/inout	signal	signal	n/a
	variable[3]	variable	n/a

[1] default mode for functions [2] default for in mode [3] default for out/inout mode

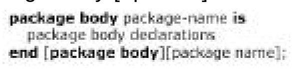
Packages and Libraries

- Provide a convenient way of referencing frequently used functions and components
- Package declaration



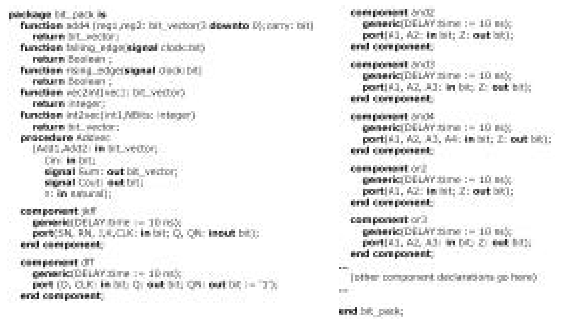
```
package package-name is
   package declarations
end [package][package-name];
```

- Package body [optional]



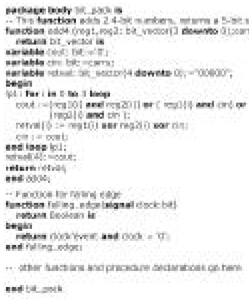
```
package body package-name is
   package body declarations
end [package body][package name];
```

Library BITLIB – bit_pack package



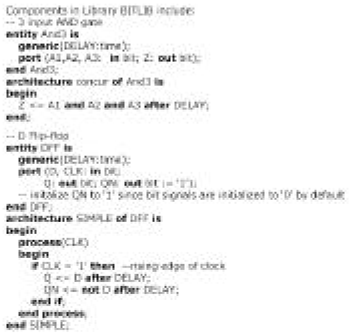
```
package bit_pack is
  function add4 (reg1,reg2: bit_vector(3 downto 0);carry: bit)
    return bit_vector;
  function falling_edge(signal clock:bit)
    return Boolean ;
  function rising_edge(signal clock:bit)
    return Boolean ;
  function vec2int(vec1: bit_vector)
    return integer;
  function int2vec(int1,NBits: integer)
    return bit_vector;
  procedure Addvec
    (Add1,Add2: in bit_vector;
      Cin: in bit;
      signal Sum: out bit_vector;
      signal Cout: out bit;
      n: in natural);

  component jkff
    generic(DELAY:time := 10 ns);
    port(SN, RN, J,K,CLK: in bit; Q, QN: inout bit);
  end component;

  component dff
    generic(DELAY:time := 10 ns);
    port (D, CLK: in bit; Q: out bit; QN: out bit := '1');
  end component;

  component and2
    generic(DELAY:time := 10 ns);
    port(A1, A2: in bit; Z: out bit);
  end component;

  component and3
    generic(DELAY:time := 10 ns);
    port(A1, A2, A3: in bit; Z: out bit);
  end component;

  component and4
    generic(DELAY:time := 10 ns);
    port(A1, A2, A3, A4: in bit; Z: out bit);
  end component;

  component or2
    generic(DELAY:time := 10 ns);
    port(A1, A2: in bit; Z: out bit);
  end component;

  component or3
    generic(DELAY:time := 10 ns);
    port(A1, A2, A3: in bit; Z: out bit);
  end component;
--
  (other component declarations go here)
--

end bit_pack;
```

Library BITLIB – bit_pack package



```
package body bit_pack is
-- This function adds 2 4-bit numbers, returns a 5-bit sum
function add4 (reg1,reg2: bit_vector(3 downto 0);carry: bit)
   return bit_vector is
variable cout: bit:='0';
variable cin: bit:=carry;
variable retval: bit_vector(4 downto 0):="00000";
begin
lp1: for i in 0 to 3 loop
   cout := (reg1(i) and reg2(i)) or ( reg1(i) and cin) or
           (reg2(i) and cin );
   retval(i) := reg1(i) xor reg2(i) xor cin;
   cin := cout;
end loop lp1;
retval(4):=cout;
return retval;
end add4;

-- Function for falling edge
function falling_edge(signal clock:bit)
   return Boolean is
begin
   return clock'event and clock = '0';
end falling_edge;

-- other functions and procedure declarations go here

end bit_pack;
```

Library BITLIB – bit_pack package



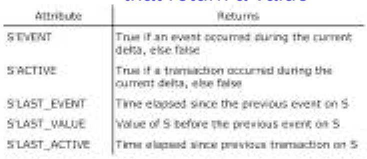
```
Components in Library BITLIB include:
-- 3 input AND gate
entity And3 is
   generic(DELAY:time);
   port (A1,A2, A3:  in bit; Z: out bit);
end And3;
architecture concur of And3 is
begin
   Z <= A1 and A2 and A3 after DELAY;
end;

-- D flip-flop
entity DFF is
   generic(DELAY:time);
   port (D, CLK: in bit;
         Q: out bit; QN: out bit := '1');
   -- initialize QN to '1' since bit signals are initialized to '0' by default
end DFF;
architecture SIMPLE of DFF is
begin
   process(CLK)
   begin
      if CLK = '1' then  --rising edge of clock
         Q <= D after DELAY;
         QN <= not D after DELAY;
      end if;
   end process;
end SIMPLE;
```

Additional Topics in VHDL

- Attributes
- Transport and Inertial Delays
- Operator Overloading
- Multivalued Logic and Signal Resolution
- IEEE 1164 Standard Logic
- Generics
- Generate Statements
- Synthesis of VHDL Code
- Synthesis Examples
- Files and Text IO

Signal Attributes

Attributes associated with signals
that return a value



Attribute	Returns
S'EVENT	True if an event occurred during the current delta, else false
S'ACTIVE	True if a transaction occurred during the current delta, else false
S'LAST_EVENT	Time elapsed since the previous event on S
S'LAST_VALUE	Value of S before the previous event on S
S'LAST_ACTIVE	Time elapsed since previous transaction on S

A'event – true if a change in S has just occurred

A'active – true if A has just been reevaluated, even if A does not change

Signal Attributes (cont’d)

- Event
  - – occurs on a signal every time it is changed
- Transaction
  - – occurs on a signal every time it is evaluated
- Example:

```
A <= B  - -  B changes at time T
```

	A’event	B’event
T		
T + 1d		

Signal Attributes (cont’d)

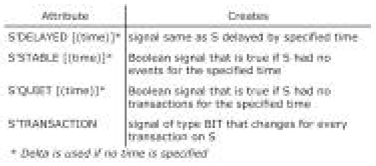
```
entity test is
end;
architecture bmtest of test is
   signal A : bit;
   signal B : bit;
   signal C : bit;
begin
   A <= not A after 20 ns;
   B <= '1';
   C <= A and B;
  process(A, B, C)
   variable Aev : bit;
   variable Aac : bit;
   variable Bev : bit;
   variable Bac : bit;
   variable Cev : bit;
   variable Cac : bit;
begin
   if (A'event ) then Aev := '1';
   else Aev := '0';
   end if;
   if (A'active) then Aac := '1';
   else Aac := '0';
   end if;
   if (B'event ) then Bev := '1';
   else Bev := '0';
   end if;
   if (B'active) then Bac := '1';
   else Bac := '0';
   end if;
   if (C'event ) then Cev := '1';
   else Cev := '0';
   end if;
   if (C'active) then Cac := '1';
   else Cac := '0';
   end if;

end process;

end bmtest;
```

Signal Attributes (cont’d)

```
ns          /test/a   /test/line__15/bev
         delta     /test/b   /test/line__15/bac
                     /test/c   /test/line__15/cev
            /test/line__15/aev   /test/line__15/cac
              /test/line__15/aac
        0  +0        0 0 0 0 0          0 0 0 0
        0  +1        0 1 0 0 0          1 1 0 1
       20  +0        1 1 0 1 1          0 0 0 0
       20  +1        1 1 1 0 0          0 0 1 1
       40  +0        0 1 1 1 1          0 0 0 0
       40  +1        0 1 0 0 0          0 0 1 1
```

Signal Attributes (cont’d)

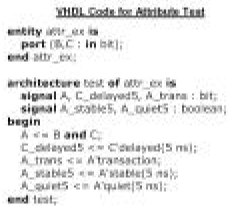
Attributes that create a signal



Attribute	Creates
S'DELAYED [(time)]*	signal same as S delayed by specified time
S'STABLE [(time)]*	Boolean signal that is true if S had no events for the specified time
S'QUIET [(time)]*	Boolean signal that is true if S had no transactions for the specified time
S'TRANSACTION	signal of type BIT that changes for every transaction on S

** Delta is used if no time is specified*

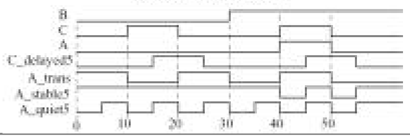
Examples of Signal Attributes



VHDL Code for Attribute Test

```
entity attr_ex is
   port (B,C : in bit);
end attr_ex;

architecture test of attr_ex is
   signal A, C_delayed5, A_trans : bit;
   signal A_stable5, A_quiet5 : boolean;
begin
   A <= B and C;
   C_delayed5 <= C'delayed(5 ns);
   A_trans <= A'transaction;
   A_stable5 <= A'stable(5 ns);
   A_quiet5 <= A'quiet(5 ns);
end test;
```

Waveforms for Attribute Test



Using Attributes for Error Checking

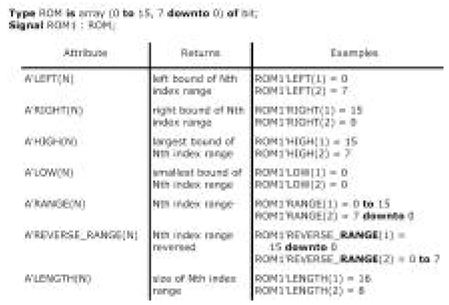
```
check: process
begin
   wait until rising_edge(Clk);
   assert (D’stable(setup_time))
        report(“Setup time violation”)
        severity error;
   wait for hold_time;
   assert (D’stable(hold_time))
        report(“Hold time violation”)
        severity error;
end process check;
```

Assert Statement

```
assert boolean-expression
   report string-expression
   severity severity-level
```

- If boolean expression is false
  display the string expression on the monitor
- Severity levels: Note, Warning, Error, Failure

Array Attributes



```
Type ROM is array (0 to 15, 7 downto 0) of bit;
Signal ROM1 : ROM;
```

Attribute	Returns	Examples
A'LEFT(N)	left bound of Nth index range	ROM1'LEFT(1) = 0 ROM1'LEFT(2) = 7
A'RIGHT(N)	right bound of Nth index range	ROM1'RIGHT(1) = 15 ROM1'RIGHT(2) = 0
A'HIGH(N)	largest bound of Nth index range	ROM1'HIGH(1) = 15 ROM1'HIGH(2) = 7
A'LOW(N)	smallest bound of Nth index range	ROM1'LOW(1) = 0 ROM1'LOW(2) = 0
A'RANGE(N)	Nth index range	ROM1'RANGE(1) = 0 **to** 15 ROM1'RANGE(2) = 7 **downto** 0
A'REVERSE_RANGE(N)	Nth index range reversed	ROM1'REVERSE_**RANGE**(1) = 15 **downto** 0 ROM1'REVERSE_**RANGE**(2) = 0 **to** 7
A'LENGTH(N)	size of Nth index range	ROM1'LENGTH(1) = 16 ROM1'LENGTH(2) = 8

A can be either an array name or an array type.

Array attributes work with signals, variables, and constants.

Recap: Adding Vectors

```
-- This procedure adds two n-bit bit_vectors and a carry and
-- returns an n-bit sum and a carry.  Add1 and Add2 are assumed
-- to be of the same length and dimensioned n-1 downto 0.
procedure Addvec
   (Add1,Add2: in bit_vector;
      Cin: in bit;
      signal Sum: out bit_vector;
      signal Cout: out bit;
      n:in positive) is
      variable C: bit;
begin
   C := Cin;
   for i in 0 to n-1 loop
      Sum(i) <= Add1(i) xor Add2(i) xor C;
      C := (Add1(i) and Add2(i)) or (Add1(i) and C) or (Add2(i) and C);
   end loop;
   Cout <= C;
end Addvec;
```

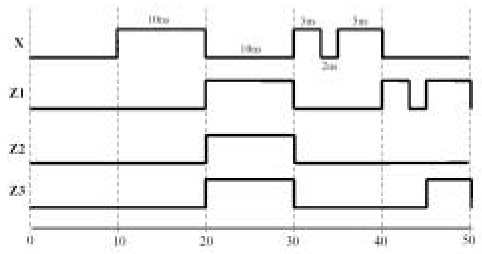
Note: Add1 and Add2 vectors must be dimensioned as *N-1 downto 0*.

Use attributes to write more general procedure that places
no restrictions on the range of vectors other than the lengths must be same.

Procedure for Adding Bit Vectors

```
-- This procedure adds two bit_vectors and a carry and returns a sum
-- and a carry.  Both bit_vectors should be of the same length.

procedure Addvec2
   (Add1,Add2: in bit_vector;
   Cin: in bit;
   signal Sum: out bit_vector;
   signal Cout: out bit) is
   variable C: bit := Cin;
   alias n1 : bit_vector(Add1'length-1 downto 0) is Add1;
   alias n2 : bit_vector(Add2'length-1 downto 0) is Add2;
   alias S : bit_vector(Sum'length-1 downto 0) is Sum;

begin
   assert ((n1'length = n2'length) and (n1'length = S'length))
      report "Vector lengths must be equal!"
      severity error;
   for i in s'reverse_range loop
      S(i) <= n1(i) xor n2(i) xor C;
      C := (n1(i) and n2(i)) or (n1(i) and C) or (n2(i) and C);
   end loop;
   Cout <= C;
end Addvec2;
```

Transport and Inertial Delay



```
Z1 <= transport X after 10 ns;     -- transport delay
Z2 <= X after 10 ns;               -- inertial delay
Z3 <= reject 4 ns X after 10 ns;   -- delay with specified rejection pulse width
```

Transport and Inertial Delay (cont'd)

```
Z3 <= reject 4 ns X after 10 ns;
```

Reject is equivalent to a combination of inertial and transport delay:

```
Zm <= X after 4 ns;
Z3 <= transport Zm after 6 ns;
```

Statements executed at time T
– B at T+1, C at T+2

```
A <= transport B after 1 ns;
A <= transport C after 2 ns;
```

Statements executed at time T
– C at T + 2:

```
A <= B after 1 ns;
A <= C after 2 ns;
```

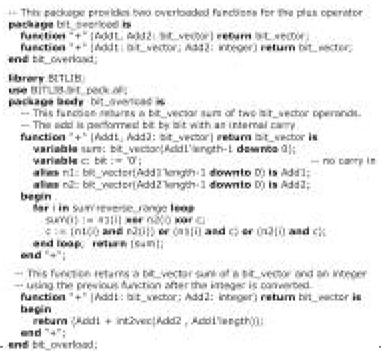
Statements executed at time T
– C at T + 1:

```
A <= transport B after 2 ns;
A <= transport C after 1 ns;
```

Operator Overloading

- Operators +, - operate on integers
- Write procedures for bit vector addition/subtraction
  - addvec, subvec
- Operator overloading allows using + operator to implicitly call an appropriate addition function
- How does it work?
  - When compiler encounters a function declaration in which the function name is an operator enclosed in double quotes, the compiler treats the function as an operator overloading ("+")
  - when a "+" operator is encountered, the compiler automatically checks the types of operands and calls appropriate functions

VHDL Package with Overloaded Operators



```
-- This package provides two overloaded functions for the plus operator
package bit_overload is
  function "+" (Add1, Add2: bit_vector) return bit_vector;
  function "+" (Add1: bit_vector; Add2: integer) return bit_vector;
end bit_overload;

library BITLIB;
use BITLIB.bit_pack.all;
package body bit_overload is
  -- This function returns a bit_vector sum of two bit_vector operands.
  -- The add is performed bit by bit with an internal carry
  function "+" (Add1, Add2: bit_vector) return bit_vector is
    variable sum: bit_vector(Add1'length-1 downto 0);
    variable c: bit := '0';                                   -- no carry in
    alias n1: bit_vector(Add1'length-1 downto 0) is Add1;
    alias n2: bit_vector(Add2'length-1 downto 0) is Add2;
  begin
    for i in sum'reverse_range loop
      sum(i) := n1(i) xor n2(i) xor c;
      c := (n1(i) and n2(i)) or (n1(i) and c) or (n2(i) and c);
    end loop;  return (sum);
  end "+";

 -- This function returns a bit_vector sum of a bit_vector and an integer
 -- using the previous function after the integer is converted.
  function "+" (Add1: bit_vector; Add2: integer) return bit_vector is
  begin
    return (Add1 + int2vec(Add2 , Add1'length));
  end "+";
end bit_overload;
```

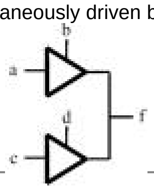
Overloaded Operators

- A, B, C – bit vectors
- A <= B + C + 3 ?
- A <= 3 + B + C ?

- Overloading can also be applied to procedures and functions
  - procedures have the same name – type of the actual parameters in the procedure call determines which version of the procedure is called

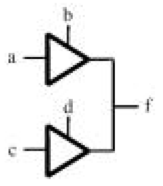
Multivalued Logic

- Bit (0, 1)
- Tristate buffers and buses => high impedance state 'Z'
- Unknown state 'X'
  - e. g., a gate is driven by 'Z', output is unknown
  - a signal is simultaneously driven by '0' and '1'



Tristate Buffers

```
use WORK.fourpack.all;
entity t_buff_exmpl is
   port (a,b,c,d : in X01Z;   -- signals are
         f: out X01Z);         -- 4 valued
end t_buff_exmpl;

architecture t_buff_conc of t_buff_exmpl is
begin
   f <= a when b = '1' else 'Z';
   f <= c when d = '1' else 'Z';
end t_buff_conc;

architecture t_buff_bhv of t_buff_exmpl is
begin
   buff1: process (a,b)
   begin
      if (b='1') then f<=a;
      else
         f<='Z';         --"drive" the output high Z when not enabled
      end if;
   end process buff1;

   buff2: process (c,d)
   begin
      if (d='1') then f<=c;
      else
         f<='Z';         --"drive" the output high Z when not enabled
      end if;
   end process buff2;
end t_buff_bhv;
```



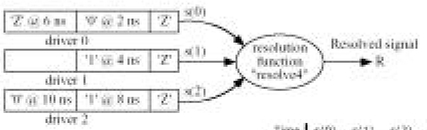
Resolution function to determine the actual value of f since it is driven from two different sources

Signal Resolution

- VHDL signals may either be resolved or unresolved
- Resolved signals have an associated resolution function
- Bit type is unresolved –
  - there is no resolution function
  - if you drive a bit signal to two different values in two concurrent statements, the compiler will generate an error

Signal Resolution (cont'd)

```
signal R : X01Z := 'Z'; ...
R <= transport '0' after 2 ns, 'Z' after 6 ns;
R <= transport '1' after 4 ns;
R <= transport '1' after 8 ns, '0' after 10 ns;
```



	'X'	'0'	'1'	'Z'
'X'	'X'	'X'	'X'	'X'
'0'	'X'	'0'	'X'	'0'
'1'	'X'	'X'	'1'	'1'
'Z'	'X'	'0'	'1'	'Z'

Time	s(0)	s(1)	s(2)	R
0	'Z'	'Z'	'Z'	'Z'
2	'0'	'Z'	'Z'	'0'
4	'0'	'1'	'Z'	'X'
6	'Z'	'1'	'Z'	'1'
8	'Z'	'1'	'1'	'1'
10	'Z'	'1'	'0'	'X'

Resolution Function for X01Z

```
package fourpack is
   type u_x01z is ('X','0','1','Z');       -- u_x01z is unresolved
   type u_x01z_vector is array (natural range <>) of u_x01z;
   function resolve4 (s:u_x01z_vector) return u_x01z;
   subtype x01z is resolve4 u_x01z;
   -- x01z is a resolved subtype which uses the resolution function resolve4
   type x01z_vector is array (natural range <>) of x01z;
end fourpack;

package body fourpack is
   type x01z_table is array (u_x01z,u_x01z) of u_x01z;
   constant resolution_table : x01z_table := (
      ('X','X','X','X'),
      ('X','0','X','0'),
      ('X','X','1','1'),
      ('X','0','1','Z'));
   function resolve4 (s:u_x01z_vector) return u_x01z is
      variable result : u_x01z := 'Z';
   begin
      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 resolve4;
end fourpack;
```

Define AND and OR for 4-valued inputs?

AND and OR Functions Using X01Z

AND	‘X’	‘0’	‘1’	‘Z’
‘X’	‘X’	‘0’	‘X’	‘X’
‘0’	‘0’	‘0’	‘0’	‘0’
‘1’	‘X’	‘0’	‘1’	‘X’
‘Z’	‘X’	‘0’	‘X’	‘X’

OR	‘X’	‘0’	‘1’	‘Z’
‘X’	‘X’	‘X’	‘1’	‘X’
‘0’	‘X’	‘0’	‘1’	‘X’
‘1’	‘1’	‘1’	‘1’	‘1’
‘Z’	‘X’	‘X’	‘1’	‘X’

IEEE 1164 Standard Logic

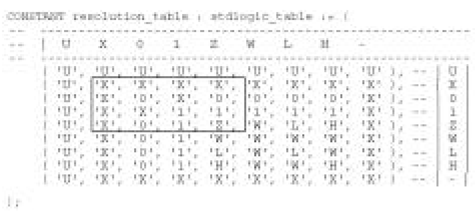
- 9-valued logic system
  - ‘U’ – Uninitialized
  - ‘X’ – Forcing Unknown
  - ‘0’ – Forcing 0
  - ‘1’ – Forcing 1
  - ‘Z’ – High impedance
  - ‘W’ – Weak unknown
  - ‘L’ – Weak 0
  - ‘H’ – Weak 1
  - ‘-’ – Don’t care

If forcing and weak signal are tied together, the forcing signal dominates.

Useful in modeling the internal operation of certain types of ICs.

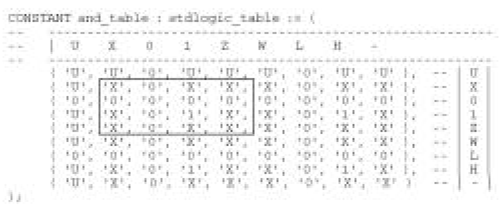
In this course we use a subset of the IEEE values: X10Z

Resolution Function for IEEE 9-valued



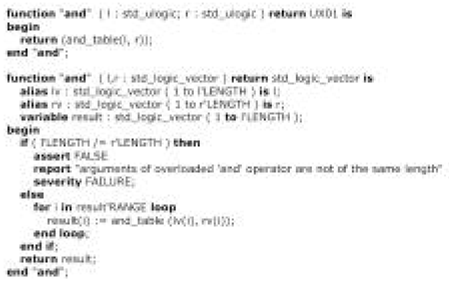
```
CONSTANT resolution_table : stdlogic_table := (
--  ---------------------------------------------------------
--  | U    X    0    1    Z    W    L    H    -
--  ---------------------------------------------------------
    ( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
    ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
    ( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
    ( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
    ( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
    ( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
    ( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
    ( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
    ( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' )  -- | - |
);
```

AND Table for IEEE 9-valued



```
CONSTANT and_table : stdlogic_table := (
--  ---------------------------------------------------------
--  | U    X    0    1    Z    W    L    H    -
--  ---------------------------------------------------------
    ( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U' ),  -- | U |
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | X |
    ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | 0 |
    ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | 1 |
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | Z |
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ),  -- | W |
    ( '0', '0', '0', '0', '0', '0', '0', '0', '0' ),  -- | L |
    ( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ),  -- | H |
    ( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' )   -- | - |
);
```

AND Function for std_logic_vectors



```
function "and"  ( l : std_ulogic; r : std_ulogic ) return UX01 is
begin
   return (and_table(l, r));
end "and";

function "and"  ( l,r : std_logic_vector ) return std_logic_vector is
   alias lv : std_logic_vector ( 1 to l'LENGTH ) is l;
   alias rv : std_logic_vector ( 1 to r'LENGTH ) is r;
   variable result : std_logic_vector ( 1 to l'LENGTH );
begin
   if ( l'LENGTH /= r'LENGTH ) then
      assert FALSE
      report "arguments of overloaded 'and' operator are not of the same length"
      severity FAILURE;
   else
      for i in result'RANGE loop
         result(i) := and_table (lv(i), rv(i));
      end loop;
   end if;
   return result;
end "and";
```