Assignment

- Homework #4: Problem 3.7 (a-c, e, f) 3.42, 4.13.
- Submit hard copy of your code and simulations for all three problems.
- For 3.7 f, just a couple of paragraphs.
- For 3.42, hardcopy can be of part c and part d. Include justification of part b.
- Due 10/30.

IEEE Logic System

The Problem

- The advent of VHDL solved the inter-organization communication problem as far as HDLs were concerned
- Different organizations were using different multivalued logic systems so models were still not portable
- Needed: a standard logic system that could be used to exchange behavioral models

IEEE Packages

- IEEE defined a 9 valued logic system
- The IEEE Developed two packages for this system:
  - STD_LOGIC_1164.VHD
    - Defines the basic value system and associated functions
    - Used as is by vendors
  - NUMERIC_STD.VHD
    - Provides overloaded arithmetic and other operators for synthesis
    - Vendors have developed their own versions of this

Package

STD_LOGIC_1164.VHD
(IEEE Standard)

type STD_ULOGIC is        -- Unresolved
{ '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
};
**Uninitialized Value (U)**

- Left most value of the type, thus the default initialization value.
- Used to model initialization of sequential circuits:
  - Initial state of the circuit is U
  - Does the logic change this state to a 0 or a 1? If not, the logic is not correctly designed.

**Z and -**

- Z represents high impedance, i.e., the output of a tri-state buffer that is turned off.
- - represents a don’t care
  - synthesis: represents a logic don’t care that can be used for logic minimization
  - simulation: acts like an X

**Strengths**

- Needed to model bus contention
- Logic values have strength
  - 0 strong zero; 1 strong one
  - L weak zero; H weak one
  - Models the effect of source impedance
- A strong 0 dominates a weak 1
- A strong 1 dominates a weak 0

**Unknown Values**

- X: value is 0 or 1
- W: value is L or H
- X and W are unknown values that arise from:
  - bus contention
  - error conditions, e.g., a flip flop has an unknown state
STD_LOGIC Definition

- type STD_ULOGIC_VECTOR is array (Natural range <>)
  of STD_ULOGIC;
- type STD_LOGIC_TABLE is array (STD_ULOGIC, STD_ULOGIC)
  of STD_ULOGIC;
- function RESOLVED (S : STD_ULOGIC_VECTOR)
  return STD_ULOGIC;
- subtype STD_LOGIC is RESOLVED STD_ULOGIC;
- type STD_LOGIC_VECTOR is array (Natural range <>)
  of STD_LOGIC;

Common Subtypes

- subtype X01 is RESOLVED STD_ULOGIC range 'X' to '1';
  -- ('X','0','1')
- subtype X01Z is RESOLVED STD_ULOGIC range 'X' to 'Z';
  -- ('X','0','1','Z')
- subtype UX01 is RESOLVED STD_ULOGIC range 'U' to '1';
  -- ('U','X','0','1')
- subtype UX01Z is RESOLVED STD_ULOGIC range 'U' to 'Z';
  -- ('U','X','0','1','Z')

Resolution Basics

Strong dominates weak, i.e.
0 R H 0 = 0
1 R L 1 = 1
resolving opposite values of the same strength yields unknowns of
0 R 1 = X
L R H = H R L = W

Common Subtypes
OVERLOADING the "and" FUNCTION

function "and" ( L : STD_ULOGIC; R : STD_ULOGIC ) return UX01 is
begin
return (AND_TABLE(L, R));
end "and";

Overloaded Operators
(For STD_ULOGIC, STD_LOGIC,
STD_ULOGIC_VECTOR and
STD_LOGIC_VECTOR)

and        not
nand       xor
or         xnor
nor

Type Conversion Functions

function To_Bit (S : STD_ULOGIC ; XMAP : Bit := '0' ) return bit;
function To_BitVector (S : STD_LOGIC_VECTOR ;
XMAP : Bit := '0' ) return bit_vector;

Conversion Function Example

function To_StdLogicVector (B: Bit_vector) return std_logic_vector is
alias BV: Bit_Vector (B'length-1 downto 0 ) is B;
variable RESULT: Std_Logic_Vector (B'length-1 downto 0 );
begins
for i in RESULT'range loop
case BV(i) is
when '0' => RESULT(i) := '0';
when '1' => RESULT(i) := '1';
end case;
end loop;
return RESULT;
end;

Converting The Other Way

function To_Bitvector(S: Std_Ulogic_Vector; XMAP: Bit := '0')
return Bit_Vector is
alias SV: Std_Ulogic_Vector (S'length-1 downto 0 ) is S;
variable RESULT: Bit_Vector (S'length-1 downto 0 );
begins
for i in RESULT'range loop
case SV(i) is
when '0' | 'L' => RESULT(i) := '0';
when '1' | 'H' => RESULT(i) := '1';
when others => RESULT(i) := XMAP;
end case;
end loop;
return RESULT;
end;
Edge Detection Functions

function Rising_Edge (signal S: Std_Ulogic)
return Boolean is
begin
return (S'event and (To_X01(S)) = '1') and
(To_X01(S'last_value) = '0'));
end;

function Falling_Edge (signal S: Std_Ulogic)
return boolean is
begin
return (S'event and (To_X01(S) = '0') and
(To_X01(S'last_value) = '1'));
end;

Package STD_LOGIC_ARITH
(Synopsys Package)

Type Definitions

type UNSIGNED is array (NATURAL range <>)
of STD_LOGIC;
type SIGNED is array (NATURAL range <>)
of STD_LOGIC;
subtype SMALL_INT is INTEGER range 0 to 1;

Overloaded + and -

• Operand Type: Signed, Unsigned, Integer,
and Std_Ulogic
• Returned Type: Signed, Unsigned, and
Std_Logic_Vector
• Must type caste Std.Logic_Vectors to
Signed or Unsigned to use them

Adder Model

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
entity SIMP_ADD is
port(A,B: in STD_LOGIC_VECTOR(3 downto 0);
CIN: in STD_LOGIC;
C: out STD_LOGIC_VECTOR(3 downto 0);
CAR_OUT: out STD_LOGIC);
end SIMP_ADD;

architecture ALG of SIMP_ADD is
signal  PADDED_CIN: STD_LOGIC_VECTOR(3 downto 0);
signal A_UNSIGNED, B_UNSIGNED: UNSIGNED(3 downto 0);
signal C_UNSIGNED: UNSIGNED(4 downto 0);
begin
A_UNSIGNED <= UNSIGNED(A);
B_UNSIGNED <= UNSIGNED(B);
PADDED_CIN <= "000"&CIN;
C_UNSIGNED <= CONV_UNSIGNED(A_UNSIGNED,5)
+ B_UNSIGNED+UNSIGNED(PADDED_CIN);
C  <= STD_LOGIC_VECTOR(C_UNSIGNED(3 downto 0));
CAR_OUT  <= C_UNSIGNED(4);
end ALG;
Other Operators

- Twos Complement (-X) of a number X
- Absolute value of signed numbers
- Multiplication:
  - operands: Signed and Unsigned, and Signed and Unsigned mixed
  - results: Signed, Unsigned and Std_Logic_Vector
- Relational Operators: < <= > >= = /=
  - operands: Signed, Unsigned, and Integer and mixtures of these
  - results: Boolean
- Shift left & shift right for Signed and Unsigned

Type Conversion & Extending

- Convert Integer, Signed, and Unsigned and Std_Ulogic to:
  - Integer
  - Signed
  - Unsigned
  - StdLogic_Vector
- Zero and sign extend StdLogic_Vector

Packages

STD_LOGIC_UNSIGNED
and
STD_LOGIC_SIGNED

Purpose

- Define various arithmetic functions for signals, variables, and ports defined as type StdLogic_Vector.
- The packages provide the following arithmetic functions:
  - StdLogic_Signed-Declares signed (2's-complement) arithmetic functions.
  - StdLogic_Unsigned-Declares unsigned arithmetic functions.

Advantages

- Using the StdLogic_Vector type and these packages for arithmetic eliminates the need for you to do manual type casting, reducing the potential for errors in your code
- In addition, consistent use of the StdLogic_Vector type improves the readability of your code

Arithmetic Operations

- Unary: negation and absolute value in signed package
- Binary: + and -
  - operands (mixed): StdLogic_Vector, Integer, and StdLogic
  - result: StdLogic_Vector
- Relational: = /= <= >= < >
  - operands (mixed): StdLogic_Vector & Integer
  - result: Boolean
Using std_logic_signed for 2's Complement Arithmetic

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_SIGNED.all;
entity ADD_TREE_SIGNED is
  port ( A, B, C: in  Std_Logic_Vector (7 downto 0);
        SUM   : out Std_Logic_Vector (7 downto 0));
end ADD_TREE_SIGNED;
architecture BEHAVIORAL of ADD_TREE_SIGNED is
begin
  SUM <= A + B + C; --This is now 2's-
       --complement addition
end BEHAVIORAL;

Features

• The packages do not perform the arithmetic functions themselves. Rather, the packages convert operands in an expression to a type usable by the Std_Logic_Arith package, then call that package to do the arithmetic.

STD_LOGIC_MISC.VHD Package

Features

• The package provides type conversion functions to convert resolved types to and from unresolved types (Drive and Sense functions).
• The Drive and Sense functions are overloaded, since they provide conversions for different types

Drive Functions

Drive (V: STD_ULOGIC_VECTOR); return STD_LOGIC_VECTOR
Drive (V: STD_LOGIC_VECTOR); return STD_ULOGIC_VECTOR

Sense Functions

Sense (V: STD_ULOGIC; vZ, vU, vDC: STD_ULOGIC) return STD_LOGIC;
Sense (V: STD_ULOGIC_VECTOR; vZ, vU, vDC: STD_ULOGIC_VECTOR) return STD_LOGIC_VECTOR;
Sense (V: STD_LOGIC_VECTOR; vZ, vU, vDC: STD_ULOGIC) return STD_ULOGIC_VECTOR;

Other Features

• Reduction operators:
  – for types std_logic_vector and std_ulogic_vector
  – operations: AND, NAND, OR, NOR, XOR, XNOR
Features

- Package contains:
  - read and write procedures to convert ASCII text data to and from std_logic and std_logic_vector types
  - procedures to read and write vector objects in octal and hexadecimal formats rather than binary format

Package
std_logic_textio.vhd

READ and WRITE Procedures For STD_ULOGIC

procedure READ (L: inout LINE; VALUE: out STD_ULOGIC);

procedure READ (L: inout LINE; VALUE: out STD_ULOGIC; GOOD: out BOOLEAN);

procedure WRITE (L: inout LINE; VALUE: in STD_ULOGIC; JUSTIFIED: in SIDE := RIGHT; FIELD: in WIDTH := 0);

oread, owrite, hread, and hwrite Procedures

procedure OREAD (L: inout LINE; VALUE: out STD_ULOGIC_VECTOR);

procedure OREAD (L: inout LINE; VALUE: out STD_ULOGIC_VECTOR; GOOD: out BOOLEAN);

procedure OWRITE (L: inout LINE; VALUE: in STD_ULOGIC_VECTOR; JUSTIFIED: in SIDE := RIGHT; FIELD: in WIDTH := 0);

procedure HREAD (L: inout LINE; VALUE: out STD_ULOGIC_VECTOR);

procedure HREAD (L: inout LINE; VALUE: out STD_ULOGIC_VECTOR; GOOD: out BOOLEAN);

procedure HWRITE (L: inout LINE; VALUE: in STD_ULOGIC_VECTOR; JUSTIFIED: in SIDE := RIGHT; FIELD: in WIDTH := 0);