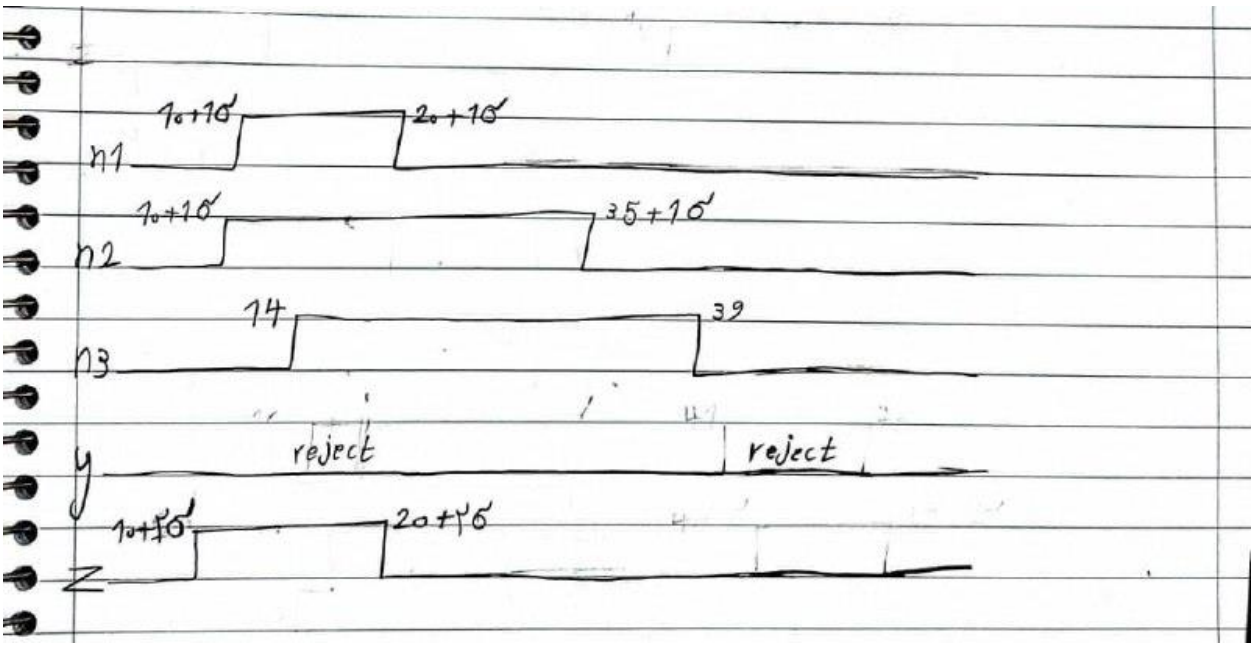
فربد فولادی-98243045

عرفان رفیعی اسکوئی-98243027

سوال اول)



سوال دوم)

a. Two of the most widely used types are `std_logic` and `std_ulogic`. The difference between them is that the former is resolved while the latter isn't.
Let's first have a look at the `std_ulogic` type, the unresolved version of the two. Below is an excerpt of the type declaration taken from an implementation of the std_logic_1164 package.

```
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
                     );
```

The `std_ulogic` is simply an enumerated type that lists the possible values as enumeration literals. When a signal or variable is declared with `std_ulogic` as the type, it can represent any one of these values and none other.
Let's talk about what does it mean that a type is *unresolved*?
If multiple drivers are driving different values onto a signal of the `std_ulogic` type, that's not going to work. Conflicting drivers is an error in VHDL. The simulation won't compile, or the design won't synthesize. Driver conflicts are not resolved with the `std_ulogic` type. The type doesn't have a built-in mechanism to determine what the signal value should be, so it's an error.
The `std_logic` can represent the same values as the `std_ulogic`, but it's a resolved type. What does that mean? To find out, let's once again refer to the implementation of the standard. Below is the declaration taken from the IEEE 1164 package.

```
subtype STD_LOGIC is resolved STD_ULOGIC;
```

The `subtype` keyword in VHDL is normally used for declaring a type with a limited range from the base type. But it can also be used for specifying a *resolution function*. That's what the above statement is saying. The `std_logic` is a subtype of `std_ulogic`, and the name of the resolution function is "resolved".
Now, let's examine the "resolved" function:

```
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 function resolved;
```

It accepts one parameter, an array of `std_ulogic` representing all the simultaneous drivers. It then compares all the elements in the vector to each other, one by one, reducing them to a single `std_ulogic` value which is returned.

For comparing the values, what seems to be a different function named “resolution_table” is called. But it’s actually a two-dimensional array:

```
type stdlogic_table is array(STD_ULOGIC, STD_ULOGIC) of STD_ULOGIC;

  ----------------------------------------------------------------------
  -- resolution function
  ----------------------------------------------------------------------
  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')   -- | - |
             );
```

The resolution function governs the final value that a signal will have in case of multiple drivers.

b.

```
--============================================================================
  -- Conversion Functions
  --============================================================================

  -- Id: D.1
  function TO_INTEGER (ARG : BIT_VECTOR) return NATURAL;
    attribute builtin_subprogram of
        TO_INTEGER[BIT_VECTOR return NATURAL]: function is
"numstd_conv_integer_un";
  -- Result subtype: NATURAL. Value cannot be negative since parameter is an
  --             UNSIGNED vector.
  -- Result: Converts the UNSIGNED vector to an INTEGER.

  -- Id: D.3
  function To_BitVector (ARG, SIZE : NATURAL) return BIT_VECTOR;
    attribute builtin_subprogram of
        To_BitVector [NATURAL, NATURAL return BIT_VECTOR]: function is
"numbit_conv_unsigned_nu";
  -- Result subtype: bit_vector(SIZE-1 downto 0)
  -- Result: Converts a non-negative INTEGER to an UNSIGNED vector with
  --         the specified size.

  function To_BitVector (ARG : NATURAL; SIZE_RES : BIT_VECTOR)
    return BIT_VECTOR;
    attribute builtin_subprogram of
        To_BitVector [NATURAL, BIT_VECTOR return BIT_VECTOR]: function is
"numbit_conv_unsigned_nuu";
  -- Result subtype: STD_LOGIC_VECTOR(SIZE_RES'length-1 downto 0)

-- begin LCS-2006-130
  alias To_Bit_Vector is
    To_BitVector[NATURAL, NATURAL return BIT_VECTOR];
  alias To_BV is
    To_BitVector[NATURAL, NATURAL return BIT_VECTOR];

  alias To_Bit_Vector is
    To_BitVector[NATURAL, BIT_VECTOR return BIT_VECTOR];
  alias To_BV is
    To_BitVector[NATURAL, BIT_VECTOR return BIT_VECTOR];

end package NUMERIC_BIT_UNSIGNED;
FUNCTION To_bitvector ( s : std_logic_vector93 ; xmap : BIT := '0') RETURN
BIT_VECTOR;
```

```
    FUNCTION To_bitvector ( s : std_ulogic_vector93; xmap : BIT := '0') RETURN
BIT_VECTOR;

    FUNCTION To_StdLogicVector  ( b : BIT_VECTOR        ) RETURN
std_logic_vector93;
    FUNCTION To_StdLogicVector  ( s : std_ulogic_vector93 ) RETURN
std_logic_vector93;
    FUNCTION To_StdULogicVector ( b : BIT_VECTOR        ) RETURN
std_ulogic_vector93;
    FUNCTION To_StdULogicVector ( s : std_logic_vector93  ) RETURN
std_ulogic_vector93;
```

C.

```
--------------------------------------------------------------------
  -- resolution function
  ------------------------------------------------------------------
  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')   -- | - |
             );
```

d.

```
--------------------------------------------------------------------
  -- resolution function
  ------------------------------------------------------------------
  function resolved (s : STD_ULOGIC_VECTOR) return STD_ULOGIC;


  ------------------------------------------------------------------
  -- logic state system  (resolved)
  ------------------------------------------------------------------
  subtype STD_LOGIC is resolved STD_ULOGIC;

  ------------------------------------------------------------------
  -- unconstrained array of resolved std_ulogic for use in declaring
  -- signal arrays of resolved elements
  ------------------------------------------------------------------
  subtype STD_LOGIC_VECTOR is (resolved) STD_ULOGIC_VECTOR;

  ------------------------------------------------------------------
  -- 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')
```

سوال سوم)

a. TYPE NIBBLE IS ARRAY (0 to 1023) OF STD_LOGIC;
   TYPE MEM IS ARRAY (0 to 1023) OF NIBBLE;

b. TYPE matrix IS ARRAY (0 to 1023 , 0 to 1023) OF STD_LOGIC;

c. TYPE student IS RECORD
        studentID: INTEGER RANGE 1 to 100;
        fieldOfStudy: fieldOfStudy_Name;
        termNumber: INTEGER RANGE 1 to 10;
   END RECORD;

d. TYPE floor_state_type IS (floor1, floor2, floor3);
   signal floor_state : floor_state_type;