Filename="ch2.doc"

2.0 VHDL Data Types

2.1 Signal Attributes

- Specific values associated with signals.
- Format:
  signal_name' attribute_designator

Example:
clock'ACTIVE

2.1.1 Signal Attributes Which Define Another Signals

1. S'DELAYED(T) is a signal which echoes the value of the prefix signal, delayed by the specified time factor. If T=0, the value is equal to S after a delta delay (i.e. in the next simulation cycle).

2. S'QUIET(T) is a boolean signal whose value is TRUE if S has not had a transaction (i.e. not active) for the length of time T. If T=0,FALSE during simulation cycle in which S was assigned to and then will return to TRUE.

3. S'STABLE(T) is a boolean signal whose value is TRUE if S has not had an event (i.e. not changed value) for the length of time T. If T=0, the value will be FALSE during the simulation cycle in which S changed and then will return to TRUE.

4. S'TRANSACTION is a bit signal whose value toggles each time a transaction occurs on S (i.e. S is active).

2.1.2 Signal Attributes Which Provide Information About Signals

1. S'EVENT is TRUE if an event has occured on S during the current simulation cycle (i.e. if S has changed value during the cycle).

2. S'ACTIVE is TRUE if a transaction has occured on S during the current simulation cycle.

3. S'LAST_EVENT returns the amount of time which has elapsed since the last event on S (i.e. since S last changed value).

4. S'LAST_ACTIVE returns the amount of time which has elapsed since the last transaction on S (i.e. since S was last active).

5. S'LAST_VALUE returns the value of S before the last event on S.

Signal Attribute	Supported Attribute
'active	No
'delayed[(t)]	No
'event	Yes
'last_active	No
'last_event	No
'last_value	Yes
'quiet[(t)]	No
'stable	Yes
'transaction	Yes

2.1.3 Signal Attribute Relationships

An activity is any change on the signal value. A change from `1' to `X' is an example of an activity, and a change from `1' to `1' is an activity. The only criteria is that something happened. However an event requires a change in value of the signal. A change from `1' to `X' is an event, but a change from `1' to `1' is not an event. All events represent activities, but not all activities represent events.

IF S'STABLE is given without a time expression, the expression defaults to 0 ns which means that the check is for stability of the signal at this exact instance in time. This is equivalent to S'EVENT.

S'EVENT is more efficient than S'STABLE. Simulator will take more work to evaluate S'STABLE.

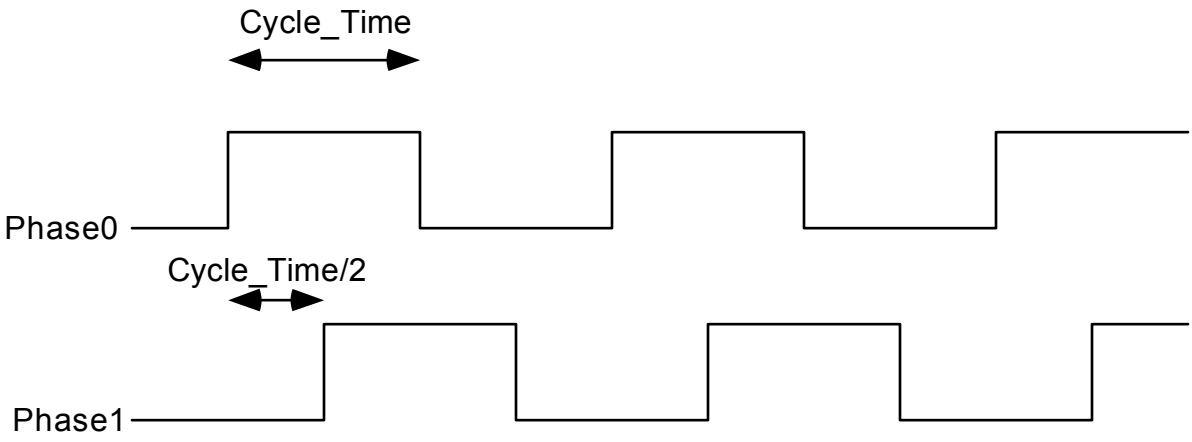
2.1.3 Examples

Example 1:

```
SIGNAL S: INTEGER;
S <= 1 AFTER 1 ns, 2 AFTER 2 ns;
The value of S'DELAYED(10ns) will be 1 after 11 ns and 2 after 12 ns.
```

Example 2: Phase-shifted Clock generation

```
ENTITY clock IS
    GENERIC(Cycle_Time: TIME:=25 ns);
    PORT(Phase0,Phase1:OUT BIT);
END clock;

ARCHITECTURE behavior OF clock IS
    SIGNAL ControlSignal:BIT:='0';
BEGIN
    ControlSignal <= NOT ControlSignal AFTER Cycle_Time;
    Phase0 <= ControlSignal;
    Phase1 <= ControlSignal'DELAYED(Cycle_Time/2);          --(Not supported by Autologic II)
END behavior;
```



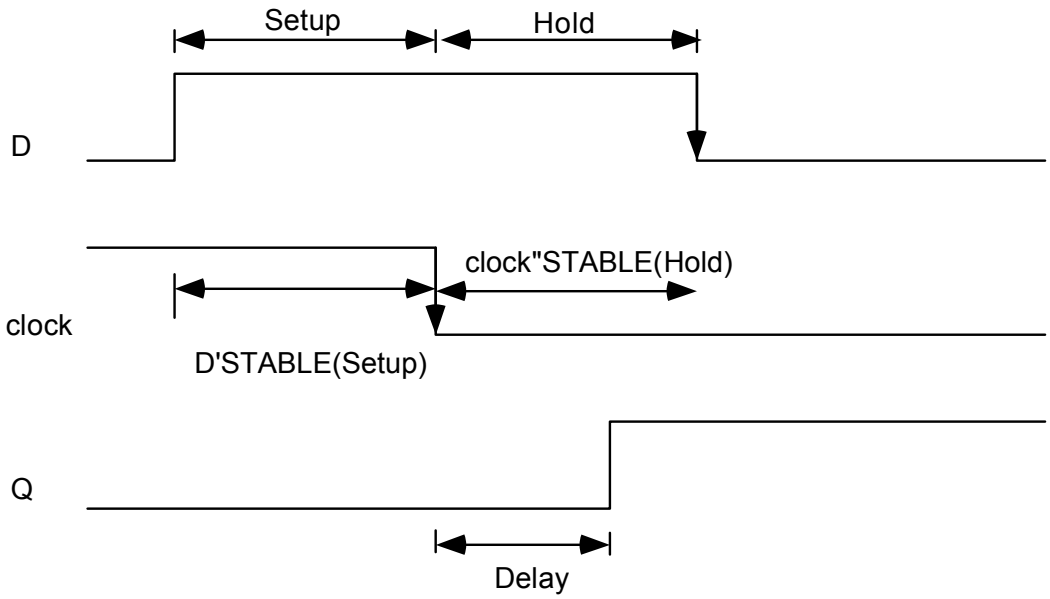
Example 3: Detecting Rising Clock Edge:

```
    clock'EVENT AND clock='1';
(or) NOT clock'STABLE AND clock='1';
(or) NOT clock'QUIET  AND clock='1';              (Not supported by Autologic II)
(or) clock'LAST_VALUE='0' AND clock='1';          (Preferred Method)
```

Example 4: Detecting Falling Clock Edge:

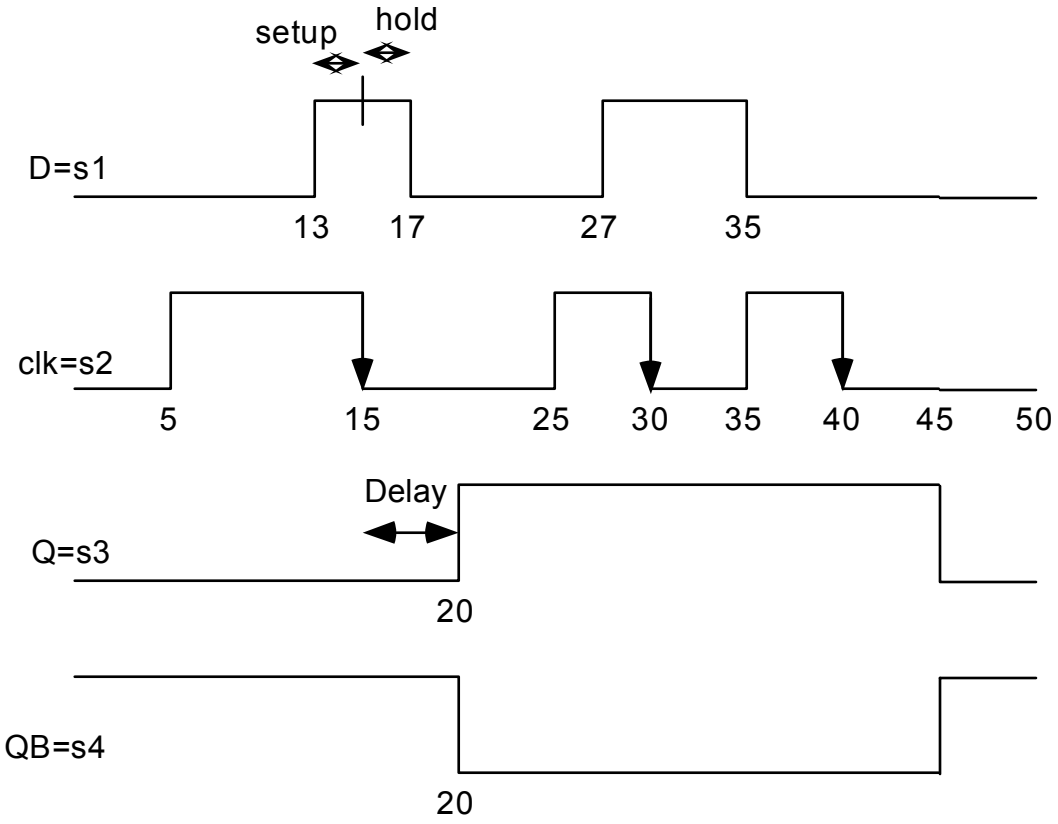
```
    clock'EVENT AND clock='0';
(or) NOT clock'STABLE AND clock='0';
(or) clock'LAST_VALUE='1' AND clock='0';          (Preferred Method )
```

Example 5: Checking Setup and Hold Time of D Flip Flop:




```
ENTITY DFF IS
      GENERIC(Setup,Hold,Delay:TIME:=0 ns);
      PORT(D,clock:IN BIT:='0';Q:OUT BIT:='0';QB:OUT BIT:='1');
BEGIN -- passive process (no signal assignment) only for checking
     -- generic constraints.
Check_Setup_and_Hold_Times:
      PROCESS(D,clock)
      BEGIN
                -- Check for setup time
                IF NOT clock'STABLE AND clock='0' THEN
                ASSERT D'STABLE(Setup)
                          REPORT "D changed within setup interval"
                                    SEVERITY Warning;
                END IF;
                -- Check for hold time
                IF NOT D'STABLE AND clock='0' THEN
                ASSERT  clock'STABLE(Hold)
                          REPORT "D changed within hold interval"
                                    SEVERITY Warning;
```

```
            END IF;
            --Check for Delay Time
            ASSERT (Delay>=Hold)
                  REPORT "Delay>=Hold Violation"
                        SEVERITY Warning;
    END PROCESS;
END DFF;

ARCHITECTURE  one OF DFF IS
    SIGNAL value: BIT;
BEGIN
    PROCESS(D,clock)
    BEGIN
            IF((NOT clock'LAST_VALUE) AND (clock = `0')) THEN
            value <= D;
            END IF;
    END PROCESS;
    Q <= TRANSPORT value AFTER Delay;
    QB <= TRANSPORT NOT value AFTER Delay;
END one;

-- Setup Test Bench
ENTITY tb IS  -- no IO
END tb;

ARCHITECTURE setup_one OF tb IS
    COMPONENT FF
            GENERIC(Setup,Hold,Delay:TIME);
            PORT(D,clock: IN BIT;Q,QB:OUT BIT);
    END COMPONENT;
    FOR u1:FF USE ENTITY WORK.DFF(one);
    SIGNAL s1,s2,s3,s4:BIT;
BEGIN
FF_1: FF
    GENERIC MAP(2 ns, 1 ns, 5 ns)
    PORT MAP(s1,s2,s3,s4);

   s1 <= `1' AFTER 13 ns;
       `0' AFTER 17 ns;
       `1' AFTER 27 ns;
       `0' AFTER 35 ns;

   s2 <= `1' AFTER 5 ns;
       `0' AFTER 15 ns;
       `1' AFTER 25 ns;
       `0' AFTER 30 ns;
       `1' AFTER 35 ns;
       `0' AFTER 40 ns;
END setup_one;
```



2.2 Literal

- Integer literals may be expressed in any base from 2 to 16.
- Any two adjacent digits may be separated by a single underscore (_).

Examples:

```
12343        --base 10 integer literal
2#10011110#  --base 2 (binary) integer literal
8#720#       --base 8 (octal) integer literal
16#FFFF0ABC# --base 16 (hex) integer literal
16#FFFF_0ABC# --using (_) for readability
```

- Floating point literals differs from an integer literal in that it contains a dot(.).

Examples:

```
65971.333333
65_971.333_333
8#43.6#
```

- A character literal consists of a single character and can be any character which can be entered. Identifiers are names and must follow the VHDL rules for creating names. To distinguish the difference, character literals must be entered in single quote marks. That is, a is an identifier, while `a' is a character.
- Character literals are case sensitive. When they appear within the model,they must appear in single quotes and they must be the same case.

- **Identifiers** are not case sensitive and can include letters, numbers and the underscore character (_). It must start with a letter and it cannot end with an underscore.

2.3 Typing

- Object's TYPE determine values it may assume and operations which may be performed on it.
- Aids in design verification:
  Data Paths
  .Object Values
- Type Classifications:
  Scalar
  Composite
  File
  Access (Pointer)

The type of an object indentifies the values the object may assume. When a value is assigned to a signal, the value is checked to be sure that it is within the allowable set of values. If it is not, an error message is issued. This is particularly useful for values assigned from a arithmetic computation.

2.3.1 Scalar Types

Scalar types (no structure) include all numeric, enumeration, and physical object types. Types which are made up of real numbers, integer, quantities with associated physical units such as times, and object which are made up of character literals or identifiers are all scalar types.

2.3.1.1 Integer Types

- Integers are the unbounded set of positive and negative whole numbers.
- 32 bit limitation restricts range.
- Upper and lower range constraints must be integer range.
- Declaration format:
  TYPE type_name IS RANGE int_range_constraint;
- Predefined integer type:
  TYPE integer IS RANGE –2147483648=[$-2^{(31-1)}$] TO 2147483647 = [$2^{(31-1)}$ -1];

RANGE

- Identifies subset of values.
- May be used with type declarations or object declarations
- Format:
  RANGE begin direction end
- Direction may be:
  Ascending - TO
  Descending - DOWNTO

Examples:

```
TYPE day IS RANGE 1 TO 31;
TYPE voltage IS RANGE 12 DOWNTO -12;
SIGNAL in_volts:voltage RANGE 5 DOWNTO 0; -- object declaration
   with range a subset of the full range of voltage.
```

- When the range clause does not appear in an object declaration, the object assumes the full range of the type which appears in them declaration.

Example :

SIGNAL output : voltage means that output ranges from 12 to -12.

2.3.1.2 Floating Point Types

- Floating Points are the unbounded set of positive and negative numbers which contain a decimal point.
- 32 bit limitation restricts range.
- Upper and lower range constraints must contain a decimal point.
- Declaration format:
  TYPE type_name IS RANGE range_constraint;
- Predefined floating point type:
  TYPE real IS RANGE -1.79769E308 TO 1.79769E308;

2.3.1.3 Enumeration Types

- Lists of identifiers or character literals.
- Identifiers follow standard naming rules
- Character literals are all upper and lower case alpha characters numbers, and special characters.
- Declaration Format:
  TYPE type_name IS (enumeration_ident_list);
- Predefined enumeration types:
  TYPE bit IS (`0','1');
  TYPE boolean IS (false,true);
  TYPE severity_level IS (note,warning,error,failure);
  TYPE character IS (`a','b','c',...);

Examples:

TYPE Two_level_logic IS (`0','1');
TYPE Three_level_logic IS (`0','1','Z');
TYPE Four_level_logic IS (`X','0','1','Z');
TYPE Opcode IS (Add,Add_with_carry,Sub,Sub_with_carry,Complement);
TYPE qsim_state IS (`0','1','X','Z');

2.3.1.4 Physical Types

- Describes objects in terms of a base unit, multiples of base unit, and a specified range.
- Declaration format:
  TYPE type_name IS RANGE range_constraints
  UNITS
  base_unit;
  [-- multiples;]
  END UNITS;
- Predefined physical type:
  TYPE time IS RANGE -2**(31-1) TO 2**(31-1)
  UNITS
  fs; --femtosecond =10^{-15} sec
  ps = 1000 fs; --picosecond =10^{-12} sec
  ns = 1000 ps; --nanosecond =10^{-9} sec
  us = 1000 ns; --microsecond =10^{-6} sec
  ms = 1000 us; --millisecond =10^{-3} sec
  sec =1000 ms; --second

```
  min =60 sec;  --minute
  hr  =60 min;  --hour
END UNITS;
```

Example:

```
TYPE Resistance IS RANGE 1 TO 10E9
  UNITS
    ohm;  --the base unit.
    kohm=1000 ohm; --secondary unit, multiple of base unit.
  END UNITS;
```

2.3.1.5 Scalar Subtypes

- Subsets of specified types
- Do not define new types
- Range constraints must be within defining type's range.
- Declaration format:
  SUBTYPE name IS type_name RANGE constraints;
- Predefined scalar subtypes:
  SUBTYPE natural IS integer RANGE 0 TO 2147483647;
  SUBTYPE positive IS integer RANGE 1 TO 2147483647;
- A subtype declares a contiguous subset of values of a specified type.

2.3.2 Composite Types

There are two kinds of composite types: arrays and records.

2.3.2.1 Array Types

- Multiple values of same type under single identifier.
- One or more dimensions. (Autologic only support 2D).
- Values referenced by indices.
- Indice's type must be integer or enumeration.
- Declaration format:
  TYPE array_type_name IS ARRAY (range_constraints) OF type;
- Predefined array types:
  TYPE string IS ARRAY (positive RANGE <>) OF character;
  TYPE bit_vector IS ARRAY (natural RANGE <>) OF bit;

Example:

```
TYPE Column IS RANGE 1 TO 80;
TYPE Row IS RANGE 1 TO 24;
TYPE Matrix IS ARRAY (Row,Column) OF boolean;
```

Array Range:
Constrained or unconstrained.

- Boundaries of constrained array are stated:
  TYPE array_1 IS ARRAY (integer RANGE -10 TO 25) OF bit;
  TYPE array_1_too IS ARRAY (-10 TO 25) OF bit;
  (NOTE: integer is optional)
- Boundaries of unconstrained array are left open:

```
TYPE array_2 IS ARRAY (integer RANGE <>) OF bit;
```

- Boundaries can be enumerated types:

```
TYPE pet IS (dog,cat,bird,horse,kid);
TYPE pet_it IS ARRAY (pet RANGE dog TO cat) OF bit;
TYPE pet_too IS ARRAY (pet RANGE <>) OF bit;
```

Array Subtypes:

Subsets of specified array types.
Do not define a new array type.
TYPE that SUBTYPE is based on must be an unconstrained array.
Declaration format:

```
SUBTYPE name IS (array_name RANGE range_constraint);
```

Example:

```
TYPE data IS ARRAY (natural RANGE <>) OF bit;
SUBTYPE low_range IS (data RANGE 0 TO 7);
SUBTYPE high_range IS (data RANGE 8 TO 15);
```

There are several advantages of subtypes. The primary advantage is to clarify what is being done in the model. They make it easier to visualize what is being stored and why by breaking large groupings of values into smaller groupings. Each "smaller grouping" can have a name which more descriptively tells what values it represents.

Array Slices:

- Consecutive positions of one-dimensional arrays.
- Created by by appending a parenthesized discrete range to the name of the one-dimensional array.

Example:

```
TYPE Byte IS ARRAY (7 DOWNTO 0) OF bit;
TYPE Memory IS ARRAY (0 TO 2**16-1) OF Byte;

SIGNAL S_byte: Byte;
SIGNAL S_memory:Memory;

S_byte(0)  -- refers to element 0
S_byte(3 DOWNTO 1) -- slice of three elements
S_memory(2**15-1 TO 2**16-1) -- slice of 2**15 elements
S_byte - refers to the entire array.
```

Example of using slice names:

```
PROCESS
   TYPE ref_array IS ARRAY(positive RANGE<>)OF Integer;
   VARIABLE  array_a:ref_array(1 TO 12);
   VARIABLE  array_b:ref_array(1 TO 4);
BEGIN
  FOR i IN 1 TO 12 LOOP
     array_a(i):=i+10;
  END LOOP;
  array_b:=array_a(6 TO 9)
END PROCESS;
```

Array Example:

```
TYPE bit_nibble IS ARRAY(3 DOWNTO 0) OF BIT;
TYPE bit_byte IS ARRAY(7 DOWNTO 0) OF BIT;
SIGNAL sq4: bit_nibble;
SIGNAL sq8: bit_byte;

sq4 <= sq8(2)&sq8(3)&sq8(4)&sq(5); -- reversing sq8 into sq4
sq8 <= sq(0)&sq8(7 DOWNTO 1) -- rotate right sq8 by 1
sq8 <= sq8(6 DOWNTO 0)&sq8(7)  -- rotate left sq8 by 1
```

Array Initialization:

1. Initial values for a one-dimensional array type signal must be placed in a set of parenthesis and should follow the := symbol in the signal declarations. The initial values of individual array elementsshould be separatedby commas.

```
SIGNAL sq4: bit_nibble :=(`1','0','1','1');
```

2. Nested sets of parentheses as should be used for multi-dimensional arrays. In this case, the top level set of parentheses corresponds to the left-most range of the array.

```
TYPE bit_4by8 IS ARRAY(3 DOWNTO 0, 0 TO 7) OF BIT;
SIGNAL sq_4_8: bit_4by8 :=
        (
                (`0','0','0','0','1','1','1','1'),
                (`0','0','0','1','1','1','1','1')
                (`0','0','1','1','1','1','1','1')
                (`0','1','1','1','1','1','1','1')
        );
```

Computer Memory Example:

```
TYPE memory IS ARRAY(0 TO 11) OF std_logic_vector(0 TO 7);
SIGNAL M: memory :=
        ("00000001", "00000010",
         "11001100", "00000011",
         "00001001","00000100",
         "00001010", "00000101",
         "00001011", "11101110",
         "00000000", "00000010");
```

2.3.2.2 Predefined Array Attributes

- Return information about index values.
- Formats:
  array_name'ATTRIBUTE
- ATTRIBUTES : LEFT,RIGHT,HIGH,LOW,RANGE,REVERSE_RANGE,LENGTH

Example:

```
SIGNAL ray:BIT_VECTOR(0 TO 7);

ray'LEFT        - returns 0
```

```
ray'RIGHT        - returns 7
ray'HIGH         - returns 7
ray'LOW          - returns 0
ray'RANGE        - returns 0 TO 7
ray'REVERSE_RANGE - returns 7 DOWNTO 0
ray'LENGTH       - returns 8
```

- `RIGHT and `HIGH return the same value when the range is declared as "value TO value". In this case the highest index value appears on the right side ofrange declaration. However, if the range is declared as "value DOWNTO value", the highest appears on the left side of the declaration. In this case `RIGHT and `HIGH would return different values. The same explanation applies to `LEFT and `LOW.

Example:

```
TYPE array2 IS ARRAY(Integer RANGE <>,Integer RANGE <>) OF Integer;
VARIABLE matrix:Array2(1 TO 3, 9 DOWNTO 6)
  matrix'high(1) -- returns 3
  matrix'high(2) -- returns 9
  matrix'left(1)  -- returns 1
  matrix'left(2)  -- returns 9
```

Array Attribute	Supported by Autologic II
‘high[(n)]	Yes
‘left[(n)]	Yes
‘length[(n)]	Yes
‘low[(n)]	Yes
‘range[(n)]	Yes
‘reverse_range[(n)]	Yes
‘right[(n)]	Yes

2.3.2.3Record Types

- Records are heterogeneous composite types; that is, the elements of a record can be of various types.
- A record type definition specifies one or more elements, each element having a different name and possibly a different type.
- Declaration format:

```
RECORD
  element_declaration
  {element_declaration}
END RECORD;
```

Example:

```
TYPE Opcode IS (Add,Add_with_carry,Sub,Sub_with_carry,Complement);
TYPE Address IS RANGE 16#0000# TO 16#FFFF#;
TYPE Instruction IS
RECORD
  Op_field  :Opcode;
  Operand_1 :Address;
  Operand_2 :Address;
END RECORD;
```

2.3.2.4 Referencing Elements of Composites

- An object of a composite type may be referenced in its entirety or by element.
- A simple name of an array or record is a reference to the entire array or record.
- An Indexed name is used to reference an element of an array. An indexed name consists of the name of the object, followed by a parenthesized list of index expressions, one index expression for each dimension of the array.

Example:

```
TYPE Column IS RANGE 1 TO 80;
TYPE Row IS RANGE 1 TO 24;
TYPE Matrix IS ARRAY (Row,Column) OF boolean;
SIGNAL S: Matrix;

S(1,1)  --reference element (1,1)
S(3,14) --reference element (3,14)
```

- A selected name is a reference to an element of a record. A selected name consists of the name of the object, followed by a dot (.), followed by the field name of the record.

Example:

```
TYPE Fraction IS
RECORD
  Numerator: Integer;
  Denominator: Integer;
END RECORD;

SIGNAL S: Fraction;

S.Numerator   --reference numerator field of S.
S.Denominator --reference denominator field of S.
```