PART 9

Standards

- MVL9, logic value system
- Logic type, Operators, Conversions
- VHDL'93
- Operators, Delay model
- Instantiation, Binding
- Attributes, Signal assignments
- Report, Postponed process
- Miscellaneous

Standards.MVL9.types

```
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
               );
```

```
TYPE std_ulogic_vector IS
     ARRAY ( NATURAL RANGE <> ) 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;
```

- Unresolved 9 value system
- The *resolved* function resolves *std_ulogic* values
- The *std_logic* and *std_logic_vector* are the industry standard

Standards.MVL9.subsets

```
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')
```

- Subtypes provide mapping to common value systems
- X01, X01Z, UX01, UX01Z
- The *std_logic* is a superset of most common value systems

Standards.MVL9.overloading

```
FUNCTION "and"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "nand" ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "or"   ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "nor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "xor"  ( l : std_ulogic; r : std_ulogic ) RETURN UX01;
FUNCTION "not"  ( l : std_ulogic                 ) RETURN UX01;
```

```
FUNCTION "and"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "and"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "nand" ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "nand" ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "or"   ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "or"   ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "nor"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "nor"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;

FUNCTION "xor"  ( l, r : std_logic_vector  ) RETURN std_logic_vector;
FUNCTION "xor"  ( l, r : std_ulogic_vector ) RETURN std_ulogic_vector;
```

- Overloading logical operators
- Vectorized operators are overloaded
- Vectorized operators are for resolved and unresolved types

Standards.MVL9.conversions

```
FUNCTION To_bit       ( s : std_ulogic;        xmap : BIT := '0') RETURN BIT;
FUNCTION To_bitvector( s: std_logic_vector; xmap: BIT := '0') RETURN    BIT_VECTOR;
FUNCTION To_bitvector( s: std_ulogic_vector; xmap: BIT := '0') RETURN  BIT_VECTOR;

FUNCTION To_StdULogic       ( b : BIT               ) RETURN std_ulogic;
FUNCTION To_StdLogicVector  ( b : BIT_VECTOR        ) RETURN std_logic_vector;
FUNCTION To_StdLogicVector  ( s : std_ulogic_vector ) RETURN std_logic_vector;
FUNCTION To_StdULogicVector ( b : BIT_VECTOR        ) RETURN std_ulogic_vector;
FUNCTION To_StdULogicVector ( s : std_logic_vector  ) RETURN std_ulogic_vector;

FUNCTION To_X01  ( s : std_logic_vector  ) RETURN  std_logic_vector;
FUNCTION To_X01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
FUNCTION To_X01  ( s : std_ulogic        ) RETURN  X01;
FUNCTION To_X01  ( b : BIT_VECTOR        ) RETURN  std_logic_vector;
FUNCTION To_X01  ( b : BIT_VECTOR        ) RETURN  std_ulogic_vector;
FUNCTION To_X01  ( b : BIT               ) RETURN  X01;

FUNCTION To_X01Z ( s : std_logic_vector  ) RETURN  std_logic_vector;
FUNCTION To_X01Z ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
FUNCTION To_X01Z ( s : std_ulogic        ) RETURN  X01Z;
FUNCTION To_X01Z ( b : BIT_VECTOR        ) RETURN  std_logic_vector;
FUNCTION To_X01Z ( b : BIT_VECTOR        ) RETURN  std_ulogic_vector;
FUNCTION To_X01Z ( b : BIT               ) RETURN  X01Z;

FUNCTION To_UX01  ( s : std_logic_vector  ) RETURN  std_logic_vector;
FUNCTION To_UX01  ( s : std_ulogic_vector ) RETURN  std_ulogic_vector;
FUNCTION To_UX01  ( s : std_ulogic        ) RETURN  UX01;
FUNCTION To_UX01  ( b : BIT_VECTOR        ) RETURN  std_logic_vector;
FUNCTION To_UX01  ( b : BIT_VECTOR        ) RETURN  std_ulogic_vector;
FUNCTION To_UX01  ( b : BIT               ) RETURN  UX01;
```

- Conversion functions for common logic value systems
- Conversion to systems without strength
- Vectorized conversion functions are included

Standards.MVL9.miscellaneous

```
-- Edge detection
 FUNCTION rising_edge  (SIGNAL s : std_ulogic) RETURN BOOLEAN;
 FUNCTION falling_edge (SIGNAL s : std_ulogic) RETURN BOOLEAN;
```

```
-- Checking for unknown
FUNCTION Is_X ( s : std_ulogic_vector ) RETURN  BOOLEAN;
FUNCTION Is_X ( s : std_logic_vector  ) RETURN  BOOLEAN;
FUNCTION Is_X ( s : std_ulogic        ) RETURN  BOOLEAN;
```

- Functions for edge detection
- Functions for unknown check
- Checking for 'X' in scalar, vectorized, and resolved

Standards.VHDL'93.shift_and_rotate

SLL:	Shift	Left	Logical	Use fill_value
SRL:	Shift	Right	Logical	Use fill_value
SLA:	Shift	Left	Arithmetic	
SRL:	Shift	Right	Arithmetic	
ROL:	Rotate	Left		
ROL:	Rotate	Right		

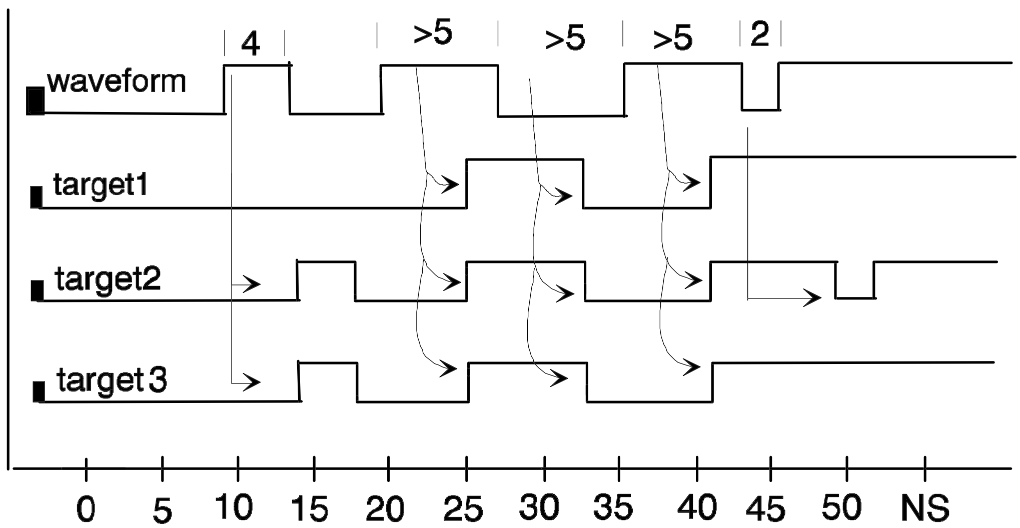
```
a_vector  SLL  3
```

- Use 'LEFT of element of ARRAY for fill_value
- SHIFT Logical:
  Shift right (left) use fill_value on left (right)
  Shift right (left) ignore right (left) most value
- SHIFT Arithmetic:
  Shift right (left) use left (right) most on left (right)
  Shift right (left) ignore right (left) most value
- ROTATE
  Rotate right (left) use right (left) most on left (right)

Standards.VHDL'93.inertial_reject

NEW DELAY MODELING

```
ARCHITECTURE delay OF example IS
    SIGNAL target1, target2, target3, waveform : BIT;
BEGIN
    -- the following illustrates inertial delay
    target1 <= waveform AFTER 5 NS;
    -- the following illustrates transport delay
    target2 <= TRANSPORT waveform AFTER 5 NS;
    -- the following illustrates reject
    target3 <= REJECT 3 NS INERTIAL waveform AFTER 5 NS;
END delay;
```



- TRANSPORT t_t does not reject any pulse
- INERTIAL t_i rejects all pulses less than t_i time expression
- REJECT t_r INERTIAL t_i rejects all pulses less than t_r

Standards.VHDL'93.direct_instantiation

Component Binding in VHDL'87:

Component Declaration + ***Configuration Specification***
or
Component Declaration + ***Component Configuration***

Configuration may be eliminated for default

VHDL'93 allows Component instantiation without need for:

Component Declaration and ***Binding Specification***

Component Instantiation will be used as default

```
ARCHITECTURE . . .
. . .
BEGIN
  . . .
      c1: ENTITY library_name.and2 (single_delay)
            GENERIC MAP ( . . .)    PORT MAP ( . . .);
. . .
END . . .
```

- Direct instantiation of a component matches its interface
- No need for Declaration and/or Binding of Component

Standards.VHDL'93.incremental_binding

```
ARCHITECTURE flexible OF nibble_comparator IS
  COMPONENT comp1
    PORT (a, b, gt, eq, lt : IN BIT; a_gt_b, a_eq_b, a_lt_b : OUT BIT);
  END COMPONENT;
  FOR ALL:comp1 USE ENTITY WORK.bit_comparator_t  passed_delay);
  SIGNAL im : BIT_VECTOR ( 0 TO 8 );
BEGIN
  c0: comp1 PORT MAP ( . . . );
  c1to2 :  FOR i IN 1 TO 2 GENERATE   c: comp1 PORT MAP ( . . . );
  END GENERATE;
  c3: comp1 PORT MAP ( . . . );
END flexible;

USE WORK.ALL;
CONFIGURATION passed_bit_level OF nibble_comparator_test_bench IS
  FOR customizable
    FOR a1 : comp4  USE ENTITY WORK.nibble_comparator(flexible);
      FOR flexible
        FOR c0, c3: comp1
          GENERIC MAP (tplh1 => 2 NS, tplh2 => 3 NS,
              tplh3 => 4 NS, tphl1 => 4 NS,  tphl2 => 5 NS, tphl3 => 6 NS);
         END FOR;
        FOR c1to2
          FOR c: comp1
            GENERIC MAP (tplh1 => 2 NS, tplh2 => 3 NS,
                tplh3 => 4 NS, tphl1 => 4 NS,  tphl2 => 5 NS, tphl3 => 6 NS);
          END FOR;
        END FOR;
      END FOR;
    END FOR;
  END FOR;
END passed_bit_level;
```

- Can split binding between:
  Configuration Specification and ***Component Configuration***
- If Configuration Specification is used, entity aspect is needed

Standards.VHDL'93.attributes_1

'ASCENDING 'IMAGE 'VALUE

```
TYPE state IS (reset, got1, got10, got101);
TYPE state_vector IS ARRAY (NATURAL RANGE <>) OF state;
--
FUNCTION one_of (sources : state_vector) RETURN state IS
  USE STD.TEXTIO.ALL;
  VARIABLE l : LINE;
  FILE flush : TEXT OPEN WRITE_MODE IS "/dev/tty";
BEGIN
  FOR i IN sources'RANGE LOOP
--   CASE sources(i) IS   -- state_string is a STRING VARIABLE
--     WHEN reset  => state_string := "reset  ";
--     WHEN got1   => state_string := "got1   ";
--     WHEN got10  => state_string := "got10  ";
--     WHEN got101 => state_string := "got101 ";
--   END CASE;
    WRITE (l, state'IMAGE (sources(i)), LEFT, 7);
  END LOOP;
  WRITELINE (flush, l);
  RETURN sources (sources'LEFT);
END one_of;
```

- 'ASCENDING returns TRUE if array is ascending
- 'IMAGE generates string of any type
  a_type'IMAGE (value_having_a_type)
- 'VALUE generates type value from string
  a_type 'VALUE (a_string_valid_in_a_type)
- In the example, use of 'IMAGE eliminates need for CASE

Standards.VHDL'93.attributes_2

'SIMPLE_NAME
'PATH_NAME
'INSTANCE_NAME

```
USE WORK.ALL;
CONFIGURATION average_delay OF nibble_comparator_test_bench IS
   FOR customizable
      FOR a1 : comp4
          USE ENTITY WORK.nibble_comparator(iterative);
      END FOR;
   END FOR;
END average_delay;
```

In bit_comparator(fixed_delay) a'PATH_NAME IS
a1: c1TO2: c(2)

In bit_comparator(fixed_delay) a'INSTANCE_NAME
contains all library, configurations, entities and architectures

- 'SIMPLE_NAME returns string of a name
- 'PATH_NAME is a complete list of labels in path of a name
- 'INSTANCE_NAME is a complete list of labels, libraries and configurations in path of a name

Standards.VHDL'93.groups

```
PACKAGE utility_attributes IS
   TYPE timing IS RECORD
       rise, fall : TIME;
   END RECORD;
   ATTRIBUTE delay : timing;
   ATTRIBUTE sub_dir : STRING;
   GROUP delayed_signals IS (SIGNAL <>);  -- GROUP Template
END utility_attributes;
--
USE WORK.utility_attributes.ALL;
--  FROM PACKAGE USE: delay, sub_dir
ENTITY brief_d_flip_flop IS
   PORT (d, c : IN BIT; q : OUT BIT);
   GROUP io_signals : delayed_signals (d, c, q);  -- GROUP Declaration
   ATTRIBUTE sub_dir OF brief_d_flip_flop : ENTITY IS "/user/vhdl";
   ATTRIBUTE delay OF io_signals : GROUP IS (8 NS, 10 NS);
END brief_d_flip_flop;
--
ARCHITECTURE attributed_falling_edge OF brief_d_flip_flop IS
   SIGNAL tmp : BIT;
BEGIN
   tmp <= d WHEN ( c = '0' AND NOT c' STABLE ) ELSE tmp;
   q <= '1' AFTER q'delay.rise WHEN tmp = '1' ELSE  '0' AFTER q'delay.fall;
END attributed_falling_edge;
```

Can use: c'delay.rise, c'delay.fall, r'delay.rise, r'delay.fall

- Group template declaration specifies entity classes in a group
- Use <> for any number of a specific entity class in a group
- Group declaration specifies members of a specific group

© Zainalabedin Navabi, 1994

Standards.VHDL'93.unaffected_assignment

For Selected Signal Assignment:

```
... UNAFFECTED WHEN OTHERS
```

For Conditional Signal Assignment:

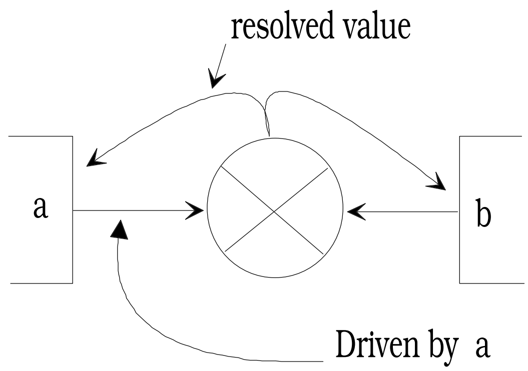
```
... ELSE UNAFFECTED
```

```
ENTITY d_flipflop IS
  GENERIC (delay1 : TIME := 4 NS; delay2 : TIME := 5 NS);
  PORT (d, c : IN BIT; q : OUT BIT);
END d_flipflop;
--
ARCHITECTURE with_internal OF d_flipflop IS
  SIGNAL internal_q : BIT;
BEGIN
  internal_q <= d WHEN ( c = '1' AND NOT c'STABLE ) ELSE internal_q;
  q <= internal_q AFTER delay1;
END with_internal;


ARCHITECTURE simpler OF d_flipflop IS
BEGIN
  q  <= d WHEN ( c = '1' AND NOT c'STABLE ) ELSE UNAFFECTED;
END simpler;
```

- The *with_internal* requires *internal_q* since *q* cannot be used on the right hand side
- The *simpler* Architecture does not use *q* output on the right hand side

Standards.VHDL'93.driving



'DRIVING returns BOOLEAN; TRUE if signal is being driven 'DRIVING_VALUE returns the driving value

```
ENTITY simple_t_flip_flop IS
   GENERIC (tq_delay : TIME := 6.3 NS);
   PORT (t : IN BIT; q : OUT BIT);
END simple_t_flip_flop;
--
ARCHITECTURE toggle OF simple_t_flip_flop IS
BEGIN
  tff: PROCESS (t)
  BEGIN
    IF t = '0' THEN
      q <= q'DRIVING_VALUE AFTER tq_delay;
    END IF;
  END PROCESS tff;
END toggle;
```

- Avoid extra delta delay by using value of *q*
- Avoid the use of *internal_state* variable or signal
- Returns locally contributed value and not resolved value

Standards.VHDL'93.file_io

File type definition:

```
TYPE   logic_data  IS  FILE  OF  CHARACTER;
```

Options in using a defined file:

```
FILE input_logic_value_file :  logic_data;
-- This option leaves file opening for a FILE_OPEN procedure

FILE  input_logic_value_file :  logic_data
                        OPEN READ_MODE IS "data.file"
-- This option includes an implicit call to FILE_OPEN
-- Can use READ_MODE, WRITE_MODE, or APPEND_MODE
```

If file declaration does not include open information:

```
FILE_OPEN (    FILE input_logic_value_file :  logic_data;
               "data.file";
               READ_MODE      );

FILE_OPEN (    ok_or_not : OUT FILE_OPEN_STATUS;
               FILE input_logic_value_file :  logic_data;
               "data.file";
               READ_MODE      );
```

To close an open file:

```
FILE_CLOSE (      input_logic_value_file      );
```

- Files can be opened by FILE_OPEN instead of by declaration

Standards.VHDL'93.file_io

```
TYPE logic_data IS FILE OF CHARACTER;
FILE input_logic_value_file : logic_data;  --  File is declared but not opened

--  File Type  logic_data  is Visible
 PROCEDURE assign_bits (
   SIGNAL target : OUT BIT;  FILE in_file : IN logic_data;  period : IN TIME)
  IS
   VARIABLE char : CHARACTER;
   VARIABLE current : TIME := 0 NS;
   VARIABLE ok_or_not : FILE_OPEN_STATUS;
  BEGIN
   FILE_OPEN ( ok_or_not;  in_file;  "data.file";  READ_MODE);
   IF ok_or_not = OPEN_OK  THEN
      WHILE NOT ENDFILE (input_value_file) LOOP
        READ (input_value_file, char);
        IF char = '0' OR char = '1' THEN
          current := current + period;
          IF char = '0' THEN
            target <= TRANSPORT '0' AFTER current;
          ELSIF char = '1' THEN
            target <= TRANSPORT '1' AFTER current;
          END IF;
        END IF;
      END LOOP;
   END IF;
  END assign_bits;
```

Calling the above procedure:

```
assign_bits (a_signal, input_logic_value_file, 1500NS)
```

- File declaration in '87 had an implicit open
- File declaration can now postpone opening for FILE_OPEN
- Modes: READ_MODE WRITE_MODE APPEND_MODE

Standards.VHDL'93.report_statement

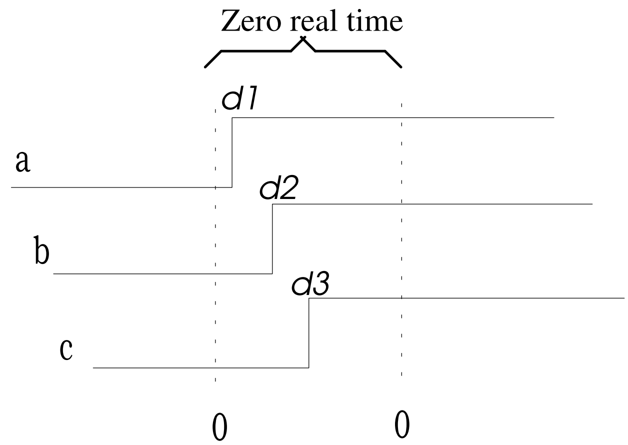
```
ASSERT make_sure_that REPORT violation SEVERITY some_action;
```

```
REPORT always_reports_some_conditions SEVERITY some_action;
```

```
ASSERT FALSE  REPORT always_reports SEVERITY some_action;
```

- Can use IF_THEN_ELSE for REPORT
- Same as ASSERT FALSE
- Less overhead than ASSERT
- Use REPORT for debugging, reporting events, etc

Standards.VHDL'93.postponed




```
PROCESS (a, b, c)            POSTPONED PROCESS (a, b, c)
BEGIN                        BEGIN
    .                            .
    .                            .
    .                            .
END PROCESS;                 END PROCESS;
```

- A PROCESS statement (left) is activated each time *a*, *b*, or *c* change
- A POSTPONED PROCESS (right) is activated after the last delta in a real time interval
- No zero delay assignments are allowed in a POSTPONED PROCESS

Standards.VHDL'93.other_changes

- Removal of 'BEHAVIORAL & 'STRUCTURE
- 'FOREIGN
- IMPURE Functions
- Using expressions for port association
- Extended CHARACTER set
- NXOR operator
- Consistent construct bracketing

```
ARCHITECTURE  xyz OF an_entity IS
. . .
END xyz ARCHITECTURE;
```

- SIGNATURE

Standards.conclusions

1. Outline: Introduction, Organization, Outline

2. Review: Behavioral description, Using process statements, Top-down design, Using available components, Wiring predefined components, Wiring from bottom to top, Generation of testbench data, Using procedures

3. VHDL_Timing: Modeling requirements, Objects & classes, Signals & variables, Concurrent & sequential assignments, Events, transactions & delta delays, Delay modeling, Sequential placement of transactions, Conventions

4. Structural_Description_of_Hardware: Wiring parts into larger designs, Start with primitives, Wire gates into general purpose components, Use iterative constructs, Generate testbenches, Show binding alternatives, Use gate-based components for a larger design

5. Design_Organization: Subprograms, Packaging, Parameter specification, Parametrization, Top level configuration, Design libraries, A complete example

6. Utilities_for_high_level_descriptions: Language aspects, Types, Over loading subprograms operators, Predefined Attributes, User defined attributes

7. Dataflow: Constructs for dataflow descriptions, Multiplexing and clocking, Multiple assignments, State machines, Open collector gates, A complete dataflow example, Load dependent timing

8. Behavioral_Descriptions: Constructs for sequential descriptions, Assertion for behavioral checks, Handshaking constructs, Timing control, Formatted I/O, MSI parts, A complete MSI based design

9. STANDARDS: MVL9: logic value system, Logic type, Operators, Conversions; VHDL'93: Operators, Delay model, Instantiation, Binding, Attributes, Signal assignments, Report, Postponed process

ERROR: undefined
OFFENDING COMMAND: :

STACK:

228