Review

- Concurrent statements
- Conditional and selected signal assignments
  - Cannot be placed inside a process
  - Equivalent to some process
- Assert statement
  - Debugging

VHDL in Action

Chapter 3
Chapter 5, Section 5.1.4
Data Types and Attributes

This week

- Basic Data types
- Object declarations
- Data type attributes

Lexical Elements

- Elements that cannot be divided by spaces, tabs, <CR>s
  - Identifiers
  - Comments
  - Delimiters (<=)
  - Literals

Literals

- Character Literal
  - '1' 'A'
- Character String Literal
  - "This is a string."
- Bit String Literal
  - B"00110110"
  - X"36"
  - B"0011_0110"

More Literals

- Abstract Literal
  - Integer
    - 21
    - 2134.5641
    - 2E10
  - Real
    - 2.0
    - 3.2E4
Data Types

- **Scalar**
  - values have only one component
  - example: integers and naturals
- **Composite**
  - types are complex objects
  - example: arrays or records
- **Access**
  - types that provide access to other types
- **File**
  - types that provide access to files

Scalar Types

- Enumeration (discrete)
- Integer (discrete, numeric)
- Physical (numeric)
- Floating point or real (numeric)

Predefined Enumeration Types

- type Bit is ('0', '1');
- type Boolean is (FALSE, TRUE);
- type Character is (NUL, ..., 'A', 'B', ..., 'a', 'b', ..., DEL)

User Defined Enumeration Types

- type COLOR is (Red, Orange, Yellow, Green, Blue);
- type STATE is (S1, S2, S3);
- type STD_ULOGIC is ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');
- type HORSE is (Mare, Stallion, Gelding, Colt, Filly);

SCALAR TYPES

**SCALAR TYPES**

**Predefined Enumeration Data Types**

- type COLOR is (Red, Orange, Yellow, Green, Blue);

**User Defined Enumeration Data Types**

- type COLOR is (Red, Orange, Yellow, Green, Blue);

**Example:**

```
signal MARKER: Color;  -- in declaration
signal CAR: Color := Yellow;
-----------------------------
MARKER <= Blue;  -- within the body of an architecture
```

SCALAR TYPES

**SCALAR TYPES**

**Subtype**

- Constrains the values of a type to be in the subtype range.
- Does not define a new type
- All subtypes of a given type have the same base type
Predefined Integer-related Types

- Integer type:
  - Subset of whole numbers ... -3, -2, -1, 0, 1, 2, ...
  - At minimum, the full 32-bit signed range is supported:
    - -2,147,483,647 to 2,147,483,647
- Natural type:
  - 0, 1, 2, 3, ...
  - subtype Natural is Integer range 0 to Integer'high
- Positive type:
  - 1, 2, 3, 4, ...
  - subtype Positive is Integer range 1 to Integer'high

Integers (cont’d)

- Declaration of integer-related types:
  - signal ANY_INT: Integer;
  - variable ANY_POS1, ANY_POS2: Positive;
  - signal ANY_NAT: Natural := 5;
- New types:
  - type COUNTER is range 0 to 100;
  - type FOOTSIZES is range 5 to 100;

VHDL is a Strongly Typed Language

- type APPLES is range 0 to 75;
- type ORANGES is range 0 to 75;
- variable A: APPLES := 25;
- variable B: ORANGES := 50;

  A := B;   -- Is this legal?
  if (A>B) then ....; -- Is this legal?

Physical Data Types

- Capture real-world measurable quantities
  - currents, torque, length, etc.
- Only one pre-defined physical type: TIME:
  type Time is range <implementation dependent>
  Units
  fs — femtoseconds
  ps = 1000 fs — picoseconds
  ns = 1000 ps — nanoseconds
  us = 1000 ns — microseconds
  ms = 1000 us — milliseconds
  sec = 1000 ms — seconds
  min = 60 sec — minutes
  hr = 60 min — hours
  END Units;

Physical (cont’d)

- Creating your own physical type:
  type RESISTANCE is range 0 to Integer'high
    units
    nohm — nano ohms
    uohm = 1000 nohm — micro ohms
    mohm = 1000 uohm — milli ohms
    ohm = 1000 mohm — ohms
    kohm = 1000 ohm — kilohms
    megohm = 1000 kohm — megohms
    END Units;

Floating Point Type

- Similar to integer types:
  signal A_FLOAT: Real;
- Declare customized REAL types:
  type ANGLE is range -90.0 to 90.0;
  type TESTSCORE is range 100.0 downto 0.0;
  type PROBABILITY is range 0.0 to 1.0;
Composite Types

- An object that holds more than one value: ARRAYS
- Predefined unconstrained array types:
  - type Bit_Vector is array (Natural range <>) of Bit;
  - type String is array (Positive range <>) of Character;

Using these predefined types:

- signal DATA_BUS: Bit_Vector(15 downto 0);
- signal INST_OP: Bit_Vector(5 to 7);

User Defined Composites

- Constrained composites
type BCD_BUS is array (3 downto 0) of Bit;
type REGFILE is array (5 downto 0) of Bit_Vector (7 downto 0);
type ARRAY2D is array (3 downto 0, 8 downto 0) of Bit;

- Unconstrained composite
type ARRAY2DB is array (Natural range <>), (Natural range <>) of Bit;

Composite Values

- Initializing and assigning values to composites:
  signal DATA_BUS: Bit_Vector(31 downto 0) := B"0111_0110_0101_0100_0011_0010_0001_0000";
signal ADDR_BUS: Bit_Vector(15 downto 0) := X"7654";

ADDR_BUS <= DATA_BUS(20 downto 5);

More Composite Values

- Initializing and assigning values to array composites:
type REGISTER is array (2 downto 0) of Bit_Vector(7 downto 0);
signal F: REGISTER := (X"12", X"DF", X"9A");

F(2) <= B"1010_0101";

Aggregates

- A structured collection of values used to initialize a signal or variable with composite type

Example: A two-dimensional array of characters

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>
```

Type CHAR2D is array (0 to 1, 0 to 1) of Character;

Variable CM1:CHAR2D := ("A","B","C","C"); -- positional

Variable CM2:CHAR2D := (1=>0, 1=>"C");

---

Composites with Enumeration Indexes

Type COLOR is (Red, Orange, Yellow, Green, Blue);

type COLOR_COUNT is array (COLOR range <>) of Natural;

Signal SUBPRISM : COLOR_COUNT (Orange to Green) := (15, 20, 30);
SUBPRISM(Orange) <= 32;
SUBPRISM(Yellow) <= 3;
**Composites Examples**

Type `COLOR` is (Red, Orange, Yellow, Green, Blue); type `ARRAY_OF_COLORS` is array (Natural range =>) of `COLOR`;

Signal `RAINBOW : ARRAY_OF_COLORS(3 to 5)`;

**Unary Operator Table**

Type `MVL4` is ('X', '0', '1', 'Z'); type `MVL4_TAB1D` is array (MVL4) of MVL4;

Constant `INV : MVL4_TAB1D` := ('X', '1', '0', 'X');

Variable `A, B : MVL4`;

`A := INV(B);`

**Binary Operator Table**

Type `MVL4D_TAB2D` is array (MVL4, MVL4) of MVL4;

Constant `AND_MVL4 : MVL4_TAB2D` :=

```
('X', '0', 'X', 'X'),
('0', '0', '0', '0'),
('X', '0', '1', 'X'),
('X', '0', 'X', 'X')
```

Signal `X, Y, Z : MVL4`;

`Z <= AND_MVL4(X, Y);`

**Records**

- Composites with heterogeneous elements
- Similar to C/C++ structures

Type `MONTH_NAME` is (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC);

Type `DATE` is record

```
DAY : Positive range 1 to 31;
MONTH : MONTH_NAME;
YEAR : Natural range 0 to 9999;
```

End record;

**Records (cont’d)**

- Using the `DATE` record:

Signal `BIRTHDAY, HOLIDAY : DATE`;

信号 `BIRTHDAY <= (16, AUG, 1943);`

HOLIDAY <= (25, DEC, 2000);

BIRTHDAY.DAY <= 0;

BIRTHDAY.MONTH <= JUN;

BIRTHDAY.YEAR <= 2000;

**Access Types**

- Also known as pointer types
- Provide a means to access dynamic objects (objects which are created and destroyed during simulation).
- Not used in this course
File Types

- Access to external data
- Uses:
  - to input test vectors and to output results,
  - to record messages from simulation, and
  - to initialize models (example, RAM)
- Covered later

Signal Attributes in VHDL

- Example:
  ```vhdl
  signal CLK: Bit;
  ```
  
  Referred to as
  a "tick"
  
  Spoken as:
  "clock tick event"

  Not: signal is of
  type BIT, attribute
  returns type
  Boolean

  Interpretation:
  Boolean function
  which is true when a
  change of value is
  occurring on this
  signal.

VHDL Signal Attributes

- In general:
  ```vhdl
  Signal_name'attribute
  ```
  
  Can return a variety of types
  
  Many built-in attributes
  - Part of the language
  - Can create your own attribute

Data Type Attribute

- 'pos
  ```vhdl
  Type_name'pos
  ```
  Function that returns the position number of a specific value from a list of values. The first value in any enum type has a position number of 0.

  ```vhdl
  A := COLOR'pos(GREEN);
  ```

Data Type Attribute

- 'val
  ```vhdl
  Type_name'val
  ```
  Function that returns the value at a specific position number in a list of values (val(0) returns the first item).

  ```vhdl
  B := COLOR'val(4);
  ```

Data Type Attribute

- 'left
  ```vhdl
  Type_name'left
  ```
  Predefined constant that specifies the left-most value in a list of values.

  ```vhdl
  A := COLOR'left;
  ```
### Data Type Attribute

**'right**

Type name'right  
Predefined constant that specifies the right-most data value in a list of values.

A := COLOR'right;

---

**'low**

Type name'low  
Predefined constant that specifies the value associated with the lowest position number in a list of values.

A := COLOR'low;

---

### Subtype Examples

**COLOR**

is

(Red, Orange, Yellow, Green, Blue)

**LONGWAVE**

is

COLOR range COLOR'left to Yellow

**SW**

is

COLOR range COLOR'right downto Orange

A := SW'right;  
-- What is the value of A?
B := SW'high;  
-- What is the value of B?
C := SW'val(2);  
-- What is the value of C?
D := SW'pos(Green);  
-- What is the value of D?

### More Attributes

**'high**

Type name'high  
Predefined constant that specifies the value associated with the highest position number in a range.

A := COLOR'high;

---

**'event**

SIG_NAME'event  
Function that returns a Boolean value that is TRUE if there is an event on signal SIG_NAME during the current simulation cycle.

signal CLK : Bit := '0';

---

**Determining if a rising edge in CLK has occurred in a process:**

DFFAC: process (CLK, RESET)
begin
if RESET = '1' then
Q <= '0';
elsif CLK'event and CLK = '1' then
-- if CLK just changed value, and is now '1', then
Q <= D;
end if;
end process;
**More Useful Attributes**

- **active**

```vhdl
SIG_NAME'active Function that returns a Boolean value that is TRUE if there is a transaction on signal SIG_NAME during the current simulation cycle.
```

```vhdl
signal CLK : Bit := '0';
assert not CLK'active
report "note: CLK has just been assigned";
```

- **stable(n)**

```vhdl
SIG_NAME'stable(n) A signal of type Boolean that is TRUE only if signal SIG_NAME has NOT changed value (has no events) for time n.
```

**Relationship Between 'stable and 'event**

<table>
<thead>
<tr>
<th>Time (ns)</th>
<th>S</th>
<th>S'table</th>
<th>S'event</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

**D Flip-Flop Timing Requirements**

- **SETUP TIME (SUT)** - D input must be stable for 10 ns prior to the rising edge of clock.
- **HOLD TIME (HT)** - D input must be stable for 2 ns after the rising edge of clock.
- **MINIMUM PULSE WIDTH (MPW)** - The clock must be high for at least 8 ns.

**Timing Assertion Development**

- Develop an expression that is true when the error occurs.
- Logically negate the expression to obtain the required assert condition.
- DeMorgan’s Theorem may be useful.
Detecting Setup Time Violation

signal STV: Boolean;
D
CLK
\\\( STV \Leftarrow \text{CLK}' \text{event and } \text{CLK}=\text{'1'} \text{ and (not } \text{D}' \text{stable}(\text{SUT})\text{);} \)
assert (not STV) report "Setup violation on D input."

Detecting Hold Time Violation

signal D: Bit;
signal STV: Boolean;
\\( STV \Leftarrow \text{CLK}' \text{event and } \text{CLK}=\text{'1'} \text{ and (not } \text{D}' \text{stable}(\text{SUT})\text{);} \)
assert (not STV) report "Setup violation on D input."

More Useful Attributes

SIG_NAME'quiet(n) A signal of type Boolean that is TRUE only if signal SIG_NAME has no transactions for time n.

More Signal Attributes

SIG_NAME'delayed(n) A signal of type equal to the base type of signal SIG_NAME

Attributes for Signals of Array Type

SIG_NAME'left Returns the left most value of the index range of SIG_NAME

### Attributes for Signals of Array Type

**'right**

- `SIGNAL_NAME.right` Returns the right most value of the index range of `SIGNAL_NAME`.
- ```
  signal DBUS : Bit_Vector ( 5 downto 0);
  ...
  DBUS(DBUS'right) <= '0'; -- which value is assigned?
  ```

**'high**

- `SIGNAL_NAME.high` Returns the upper bound of the index range of `SIGNAL_NAME`.
- ```
  signal DBUS : Bit_Vector ( 5 downto 0);
  ...
  DBUS(0) <= DBUS(DBUS'high);
  ```

**'low**

- `SIGNAL_NAME.low` Returns the lower bound of the index range of `SIGNAL_NAME`.
- ```
  signal DBUS : Bit_Vector ( 5 downto 0);
  ...
  for I in DBUS'high downto DBUS'low loop
  ```

**'ascending**

- `SIGNAL_NAME.ascending` Returns a Boolean `TRUE` if the declaration of `SIGNAL_NAME` is an ascending range.
- ```
  if DBUS.ascending then
    for I in DBUS'left to DBUS'right loop
    end loop;
  end if;
  ```

**'length**

- `SIGNAL_NAME.length` Returns the number of elements in array `SIGNAL_NAME`.
- ```
  signal DBUS : Bit_vector ( 5 downto 3);
  ...
  for i in DBUS'length-1 downto 0 loop
  ```

**'last_event**

- `SIGNAL_NAME.last_event` Returns the TIME since the last event on `SIGNAL_NAME` occurred.
- ```
  signal CLK : Bit := '0';
  ...
  if CLK'last_event < 100 ns then
  end if;
  ```

### Signal Attributes
Signal Attributes

- **'last_active**
  
  `SIG_NAME.last_active`

  Returns the TIME since the last transaction on `SIG_NAME`.

```vhdl
signal CLK : Bit := '0';
--
if CLK'last_active > CLK'last_event
  assert false report
  "CLK assigned the same value at least twice";
```

- **'last_value**
  
  `SIG_NAME.last_value`

  --Returns the previous value assigned to `SIG_NAME`.

```vhdl
signal TRAFFIC : COLOR := red;
process begin
  if TRAFFIC'last_value = red and TRAFFIC = green
    then
```

More More Attributes

- **SIG'leftof / SIG'rightof**
- **SIG'succ / SIG'pred**
- **SIG'pos**
- **SIG'vel**
- **SIG'range / SIG'reverse_range**
  - for i in SIG'range loop

Is that all?

- No. Plenty more attributes.
- Plus: *user-defined* attributes.