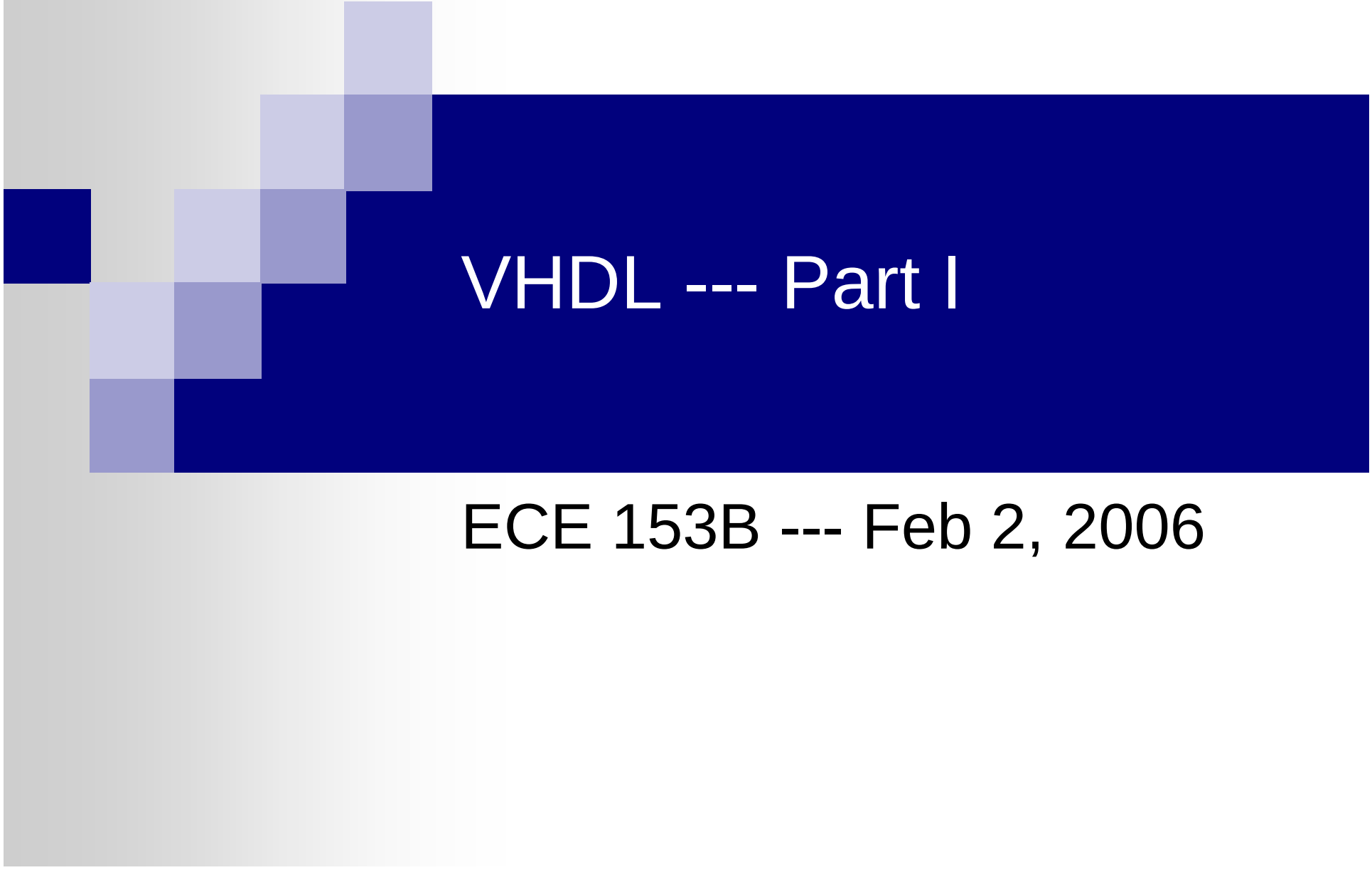
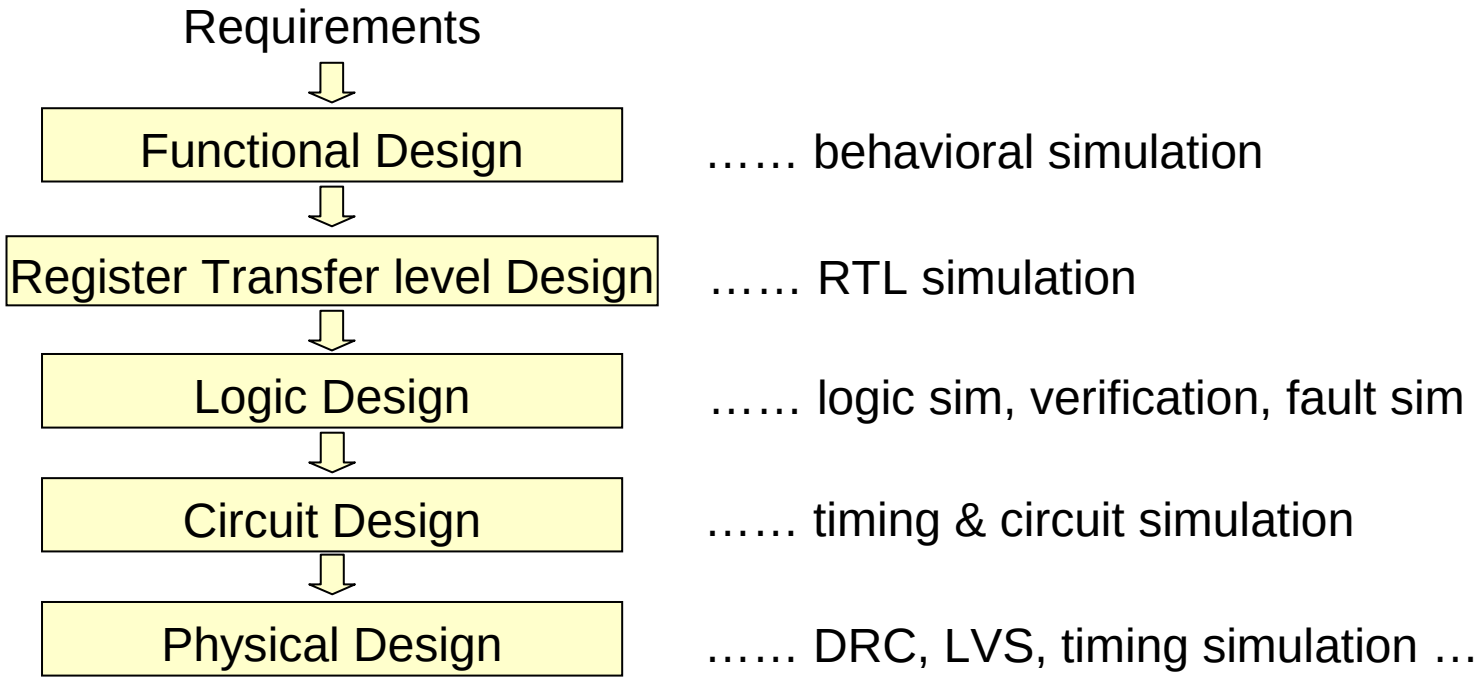


ECE 153B --- Feb 2, 2006

Hardware Description Languages

- HDLs support top-down digital systems design



VHDL --- an acronym

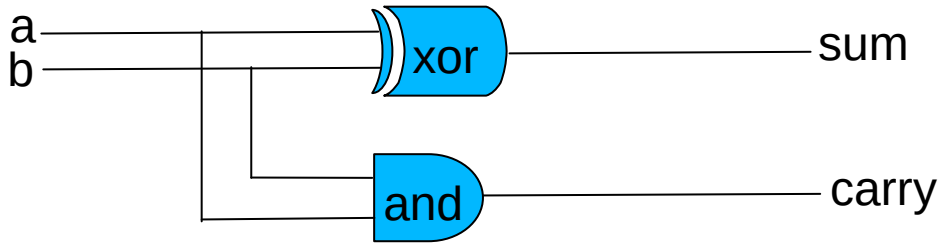
- VHSIC Hardware Description Language
- VHSIC == very high-speed integrated circuit

- Language focus is on interfaces
- Two complementary views
  - *Behavioral* – does not tell much about structure
  - *Structural* – does not give details about behavior
- VHDL allows simulation at many levels
- VHDL now an IEEE standard
  - originally IEEE std-1076 in 1987, later IEEE std-1164 when std_logic types added

Behavioral & Structural Descriptions

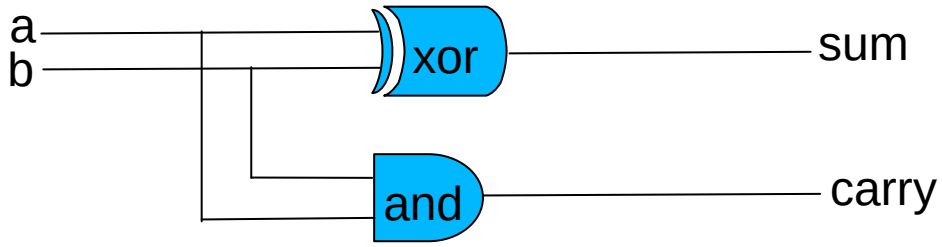
- digital systems are about signals (0 vs. 1)
- made from components, e.g.
  - gates, FF's, muxes, decoders, counters
- components interconnected by wires
  - transform inputs into outputs

Consider a Half-Adder



- a, b are inputs
- sum, carry are outputs
- xor, and are components
  - with internal functions that transform their inputs to their outputs
- interconnecting everything are wires

Half-Adder is a design “entity”




```
entity half_adder is
    port ( a, b : in bit;
           sum, carry : out bit );
end half_adder;
```

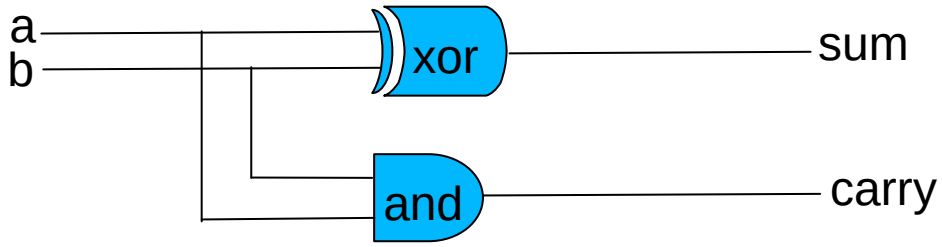
- VHDL is case insensitive
- Inputs and outputs are referred to as ports
- Ports are special programming objects & also they are signals
- Each must be declared to be a certain type, in this case “bit” (a signal that can be 0 or 1)
  - Another possibility is bit_vector – an array or vector of bits

IEEE 1164 9-valued logic system

Value	Interpretation
U	uninitialized
x	forcing unknown
0	forcing 0
1	forcing 1
Z	high impedance (floating)
w	weak unknown
L	weak 0
H	weak 1
-	don’t care

- Known as “std_ulogic” or “std_ulogic_vector”

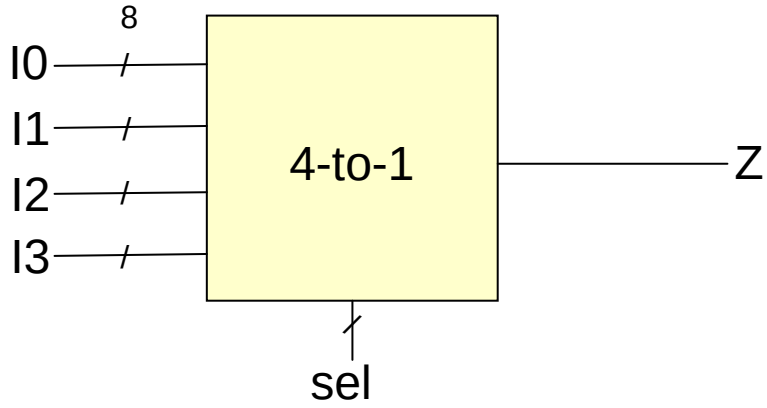
Half-Adder re-written using std_ulogic




```
entity half_adder is
    port ( a, b : in std_ulogic;
           sum, carry : out std_ulogic );
end half_adder;
```

- VHDL is case insensitive for keywords and identifiers
- Inputs and outputs are referred to as ports
- Ports are special programming objects & also they are signals
- Each must be declared to be a certain type, in this case “bit” (a signal that can be 0 or 1)
  - Another possibility is bit_vector – an array or vector of bits

A 4-to-1 multiplexer




```
entity mux is
    port ( I0, I1, I2, I3 : in std_ulogic_vector (7 downto 0);
           sel : in std_ulogic_vector (1 downto 0);
           Z : out std_ulogic_vector (7 downto 0) );
end mux;
```

Entities and Architectures

- Once we have described an entity's interface ports,
  - we can describe its internal behavior
- Every VHDL design must have at least one entity/architecture pair

```
entity mux is
    port ( I0, I1, I2, I3 : in std_ulogic_vector (7 downto 0);
           sel : in std_ulogic_vector (1 downto 0);
            Z : out std_ulogic_vector (7 downto 0) );
end mux;
architecture behav of mux is
    -- place declarations here (note that -- introduces a line comment)
    begin
    -- description of behavior here
    end behav;
```

Some Common Data Types

Type	Values	Example
bit	‘0’, ‘1’	Q <= ‘1’;
bit_vector	array of bits	data <= “00010110”;
boolean	True, False	EQ <= True;
integer	…, -2,-1,0,1,2, …	count <= count + 2;
real	1.0, -1.0E5	v1 = v2 / 5.3;
time	7ns, 100ps	Q <= ‘1’ after 6 ns;
character	‘a’,’b’,’2’,’$’	Chardata <= ‘x’;
string	array of characters	msg <= “MEM:” & addr;

VHDL descriptions

- You can write VHDL to describe a part …
  - by behavior (allows simulation of HOW & WHAT)
  - by dataflow (allows an RTL level of description)
  - by structure (allows composition from basic components)

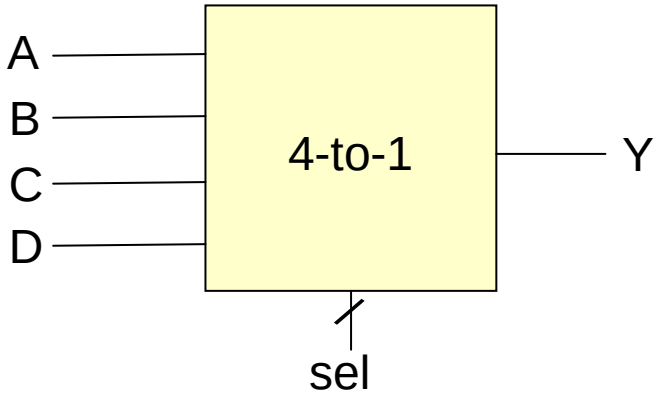
Concurrent signal assignments -- CSAs

- We can use CSAs to specify behavior
- If an event (signal transition) occurs on a signal on the rhs of a CSA,
  - The expression is evaluated and new values are scheduled for a time in the future per the optional after clause
- The order of CSAs is not significant. They are concurrent.

```
architecture concur_behav of half_adder is
    begin
    sum <= (a xor b) after 5 ns;
    carry <= (a and b) after 5 ns;
    end concur_behav;
```

A 4-to-1 multiplexer

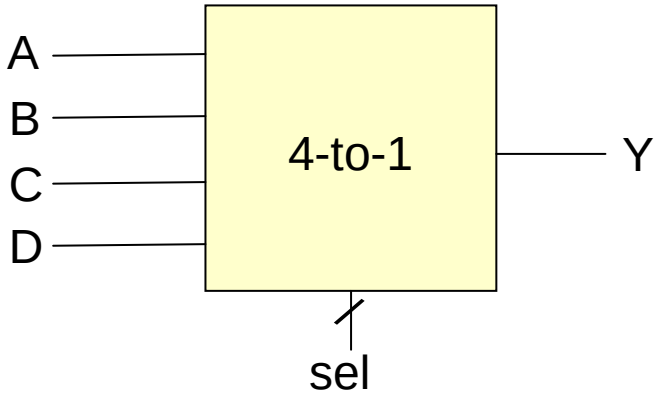
```
entity mux is
    port ( A, B, C, D : in std_ulogic;
           sel : in std_ulogic_vector (1 downto 0);
           Y : out std_ulogic);
end mux;
architecture m1 of mux is
begin
    Y <= A when (sel = "00") else
         B when (sel = "01") else
         C when (sel = "10") else
         D when (sel = "11");
end m1;
```



called a ‘conditional signal assignment’

- aka a ‘when/else’ statement

Selected signal assignment --- with/select




```
architecture m2 of mux is
begin
   with sel select
        Y <= A when "00",
             B when "01",
             C when "10",
             D when "11";
end m2;
```

- similar to when/else but does not imply priority
- must cover all possibilities else a latch will be implied

Process statement

- primary means by which sequential circuits are described

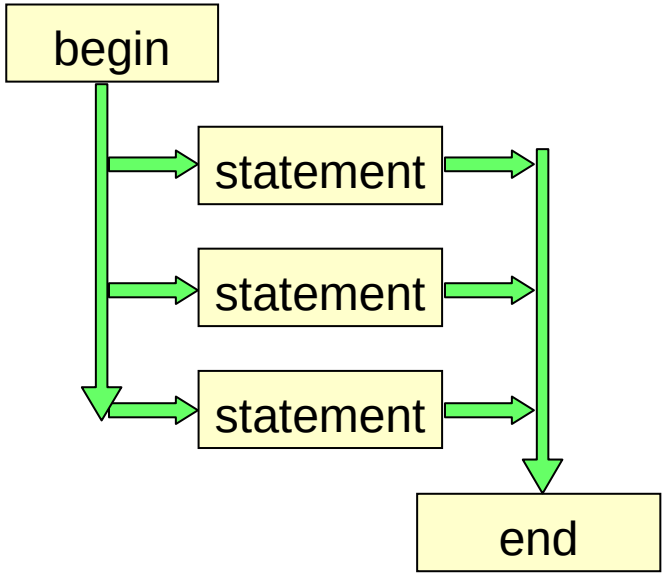
```
architecture arch_name of entity_name is
begin
    process_name: process ( sensitivity list )
                  local declaration(s);
                            :
                  begin
                  sequential stmt;
                  sequential stmt;
                           :
                  end process;
end arch_name;
```

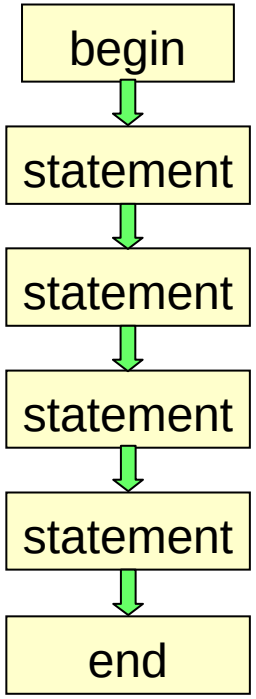
Any event (change) in any of these signals causes execution of the process

Time stands still during sequential execution of these statements.

There is another kind of process; without any sensitivity list. It uses wait until (condition) or wait for (condition) statements

Concurrent vs. Sequential Execution





Concurrent (between begin/end in architecture)

- everything happens “at once”
- no significance to stmt order

Sequential (between begin/end in a process)

- statements happen in sequence

Using a process to describe registers

```
architecture rot2 of rotate is
    signal Qreg : std_logic_vector (0 to 7);
    begin
        reg: process (rst, clk)      -- only execute this process when clk or rst changes
           begin
           if rst = ‘1’ then         -- asynchronous reset
               Qreg <= "00000000";
           elsif (clk = ‘1’ and clk’event) then      -- leading edge clocking
               if (load = ‘1’) then
                   Qreg < data;
               else
                   Qreg < Qreg(1 to 7) & Qreg(0);    -- rotate one position left
               end if;
           end if;
        end process;
    Q <= Qreg;                   -- concurrent (C/L) assignment happens concurrently with process
end rot2;
```

•For synthesis, process must be structured so as to show the intended structure.

Elements of the language

- Signals
  - Objects that connect concurrent elements
  - All ports are signals
- Variables
  - Objects used to store intermediate values between seq statements
  - Only allowed in processes & functions …. always local
- Constants
  - Assigned a value once (when declared) …. does not change
    - Constant countlimit : integer := 255;
    - Constant msg : string := “This is a string”;
    - Constant myaddr : bit_vector (15 downto 0) := X”F0F0”;
- Literals (next slide)

Literals

- Explicit data values that are assigned to objects or used in expressions
- Character literals --- 1 char ASCII values enclosed in single quotes
  - ‘A’, ‘$’, ‘g’
- String literals --- one or more ASCII characters in double quotes
  - “testing, 1-2-3” , “This is a string of characters --- i.e. a string literal”
- Bit-string literals --- special string literals to represent binary, octal, hex
  - B”01101111” ---- 8-bit binary literal
  - O”7602” ---------- 3 x 4 = 12 bits in octal
  - X”1CF2” ---------- 16-bit hexadecimal literal
- Numeric literals --- decimal integers and reals
  - 5.0 -12.9 1.6E10 2.45E-10
- Based literals
  - 2#100010001# 16#FFCC# 2#101.00#E10
- Physical literals – representing physical quantities like time, voltage, current, distance
  - 300 ns, 900 ps, 40 ma, 16 v -- always a numeric part and a unit specification

Types and Sub-Types

- Scalar types
  - represent a single value
- Composite types
  - represent a collection of values (e.g. arrays, records)
- Access types
  - ala pointers, providing references to objects
- File types
  - reference types (typically files on disk) that contain a sequence of values

Scalar Types

bit	‘1’, ‘0’	Only two possibilities
boolean	True, False	
integer	-2^{31} – (2^{31}-1)	32-bit, signed integer
character	‘a’, ‘b’, ‘@’, …	
real	floating-point number	
severity level	NOTE, WARNING, ERROR, FAILURE	Only used in the report section of an assert statement
time	100 ns	Units: fs,ps,ns,us,ms,sec,min,hr

Enumerated & Composite Types

- Enumerated Type
  - Ordered scalar subtype used to describe high-level desiugn concepts symbolically
  - `type states is (Idle, Read1, Read2, Write1, Refresh, Cleanup);`
- Composite Types

`bit_vector`	`"0010001"`	`1-D array of bit`
`string`	`"simulation fail"`	`array of character`
`record`	`any collection of values`	`user-def'd composite object`

Record Types

```
Type data_in is
   record
       ClkEnable : std_logic;
       Din : std_logic_vector (15 downto 0);
       Addr : integer range 0 to 255;
       CS : std_logic;
   end record;

signal test_rec : data_in := ( ‘0’, "1001011011110011", 165, ‘1’);
       or
test_rec.ClkEnable <= ‘0’;
```

Operators

`and,or,nand,nor,not,xor,xnor`	Combine any bit or boolean types
`=  /=  <  <=  >  >=`	Ordering any scalar or discrete array types
`+  -  &`	Any numeric type (& is concatenate)
`*  /  mod  rem  **  abs`	Mult/div, etc. …. but not for synthesis
`sll  srl  sla  sra  rol  ror`	L: any 1-D array type with bits or booleans R: integer

Attributes

- Attributes are always applied to a prefix, in this case the CLK signal

```
wait until CLK='1' and CLK'event and CLK'last_value = '0';
```

List of attributes

'Left	leftmost element index of	bit_array'Left
'Right	Rightmost element index of	bit_array'Right
'High	Upper bound of a type or subtype	
'Low	Lower bound of a type or subtype	
'Ascending	Boolean (True if type has ascending range)	
'Length	Number of elements in an array	