EE366 – CMOS VLSI Design

VHDL – Language Elements

VHDL Syntax- summary

• Identifiers, Numbers, Strings
• variables, signals, constants and types
• arrays, records
• Expressions, Operators
• Sequential vs. Concurrent Statements
• Variable Assignment
• If, Case, Loop, While, For, Null, Assert

VHDL Syntax- summary (II)

• entity declaration
• architecture declaration
• signal declaration
• blocks
• component declaration vs. instantiation
• Signal Assignment (assignment semantics)
• wait

VHDL Syntax- summary (III)

• Concurrent vs. Sequential Signal Assignment
• with .. select, when
• Packages
• Functions and Procedures
• Design Units
• Libraries, Configurations

VHDL Syntax- summary (IV)

• The Library std_logic_1164
• The type Std_logic
• Resolved Signals, Drivers
• Generate
• VITAL

Identifiers –reserved words

• All keywords are reserved, and cannot be used to name any user-defined object (signal, variable, architecture, process, component, loop variable….)
• See handout for a list of reserved words
• VHDL is case insensitive!
Identifiers – user-defined

• A valid identifier is:
  – Any sequence of letters, digits and isolated underline,
  – starting with a letter,
  – ending with a letter or a digit
• Ex.: valid: A a1 A_a
• non valid: _1 a_ a-b a__b
• Note: VHDL’93 adds \*\*\*\ identifiers
  – Ex. \74ls163\ is valid in VHDL’93

Delimiters and separators

• Separators: Space, tab, end of line.
• Delimiters:
  – Simple: “ & # ‘ ( ) * + . / ; : ; <= => <= /= **
• Comments are introduced by -- and are valid until the end of the line

Literals

• Character literal : ‘char’
  – ex.: ‘0’, ‘a’
• String literal: “string of chars”.
  – ex.: “a string”, “*****”
  – Concatenation operator: &
    • “abc&’def” is equivalent to “abcdef”
• Bit string literal: base_specifier“string of digits”
  – Ex.: “11010011”, X”E3”, O”323”
  – _ delimiter for readability: B”1101_0011”

Numbers

• They are abstract literals
• Decimal integer: with or without exponent.
  – Ex: 34 3e5 111 111E10
• Decimal real: needs the decimal point, and a digit before and after it
  – Ex: 2.0e-2
• Based: uses a different base:
  – 4#3300#

Variable vs. Signal Declaration

• variable varname : vartype;
  – Only inside processes, procedures or function
  – Local scope (VHDL’93 has global variables)
    • Ex: variable x : time :=1.0 s;
• signal signame : sigtype;
  – Inside architectures
  – Visible everywhere in the architecture
    • Ex: signal a,b : bit;
    • Ex: signal num : integer := 10;

Constants

• Constant constname : type := constvalue;
  – Deferred constants accepted
Types and Strong Typing

- VHDL is a strongly typed language:
  - Typed consistence is always checked
- Divided into 4 classes
  - Scalar
    - Discrete (enumeration and integer)
    - real
  - Physical
    - Composite (arrays and records)
    - access (like C pointers – allow dynamic variables)
    - file (useful for test vectors and output dump)

Enumeration types

- Simple list of values
  - type traffic_light is (RED, YELLOW, GREEN);
- Built-in enumeration types:
  - type bit is ('0', '1');
  - type boolean is (FALSE, TRUE);
  - type severity_level is (NOTE, WARNING, ERROR, FAILURE);
  - type character is (NUL, SOH, ..., 'A', 'B', ..., DEL);
- Practically identifiable with the numbers from 0 to n-1

Integer data types

- Predefined:
  - type integer is range min_num to max_num;
  - subtype natural is integer range 0 to integer'high;
  - subtype positive is integer range 1 to integer'high;
- User-defined:
  - type my_int is range -10 to 200;
  - subtype my_pos is my_int range 0 to my_int'high;
- Subtypes can be freely used in expressions with their defining type

Real types

- Predefined:
  - type real is range min_real_num to max_real_num;
  - max_real_num and |min_real_num| >= 1.0e+38
- User-defined:
  - type probability is range 0.0 to 1.0;
  - type reverse_real is range 200.0 downto -10.0;
- Subtypes can be freely used in expressions with their defining type

Physical types

- Predefined:
  - Type time is range min_num to max_num
    units
    fs;
  - ps = 1000 fs;
  - ns = 1000 ps;
  - us = 1000 ns;
  - ms = 1000 us;
  - sec = 1000 ms;
  - min = 60 sec;
  - hr = 60 min;
  - end units;

Arrays

- Like in classical programming language
  - type array_of_tl is array (0 to 10) of traffic_light;
- Predefined:
  - type string is array (positive range <=) of character;
  - type bit_vector is array (natural range <=) of bit;
    - Ex: signal counter : bit_vector(3 downto 0) := "0000";
- Assignment can be made by subranges:
  - Counter1(3 downto 1) <= counter2(2 downto 0);
- Single elements access: a <= Counter1(2);
Multidimensional arrays

• Arrays of arrays can be defined
  – type archar is array 0 to 10 of character;
  – type arrarchar is array 0 to 4 of archar;
  – Accessed as char_var := arrarchar(3)(5)
• Or, purely multidimensional arrays as
  – Type doublearrint is array(0 to 2, 0 to 1) of integer;
  – Accessed as int_var :=doublearrint(2,1)
• Array assignment (in both cases) is done as:
  – Doublearrint_var :=( (1,2), (3,4), (5,6) )

Operations

• Logical: and or nand nor xor (bit or boolean)
• Relational: = /= < <= > >= (same base type->boolean)
• Adding: + - & (same base type)
• Sign: + - (numeric)
• Multiplicative: * / mod rem (integer and/or real)
• Miscellaneous: ** abs not

Expressions

• Use operators, and parentheses.
• And, or are associative, nand, nor are not
• VHDL ’93 has added xnor, sll, srl, sla, sra, rol, ror

Sequential Statements

• Concurrent signal assignment
• Concurrent assert
• component instantiations
• blocks

Concurrent Statements

• Classical if construct
if condition1 then
  ...
elsif condition2 then
  ...
elsif condition3 then
  ...
else
  ...
end if;

IF...THEN...ELSIF...THEN...ENDIF
CASE…IS…WHEN … END CASE
• Classical switch construct
```plaintext
case expression is
  when value1 => ...
  when value2 => ...
  when valuesn => ...
  when others => ...
end case;
```
{Needed if values don’t cover all choices!}

What if you don’t…
• …want to do anything for some choices?
• Use null statement
```plaintext
case expression is
  ...
  when others => NULL;
end case;
```

LOOP … END LOOP
• Declares a loop to be executed
• Without controls (while, for, exit) it executes forever
```plaintext
[looplabel:] loop … end loop;
```

WHILE
• Exits a loop when condition fails to be true
```plaintext
while(condition) loop … end loop;
```

FOR
• Repeats the loop a number of times
```plaintext
for loopvar in a to b loop … end loop;
```
Ex.: for i in 0 to 100 loop sum:=sum+i end loop;
• loopvar
  – doesn’t have to be declared
  – can be used inside the loop
  – cannot be modified in the loop
• A to/downto b can be any discrete range
  – Ex: for trafl in traffic_light loop … end loop;

NEXT
• Skips to the end of the loop
```plaintext
loop …; next; …; end loop;
```
• In case of indented loop the label could be used to specify the end loop to go to by
```plaintext
next(looplabel);
```
EXIT

- Exits to the end of the loop
  `loop ...; exit; ...; end loop;`
- In case of indented loop the label could be used to specify the end loop to go to by `exit(looplabel);`
- Exit can have an exit condition:
  - `exit looplabel when condition;`

Signal and Variable assignment

- Signal assignment:
  - `signname <= expression after timeexpr;`
  - Transport vs. inertial
    - `signname <= [inertial] expression after timeexpr;`
    - `signname <= transport expression after timeexpr;`
  - If outside a process is a process by itself ~ to
    - `process (signals in expression) begin
      signname <= expression after timeexpr;
    end process;`
- Variable assignment
  - Only inside a process;
  - `signname := expression;`

Inertial vs transport delays

- Inertial delay (default) overwrites old events in the event list (as shown in previous lecture)
  - Result: glitches with delay < assignment delay are not propagated (the assignment has inertial)
  - Models gates
- Transport delay leaves old events in the list
  - Result every event scheduled is actually propagated
  - Models wires
- VHDL'93 adds `reject` to model more precisely pulse reject

Multiple assignment

- A single statement can describe multiple assignments on the same signal:
  - Ex `a <= '1' after 1 ns, '0' after 2 ns, '1' after 5 ns`
  - Only the first can be inertial! Otherwise all others will overwrite the first!

with … select

- Used to control assignment based on the value of an object (variable or signal):
  - `with object select
    sgnl <= val1 when selval1,
    sgnl <= val2 when selval2,
    ...
    sgnl <= valdef when others;`

when … else

- Used to control assignment based on the value of boolean expressions
  - `sgnl <= val1 when expr1 else
    val2 when expr2 else
    ...
    valn-1 when exprn-1 else
    valn;`
**ASSERT…REPORT…SEVERITY…**

- Used to signal the simulator when a desired condition is not met
- `assert condition report string severity severity_level;`
- If `condition` is not true, then it will tell the simulator there is a violated assertion, whose `severity_level` is given reporting `string`
  - Ex. `assert (data_last_event>t_setup) report "setup time violation!" severity ERROR;`
- `report` can be used independently in VHDL '93

**component** Instantiation

- The signal mapping can be done in 3 different ways: (suppose the component has a `port(first, second: in bit; third: out bit)`
  - Positional notation:
    - `port map (a, b, c)`
  - Index notation
    - `port map (1=>a, 3=>c, others=>b)`
  - Named notation
    - `port map (first=>a, third=>c, second=>b)`
- Can be variously combined

**blocks**

- Help in organizing the structure inside an architecture
- `label: block
deaclerative part
begin
end block label;`
- `guarded` blocks
  - `block/guard_expression)
begin
  signal_val after time_expr;
end block;`

**generics and generic instantiation**

- `entity and is
generic delay : time := 0 ns;
port (a, b: in bit; c: out bit);
end and;`
- `component and
generic delay : time := 0 ns;
port (a, b: in bit; c: out bit);
end component;`
- Instantiated using `generic map`. Same syntax as port map
  - `An1: and generic map (2 ns) port map (1,2,o1);`