assignVHDL header:

Ieee library

Std\_logic\_1164

Use ieee.numeric\_std.all;

Synthesizable code:

* No initialization statement;
* No system call;
* No timing information (delay);
* Loops unroll fixed index limit;
* Incomplete “case” or “if” statement => latch;
* No Real, No time type, no force/release, no table, no assingment to registers, limited use of ‘x’ (donot care);

|  |  |
| --- | --- |
| Library | LIBRARY library\_name;  USE library\_name.package\_name.package\_parts; |
| Entity | ENTITY entity\_name IS  PORT (  port\_name : signal\_mode signal\_type;  port\_name : signal\_mode signal\_type;  ...);  END entity\_name;  Mode: IN, OUT, INOUT, or BUFFER  ENTITY full\_adder IS  PORT (a, b, cin: IN BIT;  s, cout: OUT BIT);  END full\_adder; |
| Architecture | ARCHITECTURE architecture\_name OF entity\_name IS  [declarations]  BEGIN  (code)  END architecture\_name;  ARCHITECTURE dataflow OF full\_adder IS  BEGIN  s <= a XOR b XOR cin;  cout <= (a AND b) OR (a AND cin) OR  (b AND cin);  END dataflow; |
| Process | The PROCESS is executed every time a signal declared in its sensitivity list changes. |
| Data types | Package standard of library std: Defines **BIT, BOOLEAN, Natural, INTEGER, and REAL**  Package std\_logic\_1164 of library ieee: Defines **STD\_LOGIC and STD\_ULOGIC**  std\_logic\_arith of library ieee: Defines **SIGNED and UNSIGNED**  Packages std\_logic\_signed and std\_logic\_unsigned |
| User-Defined Data Types | TYPE integer IS RANGE -2147483647 TO +2147483647;  -- This is indeed the pre-defined type INTEGER.  TYPE natural IS RANGE 0 TO +2147483647;  -- This is indeed the pre-defined type NATURAL.  TYPE my\_integer IS RANGE -32 TO 32;  -- A user-defined subset of integers.  TYPE student\_grade IS RANGE 0 TO 100;  -- A user-defined subset of integers or naturals. |
| User-defined enumerated types | TYPE bit IS ('0', '1');  -- This is indeed the pre-defined type BIT  TYPE my\_logic IS ('0', '1', 'Z');  -- A user-defined subset of std\_logic.  TYPE bit\_vector IS ARRAY (NATURAL RANGE <>) OF BIT;  -- This is indeed the pre-defined type BIT\_VECTOR.  TYPE student\_grade IS RANGE 0 TO 100;  -- A user-defined subset of integers or naturals.  TYPE state IS (idle, forward, backward, stop);  -- An enumerated data type, typical of finite state machines.  TYPE color IS (red, green, blue, white);  -- Another enumerated data type. |
| Subtype | SUBTYPE natural IS INTEGER RANGE 0 TO INTEGER'HIGH;  -- As expected, NATURAL is a subtype (subset) of INTEGER.  SUBTYPE my\_color IS color RANGE red TO blue;  -- Since color=(red, green, blue, white), then  -- my\_color=(red, green, blue). |
| Array | TYPE type\_name IS ARRAY (specification) OF data\_type;  TYPE row IS ARRAY (7 DOWNTO 0) OF STD\_LOGIC; -- 1D array  TYPE matrix IS ARRAY (0 TO 3) OF row; -- 1Dx1D array  SIGNAL x: matrix; -- 1Dx1D signal  TYPE matrix IS ARRAY (0 TO 3) OF STD\_LOGIC\_VECTOR(7 DOWNTO 0);  TYPE matrix2D IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD\_LOGIC; -- 2D array |
| Record | TYPE birthday IS RECORD  day: INTEGER RANGE 1 TO 31;  month: month\_name;  END RECORD; |
| Assignment Operators | <= Used to assign a value to a SIGNAL.  := Used to assign a value to a VARIABLE, CONSTANT, or GENERIC. Used also for establishing initial values.  => Used to assign values to individual vector elements or with OTHERS. |
| Logical Operators | NOT  AND  OR  NAND  NOR  XOR  XNOR |
| Arithmetic Operators | + Addition  - Subtraction  \* Multiplication  / Division  \*\* Exponentiation  MOD Modulus  REM Remainder  ABS Absolute value |
| Comparison Operators | = Equal to  /= Not equal to  < Less than  > Greater than  <= Less than or equal to  >= Greater than or equal to |
| Shift Operators | sll Shift left logic positions on the right are filled with ‘0’s  srl Shift right logic positions on the left are filled with ‘0’s  sla, sra, rol, ror |
| Data Attributes | \_ d’LOW: Returns lower array index  \_ d’HIGH: Returns upper array index  \_ d’LEFT: Returns leftmost array index  \_ d’RIGHT: Returns rightmost array index  \_ d’LENGTH: Returns vector size  \_ d’RANGE: Returns vector range  \_ d’REVERSE\_RANGE: Returns vector range in reverse order  If the signal is of enumerated type, then:  \_ d’VAL(pos): Returns value in the position specified  \_ d’POS(value): Returns position of the value specified  \_ d’LEFTOF(value): Returns value in the position to the left of the value specified  \_ d’VAL(row, column): Returns value in the position specified; etc. |
| Signal Attributes | \_ s’EVENT: Returns true when an event occurs on s  \_ s’STABLE: Returns true if no event has occurred on s  \_ s’ACTIVE: Returns true if s = ‘1’  \_ s’QUIET <time>: Returns true if no event has occurred during the time specified  \_ s’LAST\_EVENT: Returns the time elapsed since last event  \_ s’LAST\_ACTIVE: Returns the time elapsed since last s = ‘1’  \_ s’LAST\_VALUE: Returns the value of s before the last event; etc. |
| User-Defined Attributes | ATTRIBUTE attribute\_name: attribute\_type;  ATTRIBUTE attribute\_name OF target\_name: class IS value;  ATTRIBUTE number\_of\_pins: INTEGER; -- declaration  ATTRIBUTE number\_of\_pins OF nand3: SIGNAL IS 3; -- specification  ...  inputs <= nand3'number\_of\_pins; -- attribute call, returns 3  TYPE color IS (red, green, blue, white);  ATTRIBUTE enum\_encoding OF color: TYPE IS "11 00 10 01"; |
| Operator Overloading | FUNCTION "+" (a: INTEGER, b: BIT) RETURN INTEGER IS  BEGIN  IF (b='1') THEN RETURN a+1;  ELSE RETURN a;  END IF;  END "+"; |
| GENERIC | GENERIC (parameter\_name : parameter\_type := parameter\_value);  GENERIC (n: INTEGER := 8; vector: BIT\_VECTOR := "00001111"); |
| Concurrent Code | Operators;  The WHEN statement (WHEN/ELSE or WITH/SELECT/WHEN);  The GENERATE statement;  The BLOCK statement. |
| WHEN / ELSE | assignment WHEN condition ELSE  assignment WHEN condition ELSE  ...;  outp <= "000" WHEN (inp='0' OR reset='1') ELSE  "001" WHEN ctl='1' ELSE  "010"; |
| WITH / SELECT / WHEN | WITH identifier SELECT  assignment WHEN value,  assignment WHEN value,  ...;  WITH control SELECT  output <= "000" WHEN reset,  "111" WHEN set,  UNAFFECTED WHEN OTHERS; |
|  | WHEN value -- single value  WHEN value1 to value2 -- range, for enumerated data types only  WHEN value1 | value2 |... -- value1 or value2 or ... |
| GENERATE  FOR / GENERATE | label: FOR identifier IN range GENERATE  (concurrent assignments)  END GENERATE;  G1: FOR i IN x'RANGE GENERATE  z(i) <= x(i) AND y(i+8);  END GENERATE; |
| IF / GENERATE nested inside  FOR /GENERATE | label1: FOR identifier IN range GENERATE  ...  label2: IF condition GENERATE  (concurrent assignments)  END GENERATE;  ...  END GENERATE; |
| BLOCK  Simple BLOCK | label: BLOCK  [declarative part]  BEGIN  (concurrent statements)  END BLOCK label; |
| Nested BLOCK | label1: BLOCK  [declarative part of top block]  BEGIN  [concurrent statements of top block]  label2: BLOCK  [declarative part nested block]  BEGIN  (concurrent statements of nested block)  END BLOCK label2;  [more concurrent statements of top block]  END BLOCK label1; |
| Guarded BLOCK | label: BLOCK (guard expression)  [declarative part]  BEGIN  (concurrent guarded and unguarded statements)  END BLOCK label; |
| Sequential Code | IF, WAIT, CASE, and LOOP. |
| PROCESS | [label:] PROCESS (sensitivity list)  [VARIABLE name type [range] [:= initial\_value;]]  BEGIN  (sequential code)  END PROCESS [label]; |
| IF | IF conditions THEN assignments;  ELSIF conditions THEN assignments;  ...  ELSE assignments;  END IF; |
| WAIT | WAIT UNTIL signal\_condition;  WAIT ON signal1 [, signal2, ... ];  WAIT FOR time;  PROCESS -- no sensitivity list  BEGIN  WAIT UNTIL (clk'EVENT AND clk='1');  IF (rst='1') THEN  output <= "00000000";  ELSIF (clk'EVENT AND clk='1') THEN  output <= input;  END IF;  END PROCESS; |
| CASE | CASE identifier IS  WHEN value => assignments;  WHEN value => assignments;  ...  END CASE;  WHEN value -- single value  WHEN value1 to value2 -- range, for enumerated data types only  WHEN value1 | value2 |... -- value1 or value2 or ...  CASE control IS  WHEN "00" => x<=a; y<=b;  WHEN "01" => x<=b; y<=c;  WHEN OTHERS => x<="0000"; y<="ZZZZ";  END CASE; |
| LOOP |  |
| FOR / LOOP | [label:] FOR identifier IN range LOOP  (sequential statements)  END LOOP [label];  FOR i IN 0 TO 5 LOOP  x(i) <= enable AND w(i+2);  y(0, i) <= w(i);  END LOOP; |
| WHILE / LOOP | [label:] WHILE condition LOOP  (sequential statements)  END LOOP [label]; |
| EXIT | [label:] EXIT [label] [WHEN condition]; |
| NEXT | [label:] NEXT [loop\_label] [WHEN condition]; |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Simulation Software:

1. Active HDL from Aldec
2. ModelSim from Mentor
3. IUS from Cadance
4. VCS from Synopsys
5. SystemC

Verilog

Module myXor (A, B, C);

Input A,B;

Output C;

Xor x(A, B, C); or assign C = A ^ B; // assign used for combinational circuit

EndModule

Module myTest (A,B,C,O)

Input A,B,C;

Output O;

Assign O = ( B & C ) | ( ~A ) ;

EndModule;

Assign out= (s) ? in1 : in2 ; // procedural continuous assignment

Assign out = (s==0) ? in1:

(s==1) ? in2 :

(s==2) ? in3: in4;

جلسه 9

Generic parameters

`define W 32;

Module mux (a,b,c,s); parameter w=3; input [w-1:0] a; ….

Mux #(5) myMux (.a ……)

Mux myMux(…..) defparam myMux.W=5;

جلسه 10

Case (variable)

2’b00 : r<=1;

2’b01: r<=2;

Default: r<=3;

endcase

MegaWing Hex to SevenSegment convert

|  |  |
| --- | --- |
| 0 | C0 |
| 1 | F9 |
| 2 | A4 |
| 3 | B0 |
| 4 | 99 |
| 5 | 92 |
| 6 | 82 |
| 7 | D8 |
| 8 | 80 |
| 9 | 90 |