3.1 Major Language Constructs

- 3.1.1 Design Entities
  - A design entity consists of
    - An interface description (________ in VHDL)
    - One or more internals (________ in VHDL)

```vhdl
entity ONES_CNT is
  port (A: in BIT_VECTOR(2 downto 0);
        C: out BIT_VECTOR(1 downto 0));

  ------ Truth table:
  ---|A2   A1   A0  |  C1  C0 |
  --------| A2 | A1 | A0 | C1 | C0 |
  -----------|-----|-----|----|----|----|
  --- |0    0   0  |  0   0 |
  --- |0    0   1  |  0   1 |
  --- |0    1   0  |  0   1 |
  --- |0    1   1  |  1   0 |
  --- |1    0   0  |  0   1 |
  --- |1    0   1  |  1   0 |
  --- |1    1   0  |  1   0 |
  --- |1    1   1  |  1   1 |
  --------

end ONES_CNT;
```
### 3.1.2 Architectural Bodies - algorithmic

```
architecture ALGORITHMIC of ONES_CNT is
begin
  process(A)
  variable NUM: INTEGER range 0 to 3;
  begin
    NUM := 0;
    for I in 0 to 2 loop
      if A(I) = '1' then
        NUM := NUM + 1;
      end if;
    end loop;
    case NUM is
      when 0 => C <= "00";
      when 1 => C <= "01";
      when 2 => C <= "10";
      when 3 => C <= "11";
    end case;
  end process;
end ALGORITHMIC;
```

### 3.1.2 Architectural Bodies - dataflow

- Note: Unlike C, C++, in VHDL, and, or have equal precedence.

```
architecture DATA_FLOW of ONES_CNT is
begin
  C(1) <= (A(1) and A(0)) or (A(2) and A(0))
    or (A(2) and A(1));
  C(0) <= (A(2) and not A(1) and not A(0))
    or (not A(2) and not A(1)and A(0))
    or (A(2) and A(1) and A(0))
    or (not A(2) and A(1) and not A(0));
end DATA_FLOW;
```
3.1.2 Architectural Bodies - structural (lowest level)

```vhdl
use work.all;
entity AND2 is
  port (I1, I2: in BIT; O: out BIT);
end AND2;
architecture BEHAVIOR of AND2 is
begin
  O <= I1 and I2;
end BEHAVIOR;

use work.all;
entity OR3 is
  port (I1, I2, I3: in BIT; O: out BIT);
end OR3;
architecture BEHAVIOR of OR3 is
begin
  O <= I1 or I2 or I3;
end BEHAVIOR;

use work.all;
entity XOR2 is
  port (I1, I2: in BIT; O: out BIT);
end XOR2;
architecture BEHAVIOR of XOR2 is
begin
  O <= I1 xor I2;
end BEHAVIOR;
```

3.1.2 Architectural Bodies - structural (intermediate level - MAJ3)

```vhdl
use work.all;
entity MAJ3 is
  port (X: in BIT_VECTOR(2 downto 0); Z: out BIT);
end MAJ3;
architecture AND_OR of MAJ3 is
  component AND2C
    port (I1, I2: in BIT; O: out BIT);
  end component;
  component OR3C
    port (I1, I2, I3: in BIT; O: out BIT);
  end component;
  for all: AND2C use entity AND2(BEHAVIOR);
  for all: OR3C use entity OR3(BEHAVIOR);
  signal A1, A2, A3: BIT;
begin
  G1: AND2C port map (X(0), X(1), A1);
  G2: AND2C port map (X(0), X(2), A2);
  G3: AND2C port map (X(1), X(2), A3);
  G4: OR3C port map (A1, A2, A3, Z);
end AND_OR;
```
3.1.2 Architectural Bodies - structural (intermediate level - OPAR)

```vhdl
use work.all;
entity OPAR3 is
  port (X: in BIT_VECTOR(2 downto 0); Z: out BIT);
end OPAR3;
architecture STRUCT of OPAR3 is
  component XOR2C
    port (I1,I2: in BIT; O: out BIT);
  end component;
  for all: XOR2C use entity XOR2(BEHAVIOR);
signal A1,A2,A3: BIT;
begin
  G1: XOR2C port map (X(0),X(1),A1);
  G2: XOR2C port map (A1,X(2),Z);
end STRUCT;
```

3.1.2 Architectural Bodies - structural (highest level)

```vhdl
use work.all;
architecture STRUCTURAL of ONES_CNT is
  component MAJ3C
    port (X: in BIT_VECTOR(2 downto 0); Z: out BIT);
  end component;
  component OPAR3C
    port (X: in BIT_VECTOR(2 downto 0); Z: out BIT);
  end component;
  for all: MAJ3C use entity MAJ3(AND_OR);
  for all: OPAR3C use entity OPAR3(STRUCT);
begin
  COMPONENT_1: MAJ3C port map (A,C(1));
  COMPONENT_2: OPAR3C port map (A,C(0));
end STRUCTURAL;
```
3.1.3 Model Testing

- We use a testbench. There are two philosophies

a. ____________________________________
b. ____________________________________

---

3.1.3 Model Testing - Ones Count Example (algorithmic)

```vhdl
tenity TEST_BENCH is
end TEST_BENCH;

use WORK.all;
architecture ONES_CNT1 of TEST_BENCH is
    signal A: BIT_VECTOR(2 downto 0);  ----Declare signals
    signal C: BIT_VECTOR(1 downto 0);
    component ONES_CNTA               ----Declare component
        port (A: in BIT_VECTOR(2 downto 0);
        C: out BIT_VECTOR(1 downto 0));
    end component;

    for L1: ONES_CNTA use entity ONES_CNT(ALGORITHMIC);
    begin
        L1: ONES_CNTA port map(A, C);
        process
            begin
                A <= "000" after 1 ns, "001" after 2 ns,
                    "010" after 3 ns, "011" after 4 ns,
                    "100" after 5 ns, "101" after 6 ns,
                    "110" after 7 ns, "111" after 8 ns;
                wait;
            end process;
        end process;
    end ONES_CNT1;
```
3.1.3 Model Testing -
Ones Count Example (structural)

entity TEST_BENCH is
end TEST_BENCH;

use WORK.all;
architecture ONES_CNT1 of TEST_BENCH is
signal A: BIT_VECTOR(2 downto 0);  ----Declare signals
signal C: BIT_VECTOR(1 downto 0);
component ONES_CNTA               ----Declare component
port (A: in BIT_VECTOR(2 downto 0);
     C: out BIT_VECTOR(1 downto 0));
end component;
for L1: ONES_CNTA use entity ONES_CNT(STRUCTURAL);
begin
L1: ONES_CNTA port map(A, C);
process
begin
A <= "000" after 1 ns, "001" after 2 ns, "010" after 3 ns, "011" after 4 ns, "100" after 5 ns, "101" after 6 ns, "110" after 7 ns, "111" after 8 ns;
wait;
end process;
end ONESS_CNT1;

3.1.3 Algorithmic Ones Count
Simulation Results

VHDL Report Generator

<table>
<thead>
<tr>
<th>TIME (ns)</th>
<th>A (2 downto 0)</th>
<th>C (1 downto 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;000&quot;</td>
<td>&quot;00&quot;</td>
</tr>
<tr>
<td>2</td>
<td>&quot;001&quot;</td>
<td>----</td>
</tr>
<tr>
<td>+1</td>
<td>----</td>
<td>&quot;01&quot;</td>
</tr>
<tr>
<td>3</td>
<td>&quot;010&quot;</td>
<td>----</td>
</tr>
<tr>
<td>4</td>
<td>&quot;011&quot;</td>
<td>----</td>
</tr>
<tr>
<td>+1</td>
<td>----</td>
<td>&quot;10&quot;</td>
</tr>
<tr>
<td>5</td>
<td>&quot;100&quot;</td>
<td>----</td>
</tr>
<tr>
<td>+1</td>
<td>----</td>
<td>&quot;01&quot;</td>
</tr>
<tr>
<td>6</td>
<td>&quot;101&quot;</td>
<td>----</td>
</tr>
<tr>
<td>+1</td>
<td>----</td>
<td>&quot;10&quot;</td>
</tr>
<tr>
<td>7</td>
<td>&quot;110&quot;</td>
<td>----</td>
</tr>
<tr>
<td>8</td>
<td>&quot;111&quot;</td>
<td>----</td>
</tr>
<tr>
<td>+1</td>
<td>----</td>
<td>&quot;11&quot;</td>
</tr>
</tbody>
</table>
3.1.4 Block Statements

- Blocks in VHDL have their own scope (visibility), may have declarative statements and executable statements, and may be guarded.
- There is a correspondence between sequential processes and concurrent blocks.
- Guarded statements are concurrent statements which are evaluated whenever a signal
  - ________________ or
  - ________________.

```
entity GUARD_EXAMP is
  port(I1, I2, CON: in BIT; O1, O2: out BIT);
end GUARD_EXAMP;

architecture DF of GUARD_EXAMP is
begin
  B: block (CON = '1')
  begin
    O1 <= guarded I1;
    O2 <= I2;
  end block B;
end DF;
```

3.1.4 Block Statements - Nested Blocks

```
arithmetic BLOCK_STRUCTURED of SYSTEM is
  ------------------ Outer Block Declaration Section
  ------------------
  Begin
  -------------- Outer Block Executable Statements
  A: block
  -------------- Inner Block A Declaration Section
  begin
  -------------- Inner Block A Executable Statements
  end block A;
  B: block
  -------------- Inner Block B Declaration Section
  begin
  -------------- Inner Block B Executable Statements
  end block B;
end BLOCK_STRUCTURED;
```
3.1.5 Processes

- Every ______ has an equivalent _______. Many ___________ have an equivalent _______.
- Equivalent Process for Guarded Block Example

3.2 Lexical Description

- Create text files only.
- All VHDL statements are terminated by a semi-colon.
- An identifier is any sequence of characters that starts with a letter and includes only _______, ______, and single _______ characters. The last character may not be an _______.
- Comments start with -- and go to the end of a line.
- ‘A’, ‘e’, and ‘I’ are examples of character literals.
- String literals are delimited by “. Use two of them to include one in a string.
- Bit string literals can be specified in binary, ______, octal, and hexadecimal _______.
- There is a difference between _____ and _____.
- No lexical element may be split between two lines.
3.4 Data Types

- A data type is a named set of values.
- A subtype is a ______________ base type
- A base type can have ______________ subtypes.

Types
  - File
  - Access
  - Composite
  - Scalar
    - Enumerated
    - Real
    - Integer
  - Array
  - Record
  - Physical

3.4.2 Scalar Data Types - Enumeration Types

- Bit, Boolean, Severity Level, and Character are examples of built-in enumeration types.
- User-defined enumeration type examples include:
- There are predefined attributes and user-defined attributes. User-defined attributes are often used by synthesis and layout tools. Example:
3.4.2 Scalar Data Types - Numeric Data Types

- The two numeric types are integer and real.
- Examples:

- Note: COUNTER and REG are not the same type, even though they have the same range.
- The analyzer doesn’t coerce, it complains.

3.4.3 Composite Data Types - Arrays

- Two predefined arrays
  - 
  - 
- Array examples

- Array attributes
3.4.3 Composite Data Types - Records, Access Types, File Types

- Record examples
- Access types
- File types
- Type marks

3.5.1 Data Objects

- Classes of Objects
  - _________________
  - _________________
  - _________________
- Variables
  - _________________
  - _________________
3.5.2 Declaration of Data Objects

- Declaration of Constants

- Declaration of Variables
  - Initial values can be specified in the ____________, these take place at ____________ or a simulation time of _______. If no initial value is given, the variable gets the _____________.

- Declaration of Signals
  - Declarations occur as _______ of entities or in ____________ regions.
3.6 Language Statements

- Two classes
  - ______________________________
  - _____________________________
- Variable Assignment Statements
  - ____________________
- Signal Assignment Statements
  - A value is scheduled to occur, the operator is _____.

- Signal Drivers
  - In each process which makes an assignment to a signal X, a driver for X is created. Multiple signal assignments within a single process cannot create _________________.
  - When multiple drivers exist, a _____________________ is needed.
- Signal Attributes
3.6 Language Statements

- **Operators and Expressions**
  - Logical Operators
    - `and`, `or`, `nand`, `nor`, `xor`
  - Relational Operators
    - `=`, `=/=`, `<`, `<=`, `>`, `>=`
  - Adding Operators
    - `+`, `-`, `&`
  - Signing Operators
    - `+`, `-`
  - Multiplying Operators
    - `*`, `/`, `Mod`, `Rem`
  - Miscellaneous Operators
    - `**`, `abs`, `not`

3.6.3 Sequential Control Statements

**Wait Statement**

```
wait on X, Y until Z = 0 for 100 ns;
```

**If Statement**

```
if CONDITION1 then
  ---sequence of statements 1
elsif CONDITION2 then
  -- sequence of statements 2
  ---any number of elsif clauses
else
  -- last sequence of statements
end if;
```
3.6.3 Sequential Control Statements

**Case Statement**

```
case EXPRESSION is
  when CHOICES1 => -- sequence of statements 1
  when CHOICES2 => -- sequence of statements 2
  .
  .
  .
  when OTHERS => -- last sequence of statements
end case;
```

**Loop Statement**

```
for NAME in RANGE loop    while CONDITION loop
  -- statements -- statements
end loop;                 end loop;
```

**Next Statement**

```
next [loop_label] [when CONDITION];
```

**Exit Statement**

```
exit [loop_label] [when CONDITION];
```

**Null Statement**

```
null;
```
3.6.4 Architecture Declarations and Concurrent Statements

Architecture Declaration

architecture ARCHITECTURE_NAME of ENTITY_NAME is
  -- architecture declaration section
  -- signals are declared here
  -- variables may not be declared here
begin
  --
  -- Concurrent statements that describe signal behavior
  --
end ARCHITECTURE_NAME;

Process Statement (with sensitivity list)

process (SENSITIVITY_SIGNAL_LIST)
  -- constant declarations
  -- variable declarations
  -- subprogram declarations
  -- signal declarations are NOT permitted here
begin
  -- sequential statements
end process [LABEL];

Note: Variables declared within a process are static. They are initialized only once at the beginning of simulation and retain their values between process activations.
3.6.4 Architecture Declarations and Concurrent Statements

Process Statement (no sensitivity list)

[LABEL:] process
-- constant declarations
-- variable declarations
-- subprogram declarations
-- signal declarations are NOT permitted here
begin
-- sequential statements
end process [LABEL];

Note: This kind of process continues to execute until a wait statement is encountered.

Equivalent Process Statements

S_LIST: process (S1, S2)
-- constant declarations
-- variable declarations
begin
-- sequential statements
end process S_LIST;

NO_LIST: process
-- constant declarations
-- variable declarations
begin
-- sequential statements
wait on S1, S2;
end process NO_LIST;
3.6.4 Architecture Declarations and Concurrent Statements

Concurrent Assert Statement
Asserts use negative logic, a message occurs if the expression is false.

Concurrent Signal Assignment Statement
These statements are evaluated whenever an event occurs on a signal which appears on the right side.
They can be conditional or selected.

LABEL: SIGNAL_NAME <= transport LABEL: with expression select
WAVEFORM1 when CONDITION1 else SIGNAL_NAME <= [transport]
WAVEFORM2 when CONDITION2 else WAVEFORM1 when CHOICE1,
. WAVEFORM2 when CHOICE2,
. WAVEFORMn when CONDITIONn else
. WAVEFORMq;
. WAVEFORMn when CHOICEn,
WAVEFORMq when others;

3.6.5 Subprograms

Functions
Formal parameters must have mode _____ and be _______ or _______. default is _______. ______ statements are not permitted in a function body.

architecture ALG of PULSE_GEN is
function INT_TO_BIN (INPUT : INTEGER; N : POSITIVE) return BIT_VECTOR is
variable FOUT: BIT_VECTOR(0 to N-1);
variable TEMP_A: INTEGER:= 0;
variable TEMP_B: INTEGER:= 0;
begin -- Begin function code.
TEMP_A := INPUT;
for I in N-1 downto 0 loop
TEMP_B := TEMP_A/(2**I);
TEMP_A := TEMP_A rem (2**I);
if (TEMP_B = 1) then FOUT(N-1-I) := '1';
else FOUT(N-1-I) := '0';
end if;
end loop;
return FOUT;
end INT_TO_BIN;
begin -- Begin architecture body
### 3.6.5 Subprograms

**Procedures**

Procedures can modify one or more of the input parameters

**modes** - , ,

**objects** - , ,

package stuff is

```vhdl
end stuff;
```

package body stuff is

```vhdl
procedure ADD(A,B: in BIT_VECTOR; CIN: in BIT; SUM: out BIT_VECTOR; COUT: out BIT) is

variable SUMV,AV,BV: BIT_VECTOR(A'LENGTH-1 downto 0);
variable CARRY: BIT;

begin

AV := A;
BV := B;
CARRY := CIN;
for I in 0 to SUMV'HIGH loop
SUMV(I) := AV(I) xor BV(I) xor CARRY;
CARRY := (AV(I) and BV(I)) or (AV(I) and CARRY) or (BV(I) and CARRY);
end loop;
COUT := CARRY;
SUM := SUMV;
end ADD;
end stuff;
```

### 3.7.1 Advanced Features of VHDL-

**Overloading**

, , , and can be

redeclared, or overloaded.

package morestuff is

```vhdl
type MVL4 is ('X', '0', '1', 'Z');
end morestuff;
```

package body morestuff is

```vhdl
function "and" (L, R: MVL4) return MVL4 is

type MVL4_TABLE is array (MVL4, MVL4) of MVL4; -- Declare a two-dimensional table type.
constant table_AND: MVL4_TABLE := -- truth table for "and" function

```

```vhdl
<table>
<thead>
<tr>
<th></th>
<th>X</th>
<th>0</th>
<th>1</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>'X'</td>
<td>'X'</td>
<td>'0'</td>
<td>'X'</td>
<td>'X'</td>
</tr>
<tr>
<td>'0'</td>
<td>'0'</td>
<td>'0'</td>
<td>'0'</td>
<td>'0'</td>
</tr>
<tr>
<td>'X'</td>
<td>'0'</td>
<td>'1'</td>
<td>'X'</td>
<td>'X'</td>
</tr>
<tr>
<td>'X'</td>
<td>'0'</td>
<td>'X'</td>
<td>'X'</td>
<td>'X'</td>
</tr>
</tbody>
</table>
```

```
begin

return table_AND(L, R);
end "and";
end morestuff;
```
3.7.1 Advanced Features of VHDL - Overloading

function INTVAL (VAL: MVL4_VECTOR) return INTEGER is
  variable SUM: INTEGER:= 0;
begin
  for N in VAL'LOW to VAL'HIGH loop
    assert not(VAL(N) = 'X' or VAL(N)='Z')
    report "INTVAL inputs not 0 or 1"
    severity WARNING;
    if  VAL(N) = '1' then
      SUM := SUM + (2**N);
    end if;
  end loop;
  return SUM;
end INTVAL;

function INTVAL(VAL:BIT_VECTOR) return INTEGER is
  variable SUM: INTEGER:=0;
begin
  for N in VAL'LOW to VAL'HIGH loop
    if  VAL(N) = '1' then
      SUM := SUM + (2**N);
    end if;
  end loop;
  return SUM;
end INTVAL;

library ieee;
use ieee.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_SIGNED.all;
entity ADD_OVERLOAD is
  port ( A, B, C: in  STD_LOGIC_VECTOR (7 downto 0);
         SUM: out STD_LOGIC_VECTOR (7 downto 0));
end ADD_OVERLOAD;
architecture PACK_SIGNED of ADD_OVERLOAD is
begin
  SUM <= A + B + C; --This is now 2's-
              --complement addition
end PACK_SIGNED;
### 3.7.2 Advanced Features of VHDL - Packages

```vhdl
package HANDY is
  subtype BITVECT3 is BIT_VECTOR (0 to 2);
  subtype BITVECT2 is BIT_VECTOR (0 to 1);
  function MAJ3 (X: BIT_VECTOR (0 to 2)) return BIT;

  ---- Other Declarations -------------------------

end HANDY;

package body HANDY is
  function MAJ3 (X: BIT_VECTOR (0 to 2)) return BIT is
    begin
      return (X(0) and X(1)) or (X(0) and X(2)) or (X(1) and X(2));
    end MAJ3;

  ---- Other subprogram declarations ---------------

end HANDY;
```

All or part of a package is made visible with a use statement. Ex:

**Packages available**

- STANDARD
- STD_LOGIC_1164
- STD_LOGIC_ARITH
- STD_LOGIC_SIGNED
- STD_LOGIC_UNSIGNED
- MATH_REAL
- MATH_COMPLEX

---

Page 40 of 50
package SIG is
  signal X: INTEGER := 1;
end SIG;

use work.SIG.all;

entity Y is
  signal X: INTEGER := 2;
end Y;

architecture Z of Y is
  signal Z1,Z2,Z3,Z4,Z5: INTEGER := 0;
  function R return INTEGER is
    variable X: INTEGER := 3;
  begin
    return X; -- Returns value of 3.
  end R;
begin
  B: block
    signal X: INTEGER := 4;
    signal Z6: INTEGER := 0;
  begin
    Z6 <= X + Y.X; -- Z6 = 6
  end block B;

  P1: process
    variable X: INTEGER :=5;
  begin
    Z5 <= X; -- Z5=5
    wait;
  end process;
  Z1 <= work.SIG.X; -- Z1=1
  Z2 <= X; -- Z2=2
  Z3 <= R; -- Z3=3
  Z4 <= B.X; -- Z4=4
end Z;

3.7.4 Advanced Features of VHDL - Libraries

Two types
  Work library
  Resource libraries

Libraries contain
  primary units - ________, ________, ______________
  secondary units - ______________, ______________

Libraries have
  logical names
  physical names
3.7.5 Advanced Features of VHDL - Configurations

Configurations bind instantiated components to a library model. They occur in two places __________, __________

architecture STRUCTURAL of TWO_CONSECUTIVE is
begin
  signal Y0,Y1,A0,A1: BIT := '0';
  signal NY0,NX: BIT := '1';
  signal ONE: BIT := '1';

  component EDGE_TRIGGERED_D
    port(CLK,D,NCLR: in BIT; Q,QN: out BIT);
  end component;
  for all: EDGE_TRIGGERED_D use entity work.EDGE_TRIG_D(BEHAVIOR);

  component INVG
    port(I: in BIT; O: out BIT);
  end component;
  for all: INVG use entity INV(BEHAVIOR);

  component AND3G
    port(I1,I2,I3: in BIT; O: out BIT);
  end component;
  for all: AND3G use entity AND3(BEHAVIOR);

  component OR2G
    port(I1,I2: in BIT; O: out BIT);
  end component;
  for all: OR2G use entity OR2(BEHAVIOR);

  Configurations bind instantiated components to a library model.

architecture PARTS of TWO_CONSECUTIVE is
begin
  for STRUCTURAL_a
    for all: EDGE_TRIGGERED_D
      use entity work.EDGE_TRIG_D(BEHAVIOR);
    end for;
    for all: INVG
      use entity work.INV(BEHAVIOR);
    end for;
    for all: AND3G
      use entity work.AND3(BEHAVIOR);
    end for;
    for all: OR2G
      use entity work.OR2(BEHAVIOR);
    end for;
  end for;
end PARTS;
3.7.6 Advanced Features of VHDL - File I/O

Two types of files

Formatted I/O must be written by a VHDL simulator

Text I/O

You must have a use clause

Reading and writing are two step processes

File -> line -> variable -> signal

Variable -> line -> file

entity OBVS is
  generic(N: INTEGER; PER: TIME);
  port(GEN: in BIT);
end OBVS;

use work.all;

architecture FIO of OBVS is

  type INP_COMB is file of BIT_VECTOR;
  file OUTVECT: INP_COMB is out "TEST.VECT";
  signal VECTORS: BIT_VECTOR(N-1 downto 0);
  signal SYNC: BIT;

  component PG
    generic(N: INTEGER; PER: TIME);
    port(START: in BIT;
         PGOUT: out BIT_VECTOR(N-1 downto 0);
         SYNC: inout BIT);
  end component;

  for C1: PG  use entity work.PULSE_GEN(ALG);

  begin
    C1: PG
      generic map(N => N, PER => PER)
      port map(START => GEN, PGOUT => VECTORS,
               SYNC => SYNC);

    WRITE_VECT: process(SYNC)
      variable V: BIT_VECTOR(N-1 downto 0);
      begin
        V := VECTORS; WRITE(OUTVECT,V);
      end process WRITE_VECT;
  end FIO;

entity PULSE_GEN is
  generic(N: INTEGER; PER: TIME);
  port(START: in BIT; PGOUT:out BIT_VECTOR(N-1 downto 0);
       SYNC:inout BIT);
end PULSE_GEN;

architecture ALG of PULSE_GEN is

  begin  -- Begin architecture body
    process(START,SYNC)
      variable CNT: INTEGER:= 0;
      begin  -- Begin process
        if START'EVENT and START='1' then
          CNT := 2**N-1;
        end if;
        PGOUT <=  INT_TO_BIN(CNT,N) after PER;
        if CNT /= -1 and START='1' then
          SYNC <= not SYNC after PER;
          CNT := CNT -1;
        end if;
      end process;
  end ALG;
### 3.7.6 Advanced Features of VHDL - File I/O

**Entity IBVS**

```vhdl
entity IBVS is
  generic(N:INTEGER;PER: TIME);
  port(PLAY: in BIT; BVOUT: out BIT_VECTOR(N-1 downto 0));
end IBVS;
architecture FIO of IBVS is
begin
  type INP_COMB is file of BIT_VECTOR;
  file INVECT: INP_COMB is in "TEST.VECT";
  process
  variable LENGTH: NATURAL := N;
  variable V: BIT_VECTOR(LENGTH-1 downto 0);
  begin
    wait on PLAY until PLAY = '1';
    loop
      exit when ENDFILE(INVECT);
      READ(INVECT,V,LENGTH);
      BVOUT <= V;
      wait for PER;
    end loop;
  end process;
end FIO;
```

**Entity TBVS**

```vhdl
entity TBVS is
  generic(N:INTEGER;PER: TIME);
  port(PLAY: in BIT; BVOUT: out BIT_VECTOR(N-1 downto 0));
end TBVS;
architecture TIO of TBVS is
begin
  process
  variable VLINE: LINE;
  variable V: BIT_VECTOR(N-1 downto 0);
  file INVECT: TEXT is "TVECT.TEXT";
  begin
    wait on PLAY until PLAY = '1';
    loop
      READLINE(INVECT,VLINE);
      READ(VLINE,V);
      BVOUT <= V;
      wait for PER;
    end loop;
  end process;
end TIO;
```