Announcement

- Project 2: Assigned today, due 9/30 beginning of class.
- Modeling a binary/Gray counter.
- As always, start early...

VHDL in Action

Chapter 3
Sequential Statements

Review

- Architecture body
  - Concurrent signal assignments
  - Component instantiations
  - Processes
- Testbenches
  - Portable way to validate a model
  - Stimulus only, or stimulus + validation

Statements

- Sequential statements
  - Inside process blocks
  - Similar to "normal" programming
- Concurrent statements
  - Outside of process blocks within an architecture body
  - Concurrent == all evaluated at the same time.

Sequential Statements

- Remember that statements within a PROCESS block are executed sequentially.
- Because of this, PROCESS code is similar to software (C/C++/Pascal/...)

```vhdl
process (sensitivity_list)
declaration_section
begin
  process_body
end process;
```

Process Execution with no sensitivity list.

1. Declarations elaborated
2. Body is executed
3. Repeat: return to BEGIN
4. Loop forever
Process Execution with Sensitivity Lists

1. Declarations elaborated
2. Body is executed
3. Loop to BEGIN
4. Wait until signals in sensitivity list change.

process (A, B, C)
  -- Variable Declarations
begin
  -- Sequential Statements
end process;

WAIT Statement

wait [on sensitivity_list] [until condition] [for time_expr];
- Suspends the enclosing process.
- Example:
  process
    variable A: Bit := '0';
    begin
      wait for 100 ns;
      A := not A;
    end process;

Note on WAITs

- Two types of process:
  - with sensitivity list
  - without sensitivity list
- Process without sensitivity list must have WAIT statement(s) to prevent infinite loops
- Process with sensitivity list may NOT have WAIT statements.

Using WAITs

process
begin
  --sequential statement
  wait ...
  --sequential statement
  wait ...
  --sequential statement
  wait ...
  --sequential statement
  end process;
Comparison

- The following are equivalent:
  - process ( S1, S2 ) begin
    X1 <= S1 or S2;
    end process;
  - process begin
    X1 <= S1 or S2;
    wait on S1, S2;
    end process;

Relational Operators

- Express relationships between signals or variables:
  - =
  - /=
  - <
  - <=
  - >
  - >=

Sequential Statements

**IF Statement**

- Usage very similar to C/C++
  process (A)
  begin
    if (A = '1') then
      B <= C + 1;
    else
      C <= C - 1;
    end if;
  end process;

- More generally:
  if condition1 then
    sequential statement(s)1
  elsif condition2 then
    sequential statement(s)2
  elsif condition3 then
    sequential statement(s)3
  else
    sequential statement(s)4
  end if;

**CASE Statement**

- Similar to SWITCH/CASE in C/C++
  process (LIGHT_COLOR)
  begin
    case LIGHT_COLOR is
      when RED => TRAFFIC <= FALSE;
      when YELLOW => TRAFFIC <= TRUE;
      when GREEN => TRAFFIC <= TRUE; DONTWALK <= '1';
      when others => LIGHT_BROKEN <= TRUE;
    end case;
  end process;

- More generally:
  case discriminant is
    when choice1 => sequential_statements[1];
    when choice2 => sequential_statements[2];
    when choice3 => sequential_statements[3];
    o
    when others => sequential_statements[n];
  end case;

  **NOTE:** Choice conditions must be mutually exclusive and must include all possible values of discriminant.
**LOOP Statement**

- Similar to FOR and DO/WHILE statements in C/C++
  - process
  - begin
  - wait until clk = '1' and clk'event;
  - for i in 0 to 9
  - loop
  - a(i) <= a(i+1);
  - end loop;
  - end process;

- More generally:
  - label:
  - for parameter in discrete_range
  - loop
  - sequential statements
  - end loop;

**LOOP Statement**

- Another variation:
  - process
  - variable count : integer;
  - begin
  - sum <= 12; count := 0;
  - while sum <= A loop
  -   wait until clken = '1';
  -   count := count + 1;
  -   sum <= sum * 12;
  - end loop;
  - end process;

- More generally:
  - label:
  - while boolean_condition
  - loop
  - sequential statements
  - end loop;

**LOOP Statement**

- General Loop
  - label:
  - loop
  - sequential statements
  - end loop;

- NEXT statements can be used to modify the execution of a LOOP statement.
- Terminates execution of a particular loop iteration (skipping iterations).

- Forms:
  1. next;
  2. next when condition ;
  3. next label ;
  4. next label when condition ;
**NEXT Statement**

- **Example:**
  ```vhdl```
  ```
  process
  begin
  FRED: for i in 10 downto 0
  loop
  sequential_statements1
  next FRED when i = 3;
  sequential_statements2
  end loop;
  end process;
  ```
  ```
  ```
```

**EXIT Statement**

- **EXIT statements can be used to terminate the execution of a LOOP statement.**
- **Terminates all remaining iterations of a loop**
- **Forms:**
  1. `exit;`
  2. `exit when condition;`
  3. `exit label;`
  4. `exit label when condition;`

**EXIT Statement**

- **Example:**
  ```vhdl```
  ```
  process
  begin
  SAM: while DONE = false
  loop
  -- sequential_statements1
  exit when DONE = true;
  -- sequential_statements2
  end loop;
  end process;
  ```
  ```
  ```
```

**NULL Statement**

- **Does nothing -- filler**
  ```vhdl```
  ```
  process (SEL)
  begin
  case SEL is
  when 0 => decout <= "00";
  when 1 => decout <= "01";
  when 2 => decout <= "10";
  when 3 => decout <= "11";
  when others => null;
  end case;
  end process;
  ```
  ```
  ```
```

**Summary**

- **Sequential constructs**
  - Loops:
    - For/while
    - Case
  - If/then/elsif/else
  - Process execution
  - Sensitivity lists vs. wait statements
  - Next time: Concurrent constructs