Loops

Sequences of statements that are executed repeatedly.

Types of loops:

- **For (most common usage)**
- **While**
- **Loop with exit construct (we skip this)**

General Format:

```
[loop_label:]
iteration_scheme  --FOR, WHILE
LOOP
  --sequence_of_statements;
END LOOP[loop_label];
```

For Loop

Statements are executed once for each value in the loop parameter's range

Loop parameter is implicitly declared and may not be modified from within loop or used outside loop.

Format:

```
[label:] FOR loop_parameter IN discrete_range
LOOP
--sequential_statements

END LOOP[label];
```

Example:

```
PROCESS (ray_in)
BEGIN
  --connect wires in a two busses
  label: FOR index IN 0 TO 7
  LOOP
    ray_out(index) <= ray_in(index);
  END LOOP label;
END PROCESS;
```

While Loop

Execution of statements within loop is controlled by Boolean condition.

Condition is evaluated before each repetition of loop.

Format:

```
WHILE boolean_expression
LOOP
--sequential_expressions
END LOOP;
```

Example:

```
p1:
PROCESS (ray_in)
  VARIABLE index : integer := 0;
BEGIN
  from_in_to_out:
  WHILE index < 8
  LOOP
    ray_out(index) <= ray_in(index);
    index := index + 1;
  END LOOP from_in_to_out;
END PROCESS p1;
```

Loop example creating a shift register

```
---------------------------------------------------------------------
--Shift register
---------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_vector_arith.ALL;
--

ENTITY shift_reg IS
  PORT(
      clk           : IN      STD_LOGIC;  --input clock
      reset_n       : IN      STD_LOGIC;  --reset async active low
      data_ena      : IN      STD_LOGIC;  --serial data enable
      serial_data   : IN      STD_LOGIC;  --serial data input
      parallel_data : BUFFER  STD_LOGIC_VECTOR(7 DOWNTO 0) --data out
      );
  END shift_reg;


 ARCHITECTURE beh OF shift_reg IS
 BEGIN
 shift_register:
 PROCESS (clk, reset_n, data_ena, serial_data, parallel_data)
   BEGIN
     --clocked part
     IF (reset_n = ‘0’) THEN
       parallel_data <= “00000000”;
     ELSIF (clk’EVENT AND clk = ‘1’) THEN
       IF (data_ena = ‘1’) THEN
         parallel_data(7) <= serial_data;         --get input data
         FOR i IN 0 TO 6 LOOP
          parallel_data(i) <= parallel_data(i+1); --shift other bits
         END LOOP;
       END IF;
     END IF;
   END PROCESS shift_register;
END beh;
```