Being more productive in VHDL: Generate Statements and Process Statements

Madhav Desai, Sachin Patkar

February 26, 2019

VHDL thus far

- We have introduced the following concepts
  - Concurrent assignments (describe equations).
  - Component instantiation (re-use gates).
- There are a couple of constructs that make things much easier to express.
  - Generate statements: allow us to instantiate many drivers in a compact way (what is a driver?).
  - Procedural statements: allow us to express complex behaviour of a driver using a sequential programming model.

What is a driver?

- A driver is an object which **writes** a value on a signal/port.
- A concurrent assignment is a driver.
- A component instance introduces a driver for each output port of the component instance.

Simple Drivers: Concurrent assignments

```
Sum <= A xor B xor Cin;
```

- This is a driver which is evaluated whenever there is an event on either A,B or Cin.
- A value is computed using the right-hand-side equation.
- This value is driven onto the signal/port Sum after a delay. If not specified, this delay can be thought of as infinitesimally small, positive number (strictly greater than 0, call it 0^+ or δ).

A tedious way to describe a 32-bit adder

```
C(1)   <= ((A(0) or B(0)) and Cin) or
                               (A(0) and B(0));
Sum(0) <= A(0) xor B(0) xor Cin;
C(2)   <= ((A(1) or B(1)) and C(1)) or
                               (A(1) and B(1));
Sum(1) <= A(1) xor B(1) xor C(1);
C(3)   <= ((A(2) or B(2)) and C(2)) or
                                (A(2) and B(2));
Sum(2) <= A(2) xor B(2) xor C(2);
...
Cout   <= ((A(31) or B(31)) and C(31)) or
                               (A(31) and B(31));
Sum(31) <= A(31) xor B(31) xor C(31);
```

The 32-bit adder using for-generates

```
-- A,B std_logic_vector(31 downto 0);
-- C std_logic_vector (32 downto 0);
C(0) <= Cin;

genAdd: for I in 0 to 31 generate
   C(I+1)   <= ((A(I) or B(I)) and C(I)) or
                                (A(I) and B(I));
   SUM(I) <= A(I) xor B(I) xor C(I);
end generate genAdd;

Cout <= C(32);
```

What is a for-generate?

- Like a **macro** in C/C++.
- Works at the pre-processor level and instantiates code with the generate index substituted over its specified range.

Another useful construct: if-generate

```
-- A std_logic_vector(M-1 downto 0);
-- B std_logic_vector(N-1 downto 0);
mGeqn: if (M >= N) generate
   -- B takes an N-section of A.
   B <= A(N-1 downto 0);
end generate mGeqn;

nGtm: if (M < N) generate
   -- B takes A and is padded
   B(M-1 downto 0)  <= A;
   B(N-1 downto M)  <= (others => '0');
end generate nGtm;
```

Process statements

```
      -- A,B std_logic_vector(31 downto 0);
   process(A,B,Cin)
     variable Cvar : std_logic;
   begin
       -- note this assignment.. it takes effect
       -- immediately.
       Cvar := Cin;
       for I in 0 to 31 loop
           -- this assignment takes effect only
           -- after a delta-delay.
           SUM(I) <= A(I) xor B(I) xor Cvar;
           -- immediate.
           Cvar   := ((A(I) or B(I)) and Cvar) or
                                   (A(I) and B(I));
       end loop;
       -- after delta
       Cout <= Cvar;
   end process;
```

Driver described by a process statements

- Evaluated whenever there is an event on a signal in the sensitivity list of the process statement.
- The process is executed as a sequential program until the end of the process is reached (the process then waits on the next event in the sensitivity list).
- Along the way, values being driven onto signals will get updated.

VHDL Simulation Algorithm

```
At t=0, evaluate all drivers (in any order),
put all transactions in a transaction queue.
```

VHDL Simulation Algorithm: continued

```
while (transaction-queue is non-empty) {

   move time forward to that corresponding
   to the transaction at the head of the queue.

   apply all transactions active at this
   time (in any order).

   execute all drivers which are triggered
   by events as a result of the applied transactions.
   Add resulting transactions to transaction-queue.
}
```

Observations

- The VHDL simulation algorithm is unambiguous.
- Drivers can be defined in any order, they are concurrent (that is, their behaviour at a particular instant is independent of each other).
- The algorithm attempts to mimic *ideal* gates which have a delay δ.
- Delays can be associated with a driver:

```
w <= (y0 and x0) after 5 ns;
```

VHDL Simulation Algorithm: example

```
entity Top is
  port (c: out bit);
end entity Top;
architecture Absurd of Top is
  signal a, b: bit;
begin
  a <= not a after 5 ns;
  b <= not a;
  c <= a or b;
end Absurd;
```

Recap

Do we understand the following concepts?

- Driver.
- Generate statements (for and if).
- Process statements.
- Transaction and events.
- Simulation Algorithm.

Process statements with WAIT

This is a special kind of driver, which is very useful in constructing test-benches. As an example:

```
process
begin
   a <= '1';
   wait for 5 ns;
   a <= '0';
   wait for 5 ns;
end process;
```

This produces a sequence of transactions on the signal **a**.
In general, one can describe very complicated sequences using this construct (as an example, see the sample VHDL code).

Useful reading material

- D. Perry, “VHDL Programming by Example”, Tata McGraw-Hill.
- P. Ashenden, “A VHDL Tutorial”, hep.uchicago.edu/ tangjian/SVT_sub/FTK_ATLAS/.../vhdl-tutorial.pdf.