

Recursive VHDL structures in FPGA synthesis By Rangarajan Sundaravaradan

New standards in VHDL-93 permit the writing of VHDL components that instantiate themselves. Implementation of recursive structures leads to designs easily pipelined by the addition of registers at the outputs of the recursively instantiated components. In general, the recursive structures are easier to develop and can be expressed clearly, making them easier to understand. This article explores practical applications of recursive structures in FPGA synthesis.

Most hardware structures that are designed consist of a number of instances of a basic component interconnected in a regular pattern. In order to describe such repetitive structures, VHDL provides a mechanism called the generate statement. The language permits both conditional and repetitive forms of generate statement. A recursive structure is one, which is parameterized with respect to its size and is described in terms of smaller instances of the same structure. The examples in this article cover recursion using subprograms and entities.

The Wide XOR component

XOR component using a simple function:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity XOR reduce is
  generic (N: natural := 5);
  port (data in: in std logic vector(N-1
downto 0);
        data_out: out std_logic);
end XOR reduce;
architecture one of XOR reduce is
  function
XOR reduce func(data:std logic vector)retu
rn std logic is
    variable result : std logic;
  begin
    result := '0';
    for I in data'RANGE loop
      result := result XOR data(I);
    end loop;
  return result;
end;
begin
  data out <= XOR reduce func(data in);</pre>
end one;
```

XOR component using a recursive function:

```
end XOR tree;
architecture one of XOR tree is
  function XOR_tree_func(data:
std_logic_vector) return std_logic is
    variable UPPER TREE, LOWER TREE: std logic;
    variable MID, LEN: natural;
    variable result: std logic;
    variable i data:
std_logic_vector(data'LENGTH-1 downto 0);
begin
    i_data := data;
    LEN := i data'LENGTH;
      if LEN = 1 then
        result := i_data(i_data'LEFT);
      elsif LEN = 2 then
        result := i data(i_data'LEFT) XOR
i data(i data'RIGHT);
      else
        MID := (LEN + 1)/2 + i_data'RIGHT;
        UPPER TREE :=
XOR tree func (i data (i data 'LEFT downto MID));
        LOWER TREE := XOR tree func(i data(MID-
1 downto i data'RIGHT));
        result := UPPER_TREE XOR LOWER_TREE;
      end if;
    return result;
end;
begin
  data out <= XOR tree func(data in);</pre>
```

Wide XOR using recursive components:

end one;

In this example the different levels of logic are pipelined.

```
entity xor gate is
      generic(pipeline_delay:natural:=0 ;);
       port (X : in std_logic_vector ;
                 q : out std logic ;
                 clk : in std logic ;
                 enable : in std logic
    end xor_gate ;
architecture xor_gate_a of xor_gate is
begin
a: if X'length < N generate
    --basic Xor gate component is instantiated
   here
    end generate a;
b: if X'length > N generate
constant Y: natural := (X' \text{length } -1)/N;
signal temp : std_logic_vector(0 to Y);
begin
  C : for I in 0 to Y-1 generate
    D: entity xor gate generic map
(pipeline_delay => pipeline_delay)
       Port map (X = X(N*I to (N*(I+1) -1)), q)
=> temp(i) ,
```

```
End xor_gate_a ;
```

In the above case the basic instance is parameterized to size N bits.

Wide XOR component targeted for ORCA2

Implementing the basic instance of the wide XOR component targeted to a Lucent ORCA@ device:

Each PFU in ORCA2 uses three input data buses (A[4:0], B[4:0], WD[3:0]), four control inputs (C0, CK, CE, LSR), and a carry input (CIN); the last is used for fast arithmetic functions. There is a 5-bit output bus (O[4:0]) and a carry-out (COUT) from the PFU.

 $A(4{:}0)$ and $B(4{:}0)$ are inputs to the lookup table and WD(4{:}0) are direct inputs to the flipflops.

Basic XOR gate instance can be constructed upto 11 bits in width using a(4:0),b(4:0) and the carry input(CIN).A single PFU can implement XOR gate upto 11 bits with 0 or 1 cycle of clock delay.

Construction of larger XOR functions require another component which instantiates the base 11 bit XOR gate as shown in the example above.

All trade names are trademarks of their respective vendors.

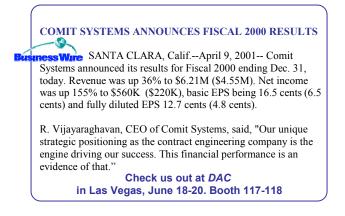
The COMIT logo and Design Advantage are registered trademarks of Comit Systems, Inc. Other Service Marks labeled as such.
 Copyright Comit Systems, Inc. 2001. All rights reserved.

The advantages with recursion are

- The component is broken into different levels of logic; pipelining the individual stages is easier.
- Timing can be improved by adopting recursive structures.
- Recursive structures are easier to develop.
- High performance in terms of speed and area can be achieved by adjusting the base component to fit the target architecture.

References.

- 1. Designer's Guide to VHDL Peter J. Ashenden
- 2. Recursive and Repetitive Hardware Models in VHDL Peter J. Ashenden
- 3. Vector Pipeline Library manual John McCluskey.







Verilog & VHDL Quick Reference Cards are now available for download to PDA! www.comit.com