Design Advantage™

Volume 2, Number 1 — The newsletter of Comit Systems, Inc. — MARCH 1999

Designing with Global Clock Buffers in FPGAs

Jaishankar Iyer

FPGAs have resources called Global Clock Buffers and the number of buffers varies from one FPGA technology to another. Global Clock Buffers may be used for high fanout signals like clock, clock enable preset logic and clear logic. Since Global Clock Buffers are dedicated resources in the FPGA architecture, it makes sense to utilize them fully.

In case we do not specifically assign Global Clock Buffers in our design, the software tools may automatically attach a Global Clock Buffer to any input port signal, which directly drives a clock pin. The software may limit the maximum number of Global Clock Buffers, which can be inserted into the synthesized netlist file, based on the architectural capabilities of the device.

In addition, some FPGA tools may allow us to assign Global Clock Buffers to high fanout signals in the constraints file which is passed on as input to the Place & Route tools. This is another way to achieve the same result.

There are two other ways that help you better control the assignment of Global Clock Buffers to the high fanout signals, as explained below :

1. By using attribue clock_buffer
2. By Instantiation

Using Attribute clock_buffer

It is possible to insert a Global Clock Buffer on any specified input port signal, regardless of whether it is a clock signal or not. Use the "clock_buffer" attribute as shown below in Example 1, to access this feature. This Global Clock Buffer replaces the input buffer and thus uses a dedicated global buffer pad. This attribute is attached to top-level port signals.

In the example below, a Global Clock Buffer is inserted on the CLR line.

Example 1

```
library ieee;
use ieee.std_logic_1164.all;
entity dff  is
    port (data  : in STD_LOGIC;
          clk   : in STD_LOGIC;
          clr   : in STD_LOGIC;
          q     : out STD_LOGIC) ;
    attribute clock_buffer : boolean;
    attribute clock_buffer of clr  : signal is
true;
end dff ;
architecture test  of dff is
begin  -- test
    process (clk,clr)
     begin
        if clr = '1' then
            q <='0';
         elsif clk'event and clk = '1' then
            q <= data;
        end if;
    end process;
end test ;
```

Instantiation

Global Clock Buffers may also be instantiated. This method is best suited to drive a Global Clock Buffer with an internally generated signal or to control the specific type of global buffer used. If, for instance, in an ORCA Series 3 design, we wish to specify the use of CLKCNTLB (bottom) clock controller, then we need to instantiate it as shown in Example 2 below.

Example 2

```
library ieee;
use ieee.std_logic_1164.all;
entity clkcnt is
       port( data   :in STD_LOGIC;
             ck_in  :in STD_LOGIC;
             ck_off :in STD_LOGIC;
             q      :out STD_LOGIC);
end clkcnt;
architecture test of clkcnt is
       signal clock :STD_LOGIC;
       component CLKCNTLB
               port( CLKIN: in STD_LOGIC;
                     SHUTOFF: in STD_LOGIC;
                      CLKOUT: out STD_LOGIC);
       end component;
begin
-- Component Instantiation
CLK_CNTL_BOTTOM: CLKCNTLB port map (CLKIN=>ck_in,
                 SHUTOFF=>ck_off, CLKOUT=>clock);

process(clock)
begin
       if(clock'event and clock='1') then
               out <= data;
       end if;
end process;
End test;
```

The VHDL code in ■ the above examples

Comit now designs AMBA compliant interfaces for on-chip digital peripherals!

Design Advantage™

The newsletter of:

COMIT® SYSTEMS

Comit Systems, Inc.
3375 Scott Blvd. Suite 330
Santa Clara, CA 95054
Phone : (408)-988-2224

Editor : Jaishankar Iyer
We welcome your comments and suggestions. Please email us at
jiyer@comit.com

Download White Paper:
Competitive Positioning for the New Millennium.
http://www.comit.com

can be compiled and synthesized.

Overcoming inconsistencies in SDF backannotation

Rohan Hubli

During backannotation using SDF files, we may come across situations where the same cell name is referenced differently in the HDL netlist file and the SDF file. This may happen due to differences in hierarchical separators, register names, net names etc.

```
  eg : U31dffrdx1_reg_1 -- VHDL/Verilog cell name
       U31dffrdx1reg_1   -- SDF cell name
```

The effect of this is an unsuccessful post-synthesis gate level simulation since the simulator cannot find the reference for the cell being addressed by the design.

We can, of course, write a shell script to identify differences in cell names in the netlist and SDF files and generate files acceptable to the simulator. This method is tedious as it requires manual intervention to identify differences in the files and fix them.

As good design practice, we can execute a set of variables before synthesis with Synopsys® Design Compiler™, which will nullify the inconsistencies in the netlist and SDF file.

For VHDL designs, the Synopsys® synthesis variable set will look like this :

```
 bus_naming_style="%s<%d>"
 default_name-rules = "sge_vhdl"
 change_names -hierarchy
 vhdlout_write_components = FALSE
 vhdlout_write_top_configuration = TRUE
 vhdlout_use_packages = { IEEE.std_logic_144 }
```

For Verilog designs, the Synopsys® synthesis variable set will look like this :

```
 bus_naming_style="%s<%d>"
 define_name_rules my_rules -allowed "A-Z _ 1-10"
                  -first_restricted "A-Z"
                  -restricted "!@#$%^&*()"
                  -last_resticted "\\-"
 change_names -rules my_rules -hierarchy.
```

This technique ■ ensures a smoother flow during