Clock Multiplication in Virtex-E and Virtex-II FPGAs

How to set up clock multiplication into Virtex-E and Virtex-II devices using VHDL or Verilog hardware description languages and Synplify synthesis software.

by Howard Walker
Technical Marketing Engineer, Xilinx
howardw@xilinx.com

CLKDLLs (Clock Delay-Locked Loops) circuits can be used in a variety of applications ranging from clock multiplication and division to phase shifting and clock deskewing. In this article, we will show you how to implement clock multiplications in both Virtex™-E and Virtex-II devices using either VHDL or Verilog™ code and Synplify™ logic synthesis software.

The CLKDLLs components on the Virtex-E device have four primary and four secondary CLKDLLs. The secondary CLKDLLs have dedicated feedback loops that can be used to generate a 4X clock using only one BUFG (global clock buffer). Virtex-E devices have four clock networks that can be used to create different clock configurations. The primary/secondary CLKDLLs must be located in the same quadrant (top right, top left, bottom right, or bottom left) to create a 4X clock.

The Virtex-II devices offer even more versatility than the Virtex-E FPGAs. The Virtex-II devices have from four to twelve DCMs (Digital Clock Managers) depending on the device size. The DCM not only offers enhanced support for all DLL functionality found in Virtex-E devices, but the DCM also provides flexible frequency synthesis, precise fine-grained phase shifting, and spread spectrum clock generation. Virtex-II devices contain 16 global clock multiplexers and buffers that can be used with the DCMs to create different clock configurations. A DCM with a BUFG and an IBUFG (dedicated input clock pad) can be used to generate a 7X clock.

Set Up a 4X Clock in a Virtex-E FPGA

The VHDL and Verilog™ code examples on the next page show how to implement a 4X clock for use in a Virtex-E device. The circuit includes two CLKDLLs (one primary and one secondary), an IBUFG, a SRL16 (shift register look-up table) and inverter for the reset control signal, and a BUFG used for the feedback from the primary CLKDLL.

The SRL16 is used to delay the reset signal to the second CLKDLL (the primary DLL), so that it will stay in reset until the first CLKDLL (the secondary DLL) has achieved a "lock" on the input clock. This insures that the second CLKDLL is sourcing a stable clock from the first CLKDLL to produce a clock signal that is 4X the frequency of the CLKIN (clock input) signal.

In all the examples in this article, we used Synplify logic synthesis software from Synplicity™ Inc.

If location constraints are required for the CLKDLLs and the associated input clock, these can be specified in a UCF (User Constraints File). Only one of the CLKDLLs or its associated global clock buffer or clock pad needs to be given a location constraint in the UCF. The Xilinx 3.1i Alliance Series™ software (or later release) will automatically place the other CLKDLL, IBUFG, or BUFG in the same quadrant (lower right, lower left, upper right, or upper left). The following constraints will place the CLKDLLs in the lower left quadrant of a Virtex-E device:

```
#UCF with LOCs for DLLs, IBUFG
and BUFG for dll_4x example

inst dll2x LOC = DLL1S;

inst dll4x LOC = DLL1P;
```

Virtex-E DLL 4X VHDL Example

```
library ieee;
use ieee.std_logic_1164.all;
—Include your vendor-specific library references for running synthesis
software here

entity dll_standard is
      port (CLKIN, RESET : in  std_logic;
      CLK2X, CLK4X, LOCKED : out std_logic);
end dll_standard;

architecture structural of dll_standard is


component IBUFG
      port (O : out std_logic;
      I : in std_logic);
end component;

component CLKDLL
      port(CLKIN, CLKFB, RST: in std_logic;
      CLK0, CLK90, CLK180, CLK270, CLK2X, CLKDV, LOCKED : out
      std_logic);
end component;

signal CLKIN_w, RESET_w, CLK2X_dll, CLK2X_g, CLK4X_dll,
CLK4X_g : std_logic;
signal LOCKED2X, LOCKED2X_delay, RESET4X: std_logic;
signal logic1 : std_logic;

begin

      logic1 <= '1';

      clkpad : IBUFG  port map (I=>CLKIN, O=>CLKIN_w);

      rstpad : IBUF   port map (I=>RESET, O=>RESET_w);

      dll2x  : CLKDLL port map (CLKIN=>CLKIN_w,
            CLKFB=>CLK2X_g,
            RST=>RESET_w, CLK0=>open, CLK90=>open,
            CLK180=>open,
            CLK270=>open, CLK2X=>CLK2X_dll, CLKDV=>open,
            LOCKED=>LOCKED2X);

      clk2xg : BUFG   port map (I=>CLK2X_dll,   O=>CLK2X_g);

      rstsrl :      SRL16  port map (D=>LOCKED2X, CLK=>CLK2X_g,
                  Q=>LOCKED2X_delay, A3=>logic1, A2=>logic1,
                  A1=>logic1, A0=>logic1);

      RESET4X <= not LOCKED2X_delay;

      dll4x  :      CLKDLL port map (CLKIN=>CLK2X_g,
                  CLKFB=>CLK4X_g, RST=>RESET4X, CLK0=>open,
                  CLK90=>open, CLK180=>open, CLK270=>open,
                  CLK2X=>CLK4X_dll, CLKDV=>open,
                  LOCKED=>LOCKED);

      clk4xg : BUFG   port map (I=>CLK4X_dll,  O=>CLK4X_g);

      CLK2X <= CLK2X_g;

      CLK4X <= CLK4X_g;

end structural;
```

Virtex-E DLL 4X Verilog Example

```
//
//Include your synthesis vendor-specific library references here

module dll_standard (CLKIN, RESET, CLK2X, CLK4X, LOCKED);

input CLKIN, RESET;

output CLK2X, CLK4X, LOCKED;

      wire CLKIN_w, RESET_w, CLK4X_dll, LOCKED2X,
      LOCKED4X;

      wire LOCKED2X_delay, RESET4X;

      wire logic1;

      assign logic1 = 1'b1;

      IBUFG clkpad (.I(CLKIN), .O(CLKIN_w));

      IBUF  rstpad (.I(RESET), .O(RESET_w));

      CLKDLL dll2x (.CLKIN(CLKIN_w), .CLKFB(CLK2X),
                  .RST(RESET_w), .CLK0(), .CLK90(), .CLK180(),
                  .CLK270(),  .CLK2X(CLK2X),
                  .CLKDV(),.LOCKED(LOCKED2X));

      SRL16  rstsrl (.D(LOCKED2X), .CLK(CLK2X),
                  .Q(LOCKED2X_delay),.A3(logic1),.A2(logic1),
                  .A1(logic1), .A0(logic1));

      assign RESET4X = ~LOCKED2X_delay;

      CLKDLL dll4x (.CLKIN(CLK2X), .CLKFB(CLK4X),
                  .RST(RESET4X), .CLK0(), .CLK90(),.CLK180(),
                  .CLK270(), .CLK2X(CLK4X_dll), .CLKDV(),
                  .LOCKED(LOCKED));

      BUFG   clk4xg (.I(CLK4X_dll),  .O(CLK4X));

endmodule
```

Set Up a 7X Clock in a Virtex-II FPGA

The following VHDL and Verilog code examples show how to implement a 7X clock multiplication in a Virtex-II device. In these examples, attributes of the DCM are passed in the hardware description language code. Alternatively, they could be specified in the UCF.

The input clock first goes through an IBUFG component and then to the CLKIN pin of the DCM. The CLK0 pin from the DCM is connected to a BUFG, and this output feeds back to the CLKFB input pin of the DCM. Using the BUFG to connect the CLK0 output pin to the CLKFB input pin is only required if you want to have the CLKFX output phase-aligned with the CLK0 output. If this is not required, then the BUFG resource (and the power it consumes) can be saved for other uses. In order to create the desired frequency, two attributes are assigned to the DCM in the source code. These are the "CLKFX_MULTIPLY" (assigned a value of "7") and "CLKFX_DIVIDE" (assigned a value of "1"). These attributes can be any value between 1 and 4,096 as long as the input frequency is from 1-300 MHz and the output frequency is from 24-300 MHz.

As with the examples before, we used the Synplify logic synthesis engine.

Using the CLKDLL and DCM in Simulations

The following hints will help guarantee the CLKDLL or DCM simulation will work correctly:

- The simulation resolution time must be set to display in pico-seconds as part of the simulation setup. In VHDL, the simulator resolution is set within the simulator. When running Verilog, set the following timing specification at the top of the main Verilog source code:
  `timescale 1 ns / 1 ps
- The simulation must run long enough for the LOCKED output signal of the Virtex-E CLKDLL and the Virtex-II DCM to go high. The CLKDLL hardware in Virtex-E using an input clock

Virtex-II VHDL DCM/DLL Example

```
library IEEE;

use IEEE.std_logic_1164.all;

- - Include your synthesis vendor-specific library references here

entity clock_distribution_block is

      port (CLK_IN, RST_DLL : in std_logic;
      CLK7X, LOCKED : out std_logic);

end clock_distribution_block;

architecture STRUCT of clock_distribution_block is

attribute CLKFX_MULTIPLY : string;

attribute CLKFX_DIVIDE   : string;

attribute CLKFX_MULTIPLY of U2 : label is "7";

attribute CLKFX_DIVIDE of U2 : label is "1";


signal CLK, CLK_int, CLK_dcm, CLKFX_int, LCK_int, RST_int: std_logic;

signal DUMMY : std_logic := '0';

component IBUFG

      port (I : in  std_logic;  O : out std_logic);

end component;

component BUFG

      port (I : in  std_logic; O : out std_logic);

end component;

component DCM is

      port (CLKFB,CLKIN,DSSEN,PSCLK,PSEN,PSINCDEC,RST : in std_logic;
            CLK0,CLK90,CLK180,CLK270,CLK2X,CLK2X180,CLKDV,
            CLKFX,CLKFX180,LOCKED,PSDONE      : out std_logic;
            STATUS  : out std_logic_vector (7 downto 0));

end component;

begin

      U1 : IBUFG port map (I => CLK_IN, O => CLK_int);

      U2 : DCM port map (CLKIN => CLK_int, CLKFB => CLK,
            RST => RST_int, DSSEN  => DUMMY, PSINCDEC => DUMMY,
            PSEN => DUMMY, PSCLK => DUMMY, CLK0 => CLK_dcm,
            CLKFX => CLKFX_int, LOCKED => LCK_int);

      U3 : BUFG port map (I => CLK_dcm, O => CLK);

      RST_int <= RST_DLL;

      CLK7X <= CLKFX_int;

      LOCKED <= LCK_int;

end architecture STRUCT;
```

Virtex-II Verilog DCM/DLL Example

```
//Include your synthesis vendor-specific library references here

module clock_distribution_block

(CLK_IN,RST_DLL,CLK7X,LOCKED);

      input    CLK_IN, RST_DLL;

      output   CLK7X, LOCKED;

wire     CLK, CLK_int, CLK_dcm;

IBUFG U1  (.I(CLK_IN), .O(CLK_int));

DCM U2  (.CLKFB(CLK), .CLKIN(CLK_int), .RST(RST_DLL),          DSSEN(1'b0),
          .PSINCDEC(1'b0), .PSEN(1'b0), .PSCLK(1'b0),
.CLK0(CLK_dcm),
          .CLK90(), .CLK180(), .CLK270(), .CLK2X(), .CLK2X180(),
          .CLKDV(), .CLKFX(CLK7X), .CLKFX180(), .LOCKED(LOCKED))
          /* synthesis CLKFX_MULTIPLY=7 CLKFX_DIVIDE=1 */;

BUFG U3 (.I(CLK_dcm), .O(CLK));

endmodule // clock_distribution_block
```

If location constraints are required for the DCM, they can be specified in a UCF. In the following example, the DCM is placed in the upper left quadrant of 2v40 device:

```
#UCF with LOCs for DCM for Virtex-II DCM/DLL example

inst U2  LOC = DCM_X0Y1;
```

running at 40-50 MHz will take up to 50 microseconds to achieve a lock signal. (Refer to the DLL clock tolerance, jitter, and phase information in the *Xilinx Data Book 2000* for more information. *The Data Book* is online at www.xilinx.com/partinfo/databook.htm.)

The DCM in a Virtex-II device will exhibit similar lock times as the CLKDLL in Virtex-E, with the exception of the use of CLKFX and CLKFX180 outputs. If these outputs are used, the lock times may be significantly shorter or longer, with a maximum of a few milliseconds for very large CLKFX_MULTIPLY attribute values. When run in a simulator, such as ModelSim XE™, the LOCKED output signal of the CLKDLL and the DCM will go high approximately 1 microsecond after the reset signal goes from high to low.

- The CLKDLLs and DCM must also be run within certain frequencies to adhere to the timing specs. (Refer to the *Xilinx Data Book 2000* for the timing specs of the Virtex-E DLL and the *Virtex-II Platform FPGA Handbook* for the timing specs of the DCM.)

In order to run a simulation on these VHDL and Verilog examples, the Virtex-E or Virtex-II library references must be included. In the VHDL example, to prevent compiler problems when the VHDL is run through your chosen synthesis software, you must add the following lines in between the "translate_off" and "translate_on" statements:

```
library unisim;

use unisim.vcomponents.all;
```

In the Verilog examples, an "`include" statement is needed to reference the path to the Virtex-E or Virtex-II libraries. In addition, a global Verilog file must be referenced and compiled in the simulator (glbl.v). Refer to "XAPP108" (www.xilinx.com/xapp/xapp108.pdf) or the "Synthesis and Simulation Design Guide" (toolbox.xilinx.com/docsan/3_1i/).

Conclusion

DLLs and DCMs offer great flexibility for clock management in demanding designs. Just make sure you refer to the specifics in your vendor's synthesis tool when using attributes in HDL designs.

For additional information about using CLKDLLs to generate or manage various clock signal configurations, refer to "XAPP132" (www.xilinx.com/xapp/xapp132.pdf).