```
------------------------------------------------------------------
-- System Generator version 12.1 VHDL source file.
-- Copyright(C) 2010 by Xilinx, Inc.  All rights reserved.  This
-- text/file contains proprietary, confidential information of Xilinx,
-- Inc., is distributed under license from Xilinx, Inc., and may be used,
-- copied and/or disclosed only pursuant to the terms of a valid license
-- agreement with Xilinx, Inc.  Xilinx hereby grants you a license to use
-- this text/file solely for design, simulation, implementation and
-- creation of design files limited to Xilinx devices or technologies.
-- Use with non-Xilinx devices or technologies is expressly prohibited
-- and immediately terminates your license unless covered by a separate
------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.conv_pkg.all;
-- synopsys translate_off
library unisim;
use unisim.vcomponents.all;
-- synopsys translate_on
entity xlclockdriver is
  generic (period: integer := 2;log_2_period: integer := 0;pipeline_regs: integer := 5;use_bufg:
integer:= 0);
  port (sysclk: in std_logic;sysclr: in std_logic;sysce: in std_logic;clk: out std_logic;clr: out std_logic;
ce: out std_logic;ce_logic: out std_logic);
end xlclockdriver;
architecture behavior of xlclockdriver is
  component bufg
    port ( i: in std_logic; o: out std_logic );
  end component;
  component synth_reg_w_init
    generic (width: integer; init_index: integer; init_value: bit_vector; latency: integer);
    port (i: in std_logic_vector(width - 1 downto 0);ce: in std_logic;  clr: in std_logic;clk: in std_logic;
```

```
    o: out std_logic_vector(width - 1 downto 0));
end component;
function size_of_uint(inp: integer; power_of_2: boolean)
  return integer
is
  constant inp_vec: std_logic_vector(31 downto 0) :=
    integer_to_std_logic_vector(inp,32, xlUnsigned);
  variable result: integer;
begin
  result := 32;
  for i in 0 to 31 loop
    if inp_vec(i) = '1' then
      result := i;
    end if;
  end loop;
  if power_of_2 then
    return result;
  else
    return result+1;
  end if;
end;
function is_power_of_2(inp: std_logic_vector)
  return boolean
is
  constant width: integer := inp'length;
  variable vec: std_logic_vector(width - 1 downto 0);
  variable single_bit_set: boolean;
  variable more_than_one_bit_set: boolean;
  variable result: boolean;
begin
  vec := inp;
  single_bit_set := false;
  more_than_one_bit_set := false;
```

```
  -- synopsys translate_off
  if (is_XorU(vec)) then
    return false;
  end if;
   -- synopsys translate_on
  if width > 0 then
    for i in 0 to width - 1 loop
      if vec(i) = '1' then
        if single_bit_set then
          more_than_one_bit_set := true;
        end if;
        single_bit_set := true;
      end if;
    end loop;
  end if;
  if (single_bit_set and not(more_than_one_bit_set)) then
    result := true;
  else
    result := false;
  end if;
  return result;
end;
function ce_reg_init_val(index, period : integer)
  return integer
is
   variable result: integer;
 begin
    result := 0;
    if ((index mod period) = 0) then
        result := 1;
    end if;
    return result;
end;
```

```
function remaining_pipe_regs(num_pipeline_regs, period : integer)
  return integer
is
   variable factor, result: integer;
begin
    factor := (num_pipeline_regs / period);
    result := num_pipeline_regs - (period * factor) + 1;
    return result;
end;

function sg_min(L, R: INTEGER) return INTEGER is
begin
    if L < R then
         return L;
    else
         return R;
    end if;
end;
constant max_pipeline_regs : integer := 8;
constant pipe_regs : integer := 5;
constant num_pipeline_regs : integer := sg_min(pipeline_regs, max_pipeline_regs);
constant rem_pipeline_regs : integer := remaining_pipe_regs(num_pipeline_regs,period);
constant period_floor: integer := max(2, period);
constant power_of_2_counter: boolean :=
  is_power_of_2(integer_to_std_logic_vector(period_floor,32, xlUnsigned));
constant cnt_width: integer :=
  size_of_uint(period_floor, power_of_2_counter);
constant clk_for_ce_pulse_minus1: std_logic_vector(cnt_width - 1 downto 0) :=
  integer_to_std_logic_vector((period_floor - 2),cnt_width, xlUnsigned);
constant clk_for_ce_pulse_minus2: std_logic_vector(cnt_width - 1 downto 0) :=
  integer_to_std_logic_vector(max(0,period - 3),cnt_width, xlUnsigned);
constant clk_for_ce_pulse_minus_regs: std_logic_vector(cnt_width - 1 downto 0) :=
  integer_to_std_logic_vector(max(0,period - rem_pipeline_regs),cnt_width, xlUnsigned);
```

```
  signal clk_num: unsigned(cnt_width - 1 downto 0) := (others => '0');
  signal ce_vec : std_logic_vector(num_pipeline_regs downto 0);
  attribute MAX_FANOUT : string;
  attribute MAX_FANOUT of ce_vec:signal is "REDUCE";
  signal ce_vec_logic : std_logic_vector(num_pipeline_regs downto 0);
  attribute MAX_FANOUT of ce_vec_logic:signal is "REDUCE";
  signal internal_ce: std_logic_vector(0 downto 0);
  signal internal_ce_logic: std_logic_vector(0 downto 0);
  signal cnt_clr, cnt_clr_dly: std_logic_vector (0 downto 0);
begin
  clk <= sysclk;
  clr <= sysclr;
  cntr_gen: process(sysclk)
  begin
    if sysclk'event and sysclk = '1'  then
      if (sysce = '1') then
        if ((cnt_clr_dly(0) = '1') or (sysclr = '1')) then
          clk_num <= (others => '0');
        else
          clk_num <= clk_num + 1;
        end if;
    end if;
    end if;
  end process;
  clr_gen: process(clk_num, sysclr)
  begin
    if power_of_2_counter then
      cnt_clr(0) <= sysclr;
    else
      if (unsigned_to_std_logic_vector(clk_num) = clk_for_ce_pulse_minus1
          or sysclr = '1') then
        cnt_clr(0) <= '1';
      else
```

```
      cnt_clr(0) <= '0';
    end if;
  end if;
end process;
clr_reg: synth_reg_w_init
  generic map (width => 1,_index => 0,  init_value => b"0000", latency => 1)
  port map ( i => cnt_clr,ce => sysce, clr => sysclr, clk => sysclk,o => cnt_clr_dly);
pipelined_ce : if period > 1 generate
    ce_gen: process(clk_num)
    begin
      if unsigned_to_std_logic_vector(clk_num) = clk_for_ce_pulse_minus_regs then
        ce_vec(num_pipeline_regs) <= '1';
      else
        ce_vec(num_pipeline_regs) <= '0';
      end if;
    end process;
    ce_pipeline: for index in num_pipeline_regs downto 1 generate
      ce_reg : synth_reg_w_init
        generic map ( width => 1, init_index => ce_reg_init_val(index, period), init_value =>
b"0000",  latency => 1)
        port map ( i => ce_vec(index downto index), ce => sysce, clr => sysclr,clk => sysclk,
          o => ce_vec(index-1 downto index-1) );
    end generate;
    internal_ce <= ce_vec(0 downto 0);
end generate;
pipelined_ce_logic: if period > 1 generate
    ce_gen_logic: process(clk_num)
    begin
      if unsigned_to_std_logic_vector(clk_num) = clk_for_ce_pulse_minus_regs then
        ce_vec_logic(num_pipeline_regs) <= '1';
      else
        ce_vec_logic(num_pipeline_regs) <= '0';
      end if;
```

```
    end process;
    ce_logic_pipeline: for index in num_pipeline_regs downto 1 generate
      ce_logic_reg : synth_reg_w_init
        generic map (width => 1,init_index => ce_reg_init_val(index, period), init_value => b"0000",
          latency => 1)
        port map (i => ce_vec_logic(index downto index), ce => sysce,clr => sysclr,clk => sysclk,
          o => ce_vec_logic(index-1 downto index-1));
    end generate;
    internal_ce_logic <= ce_vec_logic(0 downto 0);
  end generate;
  use_bufg_true: if period > 1 and use_bufg = 1 generate
    ce_bufg_inst: bufg
      port map (i => internal_ce(0),o => ce);
    ce_bufg_inst_logic: bufg
      port map (i => internal_ce_logic(0),o => ce_logic );
  end generate;
  use_bufg_false: if period > 1 and (use_bufg = 0) generate
    ce <= internal_ce(0);
    ce_logic <= internal_ce_logic(0);
  end generate;
  generate_system_clk: if period = 1 generate
    ce <= sysce;
    ce_logic <= sysce;
  end generate;
end architecture behavior;
library IEEE;
use IEEE.std_logic_1164.all;
use work.conv_pkg.all;
entity default_clock_driver is
  port (sysce: in std_logic; sysce_clr: in std_logic; sysclk: in std_logic; ce_1: out std_logic;
clk_1: out std_logic );
end default_clock_driver;
architecture structural of default_clock_driver is
```

```
  attribute syn_noprune: boolean;
  attribute syn_noprune of structural : architecture is true;
  attribute optimize_primitives: boolean;
  attribute optimize_primitives of structural : architecture is false;
  attribute dont_touch: boolean;
  attribute dont_touch of structural : architecture is true;
 signal sysce_clr_x0: std_logic;
  signal sysce_x0: std_logic;
  signal sysclk_x0: std_logic;
  signal xlclockdriver_1_ce: std_logic;
  signal xlclockdriver_1_clk: std_logic;
begin
  sysce_x0 <= sysce;
  sysce_clr_x0 <= sysce_clr;
  sysclk_x0 <= sysclk;
  ce_1 <= xlclockdriver_1_ce;
  clk_1 <= xlclockdriver_1_clk;
xlclockdriver_1: entity work.xlclockdriver
    generic map ( log_2_period => 1,period => 1,use_bufg => 0)
port map (sysce => sysce_x0,sysclk => sysclk_x0,sysclr => sysce_clr_x0,ce => xlclockdriver_1_ce,
      clk => xlclockdriver_1_clk);
end structural;
library IEEE;
use IEEE.std_logic_1164.all;
use work.conv_pkg.all;
entity digitalpid_cw is
  port ( ce: in std_logic := '1'; clk: in std_logic; -- clock period = 10.0 ns (100.0 Mhz)
    gateway_in: in std_logic_vector(15 downto 0);
    gateway_in1: in std_logic_vector(15 downto 0);
    gateway_out: out std_logic_vector(50 downto 0) );
end digitalpid_cw;
architecture structural of digitalpid_cw is
  component xlpersistentdff
```

```
    port (clk: in std_logic; d: in std_logic;  q: out std_logic);
  end component;
  attribute syn_black_box: boolean;
  attribute syn_black_box of xlpersistentdff: component is true;
  attribute box_type: string;
  attribute box_type of xlpersistentdff: component is "black_box";
  attribute syn_noprune: boolean;
  attribute optimize_primitives: boolean;
  attribute dont_touch: boolean;
  attribute syn_noprune of xlpersistentdff: component is true;
  attribute optimize_primitives of xlpersistentdff: component is false;
  attribute dont_touch of xlpersistentdff: component is true;
 signal ce_1_sg_x1: std_logic;
  attribute MAX_FANOUT: string;
  attribute MAX_FANOUT of ce_1_sg_x1: signal is "REDUCE";
  signal clkNet: std_logic;
  signal clk_1_sg_x1: std_logic;
  signal gateway_in1_net: std_logic_vector(15 downto 0);
  signal gateway_in_net: std_logic_vector(15 downto 0);
  signal gateway_out_net: std_logic_vector(50 downto 0);
  signal persistentdff_inst_q: std_logic;
attribute syn_keep: boolean;
  attribute syn_keep of persistentdff_inst_q: signal is true;
  attribute keep: boolean;
  attribute keep of persistentdff_inst_q: signal is true;
  attribute preserve_signal: boolean;
  attribute preserve_signal of persistentdff_inst_q: signal is true;
begin
  clkNet <= clk;
  gateway_in_net <= gateway_in;
  gateway_in1_net <= gateway_in1;
  gateway_out <= gateway_out_net;
  default_clock_driver_x0: entity work.default_clock_driver
```

```
    port map (
      sysce => '1',
      sysce_clr => '0',
      sysclk => clkNet,
      ce_1 => ce_1_sg_x1,
      clk_1 => clk_1_sg_x1  );
digitalpid_x0: entity work.digitalpid
    port map (
      ce_1 => ce_1_sg_x1,
      clk_1 => clk_1_sg_x1,
      gatewayin => gateway_in_net,
      gateway_in1 => gateway_in1_net,
      gateway_out => gateway_out_net );
persistentdff_inst: xlpersistentdff
    port map (
      clk => clkNet,
      d => persistentdff_inst_q,
      q => persistentdff_inst_q );
end structural;
```