IQ Phase and Gain Correction implemented with real type variables
IQCorrectionReal workspace

version 7.2
STUDENT EDITION
Active-HDL™
Complete FPGA Verification Environment
ALDEC
The Design Verification Company
Copyright © 1997-2007. All rights reserved.

ALDEC, Inc. homepage http://www.aldec.com

Contents

2 Design Hierarchy Structure

```
┌─iqgainphasecorrection_testbench (iqgainphasecorrection_testbench_read)
│ ├─DUT : iqgainphasecorrection (iqgainphasecorrection_integer)
│ │ └─correction
│ ├─READ_I_Q_SAMPLES
│ ├─COMPARE_RESULTS
│ └─DRIVE_CLOCK
├─std.standard
├─std.textio
├─ieee.std_logic_1164
├─ieee.numeric_std
├─ieee.math_real
├─ieee.std_logic_textio
├─ieee.std_logic_arith
└─ieee.std_logic_unsigned
```

3 IQCorrectionReal

3.1 IQGainPhaseCorrection_entity.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;



entity IQGainPhaseCorrection is

generic(width:natural);

port(
    clk          :in std_logic;
    x1          :in signed(width-1 downto 0);
    y1          :in signed(width-1 downto 0);
    gain_error     :out signed(width-1 downto 0);
    gain_lock   :out bit;
    phase_error       :out signed(width-1 downto 0);
    phase_lock     :out bit;
    corrected_x1   :out signed(width-1 downto 0);
    corrected_y1   :out signed(width-1 downto 0)
    );

end IQGainPhaseCorrection;
```

3.2 IQGainPhaseCorrection_testbench_read.vhd

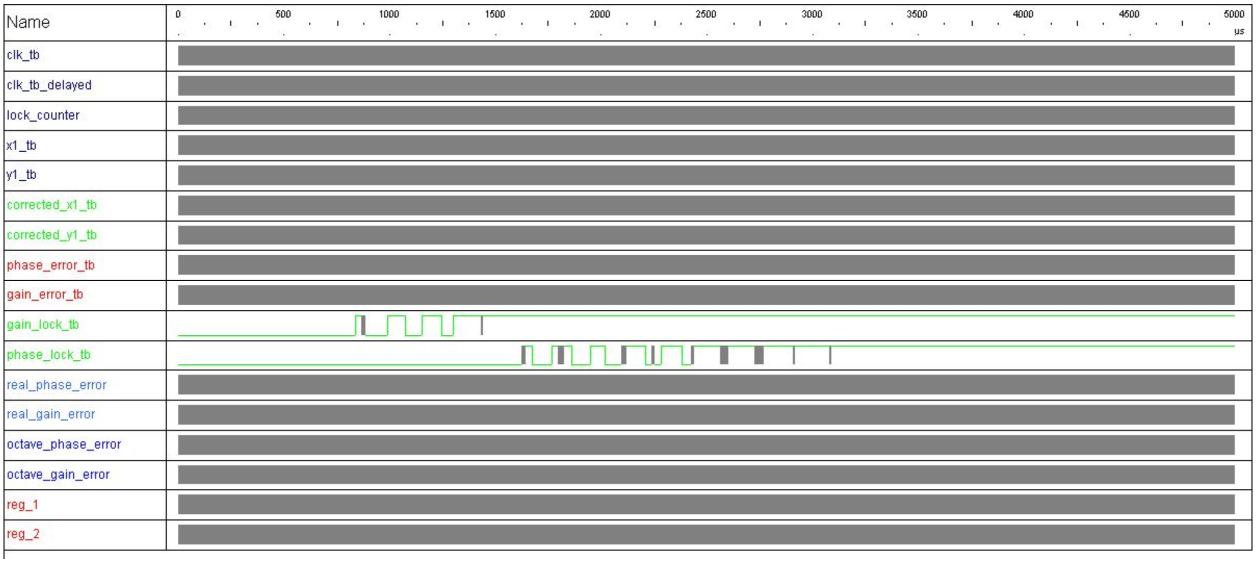
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.math_real.all;
use ieee.numeric_std.all;
use std.textio.all;
use ieee.std_logic_textio.all;

entity IQGainPhaseCorrection_testbench is
end entity;


--The read architecture reads I and Q samples from a text file.
--The values were created by the MATLAB reference model for the design.

architecture IQGainPhaseCorrection_testbench_read of IQGainPhaseCorrection_tes
tbench is

--declare the DUT as a component.
component IQGainPhaseCorrection is
    generic(width :natural);
    port(
    clk          :in std_logic;
    x1          :in signed(width-1 downto 0);
    y1          :in signed(width-1 downto 0);
    gain_error     :out signed(width-1 downto 0);
    gain_lock   :out bit;
    phase_error       :out signed(width-1 downto 0);
    phase_lock     :out bit;
    corrected_x1   :out signed(width-1 downto 0);
    corrected_y1   :out signed(width-1 downto 0)
    );
end component;


--provide signals to run the DUT.
signal clk_tb      : std_logic := '0';
signal clk_tb_delayed   : std_logic   := '0';
signal x1_tb         : signed(31 downto 0);
signal y1_tb         : signed(31 downto 0);
signal gain_error_tb    : signed(31 downto 0);
signal gain_lock_tb     : bit;
signal phase_error_tb   : signed(31 downto 0);
signal phase_lock_tb    : bit;
signal corrected_x1_tb  : signed(31 downto 0);
signal corrected_y1_tb  : signed(31 downto 0);



begin

    --connect the testbench signal to the component
    DUT:IQGainPhaseCorrection
    generic map(
    width => 32
    )
    port map(
    clk => clk_tb_delayed,
    x1 => x1_tb,
    y1 => y1_tb,
    gain_error => gain_error_tb,
    gain_lock => gain_lock_tb,
    phase_error => phase_error_tb,
    phase_lock => phase_lock_tb,
    corrected_x1 => corrected_x1_tb,
    corrected_y1 => corrected_y1_tb
    );


--Read I and Q from a text file created by MATLAB.
READ_I_Q_SAMPLES: process (clk_tb) is

--read input data into process using the readline technique
file I_data : text open READ_MODE is "I_data_octave";
file Q_data : text open READ_MODE is "Q_data_octave";
variable incoming : line;
variable local_x1 : real;
variable local_y1 : real;
variable int_x1 : integer;
variable returned_x1 : signed(31 downto 0);  --need to parameterize this
variable int_y1 : integer;
variable returned_y1 : signed(31 downto 0);  --need to parameterize this

begin

    if (clk_tb'event and clk_tb = '1') then

       if (not endfile(I_data) and not endfile(Q_data)) then

         readline(I_data, incoming);  --read in the first line.
         read(incoming, local_x1);  --get the real value from the first line
         report "Reading " & real'image(local_x1) & " from I_data.";
         local_x1 := local_x1/(1.11); --model AGC
         report "AGC applied. Result: " & real'image(local_x1) & ".";
         int_x1 := integer(trunc(local_x1*((2.0**31.0)-1.0)));  --scaled
         report "Converted real I_data to the integer " & integer'image(int_x1
) & ".";
         returned_x1 := (to_signed(int_x1, 32));
         x1_tb <= returned_x1;

         readline(Q_data, incoming);  --read in the first line.
         read(incoming, local_y1);  --get the real value from the first line
         report "Reading " & real'image(local_y1) & " from Q_data.";
         local_y1 := local_y1/(1.11); --model AGC
         report "AGC applied. Result: " & real'image(local_y1) & ".";
         int_y1 := integer(trunc(local_y1*((2.0**31.0)-1.0)));  --scaled
         report "Converted real Q_data to the integer " & integer'image(int_y1
) & ".";
         returned_y1 := (to_signed(int_y1, 32));
         y1_tb <= returned_y1;

       else
         file_close(I_data);
         file_close(Q_data);
       end if;
    end if;
end process READ_I_Q_SAMPLES;


COMPARE_RESULTS : process (clk_tb) is

--compare process output with data file using the readline technique
file phase_error : text open READ_MODE is "phase_error_estimate_octave";
file gain_error : text open READ_MODE is "gain_error_estimate_octave";
variable incoming : line;
variable filter_delay : natural := 0;

variable real_phase_error : real;
variable int_phase_error : integer;
variable octave_phase_error : signed(31 downto 0);
variable real_gain_error : real;
variable int_gain_error : integer;
variable octave_gain_error : signed(31 downto 0);

begin
    if (clk_tb'event and clk_tb = '1') then
         if (not endfile(phase_error) and not endfile(phase_error)) then
          --read in a result and compare with testbench result
          readline(phase_error, incoming);  --read in the first line.
          read(incoming, real_phase_error);  --get the real value from the fir
st line
          report "Phase error from model: " & real'image(real_phase_error) & "
.";
          int_phase_error := integer(trunc(real_phase_error*((2.0**31.0)-1.0))
);  --scaled
          report "Converted real phase_error to the integer " & integer'image(
int_phase_error) & ".";
          octave_phase_error := (to_signed(int_phase_error, 32));

          readline(gain_error, incoming);  --read in the first line.
          read(incoming, real_gain_error);  --get the real value from the firs
t line
          report "Gain error from model: " & real'image(real_gain_error) & "."
;
          int_gain_error := integer(trunc(real_gain_error*((2.0**31.0)-1.0)));
  --scaled
          report "Converted real gain_error to the integer " & integer'image(i
nt_gain_error) & ".";
          octave_gain_error := (to_signed(int_gain_error, 32));

         else
          file_close(phase_error);
          file_close(gain_error);
         end if;
    end if;
end process COMPARE_RESULTS;



DRIVE_CLOCK:process
begin
    wait for 50 ns;
    clk_tb <= not clk_tb;
    clk_tb_delayed <= not clk_tb_delayed after 1 ns;
end process;

end IQGainPhaseCorrection_testbench_read;
```

3.3 IQGainPhaseCorrection_arch_integer.vhd

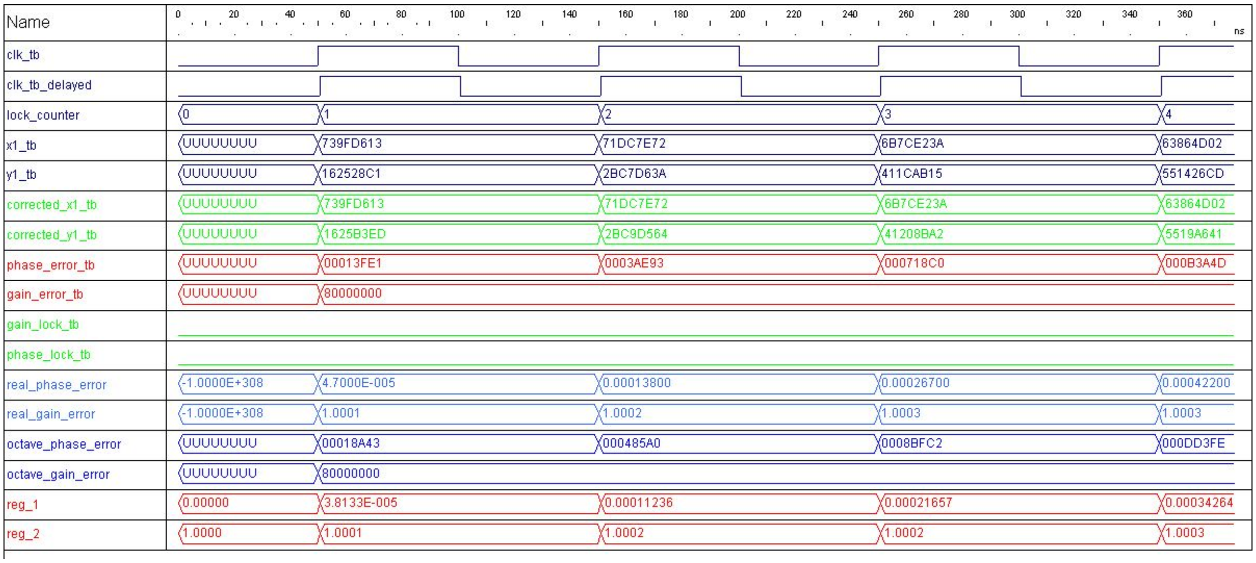
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_unsigned.all;
use ieee.math_real.all;


architecture IQGainPhaseCorrection_integer of IQGainPhaseCorrection is


begin

    correction : process (clk) is

    variable x1_real : real := 0.0;
    variable y1_real : real := 0.0;
    variable reg_1x1 : real := 0.0;
    variable y2      : real := 0.0;
    variable mu_1    : real := 0.000244;
    variable mu_2    : real := 0.000122;
    variable x1y2    : real := 0.0;
    variable reg_1   : real := 0.0;
    variable reg_2   : real := 1.0;
    variable y3      : real := 0.0;

    variable lock_counter : integer range 0 to 100 := 0;
    variable trail_reg_1 : real := 0.0;
    variable trail_reg_2 : real := 1.0;

       begin

        if clk'event and clk = '1' then
            --get the signed I and Q values. Convert them to real values.
            x1_real := real(to_integer(x1));
            x1_real := x1_real / ((2.0**31.0)-1.0);
            y1_real := real(to_integer(y1));
            y1_real := y1_real / ((2.0**31.0)-1.0);


            --phase error estimate, step size set to 0.000244
            y2 := y1_real - reg_1 * x1_real;
            reg_1 := reg_1 + mu_1*x1_real*y2;

            --convert to signed.
            phase_error <= to_signed(integer(trunc(reg_1*((2.0**31.0)-1.0))),
32);


            --gain error estimate, step size set to 0.000122
            y3 := y2 * reg_2;
            reg_2 := reg_2 + mu_2 * ((x1_real*x1_real) - (y3*y3));




            if (lock_counter = 100) then

             --if (abs(trail_reg_2 - reg_2) < 0.0005) then --early lock
             if (abs(trail_reg_2 - reg_2) < 0.00025) then  --locks later
                 gain_lock <= '1';  --gain error is settling
             else
```

```
                 gain_lock <= '0';  --gain error is not settled yet
             end if;

             --if (abs(trail_reg_1 - reg_1) < 0.001) then  --early lock
             if (abs(trail_reg_1 - reg_1) < 0.00025) then  --locks later
                 phase_lock <= '1';  --gain error is settling
             else
                 phase_lock <= '0';  --gain error is not settled yet
             end if;

             trail_reg_2 := reg_2;
             trail_reg_1 := reg_1;

            end if;


            --convert to signed.
            gain_error <= to_signed(integer(trunc(reg_2*((2.0**31.0)-1.0))), 3
2);



            corrected_x1 <= x1;  --I is passed along unchanged. No filter dela
y in this implementation.
            corrected_y1 <= to_signed(integer(trunc((y1_real*reg_2*(cos(reg_1)
))*((2.0**31.0)-1.0))), 32); --Q is corrected and then passed along.

            lock_counter := (lock_counter + 1) mod 101; --update counter that
picks out values to test for phase and gain lock.

       end if;

    end process;

end IQGainPhaseCorrection_integer;
```

3.4 Lock Signals.awf



3.5 First Samples.awf



```
clk_tb
clk_tb_delayed
lock_counter         0             1             2             3             4
x1_tb                UUUUUUUU      739FD613      71DC7E72      6B7CE23A      63864D02
y1_tb                UUUUUUUU      162528C1      2BC7D63A      411CAB15      551426CD
corrected_x1_tb      UUUUUUUU      739FD613      71DC7E72      6B7CE23A      63864D02
corrected_y1_tb      UUUUUUUU      1625B3ED      2BC9D564      41208BA2      5519A641
phase_error_tb       UUUUUUUU      00013FE1      0003AE93      000718C0      000B3A4D
gain_error_tb        UUUUUUUU      80000000
gain_lock_tb
phase_lock_tb
real_phase_error     -1.0000E+308  4.7000E-005   0.00013800    0.00026700    0.00042200
real_gain_error      -1.0000E+308  1.0001        1.0002        1.0003        1.0003
octave_phase_error   UUUUUUUU      00018A43      000485A0      0008BFC2      000DD3FE
octave_gain_error    UUUUUUUU      80000000
reg_1                0.00000       3.8133E-005   0.00011236    0.00021657    0.00034264
reg_2                1.0000        1.0001        1.0002        1.0002        1.0003
```