FINAL PROJECT

"DESIGN OF HOUGH TRANSFORM ON A CORDIC CONFIGURABLE ALGORITHM"

As Part of CourseWork

Of ECE 590

DIGITAL DESIGN IN HARDWARE DESCRIPITIVE LANGUAGE

By

Mithun Bista
Graduate Student
ID #: 990081129

Instructor: Dr. Marek Perkowski
Professor, ECE Department
Portland State University
Spring 2006

Introduction:

The Hough transform (HT) is frequently used to locate possibly occluded straight edges or lines in machine vision. Each detected edge pixel in a binary image votes for a potential edge upon which it might lie. The HT is potentially suitable for video-rate applications such as motion detection (by comparison of successive HT transformed frames), but the computational burden is high, motivating hardware implementations.

The underlying principle of the Hough transform is that there are an infinite number of potential lines that pass through any point, each at a different orientation. The purpose of the transform is to determine which of these theoretical lines pass through most *features* in an image – that is, which lines fit most closely to the data in the image [wikipedia, 1].

1.1 Implementation:

The input to a Hough transform is usually a raw image to which an edge detection operator is usually applied. Thus this way we guarantee that the set of points to be transformed are likely to be an "edge" in the image. The transform itself is quantized into an arbitrary number of bins, each representing an approximate definition of a possible line. Each feature in the edge detected image is said to vote for a set of bins corresponding to the lines that pass through it. By simple incrementing the value stored in each bin for every feature lying on that line, an array is built up which shows which lines fit most closely to the data in the image.

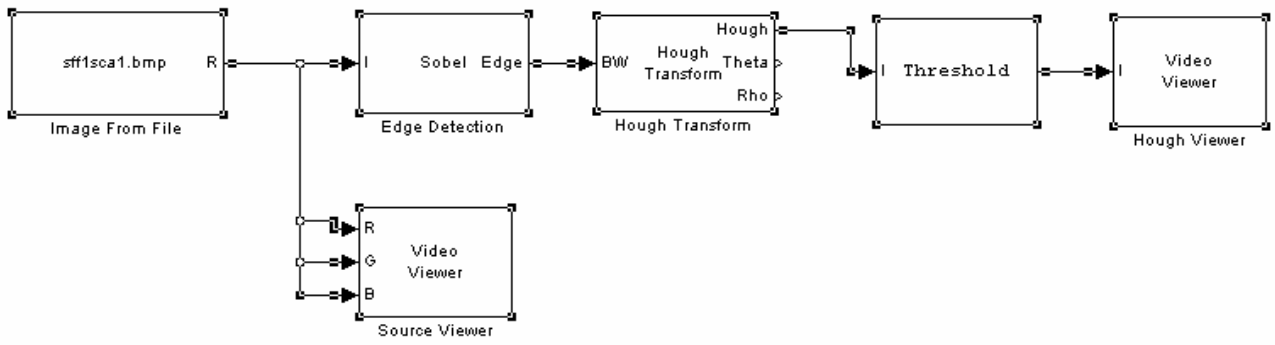
By finding the bins with the highest value, the most likely lines can be extracted. The simplest form of extracting those peaks is to apply some form of threshold operator which compares against some constant. But this threshold operator is not unique and different techniques can be applied yielding better results in different circumstances.

There are lots of variations of Hough transform, but the one that we are going to implement in hardware is called the "Standard Hough transform", and it is given by the parameterized equation

rho = x*cos (theta) + y*sin (theta).

The variable rho is the distance from the origin to the line along a vector perpendicular to the line. theta is the angle between the x-axis and this vector. The Hough function generates a parameter space matrix whose rows and columns correspond to these rho and theta values, respectively.

1.2 Block Diagram of Hough transform:



Here we see that an image from the file is passed to a Sobel edge detector block which transforms the image into a binary image with color white representing edges. Then the standard Hough transform of the edge detected image is performed, which outputs a space matrix of rho and theta values. The threshold block finds peaks in the space matrix which then corresponds to the lines or any arbitrary shapes of the feature in the original image.

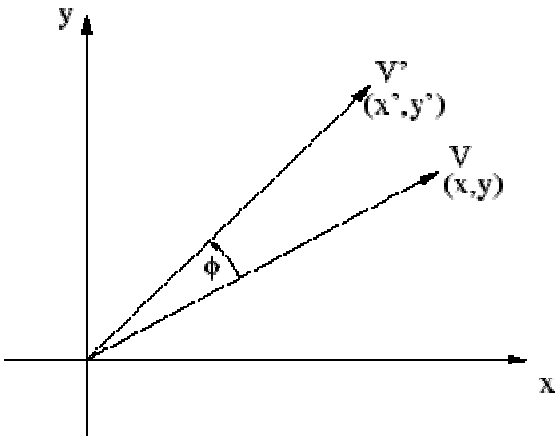
2.1 CORDIC Algorithm for HT:

We implement the Hough transform using the Cordic Algorithm which allows trigonometric angles to be calculated primarily by shifting and adding. This method is very effective because if avoids the multiplication term, so we don't require the multiplier to be used.

In 1957 Jack E. Volder [2] described the Coordinate Rotation Digital Computer or CORDIC for calculation of trigonometric functions, multiplication, division and conversion between binary and mixed radix number system. The CORDIC- algorithm provides an iterative method of performing vector rotations by arbitrary angles using only shift and adds. Volder's algorithm is derived from the general equation for vector rotation. If a vector V with components (x_i, yi) is to be rotated through an angle (ϕ) a new vector V' with components (x_i', yi') is formed by equations:

$$V' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \cdot \cos(\phi) - y \cdot \sin(\phi) \\ y \cdot \cos(\phi) + x \cdot \sin(\phi) \end{bmatrix}$$

The rotation of a vector $V = \begin{bmatrix} x \\ y \end{bmatrix}$ by the angle ϕ is given by the figure below.



Rotation of a vector V by the angle ϕ

The individual equations for x' and y' can be rewritten as:

$$x' = x * \cos(\phi) - y * \sin(\phi)$$

$$y' = y * \cos(\phi) + x * \sin(\phi)$$

and rearranged so that:

$$x' = \cos(\phi)\,[x - y * \tan(\phi)]$$

$$x' = \cos(\phi)\,[y + x * \tan(\phi)]$$

The multiplication by the tangent term can be avoided if the rotation angles and therefore $\tan(\phi)$ are restricted so that $\tan(\phi) = 2^{-i}$. In digital hardware this denotes a simple shift to the right operation. Furthermore, if those rotations are performed iteratively and in both directions every value of $\tan(\phi)$ is representable. With $\phi = \arctan\left(2^{-i}\right)$ the cosine term could also be simplified and since $\cos(\phi) = \cos(-\phi)$ it is a constant for a fixed number of iterations. This iterative rotation can now be expressed as:

$$x_{i+1} = K_i\left[x_i - y_i \cdot d_i \cdot 2^{-i}\right]$$

$$y_{i+1} = K_i\left[y_i + x_i \cdot d_i \cdot 2^{-i}\right]$$

where **Ki** = **Cos (ϕ)** is the gain, for small angles **Cos (ϕ) ~ = ϕ = 1.** For e.g.: Cos (ϕ) = Cos (1.79) = 0.99567. **di** represents in which direction the angle increments.

So how is all of this relevant for calculating the HOUGH TRANSFORM?

Consider coordinate origins at the geometrical center of an image, and then seek the length of a radius R from the origin normal to a straight edge passing between a detected pixel with Cartesian coordinates (X; Y).

R subtends an angle ϕ so that $R = X\cos\phi + Y\sin\phi$; with ϕ valid from 0 to 2π. In practice, ϕ is ranged from 0 to π as the sign of R distinguishes the position of R. Dividing through by cos ϕ and with suitable trigonometric substitutions yields:

$$R_{1X} = (X + Y\tan\theta)\cos\theta, 0 < \theta < \frac{\Pi}{2}, \Pi < \theta < \frac{3\Pi}{2}$$

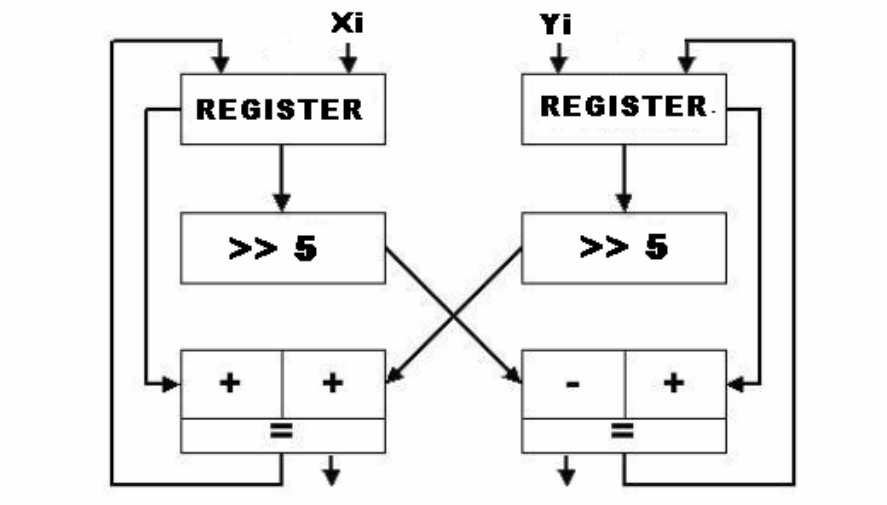
$$R_{1Y} = (Y - X\tan\theta)\cos\theta, \frac{\Pi}{2} < \theta + \frac{\Pi}{2} < \Pi, \frac{3\Pi}{2} < \theta + \frac{\Pi}{2} < 2\Pi),$$

This is also the basis of the CORDIC algorithm (as shown previously), for hardware calculation of trigonometric values by means of micro-rotations. By varying ϕ, all possible values of R are found for varying straight lines through invariant (X, Y).

The angle ϕ = $\pm 2^{-i}$ with i = 0, 1, 2, and so on. In traditional Cordic, the set of micro-rotations that was required to converge to a given angle required a look-up table to establish the direction of a micro-rotation at any stage in the iteration and also a scaling readjustment was required, but for our project, the direction is unidirectional and angle increment is of constant size. This is because we don't need to converge to a particular angle, but to calculate R at constant angles at a given pixel location.

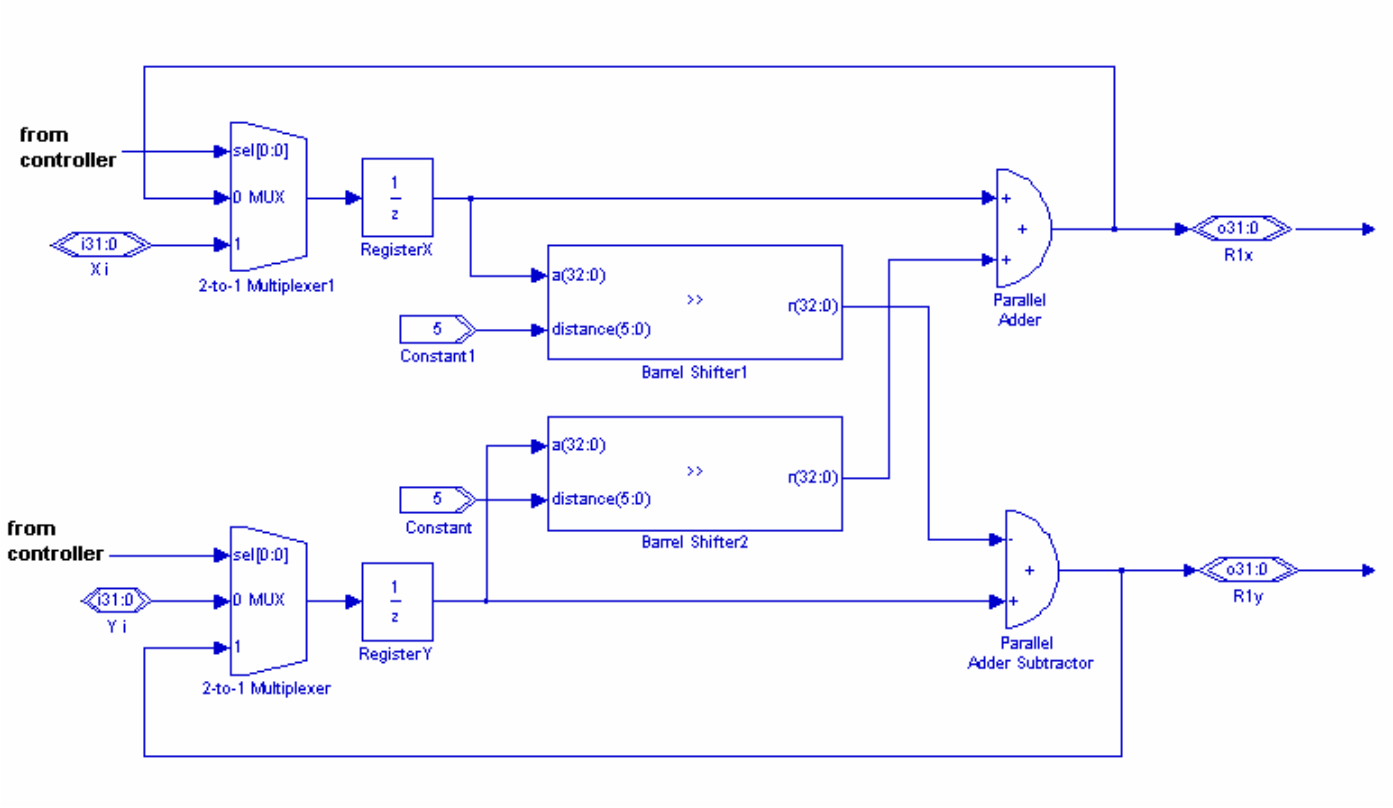
2.2 Constant Rotation CORDIC-Hough Transform:

In this project, I'll be designing the architecture for CORDIC-HT through constant rotation. The micro-rotations are of constant size step increment of ϕ = 1.79. Hence tan (ϕ) = 0.031 = 2^{-5}, which will give us a simple shift of x or y vector by five places. The Cordic-HT cell is given below:



The outputs from the cell are R1X and R1Y. We start by selecting a particular angle ($\phi = 0$) and iterate over the cell to calculate R1X and R1Y, incrementing $\phi = \phi + 1.79$ each time. As we iterate over the range from $0 <= \phi <= \pi/2$, we calculate R in the range $0 <= R <= \pi$. Thus, the cell performs parallel computing of rho values. Each time we iterate, we increment a counter (termed angle counter (not shown in above diagram)). Since for range $0 <= \phi <= \pi/2$, the number of iterations needed is 90/1.79 = 51 iterations. As the counter reaches the value of 51, we assert a stop flag to end the iteration cycle.

2.3 RTL Schematic of the Cordic-HT Cell:



The above schematic shows the bit parallel iterative structure of the Hough-Cordic algorithm. Other architecture that is viable is the bit parallel unrolled HT algorithm and the bit serial unrolled HT algorithm. In our design, we first model the components in VHDL and verify that these components work by simulating with test benches. Each component is written in its own VHDL file. In our final project file, we call these components to model the overall system.

3.1 Design, Simulation and Synthesis:

We start the design by first observing the number of components which are multiplexers, registers; barrel shifter and ripple carry adders and subtractor's.

3.1.1 Design of Barrel Shifter:

The VHDL code for the right shift barrel shifter is given to the right; the input is a 16 input bit vector. The output is a shifted version of the input (from 0 to 7).This circuit consists of three individual barrel shifters. Here the first barrel has only one '0' connected to one of the multiplexers, while the second one has two and the third has four.

The output is equal to the input when shift = 0 (i.e. shift = '000'). When one of the bit change, say for e.g. 010, a shift to two the input bit is performed.

For our HT architecture, we always want a right shift of 5 bits, so we set the shift bits as '101'.

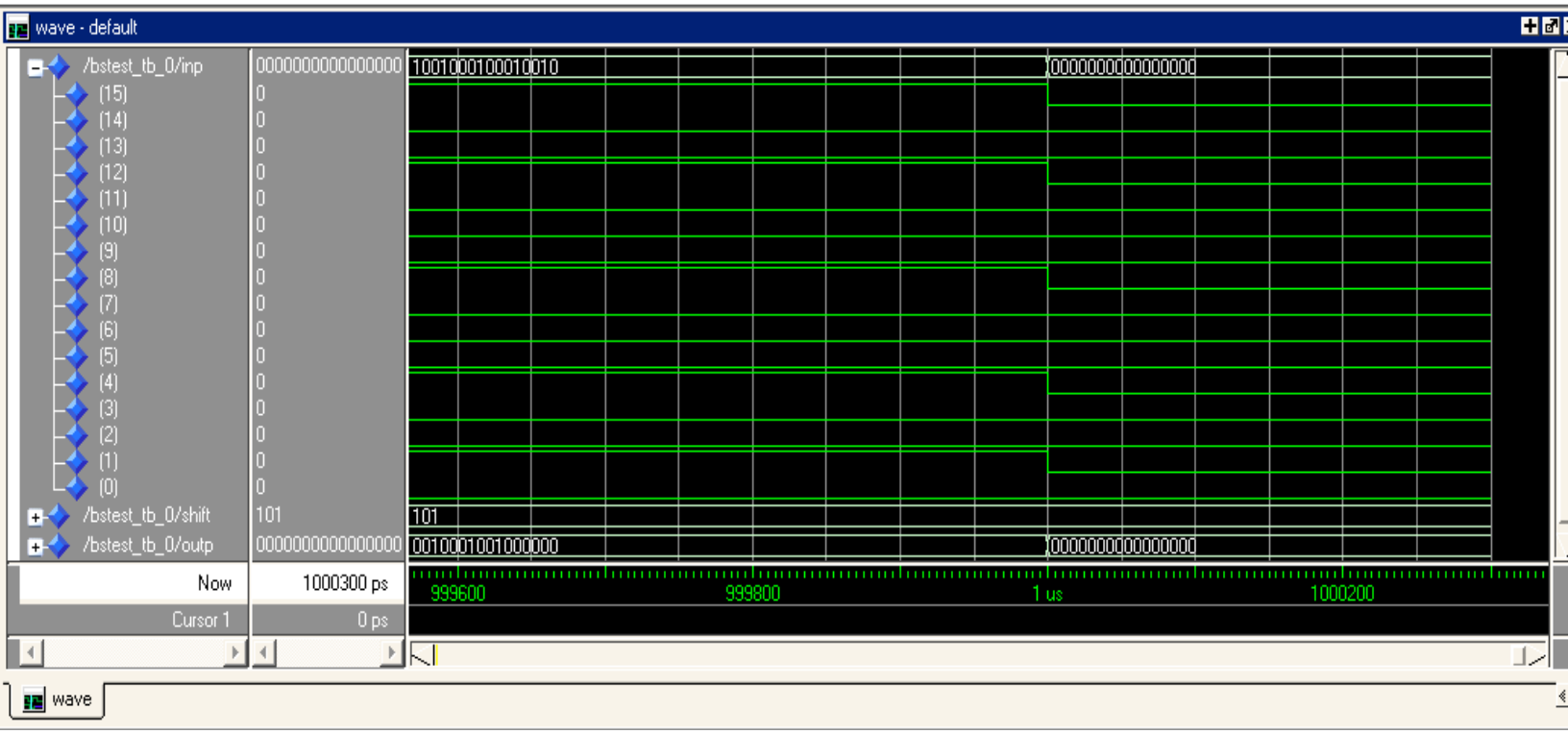
```
library ieee;
  Use ieee.std_logic_1164.all;
  ----------------------------
  entity barrel is
    generic(length: integer:=16);
    port (inp: in std_logic_vector (length-1 downto 0);
          shift: in std_logic_vector (2 downto 0);
          outp: out std_logic_vector (length-1 downto 0));
      end barrel;
  ------------------------------
  architecture behavior of barrel is
    begin
      process (inp, shift)
        variable temp1: std_logic_vector (length-1 downto 0);
        variable temp2: std_logic_vector (length-1 downto 0);
        begin
          --- 1st shifter----
          if (shift(0)='0') then
            temp1 := inp;
          else
            temp1(0) := '0';
            for i in 1 to inp'HIGH loop
            temp1(i) := inp(i-1);
          end loop;
        end if;
        -----2nd shifter------
        if (shift(1)='0') then
          temp2 := temp1;
        else
          for i in 2 to inp'HIGH loop
          temp2(i) := temp1(i-2);
        end loop;
      end if;
      ------3 rd shifter------
      if (shift(2)='0') then
        outp <= temp2;
      else
        for i in 0 to 3 loop
        outp(i) <= '0';
      end loop;
      for i in 4 to inp'HIGH loop
        outp(i) <= temp2(i-4);
      end loop;
    end if;
 end process;
 end behavior;
```

3.1.2 Test Bench Circuit for the Barrel Shifter:

```
--Test bench ckt for the barrel Shifter

        library ieee;
        use ieee.std_logic_1164.all;
        USE IEEE.STD_LOGIC_TEXTIO.ALL;
        USE IEEE.STD_LOGIC_ARITH.ALL;
        USE IEEE.STD_LOGIC_UNSIGNED.ALL;
        USE STD.TEXTIO.ALL;

        ENTITY bstest_tb_0 IS
        END bstest_tb_0;

        ARCHITECTURE testbench_arch OF bstest_tb_0 IS
          FILE RESULTS: TEXT OPEN WRITE_MODE IS "results.txt";

          COMPONENT barrel
            PORT (
              inp : In std_logic_vector (16 downTo 0);
              shift : In std_logic_vector (2 DownTo 0);
              outp : Out std_logic_vector (16 DownTo 0)
            );
          END COMPONENT;
          SIGNAL inp : std_logic_vector (16 DownTo 0) := "10010001000100100";
          SIGNAL shift : std_logic_vector (2 DownTo 0) := "101";
          SIGNAL outp : std_logic_vector (16 DownTo 0) := "00010100100100000";

          SHARED VARIABLE TX_ERROR : INTEGER := 0;
          SHARED VARIABLE TX_OUT : LINE;

          BEGIN
            UUT : barrel
            PORT MAP (
              inp => inp,
              shift => shift,
              outp => outp  );

            PROCESS
              PROCEDURE CHECK_outp(
                next_outp : std_logic_vector (16 DownTo 0);
                TX_TIME : INTEGER
              ) IS
                VARIABLE TX_STR : String(1 to 4096);
                VARIABLE TX_LOC : LINE;
                BEGIN
                IF (outp /= next_outp) THEN
                  STD.TEXTIO.write(TX_LOC, string'("Error at time="));
                  STD.TEXTIO.write(TX_LOC, TX_TIME);
                  STD.TEXTIO.write(TX_LOC, string'("ns outp="));
                  IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, outp);
                  STD.TEXTIO.write(TX_LOC, string'(", Expected = "));
                  IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, next_outp);
                  STD.TEXTIO.write(TX_LOC, string'(" "));
                  TX_STR(TX_LOC.all'range) := TX_LOC.all;
                  STD.TEXTIO.writeline(RESULTS, TX_LOC);
                  STD.TEXTIO.Deallocate(TX_LOC);
                  ASSERT (FALSE) REPORT TX_STR SEVERITY ERROR;
                  TX_ERROR := TX_ERROR + 1;
                END IF;
```

```
    END;
    BEGIN
       -- -------------  Current Time:  1000ns
       WAIT FOR 1000 ns;
       inp <= "0000000000000000";

       IF (TX_ERROR = 0) THEN
          STD.TEXTIO.write(TX_OUT, string'("No errors or warnings"));
          STD.TEXTIO.writeline(RESULTS, TX_OUT);
          ASSERT (FALSE) REPORT
            "Simulation successful (not a failure).  No problems detected."
            SEVERITY FAILURE;
       ELSE
          STD.TEXTIO.write(TX_OUT, TX_ERROR);
          STD.TEXTIO.write(TX_OUT,
             string'(" errors found in simulation"));
          STD.TEXTIO.writeline(RESULTS, TX_OUT);
          ASSERT (FALSE) REPORT "Errors found during simulation"
             SEVERITY FAILURE;
       END IF;
    END PROCESS;

END testbench_arch;
```

3.1.3 Simulation Result



The diagram clearly shows the input ports ‘inp’, shift ports ‘shift’, and output ‘outp’.

3.2.1 Design of the Carry Ripple Adder:

Here we design a 16 bit carry ripple adder to implement our Hough Cordic architecture. For each bit, a full adder unit is employed. The truth table is obvious for the adder, in which a and b represent the input bits, for our case the inputs are xi and the shifted version of yi by 5 places. The cin is the carry-in bit, s is the sum bit, and cout the carry-out bit. s must be high when two or more inputs are high is odd (parity function), while cout must be high when two or more inputs are high (majority function) .

Based on the truth table, a very simple way of computing s and cout is the following:

s = a XOR b XOR cin

cout = (a AND b) OR (a AND cin) OR (b AND cin).

Therefore, the VHDL implementation of the ripple carry adder is straightforward.

```
library IEEE;
  use IEEE.STD_LOGIC_1164.ALL;
  use IEEE.STD_LOGIC_ARITH.ALL;
  use IEEE.STD_LOGIC_UNSIGNED.ALL;
  ----------------------------------------------------------------
  entity carry_ripple_adder is
     generic (length:integer := 16);
       port (a,b: in std_logic_vector (length-1 downto 0);
            cin: in std_logic;
            clk: in std_logic;
            s: out std_logic_vector (length-1 downto 0);
            cout: out std_logic);
  end carry_ripple_adder;
  ----------------------------------------------------------------
  architecture Behavioral of carry_ripple_adder is
   begin
     process (a,b,cin,clk)
       variable carry: std_logic_vector (length downto 0);
          begin
            if(clk='1' and clk'event)then
            carry(0) := cin;
            for i in 0 to length-1 loop
            s(i) <= a(i) XOR b(i) XOR carry(i);
            carry(i+1) := (a(i) AND b(i)) OR (a(i) AND
              carry(i)) OR (b(i) AND carry(i));
            end loop;
              cout <= carry(length);
              end if;
     end process;
  end Behavioral;
```

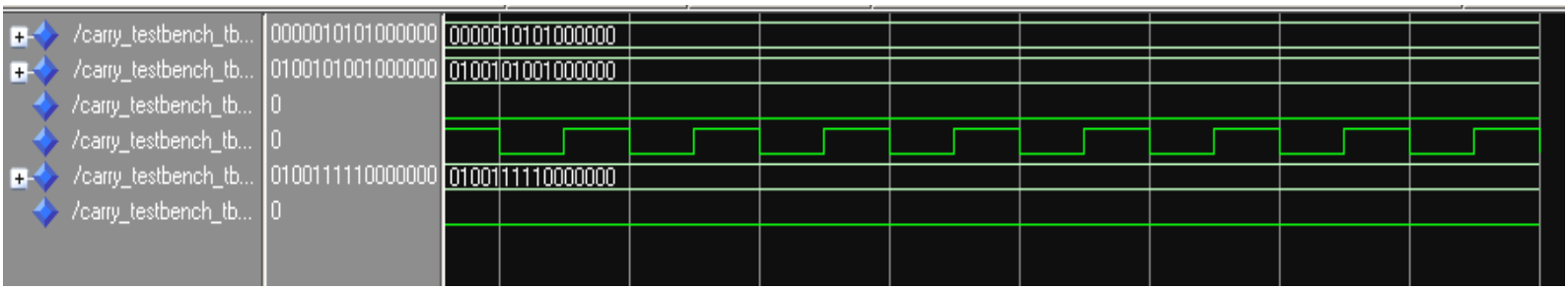
3.2.2: Test Bench Circuit:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY carry_tb_0 IS
END carry_tb_0;

ARCHITECTURE testbench_arch OF carry_tb_0 IS
  FILE RESULTS: TEXT OPEN WRITE_MODE IS "results.txt";

  COMPONENT carry_ripple_adder
    PORT (
      a : In std_logic_vector (15 DownTo 0);
      b : In std_logic_vector (15 DownTo 0);
      cin : In std_logic;
      s : Out std_logic_vector (15 DownTo 0);
      cout : Out std_logic
    );
  END COMPONENT;

  SIGNAL a : std_logic_vector (15 DownTo 0) := "0110011010010011";
  SIGNAL b : std_logic_vector (15 DownTo 0) := "0110101010100110";
  SIGNAL cin : std_logic := '0';
  SIGNAL s : std_logic_vector (15 DownTo 0) := "1001001000000000";
  SIGNAL cout : std_logic := '0';

  SHARED VARIABLE TX_ERROR : INTEGER := 0;
  SHARED VARIABLE TX_OUT : LINE;

  constant PERIOD : time := 200 ns;
  constant DUTY_CYCLE : real := 0.5;
  constant OFFSET : time := 0 ns;

  BEGIN
    UUT : carry_ripple_adder
    PORT MAP (
```

```
        a => a,
        b => b,
        cin => cin,
        s => s,
        cout => cout
    );
    PROCESS    -- clock process for cin
    BEGIN
        WAIT for OFFSET;
        CLOCK_LOOP : LOOP
            cin <= '0';
            WAIT FOR (PERIOD - (PERIOD * DUTY_CYCLE));
            cin <= '1';
            WAIT FOR (PERIOD * DUTY_CYCLE);
        END LOOP CLOCK_LOOP;
    END PROCESS;

    PROCESS
        PROCEDURE CHECK_cout(
            next_cout : std_logic;
            TX_TIME : INTEGER
        ) IS
            VARIABLE TX_STR : String(1 to 4096);
            VARIABLE TX_LOC : LINE;
            BEGIN
            IF (cout /= next_cout) THEN
                STD.TEXTIO.write(TX_LOC, string'("Error at time="));
                STD.TEXTIO.write(TX_LOC, TX_TIME);
                STD.TEXTIO.write(TX_LOC, string'("ns cout="));
                IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, cout);
                STD.TEXTIO.write(TX_LOC, string'(", Expected = "));
                IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, next_cout);
                STD.TEXTIO.write(TX_LOC, string'(" "));
                TX_STR(TX_LOC.all'range) := TX_LOC.all;
                STD.TEXTIO.writeline(RESULTS, TX_LOC);
                STD.TEXTIO.Deallocate(TX_LOC);
                ASSERT (FALSE) REPORT TX_STR SEVERITY ERROR;
                TX_ERROR := TX_ERROR + 1;
            END IF;
        END;
        PROCEDURE CHECK_s(
            next_s : std_logic_vector (15 DownTo 0);
            TX_TIME : INTEGER
        ) IS
            VARIABLE TX_STR : String(1 to 4096);
            VARIABLE TX_LOC : LINE;
            BEGIN
            IF (s /= next_s) THEN
                STD.TEXTIO.write(TX_LOC, string'("Error at time="));
                STD.TEXTIO.write(TX_LOC, TX_TIME);
                STD.TEXTIO.write(TX_LOC, string'("ns s="));
                IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, s);
                STD.TEXTIO.write(TX_LOC, string'(", Expected = "));
                IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, next_s);
                STD.TEXTIO.write(TX_LOC, string'(" "));
                TX_STR(TX_LOC.all'range) := TX_LOC.all;
                STD.TEXTIO.writeline(RESULTS, TX_LOC);
                STD.TEXTIO.Deallocate(TX_LOC);
                ASSERT (FALSE) REPORT TX_STR SEVERITY ERROR;
                TX_ERROR := TX_ERROR + 1;
            END IF;
        END;
        BEGIN
            -- -------------  Current Time:  115ns
            WAIT FOR 115 ns;
            CHECK_cout('1', 115);
```

```
        -- -------------------------------------
        WAIT FOR 1085 ns;

        IF (TX_ERROR = 0) THEN
           STD.TEXTIO.write(TX_OUT, string'("No errors or warnings"));
           STD.TEXTIO.writeline(RESULTS, TX_OUT);
           ASSERT (FALSE) REPORT
             "Simulation successful (not a failure).  No problems detected."
             SEVERITY FAILURE;
        ELSE
           STD.TEXTIO.write(TX_OUT, TX_ERROR);
           STD.TEXTIO.write(TX_OUT,
              string'(" errors found in simulation"));
           STD.TEXTIO.writeline(RESULTS, TX_OUT);
           ASSERT (FALSE) REPORT "Errors found during simulation"
              SEVERITY FAILURE;
        END IF;
      END PROCESS;

    END testbench_arch;
```

3.2.3 Simulation Result:



The simulation result shows the output 16 bit vector of (0100111110000000) when the two inputs are xi = 0000010101000000 and yi = 0100101001000000.

3.3.1 Design of Carry Ripple Subtractor:

The carry ripple subtractor is also designed in the same way as carry ripple adder, with cin set to 1 and each bit of the subtractor vector negated.

cin = 1;

s(i) = A-B = A + NOT B = A(i) XOR NOT B(i) + XOR carry(i)

carry(i+1) := (a(i) AND NOT (b(i))) OR (a(i) AND carry(i)) OR (NOT (b(i)) AND carry(i));

The VHDL code for the subtractor is given below:

```
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
```

```
use IEEE.STD_LOGIC_UNSIGNED.ALL;
----------------------------------------------------------------------------------

entity carry_ripple_subtractor is
        generic (length:integer := 16);
        port (a,b: in std_logic_vector (length-1 downto 0);
                        clk: in std_logic;
                        s: out std_logic_vector (length-1 downto 0);
                        cout: out std_logic);
end carry_ripple_subtractor;
----------------------------------------------------------------------------------
architecture Behavioral of carry_ripple_subtractor is

 begin
                process (a,b,clk)
                    variable carry: std_logic_vector (length downto 0);
                                begin
                                if(clk='1' and clk'event)then
                                carry(0) := '1';
                                for i in 0 to length-1 loop
                                                s(i) <= a(i) XOR NOT(b(i)) XOR carry(i);
                                                carry(i+1) := (a(i) AND NOT (b(i))) OR (a(i) AND
                                        carry(i)) OR (NOT (b(i)) AND carry(i));
                                end loop;
        cout <= carry(length);
                                end if;
                        end process;
end Behavioral;
```

Here the testbench and the simulation are similar to that or carry ripple adder, so I am not including those here. Similarly the design of a register is too obvious to be shown here.

3.4.1 Design of the Hough Cordic Algorithm:

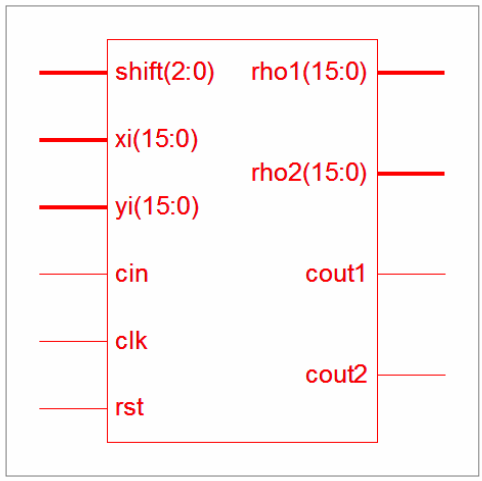
```
----------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
----------------------------------------------------------------
entity Main_project is
  generic(length:integer:=16);
 --input signals
 port(xi,yi: in std_logic_vector(length-1 downto 0);
      cin  : in std_logic;
            clk  : in std_logic;
            rst  : in std_logic;
            shift : in std_logic_vector (2 downto 0);
            cout1,cout2  : out std_logic;
            rho1, rho2: out std_logic_vector (length-1 downto 0));

end Main_project;

architecture Structural of Main_project is
signal out_r1:std_logic_vector (length-1  downto 0);
signal out_r2:std_logic_vector (length-1  downto 0);
signal out_b1:std_logic_vector (length-1  downto 0);
signal out_b2:std_logic_vector (length-1  downto 0);
```

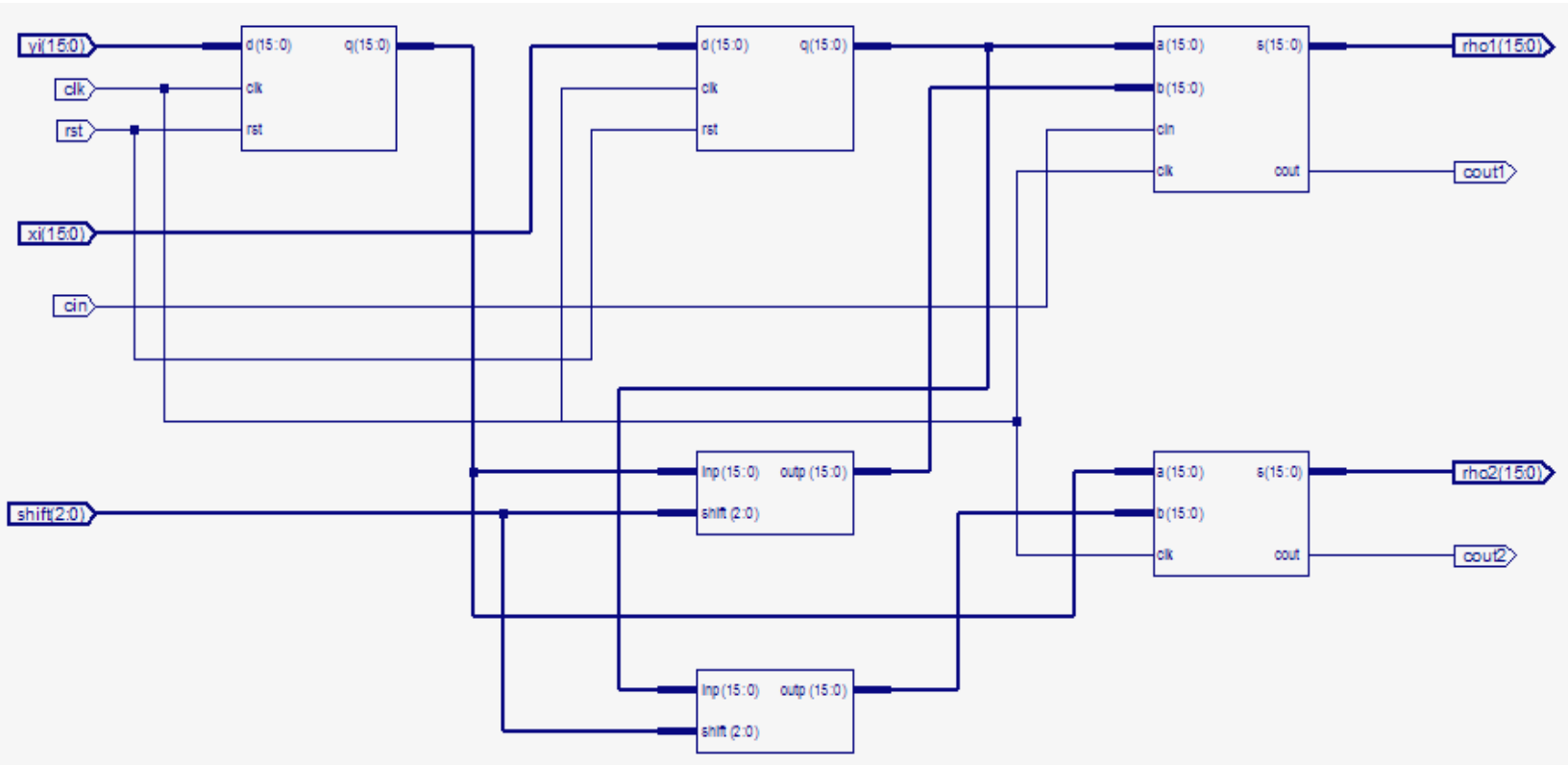
```
------------------
        component register1 is
            port(d: in std_logic_vector(length-1 downto 0);
                  clk, rst: in std_logic;
                  q: out std_logic_vector(length-1 downto 0));
        end component;
--------------------
        component barrel is
            port(inp: in std_logic_vector (length-1  downto 0);
            shift: in std_logic_vector (2 downto 0);
            outp: out std_logic_vector (length-1  downto 0));
        end component;
-------------------
         component carry_ripple_adder is
            port(a,b: in std_logic_vector (length-1 downto 0);
                  cin: in std_logic;
                  clk: in std_logic;
                  s: out std_logic_vector (length-1 downto 0);
                  cout: out std_logic);
                  end component;
------------------
      component carry_ripple_subtractor is
        port (a,b: in std_logic_vector (length-1 downto 0);
              clk: in std_logic;
              s: out std_logic_vector (length-1 downto 0);
              cout: out std_logic);
      end component;


begin
            U1: register1 PORT MAP (xi,clk,rst,out_r1);
            U2: register1 PORT MAP (yi,clk,rst,out_r2);
            U3: barrel PORT MAP (out_r2,shift,out_b1);
            U4: barrel PORT MAP (out_r1,shift,out_b2);
            U5: carry_ripple_adder PORT MAP(out_r1,out_b1,cin,clk,rho1,cout1);
            U6: carry_ripple_subtractor PORT MAP(out_r2,out_b2,clk,rho2,cout2);


end Structural;
```

This is the main project file and it uses structural modeling while most components are modeled behaviorally. The components are declared using the keyword component, and the U1, U2 shows concurrent processes.

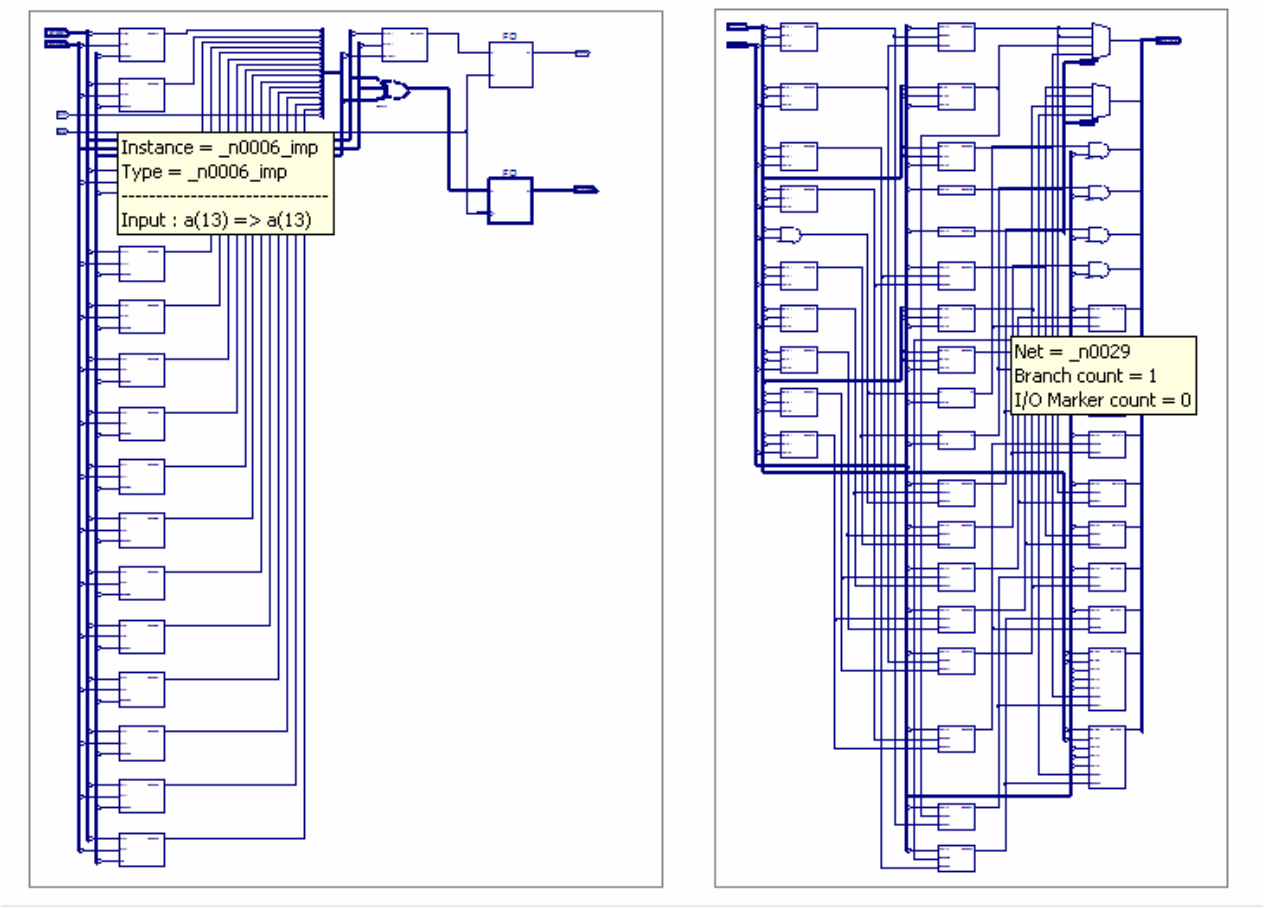
4.1 Synthesis:

This project was performed on a Xilinx Virtex II fpga, the device model # XC2V40. The RTL schematic as synthesized is given below.





Hough Cordic Cell



carry ripple adder barrel shifter

4.2 Hough Project Design Summary:

ents\VHDL\hough\hough\hough.ise - [Design Summary]

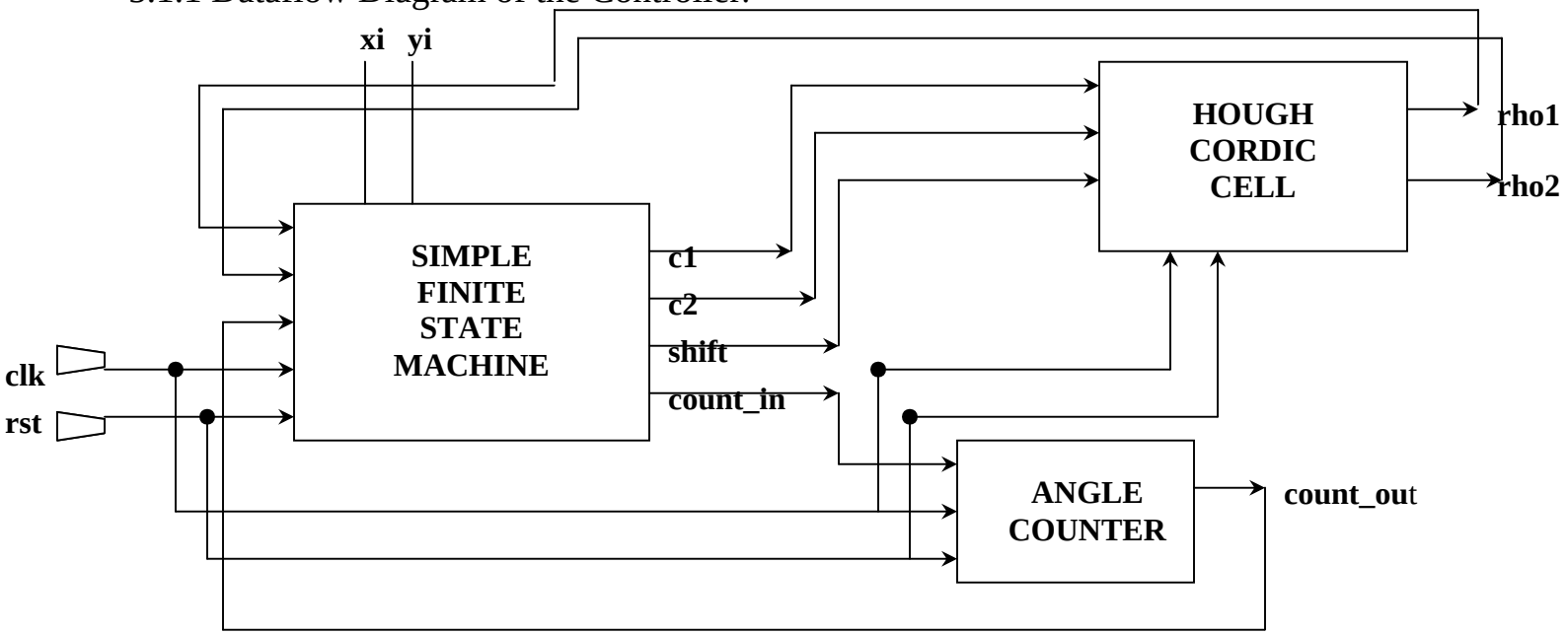
HOUGH Project Status			
Project File:	hough.ise	**Current State:**	Translated
Module Name:	Main_project	• **Errors:**	No Errors
Target Device:	xc2v40-6cs144	• **Warnings:**	11 Warnings (11 new, 0 filtered)
Product Version:	ISE, 8.1i	• **Updated:**	Thu Jun 15 00:29:59 2006

Device Utilization Summary (estimated values)			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	126	256	49%
Number of Slice Flip Flops	69	512	13%
Number of 4 input LUTs	243	512	47%
Number of bonded IOBs	72	88	81%
Number of GCLKs	2	16	12%

5.1 Controller Design:

Now the next thing to do is to design the controller for the Hough Cordic cell. The controller here is a simple finite state machine, that takes input from the angle counter, the 16 bit input pixel value xi and yi, the output of the Cordic cell rho1 and rho2 and decides how many iterations i.e. the number of rotations of the angle, is to be calculated.

5.1.1 Dataflow Diagram of the Controller:



5.1.2 VHDL Code for the finite state machine:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity FSM is
        generic(length:integer:=16); --length is set at 16 bit
        port(xi,yi,rho1,rho2  : in std_logic_vector(length-1 downto 0);
             clk, rst          : in std_logic;
              count_in         : in integer range 0 to 100;
              c1,c2            : out std_logic_vector(length-1 downto 0);
              shift            : out (2 downto 0);
             count_out         : out integer range 0 to 100);
end FSM;

-------------------------------

architecture Simple_state_machine of FSM is
        type state is (stateA,stateB);
        signal pr_state, nx_state:state;

        begin
                process(rst,clk,count_in)
```

```
                        begin
                                if(rst = '1') then
                                        pr_state <= stateA;
                                end if;
                                if (clk'event AND clk ='1') then
                                  pr_state <= nx_state;
                                end if;
      end process;

    process(xi,yi,rho1,rho2,pr_state)
          begin
            case pr_state is
                  when stateA =>
                    c1 <= xi;
                    c2 <= yi;
                    shift <= '101';
                        if (count_in /= 0 AND count_in /=100) then nx_state <= stateB;
                        else nx_state <= stateA;
                        count_out <= count_in;
                        end if;

                when stateB =>
                    c1 <= rho1;
                      c2 <= rho2;
                      shift <= '101';
                      if (count_in = 100) then nx_state <= stateA;
                      else nx_state <= stateB;
                      count_out <= count_in;
                      end if;
            end case;
   end process;
end Simple_state_machine;
```

5.1.3 VHDL code for the Angle Counter:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity counter is
          port(clk,rst: in std_logic;
              count_in: in integer range 0 to 100;
                    count_out: out integer range 0 to 100);
end counter;

architecture counter of counter is
  begin
           process(clk,rst,count_in)
                    begin
                            if(rst = '1' and clk'event and clk ='1') then
                                    count_out <= count_in + 1;
                            elsif(count_in = 100) then
                                    count_out <= 0;
                            end if;
          end process;

end counter;
```

** Here the counter resets after counting to 100, because we have angle increments at 1.78 degrees and 180/1.78 is roughly 100.

5.1.4 VHDL code for the controller:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity controller is
        generic(length:integer:=16); --length is set at 16 bit
        port(xi,yi,rho1,rho2  : in std_logic_vector(length-1 downto 0);
             clk,rst         : in std_logic;
             c1,c2           : out std_logic_vector(length-1 downto 0));
end controller;

architecture Structural of controller is
signal fsm_count_in: integer:= 0;
signal fsm_count_out: integer range 0 to 100;

          component FSM is
                    port(xi,yi,rho1,rho2  : in std_logic_vector(length-1 downto 0);
                    clk,rst         : in std_logic;
                    count_in        : in integer range 0 to 100;
                    c1,c2           : out std_logic_vector(length-1 downto 0);
                    count_out       : out integer range 0 to 100);
                end component;

                component counter is
                    port(clk,rst: in std_logic;
                    count_in: in integer range 0 to 100;
                    count_out: out integer range 0 to 100);
                end component;

        begin
    U1: FSM PORT MAP (xi,yi,rho1,rho2,clk,rst,fsm_count_in,c1,c2,fsm_count_out);
    U2: counter PORT MAP (clk,rst,fsm_count_out,fsm_count_in);
end Structural;
```

Again, I emphasize here that the controller is modeled structurally, while the components are modeled behaviorally.

5.1.5 Test Bench Circuit for Controller:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_TEXTIO.ALL;
USE STD.TEXTIO.ALL;

ENTITY controller_test_tb_0 IS
END controller_test_tb_0;

ARCHITECTURE testbench_arch OF controller_test_tb_0 IS
   FILE RESULTS: TEXT OPEN WRITE_MODE IS "results.txt";
```

```
COMPONENT controller
   PORT (
      xi : In std_logic_vector (15 DownTo 0);
      yi : In std_logic_vector (15 DownTo 0);
      rho1 : In std_logic_vector (15 DownTo 0);
      rho2 : In std_logic_vector (15 DownTo 0);
      clk : In std_logic;
      rst : In std_logic;
      c1 : Out std_logic_vector (15 DownTo 0);
      c2 : Out std_logic_vector (15 DownTo 0)
   );
END COMPONENT;

SIGNAL xi : std_logic_vector (15 DownTo 0) := "0001011001000000";
SIGNAL yi : std_logic_vector (15 DownTo 0) := "0001011000000000";
SIGNAL rho1 : std_logic_vector (15 DownTo 0) := "0010000010100000";
SIGNAL rho2 : std_logic_vector (15 DownTo 0) := "0001010010000000";
SIGNAL clk : std_logic := '1';
SIGNAL rst : std_logic := '1';
SIGNAL c1 : std_logic_vector (15 DownTo 0) := "0000000000000000";
SIGNAL c2 : std_logic_vector (15 DownTo 0) := "0000000000000000";

SHARED VARIABLE TX_ERROR : INTEGER := 0;
SHARED VARIABLE TX_OUT : LINE;

constant PERIOD : time := 200 ns;
constant DUTY_CYCLE : real := 0.5;
constant OFFSET : time := 0 ns;

BEGIN
   UUT : controller
   PORT MAP (
      xi => xi,
      yi => yi,
      rho1 => rho1,
      rho2 => rho2,
      clk => clk,
      rst => rst,
      c1 => c1,
      c2 => c2
   );

   PROCESS    -- clock process for clk
   BEGIN
      WAIT for OFFSET;
      CLOCK_LOOP : LOOP
         clk <= '0';
         WAIT FOR (PERIOD - (PERIOD * DUTY_CYCLE));
         clk <= '1';
         WAIT FOR (PERIOD * DUTY_CYCLE);
      END LOOP CLOCK_LOOP;
   END PROCESS;

   PROCESS
      PROCEDURE CHECK_c1(
         next_c1 : std_logic_vector (15 DownTo 0);
         TX_TIME : INTEGER
      ) IS
         VARIABLE TX_STR : String(1 to 4096);
         VARIABLE TX_LOC : LINE;
         BEGIN
         IF (c1 /= next_c1) THEN
            STD.TEXTIO.write(TX_LOC, string'("Error at time="));
            STD.TEXTIO.write(TX_LOC, TX_TIME);
            STD.TEXTIO.write(TX_LOC, string'("ns c1="));
```

```
            IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, c1);
            STD.TEXTIO.write(TX_LOC, string('', Expected = ''));
            IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, next_c1);
            STD.TEXTIO.write(TX_LOC, string('' ''));
            TX_STR(TX_LOC.all'range) := TX_LOC.all;
            STD.TEXTIO.writeline(RESULTS, TX_LOC);
            STD.TEXTIO.Deallocate(TX_LOC);
            ASSERT (FALSE) REPORT TX_STR SEVERITY ERROR;
            TX_ERROR := TX_ERROR + 1;
         END IF;
      END;
      PROCEDURE CHECK_c2(
         next_c2 : std_logic_vector (15 DownTo 0);
         TX_TIME : INTEGER
      ) IS
         VARIABLE TX_STR : String(1 to 4096);
         VARIABLE TX_LOC : LINE;
         BEGIN
         IF (c2 /= next_c2) THEN
            STD.TEXTIO.write(TX_LOC, string(''Error at time=''));
            STD.TEXTIO.write(TX_LOC, TX_TIME);
            STD.TEXTIO.write(TX_LOC, string(''ns c2=''));
            IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, c2);
            STD.TEXTIO.write(TX_LOC, string('', Expected = ''));
            IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, next_c2);
            STD.TEXTIO.write(TX_LOC, string('' ''));
            TX_STR(TX_LOC.all'range) := TX_LOC.all;
            STD.TEXTIO.writeline(RESULTS, TX_LOC);
            STD.TEXTIO.Deallocate(TX_LOC);
            ASSERT (FALSE) REPORT TX_STR SEVERITY ERROR;
            TX_ERROR := TX_ERROR + 1;
         END IF;
      END;
      BEGIN
         WAIT FOR 1200 ns;

         IF (TX_ERROR = 0) THEN
            STD.TEXTIO.write(TX_OUT, string(''No errors or warnings''));
            STD.TEXTIO.writeline(RESULTS, TX_OUT);
            ASSERT (FALSE) REPORT
              ''Simulation successful (not a failure).  No problems detected.''
              SEVERITY FAILURE;
         ELSE
            STD.TEXTIO.write(TX_OUT, TX_ERROR);
            STD.TEXTIO.write(TX_OUT,
               string('' errors found in simulation''));
            STD.TEXTIO.writeline(RESULTS, TX_OUT);
            ASSERT (FALSE) REPORT ''Errors found during simulation''
               SEVERITY FAILURE;
         END IF;
      END PROCESS;

END testbench_arch;
```

6.1 Conclusion:

This project builds upon the concept of image shape detection by Hough Transform and models that concept into hardware using Cordic algorithm to eliminate multipliers. The adders, shifters, controllers and many more used to build this project have certainly helped me to get started in this field and understand their concepts and how to effectively use them. The differences between behavioral and structural modeling has also been learnt and their difference's understood. How to write test benches and use simulations to verify that the digital design is correct has been understood. Also, simulation software such as Xilinx ISE and Mentor ModelSim has been learnt at least to some degree. I would personally like to thank Prof Perkowski for allowing us to do this project.

7. Bibliography:

1. Internet References:

HOUGH TRANSFORM:

http://de.wikipedia.org/wiki/Hough-Transformation
http://en.wikipedia.org/wiki/Hough_**transform**
http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm
www.web.media.mit.edu/~lifton/acad/dcom/L11-02.pdf

Selected Books:

Fundamentals of Digital Image Processing – Anil K Jain, © 1998 Prentice Hall

CORDIC:

http://cnmat.berkeley.edu/~norbert/cordic/node6.html
http://cegt201.bradley.edu/projects/proj2003/dspproj/logmul.htm
http://www.andraka.com/**cordic**.htm
Books: Digital Design, 3rd Edition, 2001 – Morris Mano

VHDL:

Selected Books:

VHDL and FPLD's – Zoran Salcic, Kluwer academic publishers, 1998