Generation Of HDL Codes Of Different Image Processing Algorithms And Efficient DCT Architecture Using MATLAB HDL Coder

Sawan Singh

UIET CSJM University Kanpur

18 May - 5 July 2016

Summer Training Project Report.

Supervised by **Dr. Kishor P Sarawadekar**.
Department Of Electronics Engineering
Indian Institute Of Technology
BHU

भारतीय प्रौद्योगिकी संस्थान काशी हिन्दू विश्वविद्यालय IIT INDIAN INSTITUTE OF TECHNOLOGY BANARAS HINDU UNIVERSITY

Acknowledgment

I am highly indebted to **Dr. Kishor P Sarawadekar** sir for his guidance and constant supervision as well as for providing necessary information regarding the project and also for their support in completing the project.My thanks and appreciations also go to my friends in developing the project and people who have willingly helped me out with their abilities.

Sawan Singh
B-Tech Part III
UIET CSJM University

INDIAN INSTITUTE OF TECHNOLOGY

Department of Electronics Engineering
BHU, Varanasi 221005

CERTIFICATE

This is to certify that the Project entitled **"Generation Of HDL Codes Of Different Image Processing Algorithms And Efficient DCT Architecture Using MATLAB HDL Coder",** submitted by **SAWAN SINGH** of University Institute Of Engineering And Technology, CSJM University Kanpur is a record of bonafide work carried out by him. This work is done during 18 May - 5 July 2016, under our guidance.

Dr. Kishor P Sarawadekar	Prof. Satyabrata Jit
SUPERVISOR	**HEAD**

Contents

Chapter 1

Introduction

HDL Code generates portable, synthesizable Verilog and VHDL code from MATLAB functions, Simulink models, and Stateflow charts. The generated HDL code can be used for FPGA programming or ASIC prototyping and design. HDL Coder provides a workflow advisor that automates the programming of Xilinx and Altera FPGAs. You can control HDL architecture and implementation, highlight critical paths, and generate hardware resource utilization estimates. HDL Coder provides traceability between your Simulink model and the generated Verilog and VHDL code, enabling code verification for high-integrity applications adhering to DO-254 and other standards.

1.1 Key Feature

- Target-independent, synthesizable VHDL and Verilog code
- Code generation support for MATLAB functions, System objects, and Simulink blocks
- Mealy and Moore finite-state machines and control logic implementations using Stateflow
- Workflow advisor for programming Xilinx and Altera application boards

- Resource sharing and retiming for area-speed tradeoff
- Code-to-model and model-to-code traceability for DO-254
- Legacy code integration

1.2 HDL Workflow Advisor

The HDL Workflow Advisor helps automate the steps and provides a guided path from MATLAB to hardware. You can see the following key steps of the workflow in the left pane of the workflow advisor.

- Fixed-Point Conversion
- HDL Code Generation
- HDL Verification
- HDL Synthesis and Analysis

1.2.1 Fixed-Point Conversion

Signal processing applications are typically implemented using floating-point operations in MATLAB. However, for power, cost, and performance reasons, these algorithms need to be converted to use fixed-point operations when targeting hardware. Fixed-point conversion can be very challenging and time-consuming, typically demanding 25 to 50 percent of the total design and implementation time. The automatic floating-point to fixed-point conversion workflow in HDL Coder can greatly simplify and accelerate this conversion process.

The floating-point to fixed-point conversion workflow consists of the following steps:

- Verify that the floating-point design is compatible with code generation.

- Propose fixed-point types based on computed ranges, either through the simulation of the testbench or through static analysis that propagates design ranges to compute derived ranges for all the variables.
- Generate fixed-point MATLAB code by applying proposed fixed-point types.
- Verify the generated fixed-point code and compare the numerical accuracy of the generated fixed-point code with the original floating point code.

1.2.2 HDL Code Generation

The HDL Code Generation step generates HDL code from the fixed-point MATLAB code. You can generate either VHDL or Verilog code that implements your MATLAB design. In addition to generating synthesizable HDL code, HDL CoderâĎć also generates various reports, including a traceability report that helps you navigate between your MATLAB code and the generated HDL code, and a resource utilization report that shows you, at the algorithm level, approximately what hardware resources are needed to implement the design, in terms of adders, multipliers, and RAMs. During code generation, you can specify various optimization options to explore the design space without having to modify your algorithm. In the Design Space Exploration and Optimization Options section below, you can see how you can modify code generation options and optimize your design for speed or area.

1.2.3 HDL Verification

Standalone HDL test bench generation: HDL Coder generates VHDL and Verilog test benches from your MATLAB scripts for rapid verification of generated HDL code. You can customize an HDL test bench using a variety of options that apply stimuli to the HDL code. You can also generate script files to automate the process of compiling and simulating your code in HDL simulators. These steps help to ensure the results of MATLAB simulation match the results of HDL simulation.

HDL Coder also works with HDL Verifier to automatically generate two types of testbenches:

- HDL cosimulation-based verification works with Mentor Graphics ModelSim and QuestaSim, where MATLAB and HDL simulation happen in lockstep.
- FPGA-in-the-Loop simulation allows you to run a MATLAB simulation with an FPGA board in strict synchronization. You can use MATLAB to feed real world data into your design on the FPGA, and ensure that the algorithm will behave as expected when implemented in hardware.

1.2.4 HDL Synthesis

Apart from the language-related challenges, programming for FPGAs requires the use of complex EDA tools. Generating a bitstream from the HDL design and programming the FPGA can be daunting tasks. HDL Coder provides automation here, by creating project files for Xilinx and Altera that are configured with the generated HDL code. You can use the workflow steps to synthesize the HDL code within the MATLAB environment, see the results of synthesis, and iterate on the MATLAB design to improve synthesis results.

Chapter 2

Low Pass Filtering Of Image

Low pass filtering remove the high frequency component, in image high frequency means the occurrence of a particular color many times.

This code is ready for HDL code generation so that the low pass can be implemented on a chip.

2.0.1 Code

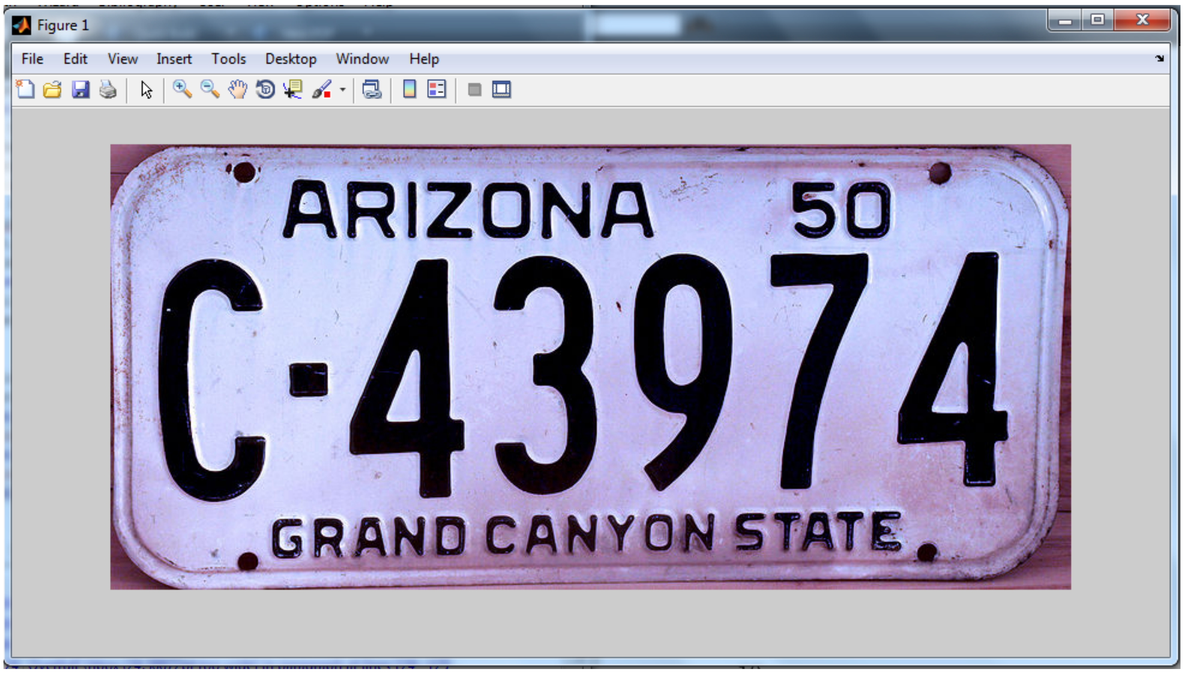
```
function [x_out, y_out,r_out,g_out,b_out]=hdllpf13(x_in, y_in, r_in, g_in, b_in)
persistent OrigImg
persistent x1 x2 y1 y2

if isempty(OrigImg)
    OrigImg = zeros(3,1);
    x1 = 0;
    x2 = 0;
    y1 = 0;
    y2 = 0;
end

D = [1/9 1/9 1/9;
    1/9 1/9 1/9;
    1/9 1/9 1/9];

RGB = [r_in; g_in; b_in];

OrigImg_1 = D*RGB;
r_out=OrigImg_1(1);
```

```
g_out=OrigImg_1(2);
b_out=OrigImg_1(3);
x_out = x2;
x2 = x1;
x1 = x_in;
y_out = y2;
y2 = y1;
y1 = y_in;
end
```

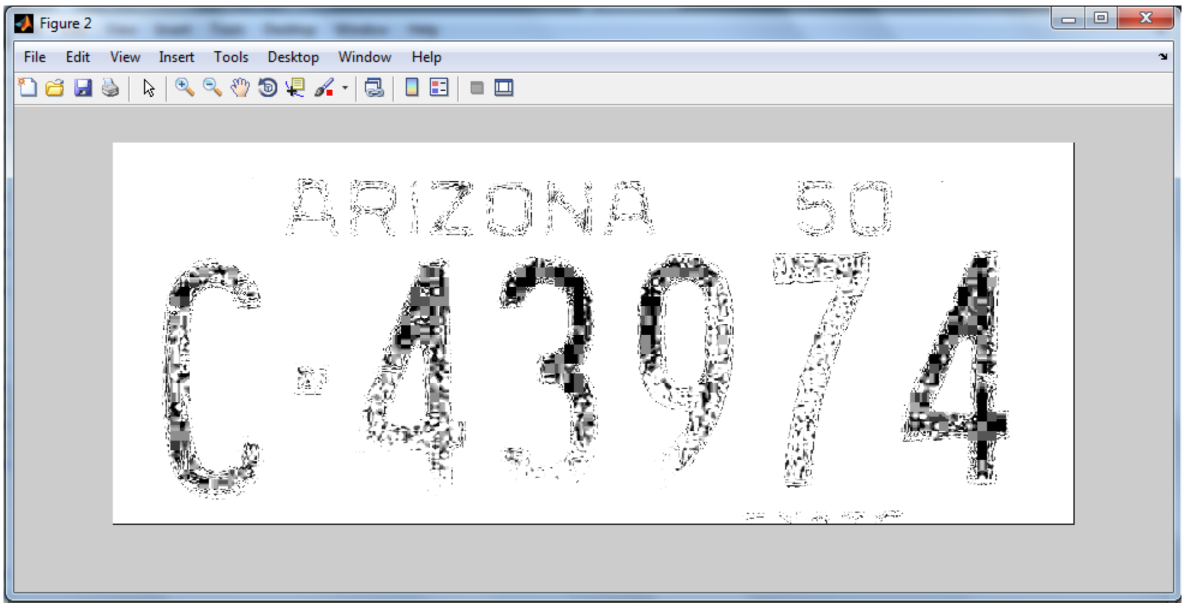
2.0.2 Test Bench

```
WIDTH = 800;%u
HEIGHT = 317;%v
rout=zeros(317,800);
ImgData = double(imread('AZ.jpg'));
    ImgOut = zeros(HEIGHT, WIDTH, 3);
    for y = 0:HEIGHT
        for x = 0:WIDTH
            if y >= 0 && y < HEIGHT && x >= 0 && x < WIDTH
                b = ImgData(y+1,x+1,1);
                g = ImgData(y+1,x+1,2);
                r = ImgData(y+1,x+1,3);
            else
                b = 0;
                g = 0;
                r = 0;
            end
            [xOut, yOut,rout,gout,bout] =hdllpf13(x, y, r, g, b);
            Iout(y+1,x+1,1)=bout;
            Iout(y+1,x+1,2)=gout;
            Iout(y+1,x+1,3)=rout;
        end
    end
    figure(1)
imshow(uint8(ImgData));
figure(2)
imshow(double(Iout),[]);
```

2.0.3 Input Image



2.0.4 Output Image



2.0.5 Generated VHDL Code

```
-- -------------------------------------------------------------
--
-- File Name: G:\AI\IIT BHU\codegen\hdllpf13\hdlsrc\hdllpf13_FixPt.vhd
-- Created: 2016-05-24 16:43:11
--
-- Generated by MATLAB 8.1, MATLAB Coder 2.4 and HDL Coder 3.2
--
--
--
-- -------------------------------------------------------------
-- Rate and Clocking Details
-- -------------------------------------------------------------
-- Design base rate: 1
--
--
-- Clock Enable  Sample Time
-- -------------------------------------------------------------
-- ce_out        1
-- -------------------------------------------------------------
--
--
-- Output Signal                 Clock Enable  Sample Time
-- -------------------------------------------------------------
-- x_out                         ce_out        1
-- y_out                         ce_out        1
-- r_out                         ce_out        1
-- g_out                         ce_out        1
-- b_out                         ce_out        1
-- -------------------------------------------------------------
--
-- -------------------------------------------------------------


-- -------------------------------------------------------------
--
-- Module: hdllpf13_FixPt
-- Source Path: hdllpf13_FixPt
-- Hierarchy Level: 0
--
-- -------------------------------------------------------------
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.hdllpf13_FixPt_pkg.ALL;
```

```
ENTITY hdllpf13_FixPt IS
  PORT( clk                              :   IN    std_logic;
        reset                            :   IN    std_logic;
        clk_enable                       :   IN    std_logic;
        x_in                             :   IN    std_logic_vector(9 DOWNTO 0);
-- ufix10
        y_in                             :   IN    std_logic_vector(8 DOWNTO 0);
-- ufix9
        r_in                             :   IN    std_logic_vector(8 DOWNTO 0);
-- ufix9
        g_in                             :   IN    std_logic_vector(8 DOWNTO 0);
-- ufix9
        b_in                             :   IN    std_logic_vector(8 DOWNTO 0);
-- ufix9
        ce_out                           :   OUT   std_logic;
        x_out                            :   OUT   std_logic_vector(9 DOWNTO 0);
-- ufix10
        y_out                            :   OUT   std_logic_vector(8 DOWNTO 0);
-- ufix9
        r_out                            :   OUT   std_logic_vector(13 DOWNTO 0);
-- ufix14_En7
        g_out                            :   OUT   std_logic_vector(13 DOWNTO 0);
-- ufix14_En7
        b_out                            :   OUT   std_logic_vector(13 DOWNTO 0)
-- ufix14_En7
        );
END hdllpf13_FixPt;


ARCHITECTURE rtl OF hdllpf13_FixPt IS

  -- Signals
  SIGNAL enb                             : std_logic;
  SIGNAL x_in_unsigned                   : unsigned(9 DOWNTO 0);
-- ufix10
  SIGNAL tmp                             : unsigned(9 DOWNTO 0);
-- ufix10
  SIGNAL x1                              : unsigned(9 DOWNTO 0);
-- ufix10
  SIGNAL x1_1                            : unsigned(9 DOWNTO 0);
-- ufix10
  SIGNAL tmp_1                           : unsigned(9 DOWNTO 0);
-- ufix10
  SIGNAL tmp_2                           : unsigned(9 DOWNTO 0);
-- ufix10
  SIGNAL x2                              : unsigned(9 DOWNTO 0);
-- ufix10
  SIGNAL x2_1                            : unsigned(9 DOWNTO 0);
-- ufix10
```

```
  SIGNAL tmp_3                           : unsigned(9 DOWNTO 0);
-- ufix10
  SIGNAL x_out_1                         : unsigned(9 DOWNTO 0);
-- ufix10
  SIGNAL x_out_2                         : unsigned(9 DOWNTO 0);
-- ufix10
  SIGNAL y_in_unsigned                   : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL tmp_4                           : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL y1                              : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL y1_1                            : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL tmp_5                           : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL tmp_6                           : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL y2                              : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL y2_1                            : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL tmp_7                           : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL y_out_1                         : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL y_out_2                         : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL r_in_unsigned                   : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL g_in_unsigned                   : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL b_in_unsigned                   : unsigned(8 DOWNTO 0);
-- ufix9
  SIGNAL RGB                             : vector_of_unsigned9(0 TO 2);
-- ufix9 [3]
  SIGNAL c                               : vector_of_unsigned25(0 TO 2);
-- ufix25_En17 [3]
  SIGNAL c_1                             : vector_of_unsigned25(0 TO 2);
-- ufix25_En17 [3]
  SIGNAL OrigImg_1                       : vector_of_unsigned14(0 TO 2);
-- ufix14_En7 [3]
  SIGNAL r_out_1                         : unsigned(13 DOWNTO 0);
-- ufix14_En7
  SIGNAL r_out_2                         : unsigned(13 DOWNTO 0);
-- ufix14_En7
  SIGNAL g_out_1                         : unsigned(13 DOWNTO 0);
-- ufix14_En7
  SIGNAL g_out_2                         : unsigned(13 DOWNTO 0);
```

```
-- ufix14_En7
  SIGNAL b_out_1                          : unsigned(13 DOWNTO 0);
-- ufix14_En7
  SIGNAL b_out_2                          : unsigned(13 DOWNTO 0);
-- ufix14_En7

BEGIN
  x_in_unsigned <= unsigned(x_in);

  tmp <= x_in_unsigned;

  x1 <= tmp;

  enb <= clk_enable;

  x1_reg_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      x1_1 <= to_unsigned(0, 10);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        x1_1 <= x1;
      END IF;
    END IF;
  END PROCESS x1_reg_process;

  tmp_1 <= x1_1;
  tmp_2 <= tmp_1;
  x2 <= tmp_2;
  x2_reg_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      x2_1 <= to_unsigned(0, 10);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        x2_1 <= x2;
      END IF;
    END IF;
  END PROCESS x2_reg_process;

  tmp_3 <= x2_1;
  x_out_1 <= tmp_3;
  x_out_2 <= x_out_1;
  x_out <= std_logic_vector(x_out_2);
  y_in_unsigned <= unsigned(y_in);
  tmp_4 <= y_in_unsigned;
  y1 <= tmp_4;

  y1_reg_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      y1_1 <= to_unsigned(0, 9);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        y1_1 <= y1;
      END IF;
    END IF;
  END PROCESS y1_reg_process;

  tmp_5 <= y1_1;
  tmp_6 <= tmp_5;
  y2 <= tmp_6;

  y2_reg_process : PROCESS (clk, reset)
  BEGIN
    IF reset = '1' THEN
      y2_1 <= to_unsigned(0, 9);
    ELSIF clk'EVENT AND clk = '1' THEN
      IF enb = '1' THEN
        y2_1 <= y2;
      END IF;
    END IF;
  END PROCESS y2_reg_process;

  tmp_7 <= y2_1;
  y_out_1 <= tmp_7;
  y_out_2 <= y_out_1;
  y_out <= std_logic_vector(y_out_2);
  r_in_unsigned <= unsigned(r_in);
  g_in_unsigned <= unsigned(g_in);
  b_in_unsigned <= unsigned(b_in);

  --spssa
  --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  --
%
  --        Generated by MATLAB 8.1, MATLAB Coder 2.4 and HDL Coder 3.2
%
  --
%
  --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  RGB(0) <= r_in_unsigned;
  RGB(1) <= g_in_unsigned;
  RGB(2) <= b_in_unsigned;

  c <= (OTHERS => to_unsigned(0, 25));

  p4_output : PROCESS (RGB, c)
```

```
    VARIABLE c1 : vector_of_unsigned25(0 TO 2);
    VARIABLE add_cast : vector_of_unsigned26(0 TO 2);
    VARIABLE mul_temp : vector_of_unsigned23(0 TO 2);
    VARIABLE add_cast_0 : vector_of_unsigned26(0 TO 2);
    VARIABLE add_temp : vector_of_unsigned26(0 TO 2);
  BEGIN
    c1 := c;
    c_1 <= c;

    FOR l IN 0 TO 2 LOOP

      FOR k IN 0 TO 2 LOOP
        add_cast(k) := resize(c1(l), 26);
        mul_temp(k) := to_unsigned(14564, 14) * RGB(k);
        add_cast_0(k) := resize(mul_temp(k), 26);
        add_temp(k) := add_cast(k) + add_cast_0(k);
        c1(l) := add_temp(k)(24 DOWNTO 0);
      END LOOP;

      c_1 <= c1;
    END LOOP;

  END PROCESS p4_output;

  OrigImg_1_gen: FOR t_0 IN 0 TO 2 GENERATE
    OrigImg_1(t_0) <= c_1(t_0)(23 DOWNTO 10);
  END GENERATE OrigImg_1_gen;
  r_out_1 <= OrigImg_1(0);
  r_out_2 <= r_out_1;
  r_out <= std_logic_vector(r_out_2);
  g_out_1 <= OrigImg_1(1);
  g_out_2 <= g_out_1;
  g_out <= std_logic_vector(g_out_2);
  b_out_1 <= OrigImg_1(2);
  b_out_2 <= b_out_1;
  b_out <= std_logic_vector(b_out_2);
  ce_out <= clk_enable;

END rtl;
```

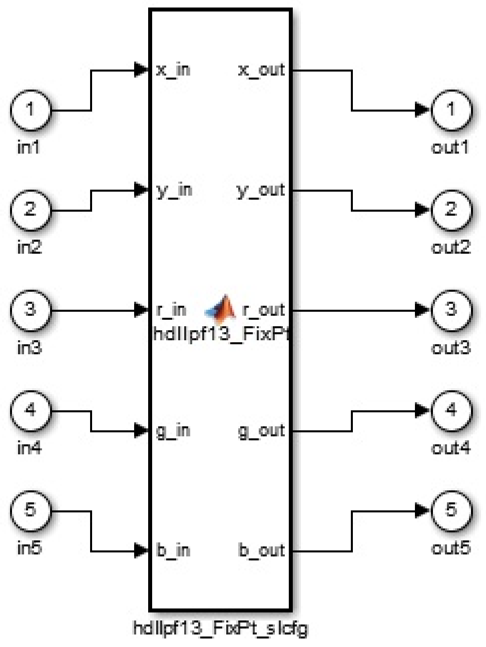
2.0.6 Generated Verilog Code

```
// ------------------------------------------------------------------
//
// File Name: G:\AI\IIT BHU\codegen\hdllpf13\hdlsrc\hdllpf13_FixPt.v
// Created: 2016-05-24 16:42:08
//
// Generated by MATLAB 8.1, MATLAB Coder 2.4 and HDL Coder 3.2
```

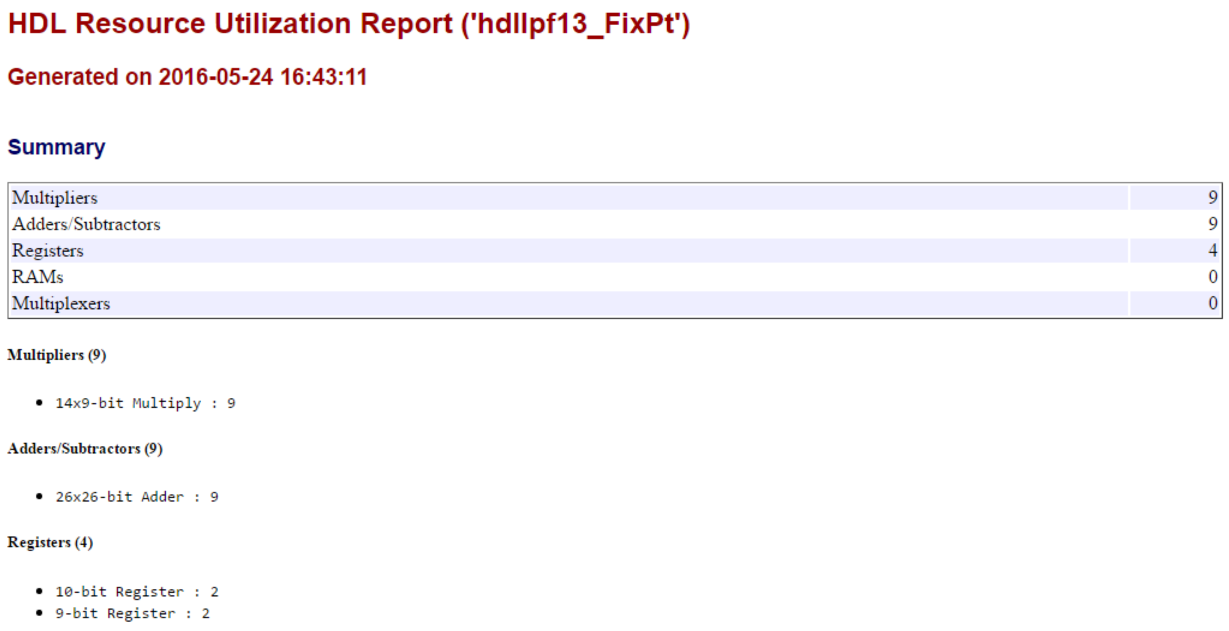
```
//
//
//
// -- ----------------------------------------------------------------
// -- Rate and Clocking Details
// -- ----------------------------------------------------------------
// Design base rate: 1
//
//
// Clock Enable  Sample Time
// -- ----------------------------------------------------------------
// ce_out        1
// -- ----------------------------------------------------------------
//
//
// Output Signal                 Clock Enable  Sample Time
// -- ----------------------------------------------------------------
// x_out                         ce_out        1
// y_out                         ce_out        1
// r_out                         ce_out        1
// g_out                         ce_out        1
// b_out                         ce_out        1
// -- ----------------------------------------------------------------
//
// ----------------------------------------------------------------


// ----------------------------------------------------------------
//
// Module: hdllpf13_FixPt
// Source Path: hdllpf13_FixPt
// Hierarchy Level: 0
//
// ----------------------------------------------------------------

`timescale 1 ns / 1 ns

module hdllpf13_FixPt
          (
           clk,
           reset,
           clk_enable,
           x_in,
           y_in,
           r_in,
           g_in,
           b_in,
           ce_out,
           x_out,
           y_out,
           r_out,
           g_out,
           b_out
          );


  input   clk;
  input   reset;
  input   clk_enable;
  input   [9:0] x_in;  // ufix10
  input   [8:0] y_in;  // ufix9
  input   [8:0] r_in;  // ufix9
  input   [8:0] g_in;  // ufix9
  input   [8:0] b_in;  // ufix9
  output  ce_out;
  output  [9:0] x_out;  // ufix10
  output  [8:0] y_out;  // ufix9
  output  [13:0] r_out;  // ufix14_En7
  output  [13:0] g_out;  // ufix14_En7
  output  [13:0] b_out;  // ufix14_En7


  wire enb;
  wire [9:0] tmp;  // ufix10
  wire [9:0] x1;  // ufix10
  reg [9:0] x1_1;  // ufix10
  wire [9:0] tmp_1;  // ufix10
  wire [9:0] tmp_2;  // ufix10
  wire [9:0] x2;  // ufix10
  reg [9:0] x2_1;  // ufix10
  wire [9:0] tmp_3;  // ufix10
  wire [9:0] x_out_1;  // ufix10
  wire [9:0] x_out_2;  // ufix10
  wire [8:0] tmp_4;  // ufix9
  wire [8:0] y1;  // ufix9
  reg [8:0] y1_1;  // ufix9
  wire [8:0] tmp_5;  // ufix9
  wire [8:0] tmp_6;  // ufix9
  wire [8:0] y2;  // ufix9
  reg [8:0] y2_1;  // ufix9
  wire [8:0] tmp_7;  // ufix9
  wire [8:0] y_out_1;  // ufix9
  wire [8:0] y_out_2;  // ufix9
  wire [8:0] RGB [0:2];  // ufix9 [3]
  wire [24:0] c [0:2];  // ufix25_En17 [3]
  reg [24:0] c_1 [0:2];  // ufix25_En17 [3]
  wire [13:0] OrigImg_1 [0:2];  // ufix14_En7 [3]
  wire [13:0] r_out_1;  // ufix14_En7
  wire [13:0] r_out_2;  // ufix14_En7
  wire [13:0] g_out_1;  // ufix14_En7
  wire [13:0] g_out_2;  // ufix14_En7
  wire [13:0] b_out_1;  // ufix14_En7
  wire [13:0] b_out_2;  // ufix14_En7
  reg signed [31:0] p4_l_1;  // int32
  reg [24:0] p4_c_1 [0:2];  // ufix25_En17 [3]
  reg signed [31:0] p4_k_1;  // int32
  reg signed [31:0] p4_t_0_1;  // int32
  reg signed [31:0] p4_t_1_1;  // int32
  reg [25:0] p4_add_cast_1 [0:2];  // ufix26_En17 [3]
  reg [22:0] p4_mul_temp_1 [0:2];  // ufix23_En17 [3]
  reg [25:0] p4_add_cast_0_1 [0:2];  // ufix26_En17 [3]
  reg [25:0] p4_add_temp_1 [0:2];  // ufix26_En17 [3]

  assign tmp = x_in;
  assign x1 = tmp;
  assign enb = clk_enable;

  always @(posedge clk or posedge reset)
    begin : x1_reg_process
      if (reset == 1'b1) begin
        x1_1 <= 10'b0000000000;
      end
      else begin
        if (enb) begin
          x1_1 <= x1;
        end
      end
    end

  assign tmp_1 = x1_1;
  assign tmp_2 = tmp_1;
  assign x2 = tmp_2;

  always @(posedge clk or posedge reset)
    begin : x2_reg_process
      if (reset == 1'b1) begin
        x2_1 <= 10'b0000000000;
      end
      else begin
        if (enb) begin
          x2_1 <= x2;
        end
      end
    end

  assign tmp_3 = x2_1;
  assign x_out_1 = tmp_3;
```

```
  assign x_out_2 = x_out_1;
  assign x_out = x_out_2;
  assign tmp_4 = y_in;
  assign y1 = tmp_4;

  always @(posedge clk or posedge reset)
    begin : y1_reg_process
      if (reset == 1'b1) begin
        y1_1 <= 9'b000000000;
      end
      else begin
        if (enb) begin
          y1_1 <= y1;
        end
      end
    end

  assign tmp_5 = y1_1;
  assign tmp_6 = tmp_5;
  assign y2 = tmp_6;

  always @(posedge clk or posedge reset)
    begin : y2_reg_process
      if (reset == 1'b1) begin
        y2_1 <= 9'b000000000;
      end
      else begin
        if (enb) begin
          y2_1 <= y2;
        end
      end
    end

  assign tmp_7 = y2_1;
  assign y_out_1 = tmp_7;
  assign y_out_2 = y_out_1;
  assign y_out = y_out_2;

  //spssa
  //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  //
%
  //         Generated by MATLAB 8.1, MATLAB Coder 2.4 and HDL Coder 3.2
%
  //
%
  //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  assign RGB[0] = r_in;
  assign RGB[1] = g_in;
  assign RGB[2] = b_in;

  assign c[0] = 25'b0000000000000000000000000;
  assign c[1] = 25'b0000000000000000000000000;
  assign c[2] = 25'b0000000000000000000000000;

  always @* begin

    for(p4_t_0_1 = 0; p4_t_0_1 <= 2; p4_t_0_1 = p4_t_0_1 + 1) begin
      p4_c_1[p4_t_0_1] = c[p4_t_0_1];
      c_1[p4_t_0_1] = c[p4_t_0_1];
    end

    for(p4_l_1 = 0; p4_l_1 <= 2; p4_l_1 = p4_l_1 + 1) begin

      for(p4_k_1 = 0; p4_k_1 <= 2; p4_k_1 = p4_k_1 + 1) begin
        p4_add_cast_1[p4_k_1] = p4_c_1[p4_l_1];
        p4_mul_temp_1[p4_k_1] = 14564 * RGB[p4_k_1];
        p4_add_cast_0_1[p4_k_1] = p4_mul_temp_1[p4_k_1];
        p4_add_temp_1[p4_k_1] = p4_add_cast_1[p4_k_1] + p4_add_cast_0_1[p4_k_1];
        p4_c_1[p4_l_1] = p4_add_temp_1[p4_k_1][24:0];
      end

      for(p4_t_1_1 = 0; p4_t_1_1 <= 2; p4_t_1_1 = p4_t_1_1 + 1) begin
        c_1[p4_t_1_1] = p4_c_1[p4_t_1_1];
      end

    end

  end

  assign OrigImg_1[0] = c_1[0][23:10];
  assign OrigImg_1[1] = c_1[1][23:10];
  assign OrigImg_1[2] = c_1[2][23:10];

  assign r_out_1 = OrigImg_1[0];
  assign r_out_2 = r_out_1;
  assign r_out = r_out_2;
  assign g_out_1 = OrigImg_1[1];
  assign g_out_2 = g_out_1;
  assign g_out = g_out_2;
  assign b_out_1 = OrigImg_1[2];
  assign b_out_2 = b_out_1;
  assign b_out = b_out_2;
  assign ce_out = clk_enable;

endmodule  // hdllpf13_FixPt
```

2.0.7 Simulink Model



2.0.8 Report



HDL Resource Utilization Report ('hdllpf13_FixPt')

Generated on 2016-05-24 16:43:11

Summary

Multipliers	9
Adders/Subtractors	9
Registers	4
RAMs	0
Multiplexers	0

Multipliers (9)

- `14x9-bit Multiply : 9`

Adders/Subtractors (9)

- `26x26-bit Adder : 9`

Registers (4)

- `10-bit Register : 2`
- `9-bit Register : 2`

Chapter 3

Edge Detection In Image

This section deals with the detection of edge in the image. Edge refers to the abrupt change in the intensity of in the image.

3.0.1 Code

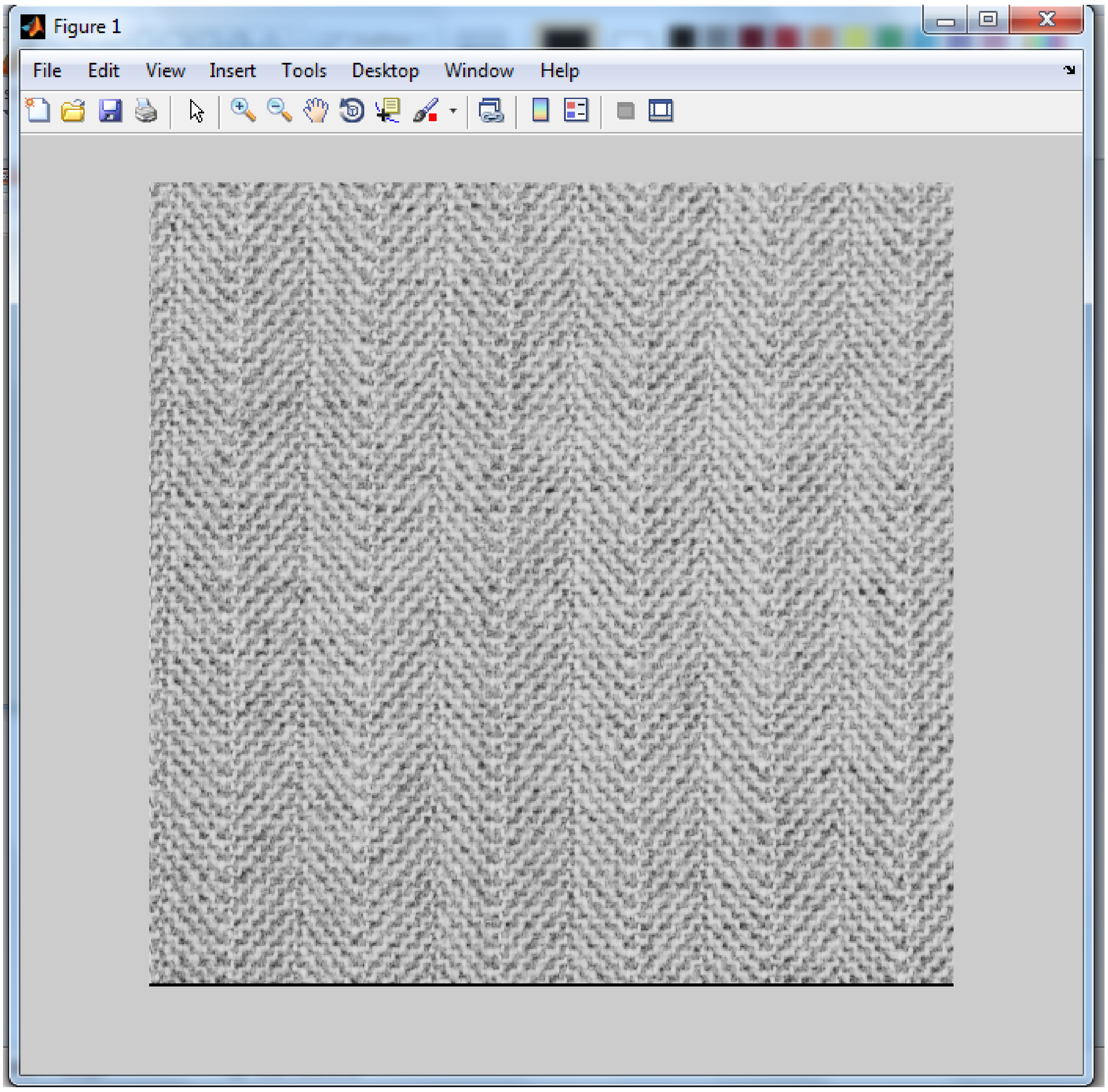
```
function [p1_out,p2_out] = vedge(b1_in,b2_in)
D = [-1 1]';
BSI = [b1_in b2_in];
OrigImg_1 = D*BSI;
%PD=abs((b1_in)-(b2_in));
    p1_out=OrigImg_1(1,1);
    p2_out=OrigImg_1(2,2);


end
```

3.0.2 Test Bench

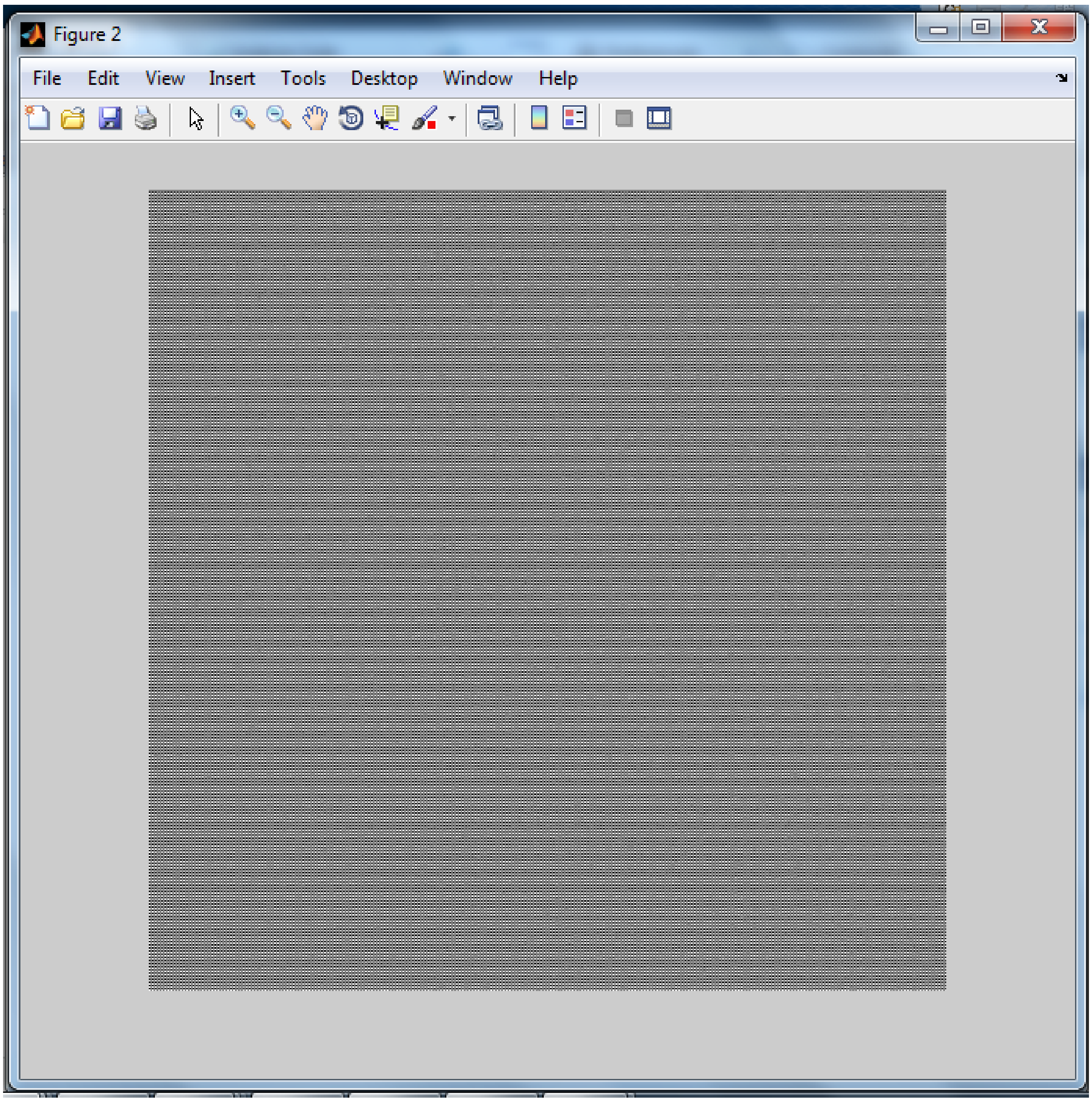
```
ImgData = double((imread('test.tiff')));
[WIDTH HEIGHT]=size(ImgData);
    ImgOut = zeros(HEIGHT, WIDTH, 3);
    for y = 1:2:HEIGHT-2

        for x = 1:2:WIDTH-2
            if y >= 0 && y < HEIGHT && x >= 0 && x < WIDTH
                p1 = ImgData(y,x+1);
                p2 = ImgData(y,x+2);

            else
```

```
                d = 0;
            end
            [a,b] =vedge(p1,p2);
            Iout(x+1,y)=a;
            Iout(x+2,y)=b;
        end

    end
imshow(uint8(ImgData));
figure(2)
imshow(Iout'>180,[]);
```

3.0.3 Input Image



3.0.4 Output Image



3.0.5 Generated VHDL Code

```
-- -------------------------------------------------------------
--
-- File Name: G:\AI\IIT BHU\codegen\vedge\hdlsrc\vedge_FixPt.vhd
-- Created: 2016-05-25 12:40:40
--
-- Generated by MATLAB 8.1, MATLAB Coder 2.4 and HDL Coder 3.2
--
--
--
-- ____________________________________________________________
-- Rate and Clocking Details
-- ____________________________________________________________
-- Design base rate: 1
--
--
-- Clock Enable  Sample Time
-- ____________________________________________________________
-- ce_out        1
-- ____________________________________________________________
--
--
-- Output Signal                 Clock Enable  Sample Time
-- ____________________________________________________________
-- p1_out                        ce_out        1
-- p2_out                        ce_out        1
-- ____________________________________________________________
--
-- ____________________________________________________________



-- ____________________________________________________________
--
-- Module: vedge_FixPt
-- Source Path: vedge_FixPt
-- Hierarchy Level: 0
--
-- ____________________________________________________________
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;
USE work.vedge_FixPt_pkg.ALL;

ENTITY vedge_FixPt IS
  PORT( clk                               :   IN    std_logic;
        reset                             :   IN    std_logic;
        clk_enable                        :   IN    std_logic;
        b1_in                             :   IN    std_logic_vector(7 DOWNTO 0);
-- ufix8
        b2_in                             :   IN    std_logic_vector(7 DOWNTO 0);
-- ufix8
        ce_out                            :   OUT   std_logic;
        p1_out                              :   OUT   std_logic_vector(8 DOWNTO 0);
-- sfix9
        p2_out                              :   OUT   std_logic_vector(7 DOWNTO 0)
-- ufix8
        );
END vedge_FixPt;

ARCHITECTURE rtl OF vedge_FixPt IS
  -- Constants
  CONSTANT nc : vector_of_signed3(0 TO 1):=(to_signed(-1,3),to_signed(1,3));--sfix3

  -- Signals
  SIGNAL enb                              : std_logic;
  SIGNAL b1_in_unsigned                   : unsigned(7 DOWNTO 0);
-- ufix8
  SIGNAL b2_in_unsigned                   : unsigned(7 DOWNTO 0);
-- ufix8
  SIGNAL p1_out_tmp                       : signed(8 DOWNTO 0);
-- sfix9
  SIGNAL p2_out_tmp                       : unsigned(7 DOWNTO 0);
-- ufix8

BEGIN
  b1_in_unsigned <= unsigned(b1_in);

  b2_in_unsigned <= unsigned(b2_in);

  enb <= clk_enable;

  vedge_FixPt_1_output : PROCESS (b1_in_unsigned, b2_in_unsigned)
    VARIABLE BSI : vector_of_unsigned8(0 TO 1);
    VARIABLE c : vector_of_signed11(0 TO 3);
    VARIABLE OrigImg_1 : vector_of_signed9(0 TO 3);
    VARIABLE add_cast : vector_of_signed64(0 TO 1);
    VARIABLE add_cast_0 : vector_of_signed64(0 TO 1);
    VARIABLE cast : vector_of_signed9(0 TO 1);
    VARIABLE mul_temp : vector_of_signed12(0 TO 1);
    VARIABLE add_cast_1 : vector_of_signed11(0 TO 1);
  BEGIN
    --spssa
    --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    --
%
    --        Generated by MATLAB 8.1, MATLAB Coder 2.4 and HDL Coder 3.2
%
    --
%
    --%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    BSI(0) := b1_in_unsigned;
    BSI(1) := b2_in_unsigned;
    c := (OTHERS => to_signed(0, 11));

    FOR l IN 0 TO 1 LOOP
      FOR m IN 0 TO 1 LOOP
        add_cast(m) := resize(to_signed(m, 32) & '0', 64);
        add_cast_0(m) := resize(to_signed(m, 32) & '0', 64);
        cast(m) := signed(resize(BSI(m), 9));
        mul_temp(m) := nc(l) * cast(m);
        add_cast_1(m) := mul_temp(m)(10 DOWNTO 0);
        c(to_integer(to_signed(l, 32) + resize(add_cast(m), 32))) := ...
                    c(to_integer(to_signed(l, 32) + resize(add_cast_0(m),32)))+...
                         add_cast_1(m);
      END LOOP;
    END LOOP;


    FOR t_0 IN 0 TO 3 LOOP
      OrigImg_1(t_0) := c(t_0)(8 DOWNTO 0);
    END LOOP;

    --PD=abs((b1_in)-(b2_in));
    p1_out_tmp <= OrigImg_1(0);
    p2_out_tmp <= unsigned(OrigImg_1(3)(7 DOWNTO 0));
  END PROCESS vedge_FixPt_1_output;

  p1_out <= std_logic_vector(p1_out_tmp);
  p2_out <= std_logic_vector(p2_out_tmp);
  ce_out <= clk_enable;

END rtl;
```

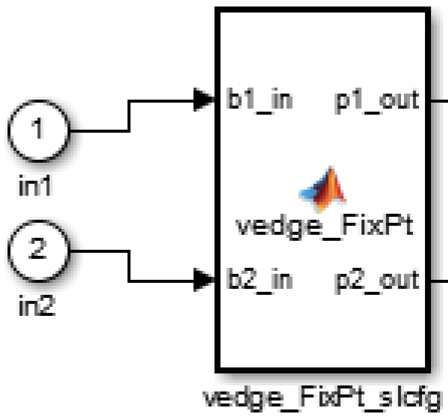
3.0.6 Generated Verilog Code

```
// -------------------------------------------------------------
//
// File Name: G:\AI\IIT BHU\codegen\vedge\hdlsrc\vedge_FixPt.v
// Created: 2016-07-02 16:41:14
//
// Generated by MATLAB 8.1, MATLAB Coder 2.4 and HDL Coder 3.2
//
//
//
// -- -------------------------------------------------------------
// -- Rate and Clocking Details
// -- -------------------------------------------------------------
// Design base rate: 1
```

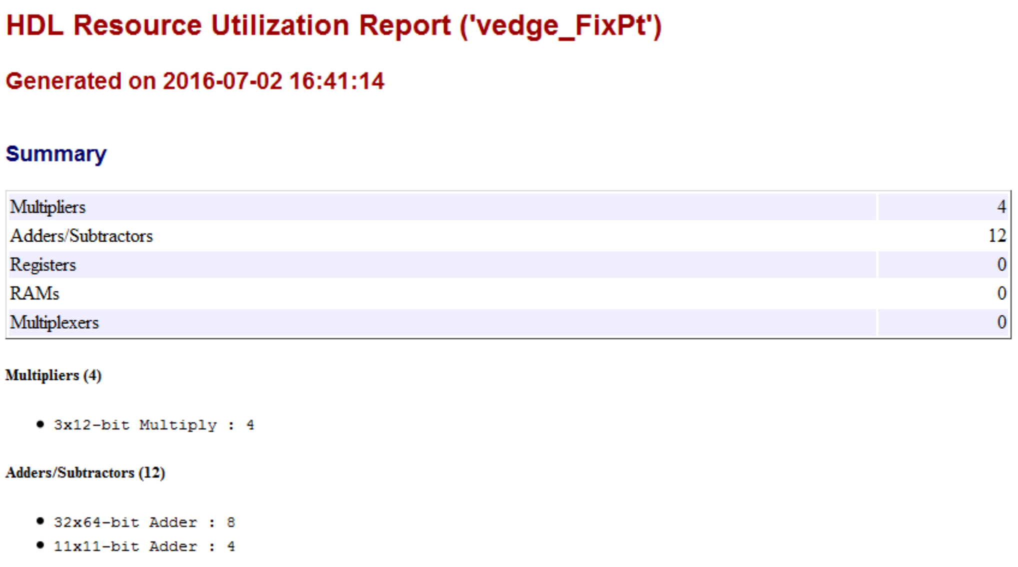
```
//
//
// Clock Enable  Sample Time
// -- -----------------------------------------------------------------
// ce_out        1
// -- -----------------------------------------------------------------
//
//
// Output Signal                  Clock Enable  Sample Time
// -- -----------------------------------------------------------------
// p1_out                         ce_out        1
// p2_out                         ce_out        1
// -- -----------------------------------------------------------------
//
// -------------------------------------------------------------------


// -------------------------------------------------------------------
//
// Module: vedge_FixPt
// Source Path: vedge_FixPt
// Hierarchy Level: 0
//
// -------------------------------------------------------------------

`timescale 1 ns / 1 ns

module vedge_FixPt
          (
           clk,
           reset,
           clk_enable,
           b1_in,
           b2_in,
           ce_out,
           p1_out,
           p2_out
          );


  input   clk;
  input   reset;
  input   clk_enable;
  input   [7:0] b1_in;   // ufix8
  input   [7:0] b2_in;   // ufix8
  output  ce_out;
  output  signed [8:0] p1_out;   // sfix9
  output  [7:0] p2_out;   // ufix8
  wire enb;
  reg signed [8:0] p1_out_1;  // sfix9
  reg [7:0] p2_out_1;  // ufix8
  reg [7:0] vedge_FixPt_BSI_1 [0:1];  // ufix8 [2]
  reg signed [10:0] vedge_FixPt_c_1 [0:3];  // sfix11 [4]
  reg signed [8:0] vedge_FixPt_OrigImg_1_1 [0:3];  // sfix9 [4]
  reg signed [31:0] vedge_FixPt_m_1;  // int32
  reg signed [31:0] vedge_FixPt_l_1;  // int32
  reg signed [31:0] vedge_FixPt_t_0_1;  // int32
  reg signed [31:0] vedge_FixPt_t_1_1;  // int32
  reg signed [2:0] vedge_FixPt_t_2_1 [0:1];  // sfix3 [2]
  reg signed [63:0] vedge_FixPt_add_cast_2 [0:1];  // sfix64 [2]
  reg signed [63:0] vedge_FixPt_add_cast_0_1 [0:1];  // sfix64 [2]
  reg signed [10:0] vedge_FixPt_add_cast_1_1 [0:1];  // sfix11 [2]


  assign enb = clk_enable;

  always @(b1_in, b2_in) begin
    vedge_FixPt_t_2_1[0] = 3'sb111;
    vedge_FixPt_t_2_1[1] = 3'sb001;
    //spssa
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    //
%
    //        Generated by MATLAB 8.1, MATLAB Coder 2.4 and HDL Coder 3.2
%
    //
%
    //%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    vedge_FixPt_BSI_1[0] = b1_in;
    vedge_FixPt_BSI_1[1] = b2_in;

    for(vedge_FixPt_t_0_1=0;vedge_FixPt_t_0_1<=3;vedge_FixPt_t_0_1= ...
                                                vedge_FixPt_t_0_1+1) begin
      vedge_FixPt_c_1[vedge_FixPt_t_0_1] = 11'sb00000000000;
    end

    for(vedge_FixPt_l_1 = 0; vedge_FixPt_l_1 <= 1; vedge_FixPt_l_1 = ...
                                                vedge_FixPt_l_1 + 1) begin
      for(vedge_FixPt_m_1 = 0; vedge_FixPt_m_1 <= 1; vedge_FixPt_m_1 = ...
                                                 vedge_FixPt_m_1 + 1) begin
        vedge_FixPt_add_cast_2[vedge_FixPt_m_1] = {{31{vedge_FixPt_m_1[31]}},...
                                                 {vedge_FixPt_m_1, 1'b0}};
        vedge_FixPt_add_cast_0_1[vedge_FixPt_m_1] = {{31{vedge_FixPt_m_1[31]}},...
                                                     {vedge_FixPt_m_1, 1'b0}};
        vedge_FixPt_add_cast_1_1[vedge_FixPt_m_1] = ...
                            vedge_FixPt_t_2_1[vedge_FixPt_l_1] *...
```

```
                             $signed({1'b0, vedge_FixPt_BSI_1[vedge_FixPt_m_1]});
        vedge_FixPt_c_1[vedge_FixPt_l_1 + ...
                        vedge_FixPt_add_cast_2[vedge_FixPt_m_1]] =...
                             vedge_FixPt_c_1[vedge_FixPt_l_1 + ...
                                vedge_FixPt_add_cast_0_1[vedge_FixPt_m_1]] +...
                                    vedge_FixPt_add_cast_1_1[vedge_FixPt_m_1];
      end
    end

    for(vedge_FixPt_t_1_1 = 0; vedge_FixPt_t_1_1 <= 3; vedge_FixPt_t_1_1 =...
                                     vedge_FixPt_t_1_1 + 1) begin
      vedge_FixPt_OrigImg_1_1[vedge_FixPt_t_1_1] = ...
                                    vedge_FixPt_c_1[vedge_FixPt_t_1_1][8:0];
    end

    //PD=abs((b1_in)-(b2_in));
    p1_out_1 = vedge_FixPt_OrigImg_1_1[0];
    p2_out_1 = vedge_FixPt_OrigImg_1_1[3][7:0];
  end


  assign ce_out = clk_enable;
  assign p1_out = p1_out_1;
  assign p2_out = p2_out_1;

endmodule  // vedge_FixPt
```

3.0.7 Simulink Model



3.0.8 Report



HDL Resource Utilization Report ('vedge_FixPt')

Generated on 2016-07-02 16:41:14

Summary

Multipliers	4
Adders/Subtractors	12
Registers	0
RAMs	0
Multiplexers	0

Multipliers (4)

- `3x12-bit Multiply : 4`

Adders/Subtractors (12)

- `32x64-bit Adder : 8`
- `11x11-bit Adder : 4`

Chapter 4

Efficient DCT Architecture

A discrete cosine transform (DCT) expresses a finite sequence of data points in terms of a sum of cosine functions oscillating at different frequencies. DCTs are important to numerous applications in science and engineering, from lossy compression of audio (e.g. MP3) and images (e.g. JPEG) (where small high-frequency components can be discarded), to spectral methods for the numerical solution of partial differential equations. The use of cosine rather than sine functions is critical for compression, since it turns out (as described below) that fewer cosine functions are needed to approximate a typical signal, whereas for differential equations the cosines express a particular choice of boundary conditions.

In particular, a DCT is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers. DCTs are equivalent to DFTs of roughly twice the length, operating on real data with even symmetry (since the Fourier transform of a real and even function is real and even), where in some variants the input and/or output data are shifted by half a sample. There are eight standard DCT variants, of which four are common. The most common variant of discrete cosine transform is the type-II DCT, which is often called simply ”the DCT”.[1][2] Its inverse, the type-III DCT, is correspondingly often called simply ”the inverse DCT” or ”the IDCT”. Two related transforms are the discrete sine transform (DST), which is equivalent to a DFT of real and odd functions, and the modified discrete cosine transform (MDCT), which is based on a DCT of overlapping data.

4.1 Architecture Used

Generally multiplier consumes large space and power in a digital circuit. To over come this problem we use shifting the numbers logically in order to multiply them with

$$2^n$$

where n is the number of bit shift towards left.

The DCT coefficient matrix has a special property. Given below is a 4x4 matrix here odd rows have the same absolute value.

$$\begin{bmatrix} d_{16} & d_{16} & d_{16} & d_{16} \\ d_8 & d_{24} & -d_{24} & -d_8 \\ d_{16} & -d_{16} & -d_{16} & d_{16} \\ d_{24} & -d_8 & d_8 & -d_{24} \end{bmatrix}$$

If we consider row 0 which is even than the first two elements are same as the other second value, but in the second row

$$d_8 = -d_8$$

and

$$d_{24} = -d_{24}$$

so we can see that the even rows are even symmetric and the odd are odd symmetric. The Even-Odd decomposition of the inverse transform of an N point input consists of the following three steps:-

- Calculate the even part using a $N/2 * N/2$ subet matrix obtained from the even columns of the inverse transform matrix.

$$\begin{bmatrix} z_0 \\ z_1 \end{bmatrix} = \begin{bmatrix} d_{16} & d_{16} \\ d_{16} & -d_{16} \end{bmatrix} \begin{bmatrix} x_0 \\ x_2 \end{bmatrix}$$

- Calculate the odd part using $N/2 * N/2$ subset matrix obtained from the odd columns of the inverse transform matrix.

$$t_0 = d_{16}x_0$$
$$t_1 = d_{16}x_2$$
$$\begin{bmatrix} z_0 \\ z_1 \end{bmatrix} = \begin{bmatrix} t_0 + t_1 \\ t_0 - t_1 \end{bmatrix}$$

- Add/subtract the odd and even parts to generate N point output.

$$\begin{bmatrix} z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} -d_{24} & d_8 \\ -d_8 & -d_{24} \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix}$$

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} z_0 - z_3 \\ z_1 - z_2 \\ z_1 + z_2 \\ z_0 + z_3 \end{bmatrix}$$

Similarly for the $8 * 8$ DCT matrix

$$\mathbf{D}_8 =$$
$$\begin{bmatrix} d_{16} & d_{16} & d_{16} & d_{16} & d_{16} & d_{16} & d_{16} & d_{16} \\ d_4 & d_{12} & d_{20} & d_{28} & -d_{28} & -d_{20} & -d_{12} & -d_4 \\ d_8 & d_{24} & -d_{24} & -d_8 & -d_8 & -d_{24} & d_{24} & d_8 \\ d_{12} & -d_{28} & -d_4 & -d_{20} & d_{20} & d_4 & d_{28} & -d_{12} \\ d_{16} & -d_{16} & -d_{16} & d_{16} & d_{16} & -d_{16} & -d_{16} & d_{16} \\ d_{20} & -d_4 & d_{28} & d_{12} & -d_{12} & -d_{28} & d_4 & -d_{20} \\ d_{24} & -d_8 & d_8 & -d_{24} & -d_{24} & d_8 & -d_8 & -d_{24} \\ d_{28} & -d_{20} & d_{12} & -d_4 & d_4 & -d_{12} & d_{20} & -d_{28} \end{bmatrix}$$

$$\begin{bmatrix} z_0 \\ z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} d_{16} & d_8 & d_{16} & d_{24} \\ d_{16} & d_{24} & -d_{16} & -d_8 \\ d_{16} & -d_{24} & -d_{16} & d_8 \\ d_{16} & -d_8 & d_{16} & -d_{24} \end{bmatrix} \begin{bmatrix} x_0 \\ x_2 \\ x_4 \\ x_6 \end{bmatrix}$$

$$\begin{bmatrix} z_4 \\ z_5 \\ z_6 \\ z_7 \end{bmatrix} = \begin{bmatrix} -d_{28} & d_{20} & -d_{12} & d_4 \\ -d_{20} & d_4 & -d_{28} & -d_{12} \\ -d_{12} & d_{28} & d_4 & d_{20} \\ -d_4 & -d_{12} & -d_{20} & -d_{28} \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \\ x_5 \\ x_7 \end{bmatrix}$$

4.2 Code for 4x4 DCT

```
function [d,Y1,Y3]=hdldct(i,j,m,n,x0,x1,x2,x3)
        t1=pi*(i-1);
        t2=(2*j-1)/(2*n);
        t3=t1*t2;
       if i==1
            d= round(sqrt(1/2)*sqrt(2/n)*cos(t3)*m);
       else
            d= round(sqrt(2/4)*cos(t3)*m);
       end
    data1=x0-x3;
    data2=x1-x2;
    k2 = double(2);
    k6 = double(6);
    k5 = double(5);
    k4 = double(4);
    k1=  double(1);
    d1 = int16(data1);
    d2 = int16(data2);
    Bn1=bitsll(d1,k6)+bitsll(d1,k4)+bitsll(d1,k2);
    Bn2=bitsll(d2,k5)+bitsll(d2,k1)+d2;
    Y1=Bn1+Bn2;
    Bn12=bitsll(d1,k5)+bitsll(d1,k1)+d1;
    Bn11=bitsll(d2,k6)+bitsll(d2,k4)+bitsll(d2,k2);
    Y3=Bn12-Bn11;
end
```

4.2.1 Test Bench

```
function test_hdldct4()
m_dct=zeros(4);
 MUL=power(2,7);
for k=1:4
    for l=1:4
        [d,y1,y3]=hdldct(k,l,MUL,4,3,3,1,1);
        m_dct(k,l)=d;
    end
end
disp(m_dct)
disp(y1)
disp(y3)
end
```

4.3 Code for 8x8 DCT

```
function [d,Y1,Y3,Y5,Y7]=hdldct8(i,j,m,n,x0,x1,x2,x3,x4,x5,x6,x7)
         t1=pi*(i−1);
         t2=(2*j−1)/(2*n);
         t3=t1*t2;
        if i==1
             d= round(sqrt(1/2)*sqrt(2/n)*cos(t3)*m);
        else
             d= round(sqrt(2/4)*cos(t3)*m);
        end

        if i==4  &&  j==4
         b1=x0−x7;
         b2=x1−x6;
         b3=x2−x5;
         b4=x3−x4;

         k1=  double(1);
         k2 = double(2);
         k3 = double(3);
         k4 = double(4);
         k5 = double(5);
         k6=  double(6);

         d1 = int16(b1);
         d2 = int16(b2);
         d3 = int16(b3);
         d4 = int16(b4);

         Y1=bitsll(d1,k6)+bitsll(d1,k4)+bitsll(d1,k3)+b1+bitsll(d2,k6)+ ...
             bitsll(d2,k3)+bitsll(d2,k1)+bitsll(d3,k5)+bitsll(d3,k4)+ ...
             bitsll(d3,k1)+bitsll(d4,k4)+bitsll(d4,k1);

         Y3=b2+bitsll(d1,k6)+bitsll(d1,k3)+bitsll(d1,k1)−b3−bitsll(d3,k6)− ...
             bitsll(d3,k4)−bitsll(d3,k3)−bitsll(d4,k5)−bitsll(d4,k4)− ...
             bitsll(d4,k1)−bitsll(d2,k4)−bitsll(d2,k1);

          Y5=−b2−bitsll(d2,k6)−bitsll(d2,k4)−bitsll(d2,k3)+b4+ ...
              bitsll(d4,k6)+bitsll(d4,k3)+bitsll(d4,k1)+bitsll(d1,k5)+ ...
              bitsll(d1,k4)+bitsll(d1,k1)+bitsll(d3,k4)+bitsll(d3,k1);

          Y7=−b4−bitsll(d4,k6)−bitsll(d4,k4)−bitsll(d4,k3)+b3+ ...
              bitsll(d3,k6)+bitsll(d3,k3)+bitsll(d3,k1)−bitsll(d2,k5)− ...
              bitsll(d2,k4)−bitsll(d2,k1)+bitsll(d1,k4)+bitsll(d1,k1);
        end
end
```

4.3.1 Test Bench

```
function test_hdldct8()
m_dct=zeros(8);
 MUL=power(2,7.5);
for k=1:8
    for l=1:8
        [d,y1,y3,y5,y7]=hdldct8(k,l,MUL,8,4,4,4,4,2,2,2,2);
        m_dct(k,l)=d;
    end
end
disp(m_dct)
disp(y1)
disp(y3)
disp(y5)
disp(y7)
end
```

4.4 Code for 16x16 DCT

```
function [d ,y1, y3, y5, y7, y9, y11, y13, y15]=hdldct16(i,j,m,n,x0,x1,x2, ...
                                  x3,x4,x5,x6,x7,x8,x9,x10, ...
                                  x11,x12,x13,x14,x15)
        t1=pi*(i-1);
        t2=(2*j-1)/(2*n);
        t3=t1*t2;
       if i==1
            d= round(sqrt(1/2)*sqrt(2/n)*cos(t3)*m);
       else
            d= round(sqrt(2/4)*cos(t3)*m);
       end
    b1=x0-x15;
    b2=x1-x14;
    b3=x2-x13;
    b4=x3-x12;
    b5=x4-x11;
    b6=x5-x10;
    b7=x6-x9;
    b8=x7-x8;

    k1=  double(1);
    k2 = double(2);
    k3 = double(3);
    k4 = double(4);
```

```
k5 = double(5);
k6=  double(6);

d1 = int32(b1);
d2 = int32(b2);
d3 = int32(b3);
d4 = int32(b4);
d5 = int32(b5);
d6 = int32(b6);
d7 = int32(b7);
d8 = int32(b8);

y1=bitsll(d1,k6)+bitsll(d1,k4)+bitsll(d1,k3)+bitsll(d1,k1)+bitsll(d2,k6)+ ...
    bitsll(d2,k4)+bitsll(d2,k2)+bitsll(d2,k1)+b2+bitsll(d3,k6)+ ...
    bitsll(d3,k4)+bitsll(d4,k6)+bitsll(d4,k2)+bitsll(d4,k1)+ ...
    bitsll(d5,k5)+bitsll(d5,k4)+bitsll(d5,k3)+b5+bitsll(d6,k5)+ ...
    bitsll(d6,k3)+bitsll(d6,k1)+b6+bitsll(d7,k4)+bitsll(d7,k3)+ ...
    bitsll(d7,k1)+bitsll(d8,k3)+b8;

y3=-bitsll(d6,k6)-bitsll(d6,k4)-bitsll(d6,k3)-bitsll(d6,k1)+bitsll(d1,k6)+ ...
    bitsll(d1,k4)+bitsll(d1,k2)+bitsll(d1,k1)+b1-bitsll(d5,k6)- ...
    bitsll(d5,k4)-bitsll(d7,k6)-bitsll(d7,k2)-bitsll(d7,k1)+ ...
    bitsll(d2,k5)+bitsll(d2,k4)+bitsll(d2,k3)+b2-bitsll(d4,k5)- ...
    bitsll(d4,k3)-bitsll(d4,k1)-b4-bitsll(d8,k4)-bitsll(d8,k3)- ...
    bitsll(d8,k1)+bitsll(d3,k3)+b3;

y5=bitsll(d7,k6)+bitsll(d7,k4)+bitsll(d7,k3)+bitsll(d7,k1)-bitsll(d4,k6)- ...
    bitsll(d4,k4)-bitsll(d4,k2)-bitsll(d4,k1)-b4+bitsll(d1,k6)+ ...
    bitsll(d1,k4)-bitsll(d3,k6)-bitsll(d3,k2)-bitsll(d3,k1)+ ...
    bitsll(d6,k5)+bitsll(d6,k4)+bitsll(d6,k3)+b6+bitsll(d8,k5)+ ...
    bitsll(d8,k3)+bitsll(d8,k1)+b8-bitsll(d5,k4)-bitsll(d5,k3)- ...
    bitsll(d5,k1)+bitsll(d2,k3)+b2;

y7=bitsll(d5,k6)+bitsll(d5,k4)+bitsll(d5,k3)+bitsll(d5,k1)-bitsll(d3,k6)- ...
    bitsll(d3,k4)-bitsll(d3,k2)-bitsll(d3,k1)-b3-bitsll(d7,k6)- ...
    bitsll(d7,k4)-bitsll(d7,k6)+bitsll(d1,k2)+bitsll(d1,k1)- ...
    bitsll(d8,k5)-bitsll(d8,k4)-bitsll(d8,k3)-b8-bitsll(d2,k5)- ...
    bitsll(d2,k3)-bitsll(d2,k1)-b2+bitsll(d6,k4)+bitsll(d6,k3)+ ...
    bitsll(d6,k1)+bitsll(d4,k3)+b4;

y9=bitsll(d4,k6)+bitsll(d4,k4)+bitsll(d4,k3)+bitsll(d4,k1)-bitsll(d6,k6)- ...
    bitsll(d6,k4)-bitsll(d6,k2)-bitsll(d6,k1)-b6-bitsll(d2,k6)- ...
    bitsll(d2,k4)-bitsll(d2,k6)+bitsll(d8,k2)+bitsll(d8,k1)+ ...
    bitsll(d1,k5)+bitsll(d1,k4)+bitsll(d1,k3)+b1+bitsll(d7,k5)+ ...
    bitsll(d7,k3)+bitsll(d7,k1)+b7-bitsll(d3,k4)-bitsll(d3,k3)- ...
    bitsll(d3,k1)-bitsll(d5,k3)-b5;

y11=-bitsll(d2,k6)-bitsll(d2,k4)-bitsll(d2,k3)-bitsll(d2,k1)-bitsll(d5,k6)- ..
    bitsll(d5,k4)-bitsll(d5,k2)-bitsll(d5,k1)-b5-bitsll(d8,k6)- ...
```

```
        bitsll(d8,k4)+bitsll(d6,k6)+bitsll(d6,k2)+bitsll(d6,k1)+ ...
        bitsll(d3,k5)+bitsll(d3,k4)+bitsll(d3,k3)+b3+bitsll(d1,k5)+ ...
        bitsll(d1,k3)+bitsll(d1,k1)+b1+bitsll(d4,k4)+bitsll(d4,k3)+ ...
        bitsll(d4,k1)+bitsll(d7,k3)+b7;

    y13=bitsll(d3,k6)+bitsll(d3,k4)+bitsll(d3,k3)+bitsll(d3,k1)+bitsll(d8,k6)+ ...
        bitsll(d8,k4)+bitsll(d8,k2)+bitsll(d8,k1)+b8-bitsll(d4,k6)- ...
        bitsll(d4,k4)-bitsll(d2,k6)-bitsll(d2,k2)-bitsll(d2,k1)- ...
        bitsll(d7,k5)-bitsll(d7,k4)-bitsll(d7,k3)-b7+bitsll(d5,k5)+ ...
        bitsll(d5,k3)+bitsll(d5,k1)+b5+bitsll(d1,k4)+bitsll(d1,k3)+ ...
        bitsll(d1,k1)+bitsll(d6,k3)+b6;

    y15=-bitsll(d8,k6)-bitsll(d8,k4)-bitsll(d8,k3)-bitsll(d8,k1)+bitsll(d7,k6)+ ..
        bitsll(d7,k4)+bitsll(d7,k2)+bitsll(d7,k1)+b7-bitsll(d6,k6)- ...
        bitsll(d6,k4)-bitsll(d6,k6)-bitsll(d6,k2)-bitsll(d6,k1)- ...
        bitsll(d4,k5)-bitsll(d4,k4)-bitsll(d4,k3)-b4+bitsll(d3,k5)+ ...
        bitsll(d3,k3)+bitsll(d3,k1)+b3-bitsll(d2,k4)-bitsll(d2,k3)- ...
        bitsll(d2,k1)+bitsll(d1,k3)+b1;
end
```

4.4.1 Test Bench

```
function test_hdldct16()
m_dct=zeros(16);
 MUL=power(2,8);
for k=1:16
    for l=1:16
        [d,y1,y3,y5,y7,y9,y11,y13,y15]=hdldct16(k,l,MUL,16,4,4,4,4,4,4,4, ...
                                            4,2,2,2,2,2,2,2,2);
        m_dct(k,l)=d;
    end
end
disp(m_dct)
disp(y1)
disp(y3)
disp(y5)
disp(y7)
disp(y9)
disp(y11)
disp(y13)
disp(y15)
end
```

4.5 Code for 32x32 DCT

```
function [d,y1, y3, y5 ,y7, y9, y11, y13, y15, y17, y19, y21, y23, y25,...
```

```
        y27, y29 ,y31]=hdldct32(i,j,m,n,x0,x1,x2,x3,x4,x5,x6,x7, ...
        x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18,x19,x20,x21, ...
        x22,x23,x24,x25,x26,x27,x28,x29,x30,x31)
    t1=pi*(i-1);
    t2=(2*j-1)/(2*n);
    t3=t1*t2;
   if i==1
        d= round(sqrt(1/2)*sqrt(2/n)*cos(t3)*m);
   else
        d= round(sqrt(2/4)*cos(t3)*m);
   end

b1=x0-x31;
b2=x1-x30;
b3=x2-x29;
b4=x3-x28;
b5=x4-x27;
b6=x5-x26;
b7=x6-x25;
b8=x7-x24;
b9=x8-x23;
b10=x9-x22;
b11=x10-x21;
b12=x11-x20;
b13=x12-x19;
b14=x13-x18;
b15=x14-x17;
b16=x15-x16;


k1=  double(1);
k2 = double(2);
k3 = double(3);
k4 = double(4);
k5 = double(5);
k6=  double(6);

d1 = int32(b1);
d2 = int32(b2);
d3 = int32(b3);
d4 = int32(b4);
d5 = int32(b5);
d6 = int32(b6);
d7 = int32(b7);
d8 = int32(b8);
d9 = int32(b9);
d10 = int32(b10);
d11 = int32(b11);
d12 = int32(b12);
```

```
d13 = int32(b13);
d14 = int32(b14);
d15 = int32(b15);
d16 = int32(b16);

y1=bitsll(d1,k6)+bitsll(d1,k4)+bitsll(d1,k3)+bitsll(d1,k1)+bitsll(d2,k6)+ ...
    bitsll(d2,k4)+bitsll(d2,k3)+bitsll(d2,k1)+bitsll(d3,k6)+ ...
    bitsll(d3,k4)+bitsll(d3,k3)+bitsll(d4,k6)+bitsll(d4,k4)+ ...
    bitsll(d4,k2)+b4+bitsll(d5,k6)+bitsll(d5,k4)+bitsll(d5,k1)+ ...
    bitsll(d6,k6)+bitsll(d6,k3)+bitsll(d6,k2)+bitsll(d6,k1)+ ...
    bitsll(d7,k6)+bitsll(d7,k3)+b7+bitsll(d8,k6)+bitsll(d8,k1)+ ...
    b8+bitsll(d9,k5)+bitsll(d9,k4)+bitsll(d9,k3)+bitsll(d9,k2)+b9+ ...
    bitsll(d10,k5)+bitsll(d10,k4)+bitsll(d10,k2)+bitsll(d10,k1)+ ...
    bitsll(d11,k5)+bitsll(d11,k3)+bitsll(d11,k2)+bitsll(d11,k1)+b11+ ...
    bitsll(d12,k5)+bitsll(d12,k2)+bitsll(d12,k1)+b12+bitsll(d13,k4)+ ...
    bitsll(d13,k3)+bitsll(d13,k2)+bitsll(d13,k1)+bitsll(d14,k4)+ ...
    bitsll(d14,k2)+bitsll(d14,k1)+bitsll(d15,k3)+bitsll(d15,k2)+b15+ ...
    bitsll(d16,k2);

y3=bitsll(d1,k6)+bitsll(d1,k4)+bitsll(d1,k3)+bitsll(d1,k1)+bitsll(d2,k6)- ...
    bitsll(d11,k4)-bitsll(d11,k3)-bitsll(d11,k1)-bitsll(d12,k6)- ...
    bitsll(d12,k4)-bitsll(d12,k3)-bitsll(d10,k6)-bitsll(d10,k4)- ...
    bitsll(d10,k2)+b10+bitsll(d2,k6)+bitsll(d2,k4)+bitsll(d2,k1)- ...
    bitsll(d13,k6)-bitsll(d13,k3)-bitsll(d13,k2)-bitsll(d13,k1)- ...
    bitsll(d9,k6)-bitsll(d9,k3)-b9+bitsll(d3,k6)+bitsll(d3,k1)+b3- ...
    bitsll(d14,k5)-bitsll(d14,k4)-bitsll(d14,k3)-bitsll(d14,k2)-b14- ...
    bitsll(d8,k5)-bitsll(d8,k4)-bitsll(d8,k2)-bitsll(d8,k1)+bitsll(d4,k5)+ ..
    bitsll(d4,k3)+bitsll(d4,k2)+bitsll(d4,k1)+b4-bitsll(d15,k5)- ...
    bitsll(d15,k2)-bitsll(d15,k1)-b15-bitsll(d7,k4)-bitsll(d7,k3)- ...
    bitsll(d7,k2)-bitsll(d7,k1)+bitsll(d5,k4)+bitsll(d5,k2)+bitsll(d5,k1)- ..
    bitsll(d16,k3)-bitsll(d16,k2)-b16-bitsll(d6,k2);

y5=-bitsll(d7,k6)-bitsll(d7,k4)-bitsll(d7,k3)-bitsll(d7,k1)+bitsll(d13,k6)+ ..
    bitsll(d13,k4)+bitsll(d13,k3)+bitsll(d13,k1)+bitsll(d1,k6)+ ...
    bitsll(d1,k4)+bitsll(d1,k3)+bitsll(d12,k6)+bitsll(d12,k4)+ ...
    bitsll(d12,k2)+b12-bitsll(d6,k6)-bitsll(d6,k4)-bitsll(d6,k1)- ...
    bitsll(d8,k6)-bitsll(d8,k3)-bitsll(d8,k2)-bitsll(d8,k1)+ ...
    bitsll(d12,k6)+bitsll(d12,k3)+b12+bitsll(d2,k6)+bitsll(d2,k1)+ ...
    b2+bitsll(d15,k5)+bitsll(d15,k4)+bitsll(d15,k3)+bitsll(d15,k2)+ ...
    b15+bitsll(d10,k5)-bitsll(d5,k4)-bitsll(d5,k2)-bitsll(d5,k1)- ...
    bitsll(d9,k5)-bitsll(d9,k3)-bitsll(d9,k2)-bitsll(d9,k1)+b9+ ...
    bitsll(d12,k5)+bitsll(d11,k2)+bitsll(d11,k1)+b11+bitsll(d3,k4)+ ...
    bitsll(d3,k3)+bitsll(d3,k2)+bitsll(d3,k1)+bitsll(d16,k4)+ ...
    bitsll(d16,k2)+bitsll(d16,k1)-bitsll(d4,k3)-bitsll(d4,k2)- ...
    b10-bitsll(d10,k2);

y7=-bitsll(d5,k6)-bitsll(d5,k4)-bitsll(d5,k3)-bitsll(d5,k1)-bitsll(d14,k6)- ..
    bitsll(d14,k4)-bitsll(d14,k3)-bitsll(d14,k1)+bitsll(d10,k6)+ ...
    bitsll(d10,k4)+bitsll(d10,k3)+bitsll(d1,k6)+bitsll(d1,k4)+ ...
    bitsll(d1,k2)+b1+bitsll(d9,k6)+bitsll(d9,k4)+bitsll(d9,k1)- ...
    bitsll(d15,k6)-bitsll(d15,k3)-bitsll(d15,k2)-bitsll(d15,k1)- ...
    bitsll(d6,k6)-bitsll(d6,k3)-b6-bitsll(d4,k6)-bitsll(d4,k1)-b4- ...
    bitsll(d13,k5)-bitsll(d13,k4)-bitsll(d13,k3)-bitsll(d13,k2)-b13+ ...
    bitsll(d11,k5)+bitsll(d11,k4)+bitsll(d11,k2)+bitsll(d11,k1)+ ...
    bitsll(d2,k5)+bitsll(d2,k3)+bitsll(d2,k2)+bitsll(d2,k1)+b2+ ...
    bitsll(d8,k5)+bitsll(d8,k2)+bitsll(d8,k1)+b8-bitsll(d16,k4)- ...
    bitsll(d16,k3)-bitsll(d16,k2)-bitsll(d16,k1)-bitsll(d7,k4)- ...
    bitsll(d7,k2)-bitsll(d7,k1)-bitsll(d3,k3)-bitsll(d3,k2)-b3-bitsll(d12,k2);

y9=-bitsll(d4,k6)-bitsll(d4,k4)-bitsll(d4,k3)-bitsll(d4,k1)-bitsll(d11,k6)- ..
    bitsll(d11,k4)-bitsll(d11,k3)-bitsll(d11,k1)+bitsll(d15,k6)+ ...
    bitsll(d15,k4)+bitsll(d15,k3)+bitsll(d8,k6)+bitsll(d8,k4)+ ...
    bitsll(d8,k2)+b8+bitsll(d1,k6)+bitsll(d1,k4)+bitsll(d1,k1)+ ...
    bitsll(d7,k6)+bitsll(d7,k3)+bitsll(d7,k2)+bitsll(d7,k1)+ ...
    bitsll(d14,k6)+bitsll(d14,k3)+b14-bitsll(d12,k6)-bitsll(d12,k1)- ...
    b12-bitsll(d5,k5)-bitsll(d5,k4)-bitsll(d5,k3)-bitsll(d5,k2)-b5- ...
    bitsll(d3,k5)-bitsll(d3,k4)-bitsll(d3,k2)-bitsll(d3,k1)- ...
    bitsll(d10,k5)-bitsll(d10,k3)-bitsll(d10,k2)-bitsll(d10,k1)-b10+ ...
    bitsll(d16,k5)+bitsll(d16,k2)+bitsll(d16,k1)+b16+bitsll(d9,k4)+ ...
    bitsll(d9,k3)+bitsll(d9,k2)+bitsll(d9,k1)+bitsll(d2,k4)+bitsll(d2,k2)+ ...
    bitsll(d2,k1)+bitsll(d6,k3)+bitsll(d6,k2)+b6+bitsll(d13,k2);

y11=bitsll(d12,k6)+bitsll(d12,k4)+bitsll(d12,k3)+bitsll(d12,k1)- ...
    bitsll(d15,k6)-bitsll(d15,k4)-bitsll(d15,k3)-bitsll(d15,k1)- ...
    bitsll(d9,k6)-bitsll(d9,k4)-bitsll(d9,k3)+bitsll(d6,k6)+ ...
    bitsll(d6,k4)+bitsll(d6,k2)+b6-bitsll(d3,k6)-bitsll(d3,k4)- ...
    bitsll(d3,k1)+bitsll(d1,k6)+bitsll(d1,k3)+bitsll(d1,k2)+ ...
    bitsll(d1,k1)-bitsll(d4,k6)-bitsll(d4,k3)+b4+bitsll(d7,k6)+ ...
    bitsll(d7,k1)+b7-bitsll(d10,k5)-bitsll(d10,k4)-bitsll(d10,k3)- ...
    bitsll(d10,k2)-b10+bitsll(d13,k5)+bitsll(d13,k4)+bitsll(d13,k2)+ ...
    bitsll(d13,k1)-bitsll(d16,k5)-bitsll(d16,k3)-bitsll(d16,k2)- ...
    bitsll(d16,k1)-b16-bitsll(d14,k5)-bitsll(d14,k2)-bitsll(d14,k1)-b14+ ...
    bitsll(d11,k4)+bitsll(d11,k3)+bitsll(d11,k2)+bitsll(d11,k1)- ...
    bitsll(d8,k4)-bitsll(d8,k2)-bitsll(d8,k1)+bitsll(d5,k3)+ ...
    bitsll(d5,k2)+b5-bitsll(d2,k2);

y13=-bitsll(d3,k6)-bitsll(d3,k4)-bitsll(d3,k3)-bitsll(d3,k1)- ...
    bitsll(d8,k6)-bitsll(d8,k4)-bitsll(d8,k3)-bitsll(d8,k1)- ...
    bitsll(d13,k6)-bitsll(d13,k4)-bitsll(d13,k3)+bitsll(d15,k6)+ ...
    bitsll(d15,k4)+bitsll(d15,k2)+b15+bitsll(d10,k6)+bitsll(d10,k4)+ ...
    bitsll(d10,k1)+bitsll(d5,k6)+bitsll(d5,k3)+bitsll(d5,k2)+ ...
    bitsll(d5,k1)+bitsll(d1,k6)+bitsll(d1,k3)+b1+bitsll(d6,k6)+ ...
    bitsll(d6,k1)+b6+bitsll(d11,k5)+bitsll(d11,k4)+bitsll(d11,k3)+ ...
    bitsll(d11,k2)+b11+bitsll(d16,k5)+bitsll(d16,k4)+bitsll(d16,k2)+ ...
    bitsll(d16,k1)-bitsll(d12,k5)-bitsll(d12,k3)-bitsll(d12,k2)- ...
    bitsll(d12,k1)+b12-bitsll(d7,k5)-bitsll(d7,k2)-bitsll(d7,k1)-b7- ...
    bitsll(d2,k4)-bitsll(d2,k3)-bitsll(d2,k2)-bitsll(d2,k1)-bitsll(d4,k4)- ...
    bitsll(d4,k2)-bitsll(d4,k1)-bitsll(d9,k3)-bitsll(d9,k2)-b9-bitsll(d14,k2);
```

```
y15=-bitsll(d7,k6)-bitsll(d7,k4)-bitsll(d7,k3)-bitsll(d7,k1)+bitsll(d9,k6)+ ..
    bitsll(d9,k4)+bitsll(d9,k3)+bitsll(d9,k1)-bitsll(d11,k6)- ...
    bitsll(d11,k4)-bitsll(d11,k3)+bitsll(d5,k6)+bitsll(d5,k4)+ ...
    bitsll(d5,k2)+b5+bitsll(d13,k6)+bitsll(d13,k4)+bitsll(d13,k1)- ...
    bitsll(d3,k6)-bitsll(d3,k3)-bitsll(d3,k2)-bitsll(d3,k1)- ...
    bitsll(d15,k6)-bitsll(d15,k3)-b15+bitsll(d1,k6)+bitsll(d1,k1)+b1- ...
    bitsll(d16,k5)-bitsll(d16,k4)-bitsll(d16,k3)-bitsll(d16,k2)-b16- ...
    bitsll(d2,k5)-bitsll(d2,k4)-bitsll(d2,k2)-bitsll(d2,k1)+ ...
    bitsll(d14,k5)+bitsll(d14,k3)+bitsll(d14,k2)+bitsll(d14,k1)+b14+ ...
    bitsll(d4,k5)+bitsll(d4,k2)+bitsll(d4,k1)+b4-bitsll(d12,k4)- ...
    bitsll(d12,k3)-bitsll(d12,k2)-bitsll(d12,k1)-bitsll(d6,k4)- ...
    bitsll(d6,k2)-bitsll(d6,k1)+bitsll(d10,k3)+bitsll(d10,k2)+b10+ ...
    bitsll(d8,k2);

y17=bitsll(d8,k6)+bitsll(d8,k4)+bitsll(d8,k3)+bitsll(d8,k1)- ...
    bitsll(d10,k6)-bitsll(d10,k4)-bitsll(d10,k3)-bitsll(d10,k1)- ...
    bitsll(d6,k6)-bitsll(d6,k4)-bitsll(d6,k3)+bitsll(d12,k6)+ ...
    bitsll(d12,k4)+bitsll(d12,k2)+b12+bitsll(d4,k6)+bitsll(d4,k4)+ ...
    bitsll(d4,k1)-bitsll(d14,k6)-bitsll(d14,k3)-bitsll(d14,k2)- ...
    bitsll(d14,k1)-bitsll(d2,k6)-bitsll(d2,k3)-b2+bitsll(d16,k6)+ ...
    bitsll(d16,k1)+b16+bitsll(d1,k5)+bitsll(d1,k4)+bitsll(d1,k3)+ ...
    bitsll(d1,k2)+b1+bitsll(d15,k5)+bitsll(d15,k4)+bitsll(d15,k2)+ ...
    bitsll(d15,k1)-bitsll(d3,k5)-bitsll(d3,k3)-bitsll(d3,k2)- ...
    bitsll(d3,k1)-b3-bitsll(d13,k5)-bitsll(d13,k2)-bitsll(d13,k1)-b13+ ...
    bitsll(d5,k4)+bitsll(d5,k3)+bitsll(d5,k2)+bitsll(d5,k1)+ ...
    bitsll(d11,k4)+bitsll(d11,k2)+bitsll(d11,k1)-bitsll(d7,k3)- ...
    bitsll(d7,k2)-b7-bitsll(d9,k2);

y19=-bitsll(d9,k6)-bitsll(d9,k4)-bitsll(d9,k3)-bitsll(d9,k1)+ ...
    bitsll(d14,k6)+bitsll(d14,k4)+bitsll(d14,k3)+bitsll(d14,k1)+...
    bitsll(d4,k6)+bitsll(d4,k4)+bitsll(d4,k3)-bitsll(d2,k6)-...
    bitsll(d2,k4)-bitsll(d2,k2)-b2+bitsll(d7,k6)+bitsll(d7,k4)+...
    bitsll(d7,k1)-bitsll(d12,k6)-bitsll(d12,k3)-bitsll(d12,k2)-...
    bitsll(d12,k1)-bitsll(d16,k6)-bitsll(d16,k3)-b16+bitsll(d11,k6)+...
    bitsll(d11,k1)+b11-bitsll(d6,k5)-bitsll(d6,k4)-bitsll(d6,k3)-...
    bitsll(d6,k2)-b6+bitsll(d1,k5)+bitsll(d1,k4)+bitsll(d1,k2)+...
    bitsll(d1,k1)+bitsll(d11,k5)+bitsll(d11,k3)-bitsll(d5,k2)-...
    bitsll(d5,k1)-b5+bitsll(d10,k5)+bitsll(d10,k2)+bitsll(d10,k1)+...
    b10-bitsll(d15,k4)-bitsll(d15,k3)-bitsll(d15,k2)-bitsll(d15,k1)-...
    bitsll(d13,k4)-bitsll(d13,k2)-bitsll(d13,k1)+bitsll(d8,k3)+...
    bitsll(d8,k2)+b8-bitsll(d3,k2);

y21=-bitsll(d1,k6)-bitsll(d1,k4)-bitsll(d1,k3)-bitsll(d1,k1)-...
    bitsll(d5,k6)-bitsll(d5,k4)-bitsll(d5,k3)-bitsll(d5,k1)-...
    bitsll(d8,k6)-bitsll(d8,k4)-bitsll(d8,k3)-bitsll(d11,k6)-...
    bitsll(d11,k4)-bitsll(d11,k2)-b11-bitsll(d14,k6)-bitsll(d14,k4)-...
    bitsll(d14,k1)+bitsll(d16,k6)+bitsll(d16,k3)+bitsll(d16,k2)+...
    bitsll(d16,k1)+bitsll(d13,k6)+bitsll(d13,k3)+b13+bitsll(d10,k6)+...
    bitsll(d10,k1)+b10+bitsll(d7,k5)+bitsll(d7,k4)+bitsll(d7,k3)+...
    bitsll(d7,k2)+b7+bitsll(d4,k5)+bitsll(d4,k4)+bitsll(d4,k2)+...
    bitsll(d4,k1)+bitsll(d1,k5)+bitsll(d1,k3)+bitsll(d1,k2)+...
    bitsll(d1,k1)+b1+bitsll(d3,k5)+bitsll(d3,k2)+bitsll(d3,k1)+b3+...
    bitsll(d6,k4)+bitsll(d6,k3)+bitsll(d6,k2)+bitsll(d6,k1)+...
    bitsll(d9,k4)+bitsll(d9,k2)+bitsll(d9,k1)+bitsll(d12,k3)+...
    bitsll(d12,k2)+b12+bitsll(d15,k2);

y23=bitsll(d6,k6)+bitsll(d6,k4)+bitsll(d6,k3)+bitsll(d6,k1)-...
    bitsll(d13,k6)-bitsll(d13,k4)-bitsll(d13,k3)-bitsll(d13,k1)-...
    bitsll(d2,k6)-bitsll(d2,k4)-bitsll(d2,k3)+bitsll(d9,k6)+...
    bitsll(d9,k4)+bitsll(d9,k2)+b9-bitsll(d16,k6)-bitsll(d16,k4)-...
    bitsll(d16,k1)-bitsll(d10,k6)-bitsll(d10,k3)-bitsll(d10,k2)-...
    bitsll(d10,k1)+bitsll(d3,k6)+bitsll(d3,k3)+b3+bitsll(d8,k6)-...
    bitsll(d5,k1)-b5+bitsll(d12,k5)+bitsll(d12,k4)+bitsll(d12,k3)+...
    bitsll(d12,k2)+b12+bitsll(d14,k5)+bitsll(d14,k4)+bitsll(d14,k2)+...
    bitsll(d14,k1)-bitsll(d7,k5)-bitsll(d7,k3)-bitsll(d7,k2)-...
    bitsll(d7,k1)-b7+bitsll(d1,k5)+bitsll(d1,k2)+bitsll(d1,k1)+b1-...
    bitsll(d8,k4)-bitsll(d8,k3)-bitsll(d8,k2)-bitsll(d8,k1)+...
    bitsll(d15,k4)+bitsll(d15,k2)+bitsll(d15,k1)+bitsll(d11,k3)+...
    bitsll(d11,k2)+b11-bitsll(d4,k2);

y25=bitsll(d3,k6)+bitsll(d3,k4)+bitsll(d3,k3)+bitsll(d3,k1)-...
    bitsll(d12,k6)-bitsll(d12,k4)-bitsll(d12,k3)-bitsll(d12,k1)-...
    bitsll(d7,k6)-bitsll(d7,k4)-bitsll(d7,k3)+bitsll(d16,k6)+...
    bitsll(d16,k4)+bitsll(d16,k2)+b16+bitsll(d8,k6)+bitsll(d8,k4)+...
    bitsll(d8,k1)-bitsll(d2,k6)-bitsll(d2,k3)-bitsll(d2,k2)-...
    bitsll(d2,k1)+bitsll(d11,k6)+bitsll(d11,k3)+b11+bitsll(d13,k6)+...
    bitsll(d13,k1)+b13-bitsll(d4,k5)-bitsll(d4,k4)-bitsll(d4,k3)-...
    bitsll(d4,k2)-b4+bitsll(d6,k5)+bitsll(d6,k4)+bitsll(d6,k2)+...
    bitsll(d6,k1)-bitsll(d15,k5)-bitsll(d15,k3)-bitsll(d15,k2)-...
    bitsll(d15,k1)-b15-bitsll(d9,k5)-bitsll(d9,k2)-bitsll(d9,k1)-b9+...
    bitsll(d1,k4)+bitsll(d1,k3)+bitsll(d1,k2)+bitsll(d1,k1)-...
    bitsll(d10,k4)-bitsll(d10,k2)-bitsll(d10,k1)-bitsll(d14,k3)-...
    bitsll(d14,k2)-b14+bitsll(d5,k2);

y27=bitsll(d10,k6)+bitsll(d10,k4)+bitsll(d10,k3)+bitsll(d10,k1)-...
    bitsll(d4,k6)-bitsll(d4,k4)-bitsll(d4,k3)+bitsll(d4,k1)-...
    bitsll(d16,k6)-bitsll(d16,k4)-bitsll(d16,k3)+bitsll(d3,k6)+...
    bitsll(d3,k4)+bitsll(d3,k2)+b3-bitsll(d11,k6)-bitsll(d11,k4)-...
    bitsll(d11,k1)-bitsll(d9,k6)-bitsll(d9,k3)-bitsll(d9,k2)-...
    bitsll(d9,k1)+bitsll(d5,k6)+bitsll(d5,k3)+b5+bitsll(d15,k6)+...
    bitsll(d15,k1)+b15-bitsll(d2,k5)-bitsll(d2,k4)-bitsll(d2,k3)-...
    bitsll(d2,k2)-b2+bitsll(d12,k5)+bitsll(d12,k4)+bitsll(d12,k2)+...
    bitsll(d12,k1)+bitsll(d8,k5)+bitsll(d8,k3)+bitsll(d8,k2)+...
    bitsll(d8,k1)+b8-bitsll(d6,k5)-bitsll(d6,k2)-bitsll(d6,k1)-b6-...
    bitsll(d14,k4)-bitsll(d14,k3)-bitsll(d14,k2)-bitsll(d14,k1)+...
    bitsll(d1,k4)+bitsll(d1,k2)+bitsll(d1,k1)-bitsll(d13,k3)-...
    bitsll(d13,k2)-b13-bitsll(d7,k2);
    y29=-bitsll(d6,k6)-bitsll(d6,k4)-bitsll(d6,k3)-bitsll(d6,k1)+...
        bitsll(d16,k6)+bitsll(d16,k4)+bitsll(d16,k3)+bitsll(d16,k1)+...
        bitsll(d5,k6)+bitsll(d5,k4)+bitsll(d5,k3)+bitsll(d7,k6)+...
        bitsll(d7,k4)+bitsll(d7,k2)+b7-bitsll(d15,k6)-bitsll(d15,k4)-...
        bitsll(d15,k1)-bitsll(d4,k6)-bitsll(d4,k3)-bitsll(d4,k2)-...
        bitsll(d4,k1)-bitsll(d8,k6)-bitsll(d8,k3)-b8+bitsll(d14,k6)+...
        bitsll(d14,k1)+b14+bitsll(d3,k5)+bitsll(d3,k4)+bitsll(d3,k3)+...
        bitsll(d3,k2)+b3+bitsll(d9,k5)+bitsll(d9,k4)+bitsll(d9,k2)+...
        bitsll(d9,k1)-bitsll(d13,k5)-bitsll(d13,k3)-bitsll(d13,k2)-...
        bitsll(d13,k1)-b13-bitsll(d2,k5)-bitsll(d2,k2)-bitsll(d2,k1)-b2-...
        bitsll(d10,k4)-bitsll(d10,k3)-bitsll(d10,k2)-bitsll(d10,k1)+...
        bitsll(d12,k4)+bitsll(d12,k2)+bitsll(d12,k1)+bitsll(d1,k3)+...
        bitsll(d1,k2)+b1+bitsll(d11,k2);

    y31=-bitsll(d16,k6)-bitsll(d16,k4)-bitsll(d16,k3)-bitsll(d16,k1)+...
        bitsll(d15,k6)+bitsll(d15,k4)+bitsll(d15,k3)+bitsll(d15,k1)-...
        bitsll(d14,k6)-bitsll(d14,k4)-bitsll(d14,k3)+bitsll(d13,k6)+...
        bitsll(d13,k4)+bitsll(d13,k2)+b13-bitsll(d12,k6)-bitsll(d12,k4)-...
        bitsll(d12,k1)+bitsll(d11,k6)+bitsll(d11,k3)+bitsll(d11,k2)+...
        bitsll(d11,k1)-bitsll(d10,k6)-bitsll(d10,k3)-b10+bitsll(d9,k6)+...
        bitsll(d9,k1)+b9-bitsll(d8,k5)-bitsll(d8,k4)-bitsll(d8,k3)-...
        bitsll(d8,k2)-b8+bitsll(d7,k5)+bitsll(d7,k4)+bitsll(d7,k2)+...
        bitsll(d7,k1)-bitsll(d6,k5)-bitsll(d6,k3)-bitsll(d6,k2)-...
        bitsll(d6,k1)-b6+bitsll(d5,k5)+bitsll(d5,k2)+bitsll(d5,k1)+b5-...
        bitsll(d4,k4)-bitsll(d4,k3)-bitsll(d4,k2)-bitsll(d4,k1)+...
        bitsll(d3,k4)+bitsll(d3,k2)+bitsll(d3,k1)-bitsll(d2,k3)-...
        bitsll(d2,k2)-b2+bitsll(d1,k2);

end
```

4.5.1 Test Bench

```
function test_hdldct32()
m_dct=zeros(32);
 MUL=power(2,8.5);
for k=1:32
    for l=1:32
        [d,y1,y3]=hdldct32(k,l,MUL,32,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,1,1, ...
            1,1,1,1,1,1,1,1,1,1,1,1,1,1);
        m_dct(k,l)=d;
    end
end
disp(m_dct)
disp(y1)
disp(y3)
disp(y5)
disp(y7)
disp(y9)
disp(y11)
disp(y13)
disp(y15)
disp(y17)
disp(y19)
disp(y21)
disp(y23)
disp(y25)
disp(y27)
disp(y29)
disp(y31)
end
```

Bibliography

Madhukar Budagavi, Arild Fuldseth, Gisle Bjontegaard, Vivienne Sze and Mangesh Sadafale. (2013). *Core Transform Design in the High Efficiency Video Coding (HEVC) Standard*,IEEE JOURNAL OF SELECTED TOPICS IN SIGNAL PROCESSING.

Pramod Kumar Meher,Sang Yoon Park,Basant Kumar Mohanty,Khoon Seong Lim,Chuohao Yeo. (2014), *Efficient Integer DCT Architectures for HEVC*, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY.