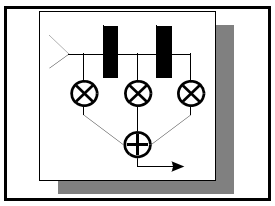
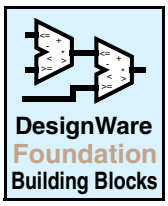




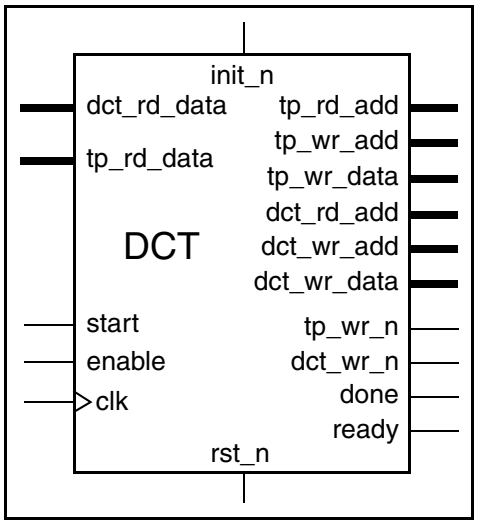
DW_dct_2d

Two Dimensional Discrete Cosine Transform

Version, STAR and Download Information: IP Directory

Features and Benefits

- Parameterized input data width and block size
- Parameterized coefficients



Applications

- Linear DCT for audio such as Dolby Digital
- NxN matrix for jpeg mjpeg/mpeg and DV video
- Dolby AC2 and AC3: 1-D DCT
- JPEG (still images): 2-D DCT spatial compression
- MPEG1 and MPEG2: 2-D DCT plus motion compensation
- H.261 and H.263: moving image compression for video conferencing and video telephony

Description

The DW_dct_2d implements the DCT-II. This is described in detail in the “Functional Description” section. The device uses coefficients and data at the input port and generates the dct data at the output. The data is read using radd, and the forward dct data is read by rows, i.e. 0,1,2 to n*n-1. The idct, or inverse dct reads data by columns, i.e. 0,n,2n,…1,n+1,2n+1, ... to n*(n-1). This device requires an intermediate ram to hold the first pass results. This ram is written by columns and read by rows. It is read continuously and written at twr_n. The inputs and outputs are listed below. If continuous stream data is begin converted, a dual length ram and a toggle method will be necessary. This will allow the initial data to be written into the transposition memory, and then toggled, and the next block will be written to the new transposition memory, while the initial data is read from the first tp block. In this way, the intermediate terms will be continually written and read.

For a detailed description of the transform performed, see “Functional Description.”

DW_dct_2d is only valid for A-2007.12-SP1 and later releases.

Table 1-1 Pin Description

Pin Name	Direction	Function
clk	input	Primary clock
rst_n	input	Asynchronous reset

Table 1-1 Pin Description (Continued)

Pin Name	Direction	Function
init_n	input	Synchronous reset
enable	input	Enable run
start	input	Start transform
dct_rd_data	input	Read data input port
tp_rd_data	input	Read data from transpose RAM
tp_rd_add	output	Read address for transpose RAM
tp_wr_add	output	Write address for transpose RAM
tp_wr_data	output	Data out to transpose RAM
tp_wr_n	output	Write to transpose RAM (active low)
dct_wr_n	output	Write signal out (active low)
done	output	Read data input done
ready	output	First transformed data ready
dct_wr_data	output	Transformed data out

Table 1-2 Parameter Description

Parameter	Value	Description
n	4 - 16 Default: 8	Size of 2 dimensional matrix n is the size of the two dimensional matrix to be transformed. It must be even and between 4 and 16, so 4, 6, 8, 10,12,14, and 16. The coefficients are generated for each size of n, using the formulas listed above. The scaling factor required for each coefficient must be applied to the coefficient before inserting it in the module.
bpp	4 - 32 Default: 8	Number of bits per pixel
reg_out	0, 1 Default: 0	Register outputs
tc_mode	0, 1 Default: 0	Input data type: two's complement Determines the data type of the input data, for forward dct mode only. If set to 0, the data is assumed to be non negative, and set to 1, the data is considered two's complement. Intermediate data and output data is always in two's complement mode.

Table 1-2 Parameter Description (Continued)

Parameter	Value	Description
rt_mode	0, 1 Default: 1	0:round data 1:truncate data Determines the output rounding or truncate. 0 means round the data to the nearest up. 1 truncates the data at the output width determined by the formula. NOTE: two's complement data is output always, and truncation will cause errors or weighting to negative numbers.
idct_mode	0, 1 Default: 0	0:dct(forward) 1:idct(inverse) Determines the direction of data throughput, and the size of the various ports. If 0, the data is assumed to be pixel or sample data and the forward DCT algorithm is used, the input is bpp, and the output is n/2+bpp. If set to 1, the input is considered to be dct coefficient data, and the input size is determined by the above formula, and the output data is sized at bpp.
co_a	16 bits Default: 23170	Coefficient A Coefficients are the various dct coefficients as calculated by the DCT formula below.
co_b	16 bits Default: 32138	Coefficient B
co_c	16 bits Default: 30274	Coefficient C
co_d	16 bits Default: 27245	Coefficient D
co_e	16 bits Default: 18205	Coefficient E
co_f	16 bits Default: 12541	Coefficient F
co_g	16 bits Default: 6393	Coefficient G
co_h	16 bits Default: 35355	Coefficient H
co_i	16 bits Default: 49039	Coefficient I
co_j	16 bits Default: 46194	Coefficient J
co_k	16 bits Default: 41573	Coefficient K
co_l	16 bits Default: 27779	Coefficient L
co_m	16 bits Default: 19134	Coefficient M

Table 1-2 Parameter Description (Continued)

Parameter	Value	Description
co_n	16 bits Default: 9755	Coefficient N
co_o	16 bits Default: 35355	Coefficient O
co_p	16 bits Default: 49039	Coefficient P

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis model	DesignWare

DW_dct_2d is only valid for A-2007.12-SP1 and later releases.

Table 1-4 Simulation Models

Model	Function
DW01.DW_DCT_2D_CFG_SIM	Design unit name for VHDL simulation
dw/dw01/src/DW_dct_2d_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_dct_2d.v	Verilog simulation model source code

1.0.1 Functional Description

Contemporary image/audio data processing applications use transform coding as an integral part of the efficient storage and transmission of signals. Current transmission and storage systems require a reduction in data stream size. Conversely, the transmission medium attempts to add redundant information to obtain the highest possible transmission quality. In order to achieve the reduction in data stream size, we need to devise a method of compression or elimination of data.

The discrete cosine transforms forms and integral part of current digital signal processing. The basis of the utility of the transform function relies on the fact that most analog waveforms perceptible to humans contain a large amount of redundant information. This redundancy exists as correlation between neighboring data samples in the digital domain. In still image processing, the redundant information is contained in the relative similarity between neighboring pixels. In order to achieve reduction in data stream size, the transform attempts to remove as much of the redundancy as is possible. Transforms perform a perfect, or lossless transformation of the data as spatial displacements into spatial frequencies. The inverse function reverses this transform and recovers the image data perfectly. The only loss involved in this transform is due to accuracy/truncation/rounding and another step outside of the transform called quantization.

The DW_dct_2d implements the DCT-II. This is described in detail below. The device uses coefficients and data at the input port and generates the dct data at the output. The data is read using radd, and the forward dct data is read by rows, i.e. 0,1,2 to n*n-1. The idct, or inverse dct reads data by columns, i.e. 0,n,2n,…1,n+1,2n+1, etc. to n*(n-1). This device requires an intermediate ram to hold the first pass results. This ram is written by columns and read by rows. It is read continuously and written at twr_n. The inputs and outputs are listed below. If continuous stream data is begin converted, a dual length ram and a toggle method will be necessary. This will allow the initial data to be written into the transposition memory, and then toggled, and the next block will be written to the new transposition memory, while the initial data is read from the first tp block. In this way, the intermediate terms will be continually written and read.

The method used for the DCT formula follows. The DCT-II is the most commonly used equation, and is the one referred to when we say DCT. The 1D DCT formulas are:

(1)

$$X_k = \sqrt{2}/N \sum_{n=0}^{N-1} x_n \cos\left[\pi/N(n + ½)k\right] \quad \text{for n = 1 to N-1}$$

$$X_k = 1/\sqrt{N} \sum_{n=0}^{N-1} x_n \cos\left[\pi/N(n + ½)k\right] \quad \text{for n = 0}$$

and the 2D DCT is:

(2)

$$Y_{(k,j)} = C_{(k)}C_{(j)}\left[\sum_{n=0}^{N-1}\sum_{m=0}^{N-1} x_{(n,m)} \cos\left[\pi/N(n + ½)k\right] \cos\left[\pi/N(n + ½)j\right]\right]$$

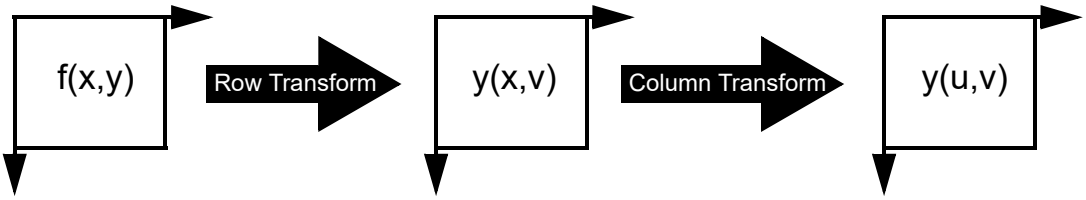
where $C_{(l)} = 1/\sqrt{2}$ for n = 0 and $\sqrt{2}/N$ for l = 1 to N-1

Separability

The DCT of (2) can be expressed as

$$Y_{(k,j)} = C_{(k)}C_{(j)} \sum_{n=0}^{N-1}\left[\cos\left[\pi/N(n + ½)k\right]\right]\sum_{m=0}^{N-1}\left[\cos\left[\pi/N(n + ½)j\right]\right]$$

This property, separability, allows the transform of a 2d image to be computed in two steps by successive 1d operations. First we will do the operation of a 1d transform on the row data, and then the second operation on the resulting data in by column.



This device uses the intermediate memory to hold the y(x,v) results listed above.

The decimal fraction coefficients used here can be generated using a simple perl script:

```
#!/usr/local/bin/perl -w
use Math::Trig;
$N  = 8;
if(defined $ARGV[0]){
  $N  = $ARGV[0];
  }
my $colcnt = 0;
my $rowcnt = 0;
  for ($rowcnt = 0; $rowcnt < $N ; $rowcnt += 1) {#col
      $Alpha = 1/sqrt($N) if $rowcnt == 0;#scaling factor
      $Alpha = sqrt(2/$N) if $rowcnt != 0;#scaling factor
      $rad = (($Alpha*(cos(((2*$colcnt +1)*$rowcnt*pi)/(2*$N)) )));
      $rad /= 1.52590;# convert to decimal fraction
      $rad *= 1000000;#scale up
      $rad += 0.5;#round up
      $rad /= 10;#round up
      #$rad = int($rad);
      $rad = sprintf("%04d",$rad ) ;#set decimal places
      print "$rad\n";
  }
```

This script will print a series of coefficients based on the 'n' number provided. For example, 'gen_coef.pl 8' will generate coefficients for and 8x8 dct matrix.

Timing Diagrams

Figure 1-1 displays timing from `start` to the first `tp_wr_n` write to transpose memory, for forward DCT.

Figure 1-1 'start' to the First 'tp_wr_n' Write to Transpose Memory

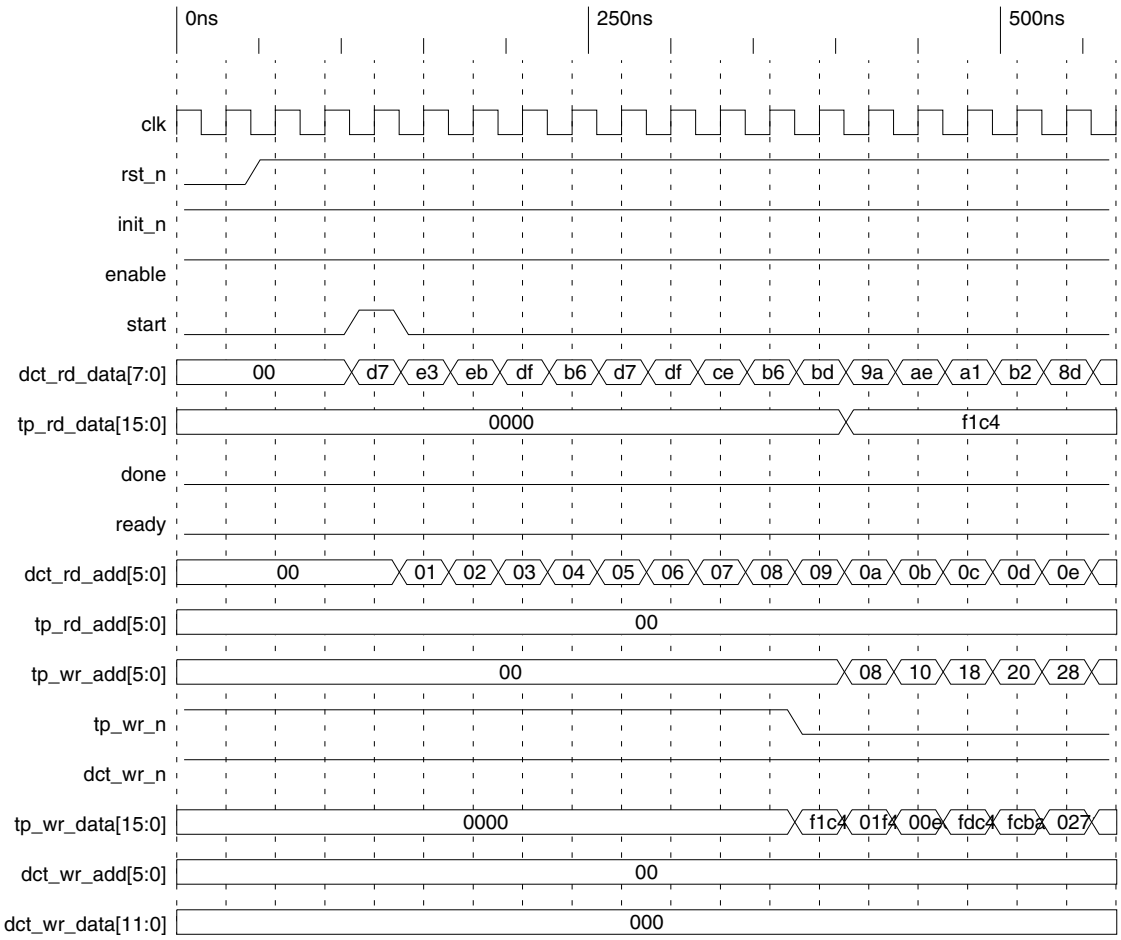


Figure 1-2 Done/Ready timing: This diagram shows, for N = 8, the timing of the signal `done`, output, to the `ready` output. The `done` signal signifies the last input data has been read, while `ready` signifies the last transform data is ready.

Figure 1-2 Done / Ready Timing

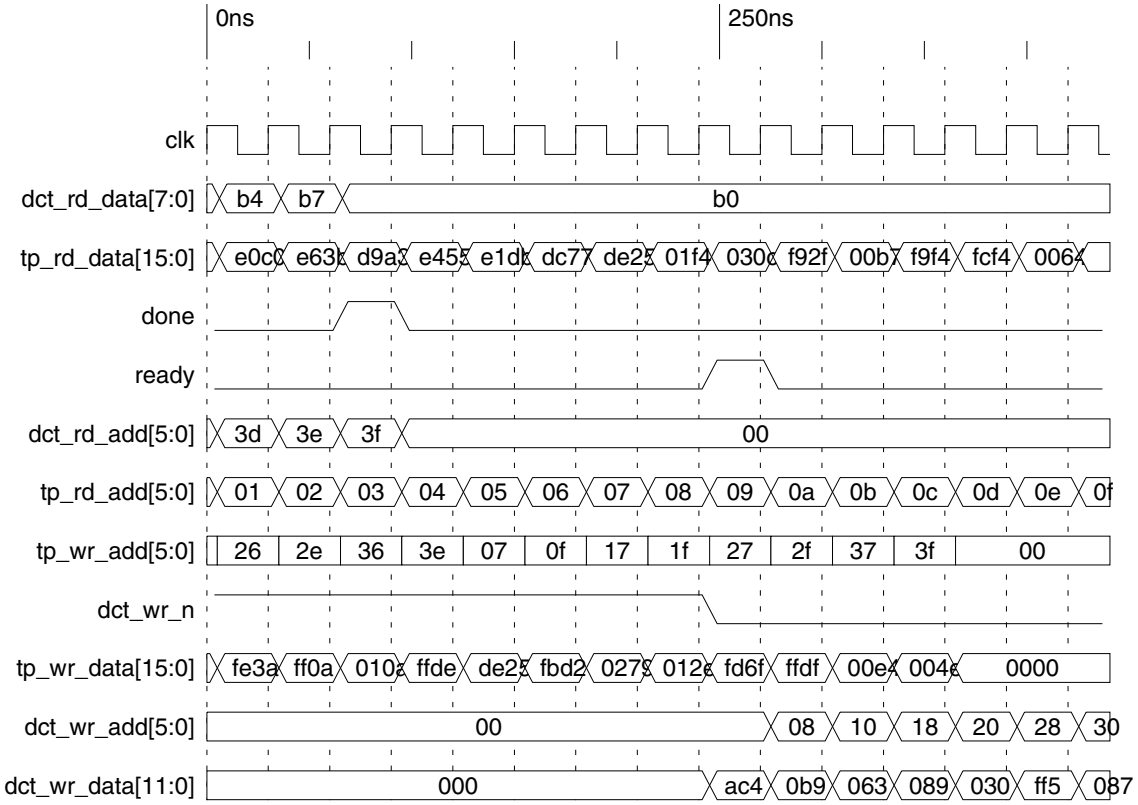


Fig.2: Done and ready timing

Related Topics

- Math – Arithmetic Overview
- DesignWare Building Block IP Documentation Overview

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE,WORK;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;

entity DW_dct_2d_inst is
  generic (
    inst_bpp : NATURAL := 8;
    inst_n : NATURAL := 8;
    inst_reg_out : NATURAL := 0;
    inst_tc_mode : NATURAL := 0;
    inst_rt_mode : NATURAL := 1;
    inst_idct_mode : NATURAL := 0;
    inst_co_a : NATURAL := 23170;
    inst_co_b : NATURAL := 32138;
    inst_co_c : NATURAL := 30274;
    inst_co_d : NATURAL := 27245;
    inst_co_e : NATURAL := 18205;
    inst_co_f : NATURAL := 12541;
    inst_co_g : NATURAL := 6393;
    inst_co_h : NATURAL := 35355;
    inst_co_i : NATURAL := 49039;
    inst_co_j : NATURAL := 46194;
    inst_co_k : NATURAL := 41573;
    inst_co_l : NATURAL := 27779;
    inst_co_m : NATURAL := 19134;
    inst_co_n : NATURAL := 9755;
    inst_co_o : NATURAL := 35355;
    inst_co_p : NATURAL := 49039
    );
  port (
    inst_clk : in std_logic;
    inst_rst_n : in std_logic;
    inst_init_n : in std_logic;
    inst_enable : in std_logic;
    inst_start : in std_logic;
    inst_dct_rd_data : in std_logic_vector(inst_bpp+(inst_n/2*inst_idct_mode)-1 downto
0);
    inst_tp_rd_data : in std_logic_vector(inst_bpp/2+inst_bpp+3+((1-inst_tc_mode)*(1-
inst_idct_mode)) downto 0);
    tp_rd_add_inst : out std_logic_vector(bit_width(inst_n*inst_n)-1 downto 0);
    tp_wr_add_inst : out std_logic_vector(bit_width(inst_n*inst_n)-1 downto 0);
    tp_wr_data_inst : out std_logic_vector(inst_bpp/2+inst_bpp+3+((1-inst_tc_mode)*(1-
inst_idct_mode)) downto 0);
    tp_wr_n_inst : out std_logic;
    dct_rd_add_inst : out std_logic_vector(bit_width(inst_n*inst_n)-1 downto 0);
    dct_wr_add_inst : out std_logic_vector(bit_width(inst_n*inst_n)-1 downto 0);
    dct_wr_n_inst : out std_logic;
```

```
    done_inst : out std_logic;
    ready_inst : out std_logic;
    dct_wr_data_inst : out std_logic_vector(inst_bpp-1+(inst_n/2*(1-inst_idct_mode))
downto 0)
    );
    end DW_dct_2d_inst;


architecture inst of DW_dct_2d_inst is

    component DW_dct_2d
      generic (
        bpp : NATURAL := 8;
        n   : NATURAL := 8;
        reg_out : NATURAL := 0;
        tc_mode : NATURAL := 0;
        rt_mode : NATURAL := 1;
        idct_mode : NATURAL := 0;
        co_a : NATURAL := 23170;
        co_b : NATURAL := 32138;
        co_c : NATURAL := 30274;
        co_d : NATURAL := 27245;
        co_e : NATURAL := 18205;
        co_f : NATURAL := 12541;
        co_g : NATURAL := 6393;
        co_h : NATURAL := 35355;
        co_i : NATURAL := 49039;
        co_j : NATURAL := 46194;
        co_k : NATURAL := 41573;
        co_l : NATURAL := 27779;
        co_m : NATURAL := 19134;
        co_n : NATURAL := 9755;
        co_o : NATURAL := 35355;
        co_p : NATURAL := 49039 );
      port (clk    : in std_logic;
        rst_n  : in std_logic;
        init_n : in std_logic;
        enable : in std_logic;
        start  : in std_logic;
        dct_rd_data : in std_logic_vector(bpp+(n/2*idct_mode)-1 downto 0);
        tp_rd_data  : in std_logic_vector(bpp/2+bpp+3+((1-tc_mode)*(1-idct_mode))
downto 0);
        tp_rd_add : out std_logic_vector(bit_width(n*n)-1 downto 0);
        tp_wr_add : out std_logic_vector(bit_width(n*n)-1 downto 0);
        tp_wr_data : out std_logic_vector(bpp/2+bpp+3+((1-tc_mode)*(1-idct_mode))
downto 0);
        tp_wr_n : out std_logic;
        dct_rd_add : out std_logic_vector(bit_width(n*n)-1 downto 0);
        dct_wr_add : out std_logic_vector(bit_width(n*n)-1 downto 0);
```

```
        dct_wr_n : out std_logic;
        done : out std_logic;
        ready : out std_logic;
        dct_wr_data : out std_logic_vector(bpp-1+(n/2*(1-idct_mode)) downto 0)
        );
    end component;

begin

    -- Instance of DW_dct_2d
    U1 : DW_dct_2d
    generic map ( bpp => inst_bpp,
                  n => inst_n,
                  reg_out => inst_reg_out,
                  tc_mode => inst_tc_mode,
                  rt_mode => inst_rt_mode,
                  idct_mode => inst_idct_mode,
                  co_a => inst_co_a,
                  co_b => inst_co_b,
                  co_c => inst_co_c,
                  co_d => inst_co_d,
                  co_e => inst_co_e,
                  co_f => inst_co_f,
                  co_g => inst_co_g,
                  co_h => inst_co_h,
                  co_i => inst_co_i,
                  co_j => inst_co_j,
                  co_k => inst_co_k,
                  co_l => inst_co_l,
                  co_m => inst_co_m,
                  co_n => inst_co_n,
                  co_o => inst_co_o,
                  co_p => inst_co_p )
    port map ( clk => inst_clk,
                  rst_n => inst_rst_n,
                  init_n => inst_init_n,
                  enable => inst_enable,
                  start => inst_start,
                  dct_rd_data => inst_dct_rd_data,
                  tp_rd_data => inst_tp_rd_data,
                  tp_rd_add => tp_rd_add_inst,
                  tp_wr_add => tp_wr_add_inst,
                  tp_wr_data => tp_wr_data_inst,
                  tp_wr_n => tp_wr_n_inst,
                  dct_rd_add => dct_rd_add_inst,
                  dct_wr_add => dct_wr_add_inst,
                  dct_wr_n => dct_wr_n_inst,
                  done => done_inst,
                  ready => ready_inst,
```

```
                    dct_wr_data => dct_wr_data_inst );

  end inst;
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_dct_2d_inst( inst_clk,
                       inst_rst_n,
                       inst_init_n,
                       inst_enable,
                       inst_start,

                      inst_dct_rd_data,
                       inst_tp_rd_data,

                       tp_rd_add_inst,
                       tp_wr_add_inst,
                       tp_wr_data_inst,
                      tp_wr_n_inst,

                       dct_rd_add_inst,
                       dct_wr_add_inst,
                       dct_wr_n_inst,

                       done_inst,
                      ready_inst,
                       dct_wr_data_inst );

parameter bpp = 8;
parameter n = 8;
parameter reg_out = 1;
parameter tc_mode = 0;
parameter rt_mode = 1;
parameter idct_mode = 1;
parameter co_a = 23170;
parameter co_b = 32138;
parameter co_c = 30274;
parameter co_d = 27245;
parameter co_e = 18205;
parameter co_f = 12541;
parameter co_g = 6393;
parameter co_h = 35355;
parameter co_i = 49039;
parameter co_j = 46194;
parameter co_k = 41573;
parameter co_l = 27779;
parameter co_m = 19134;
parameter co_n = 9755;
parameter co_o = 35355;
parameter co_p = 49039;
```

```
`define addr_width 6  // addr_width is ceil(log2(n * n))

input inst_clk;
input inst_rst_n;
input inst_init_n;
input inst_enable;
input inst_start;
input [bpp+(n/2*idct_mode)-1 : 0] inst_dct_rd_data;
input [bpp/2+bpp+3+((1-tc_mode)*(1-idct_mode)) : 0] inst_tp_rd_data;
output [`addr_width-1 : 0] tp_rd_add_inst;
output [`addr_width-1 : 0] tp_wr_add_inst;
output [bpp/2+bpp+3+((1-tc_mode)*(1-idct_mode)) : 0] tp_wr_data_inst;
output tp_wr_n_inst;
output [`addr_width-1 : 0] dct_rd_add_inst;
output [`addr_width-1 : 0] dct_wr_add_inst;
output dct_wr_n_inst;
output done_inst;
output ready_inst;
output [bpp-1+(n/2*(1-idct_mode)) : 0] dct_wr_data_inst;

    // Instance of DW_dct_2d
DW_dct_2d #(bpp, n, reg_out, tc_mode, rt_mode, idct_mode,
            co_a, co_b, co_c, co_d, co_e, co_f, co_g, co_h, co_i, co_j,
            co_k, co_l, co_m, co_n, co_o, co_p)
      U1 ( .clk(inst_clk),
               .rst_n(inst_rst_n),
               .init_n(inst_init_n),
               .enable(inst_enable),
               .start(inst_start),
               .dct_rd_data(inst_dct_rd_data),
               .tp_rd_data(inst_tp_rd_data),
               .tp_rd_add(tp_rd_add_inst),
               .tp_wr_add(tp_wr_add_inst),
               .tp_wr_data(tp_wr_data_inst),
               .tp_wr_n(tp_wr_n_inst),
               .dct_rd_add(dct_rd_add_inst),
               .dct_wr_add(dct_wr_add_inst),
               .dct_wr_n(dct_wr_n_inst),
               .done(done_inst),
               .ready(ready_inst),
               .dct_wr_data(dct_wr_data_inst) );

endmodule
```

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043

www.synopsys.com