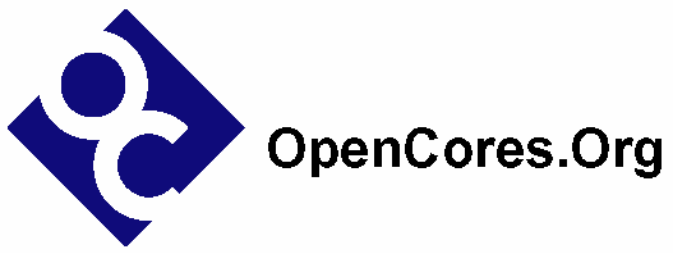

MDCT IP Core Specification

Author: Michal Krepa

Rev. 1.1

April 26, 2006

This page has been intentionally left blank.

Revision History

Rev.	Date	Author	Description
1.0	15/04/2006	Michal Krepa	Initial Draft
1.1	17/04/2006	Michal Krepa	Updated all sections
1.2	17/04/2006	Michal Krepa	Updated Testbench section
1.3	21/04/2006	Michal Krepa	Added memory information
1.4	25/04/2006	Michal Krepa	Updated as per code redesign

Contents

1 Introduction

The MDCT core is two dimensional discrete cosine transform implementation designed for use in compression systems like JPEG. Architecture is based on parallel distributed arithmetic with butterfly computation. Done as row/column decomposition where two 1D DCT units are connected through transposition matrix memory. Core has simple input interface. Design is synchronous, with single positive clock edge and no internal tri-state buffers. Latency between first latched input data and first DCT transformed output is 85 clock cycles. Design is internally pipelined, when the pipeline is full 64 point input data is transformed in 64 clock cycles to 2D DCT values. Core uses double buffered (ping-pong scheme) RAM for storing intermediate product results after first DCT stage for maximized performance. This way both 1D DCT units can work in parallel effectively creating dual stage global pipeline. MDCT core takes 8 bit input data and produces 12 bit output using 12 bit DCT matrix coefficients. This may be enhanced to be configurable in the future.

Self checking testbench is included for testing operation of the MDCT core. Testbench takes matlab-converted bitmap image, MDCT core DCT-transforms it and testbench compares input image to reconstructed one by performing behavioral IDCT. Peak signal to noise ratio is computed to see how big error is introduced by fixed point arithmetic used in core. Testbench can also perform quantization of DCT output, but when this option is enabled DCT core operation is not verified. Testbench created DCT images (reconstructed image and error image) can be converted to JPEG format by matlab scripts (included).

Core was not yet tested on real HW FPGA, so it is considered beta release.

2

Architecture

This section describes the architecture of the MDCT. Below is I/O schematic of MDCT core.

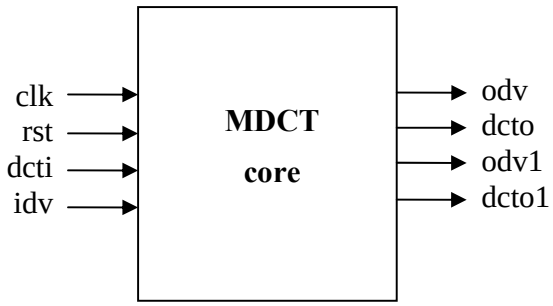

Figure 1. Top level schematic

Block diagram is presented below describing the top level of the design.

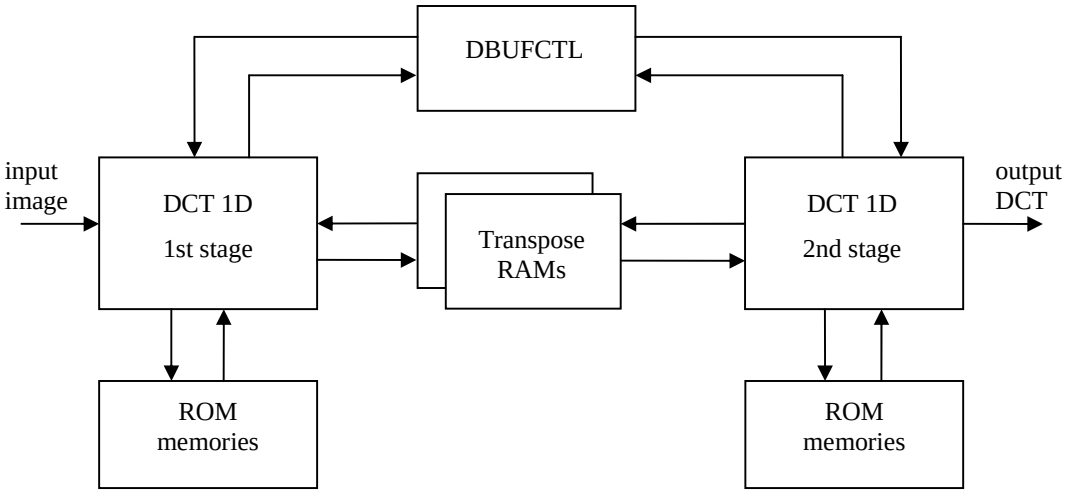

Figure 2. Block diagram of MDCT core

MDCT core architecture is based two 1D DCT units connected through transpose matrix RAM. Transposition RAM is double buffered, that is, when 2nd stage of DCT reads out data from transposition memory 1, 1st DCT stage can populate 2nd transposition memory with new data. This enables creation of dual stage global pipeline where every stage consist of 1D DCT and transposition memory. 1D DCT units are not internally pipelined, they use parallel distributed arithmetic with butterfly computation to compute DCT values. Because of parallel DA they need considerable amount of ROM memories to compute one DCT value in single clock cycle. Design based on distributed arithmetic does not use any multipliers for computing MAC (multiply and accumulate), instead it stores precomputed MAC results in ROM memory and grab them as needed. DBUFCTL block is essentially a memory arbiter between 1D DCT stages. Each stage can request memory buffer for read/write and basing on availability one of two RAM buffers is granted. Every RAM has 10 bit width data and 64 memory cells.

On chip RAMs used are dual port synchronous memory with one clock cycle delay. These RAMs are currently 64 words x 14 bits size and there are two of them to create ping-pong buffers for improved throughput.

Design uses also ROM memories for storing precomputed MAC coefficients. ROMs can be either asynchronous or synchronous. ./SOURCE/ROME.VHD and ./SOURCE/ROMO.VHD memory models included in core sources are synchronous synthesizable ROM memory models which can be used for simulation and implementation. But, beware they seems to only synthesize to block rams with Mentor tools even if block rom inference is disabled (did not try with other tools). If you want to have asynchronous distributed ROMs remove clocking and input address register from these ROMs or use ./SOURCE/XILINX/ROM(E/O).VHD – CoreGen created synchronous distributed ROM memories.

3

Operation

Interface for host communication is pretty simple. When *idv* is high *dcti* input is sampled by core and when eight input values are read internal DCT processing is started. MDCT core is always ready for new data, so there is no need for flow control. New input data can be feed into the core back to back with *idv* all the time pulled high.

Regarding output DCT handling, there is *odv* pin - when high means that *dcto* output is valid (*dcto* output is 2D DCT transformed data).

Below is example of host data transfer to MDCT core (only output values after first DCT stage are presented, ie. *odv1* and *dcto1*).

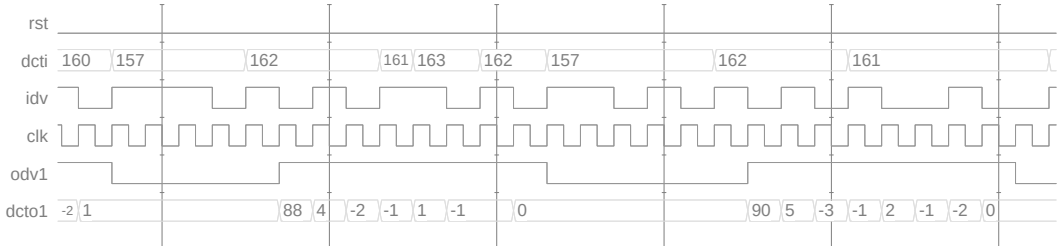

Figure 3. MDCT operation

MDCT core uses synchronous internal RAM (they are connected within core top level which is MDCT entity).

RAM write access example:

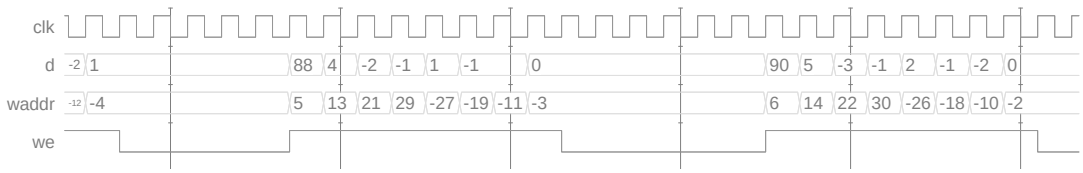

Figure 4. Synchronous RAM write access

RAM read access example:

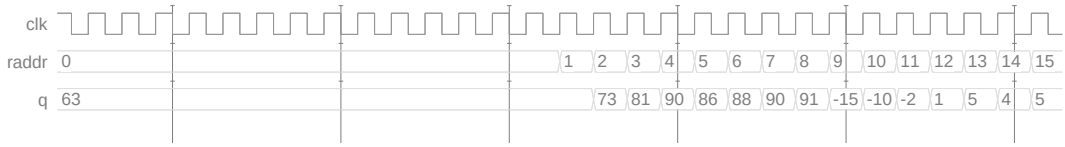

Figure 5. Synchronous RAM/ROM read access

RAM read access waveform is also applicaple to on-chip ROM memories used. There is one cycle delay between read address and output data valid.

4

Registers

There are no internal registers in the MDCT core.

5 Clocks

This section specifies all the clocks.

There is only one clock used by core.

Name	Source	Rates (MHz)			Remarks	Description
		Max	Min	Resolution		
clk	Input Pad	N/A	N/A	N/A	Duty cycle N/A	Core clock

Table 1: List of clocks

6

IO Ports

This section specifies the core IO ports.

MDCT core has following ports:

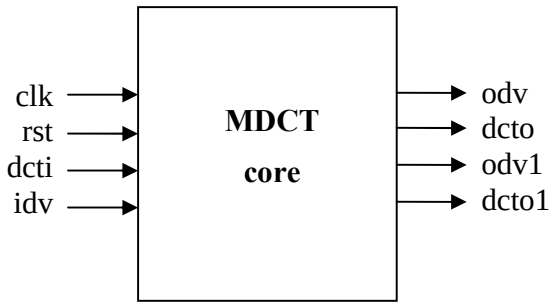

Figure 6. Core interface

PIN	width	direction	description
clk	1	input	clock
rst	1	input	async reset
dcti	8	input	input data
idv	1	input	input data valid (image sample)
odv	1	output	output data valid
dcto	12	output	output data (DCT transformed)
odv1	1	output	output data valid after 1st DCT stage (for debug)
dcto1	12	output	output data after 1st DCT stage (for debug)

Table 2. Pins description

Appendix A

Performance

Core Performance/area with current implementation.

Performance: 512 x 512 x 8 bit image is 2D DCT transformed in about 26.3 ms with 10 MHz input clock. This gives 10 mega samples per second throughput with this frequency. When the pipeline is full new 2D DCT data is output on every clock cycle. Core latency is 85 clock cycles (time for latching first input to giving first DCT output).

Area: 1550 slices and 2 RAMB16s Blockrams on Xilinx Spartan3 xc3s1000 device (45 MHz max freq) with asynchronous on chip ROM. Using synchronous distributed ROM (with one clock delay) area is larger (1620 slices) but performance is better (55 MHz).

Appendix B

Testbench

Self-veryfing testbench included which takes matlab-converted image as input. Core transforms it to DCT coefficients and behavioral IDCT testbench code reconstructs from it original image. PSNR is computed between original and reconstructed image to find out error introduced by fixed point arithmetic, for sample Lena images PSNR is 48 dB.

Matlab scripts are included for computing floating point DCT/IDCT as reference. Scripts for converting 8 bit bitmap to txt format readable by testbench and vice versa are also available.

You can see here MDCT core-transformed image and reconstructed by behavioral IDCT.

There are few Matlab scripts to aid core verification stored in ./Matlab directory:

dct_func.m

Matlab dct_func() function which computes floating point 1D DCT using butterfly method. Warning: output from this function is transposed, so applying this function twice will produce 2D DCT without need to transpose after 1st stage.

dctf.m

m-file for testing dct_func() operation. Before input is taken by dct_func() it is level shifted (128 number is subtracted from every input sample). M-file dctf.m also performs quantization after DCT for testing purposes using quantization matrix mentioned in JPEG STD.

dctm.m

Computes 2D DCT using equation:

$$X(u,v) \quad \frac{2}{N} \; C(u) \; C(v) \sum_{i=0}^{N\ 1} \sum_{j=0}^{1} x(x,y) \cos\frac{(2i \quad 1)u}{2N} \cos\frac{(2j \quad 1)v}{2N}$$

where C(u),C(v) $= 2^{-1/2}$ for u=0,v=0

$= 1$ otherwise

This equation was used as reference for checking validity of row/column method. Before input is taken to equation above it is level shifted (128 number is subtracted from every input sample).

img2txt.m

Converts bitmap (*.bmp) image to txt format readable by MDCT testbench core. If you want to use your own image for testing do the following:

Set your bitmap filename in first line of img2txt.m and copy your bitmap to ./Matlab folder. Run img2txt from Matlab – this will create txt file with extension *.txt and basename as your bmp filename. Then copy 'filename'.txt to ./SOURCE/TESTBENCH and update **constant FILEIN_NAME_C** to match your image basename. Then you are good to go with running testbench. Note that image sizes (width, height) must be multiply of DCT block size which is set to 8.

txt2img.m

Converts testbench output images (error image and reconstructed image) to JPEG format. After VHDL testbench finished testing, copy ./SOURCE/TESTBENCH/imageo.txt and ./SOURCE/TESTBENCH/imagee.txt to ./MATLAB folder and run txt2img.m to obtain imageo.jpg and imagee.jpg. Then you can visually estimate if testing went good.

Testbench configurable constants

There are few constants in MDCTTB_PKG.VHD package which configure testing environment:

ENABLE_QUANTIZATION_C
When true testbench quantizes MDCT core output with quantization matrix given by Q_MATRIX_USED constant. So far two quantization matrices are supported: Q_JPEG_STD (from JPEG standard) and Q_CANON10D (used in Canon EOS digital camera, super fine quality). Warning: when you enable quantization no error checking is performed on image being transformed.

RUN_FULL_IMAGE
When true, testbench transforms image with name from FILEIN_NAME_C constant. Can be set to false to speed up simulation – there are independent testing scenarios used before image transformation scenario.

Testbench uses random number generator (RANDOM1.VHD) by Gnanasekaran Swaminathan to create different input loading of data to be transformed in one of test cases (ie. clock cycles between consecutive *dcti* samples).

Index

This section contains an alphabetical list of helpful document entries with their corresponding page numbers.