|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI  **VIỆN ĐIỆN TỬ - VIỄN THÔNG**  logo_128  **REPORT**  **DIGITAL DESIGN II**  **Topic : Decrete Cosin Transform**   |  |  |  | | --- | --- | --- | | **Lecturers:** | **Nguyễn Đức Minh** | | | **Students:** | **Phạm Tuấn Anh** | **20182903** | |  | **Đinh Sỹ Dũng** | **20182912** | |  | **Giang Vũ Tuấn Hải** | **20182917** | |  |  |  | |  |  |  | |  |  |  |     Hà Nội, 2-2022 |

[CHAPTER 1: INTRODUCTION……………………………………………………………3](#_Toc91591792)

[CHAPTER 2: : DISCRETE COSINE TRANSFORM AN OVERVIEW 4](#_Toc91591798)

[2.1 The One-Dimensional DCT 4](#_Toc91591799)

[2.2 The Two-Dimensional DCT 5](#_Toc91591800)

[2.3 Properties of DCT 5](#_Toc91591800)

[2.3.1 Decorrelation ***5***](#_Toc91591801)

[2.3.2 Energy Compaction ***5***](#_Toc91591802)

[2.3.3 Separability ***6***](#_Toc91591803)

[2.3.4 Symmetry ***6***](#_Toc91591803)

[2.3.5 Orhogonality ***6***](#_Toc91591803)

[CHAPTER 3: IMPLEMENTATIONS OF DECRETE COSIN TRANSFORM *7*](#_Toc91591804)

[3.1 Specification 7](#_Toc91591805)

[3.1.1 Input ***7***](#_Toc91591801)

[3.1.1 Output ***7***](#_Toc91591801)

[3.2 Architecture, block diagram and FSM 8](#_Toc91591806)

[3.2.1 Architecture ***8***](#_Toc91591807)

[3.2.2 Block diagram ***9***](#_Toc91591808)

[3.2.3 FSM ***9***](#_Toc91591808)

[3.3 Algorithm and ASMD 10](#_Toc91591809)

[3.3.1 Algorithm 10](#_Toc91591810)

[3.3.2 ASMD 12](#_Toc91591811)

[3.3.2 ASMD 14](#_Toc91591811)

[3.3 Timming Analysis 18](#_Toc91591809)

[3.3 Verilog code 19](#_Toc91591809)

[CHAPTER 4: SIMULATION *23*](#_Toc91591804)

[CHAPTER 5: RESULT AND COMPARE *26*](#_Toc91591804)

[5.1 Result 26](#_Toc91591805)

[5.2 Compare 27](#_Toc91591805)

[5.3 Comment 29](#_Toc91591805)

[CHAPTER 6: CONCLUSION AND REFERENCES *30*](#_Toc91591804)

# LIST OF FIGURES

[Figure 3.1 Architechture 8](#_Toc91591431)

[Figure 3.2 Block diagram. 9](#_Toc91591432)

[Figure 3.3 FSM 9](#_Toc91591433)

[Figure 3.4 Algorithm diagram 11](#_Toc91591434)

[Figure 3.5 ASMD diagram 12](#_Toc91591435)

[Figure 3.6 Control unit and Datapath 13](#_Toc91591436)

[Figure 3.7 Load state algorithm 14](#_Toc91591437)

[Figure 3.8 One\_dct state algorithm 15](#_Toc91591438)

[Figure 3.9 1D-DCT architecture 16](#_Toc91591439)

[Figure 3.10 Transpose state algorithm 17](#_Toc91591440)

[Figure 3.11 Timming 18](#_Toc91591441)

[Figure 3.12 DCT module code 19](#_Toc91591442)

[Figure 4.1 Testbench module code 24](#_Toc91591443)

[Figure 4.2 Input data 25](#_Toc91591444)

[Figure 5.1 Model Sim wave 26](#_Toc91591444)

[Figure 5.2 Output data 27](#_Toc91591444)

[Figure 5.3 Output data of C++ code 28](#_Toc91591444)

[Figure 5.4 Compare result 29](#_Toc91591444)

# LIST OF TABLES

CHAPTER 1: INTRODUCTION

Discrete cosine transform (DCT) is widely used transform in image processing, especially for compression. Some of the applications of two-dimensional DCT involve still image compression and compression of individual video frames, while multidimensional DCT is mostly used for compression of video streams and volume spaces. Transform is also useful for transferring multidimensional data to DCT frequency domain, where different operations, like spread-spectrum data watermarking, can be performed in easier and more efficient manner. In this project we will design a simple dct system based on the traditional formula of dct

CHAPTER 2: DISCRETE COSINE TRANSFORM AN OVERVIEW

The Discrete Cosine Transform (DCT) was first proposed by Ahmed et al. (1974), and it has been more and more important in recent years. DCT has been widely used in signal processing of image data, especially in coding for compression, especially in lossy compression, for its near-optimal performance. Because of the wide-spread use of DCT's, research into fast algorithms for their implementation has been rather active ,and also, since the DCT is computation intensive, the development of highspeed hardware and real-time DCT processor design have been object of research .

Discrete cosine transform (DCT) is widely used in image processing, especially for compression. Some of the applications of two-dimensional DCT involve still image compression and compression of individual video frames, while multidimensional DCT is mostly used for compression of video streams. DCT is also useful for transferring multidimensional data to frequency domain, where different operations, like spread-spectrum, data compression, data watermarking, can be performed in easier and more efficient manner..

## **2.1 The One-Dimensional DCT**

The most common DCT definition of a 1-D sequence of length N is

where:

In this project we use N = 8

## **2.2 The Two-Dimensional DCT**

The 2-D DCT is a direct extension of the 1-D case and is given by

where:

In this project we use N = 8

## **2.3 Properties of DCT**

### 2.3.1 Decorrelation

The principle advantage of image transformation is the removal of redundancy between neighboring pixels. This leads to uncorrelated transform coefficients which can be encoded independently.

### 2.3.2 Energy Compaction

Efficacy of a transformation scheme can be directly gauged by its ability to pack input data into as few coefficients as possible. This allows the quantizer to discard coefficients with relatively small amplitudes without introducing visual distortion in the reconstructed image. DCT exhibits excellent energy compaction for highly correlatedimages

### 2.3.3 Separability

This property, known as separability, has the principle advantage that C (u, v) can be computed in two steps by successive 1-D operations on rows and columns of an image. The arguments presented can be identically applied for the inverse DCT computation

### 2.3.4 Symmetry

Another look at the row and column operations in Equation of 2D DCT reveals that these operations are functionally identical. Such a transformation is called a symmetric transformation. A separable and symmetric transform can be expressed in the form .

T = AfA

### 2.3.5 Orthogonality

In order to extend ideas presented in the preceding section, let us denote the inverse transformation of above as

CHAPTER 3: IMPLEMENTATIOS OF DISCRETE COSINE TRANSFORM

## **3.1 Specification**

### 3.1.1 Input

In this project we use input as a matrix 8x8 which represent for an image file infomation ( pixel value in term of integer having range 0 -255 )

* Positive Integer number
* Range [0 : 255]
* Represent : 8 bits

### 3.1.2 Output

The output of 2D-DCT will be a real matrix 8x8 with range from -2047 to 2047

* Real number
* Range [-2047 : 2047]
* Represent: 32 bits fixed point
  + 1 sign bit
  + 11 bits for integer part
  + 20 bits for fractional part

## **3.2 Architecture, block diagram and FSM**

### 3.2.1 Architecture

A picture containing timeline

Description automatically generated

Figure 3.1 Architecture

* Clk : clock signal
* Reset: reset signal
* Start: start signal
* Data\_in: 1 point of input data – 64 point input data will be put in one by one
* Data\_out: 1 point of ouput data – 64 point output data will be export one by one
* exportProduct: signal to start outputting data
* finish: finish signal

### 3.2.2 Block diagram

Diagram

Description automatically generated

Figure 3.2 Block diagram

### 3.2.1 FSM

Diagram

Description automatically generated

Figure 3.3 FSM

## **3.3 Algorithm and ASMD**

### 3.3.1 Algorithm

* Load:
  + Input data\_in: load and save input data to memory
  + Output data\_out ( product )
* ONE\_DCT: do 1D-DCT with each 8 points of input data
* TRANSPOSE: transpose data
* transpose\_cnt: count number of transpose

Diagram

Description automatically generated

Figure 3.4 Algorithm diagram

### 3.3.2 ASMD

Diagram

Description automatically generated

Figure 3.5 ASMD diagram

Diagram

Description automatically generated

Figure 3.6 Control unit and Data path

***Explain:***

* control: state control signal
  + 00: IDLE state
  + 01: LOAD state
  + 10: ONE\_DCT state
  + 11: TRANSPOSE state
* load\_control: load control signal
  + 0: load input data and save to memory
  + 1: start export product
* transpose\_cnt: count number of transpose
* one\_dct\_cnt: count number of ONE\_DCT ( each time ONE\_DCT will do 1D-DCT with 8 points data )

### 3.3.3 States Algorithm

1. LOAD State

Diagram

Description automatically generated

Figure 3.7 Load state algorithm

1. ONE\_DCT State

Diagram

Description automatically generated

Figure 3.8 One\_dct state algorithm

***Explain:***

* x[0:63][31:0] : 64 point data memory
* one\_dct\_cnt: count number of ONE\_DCT ( each time ONE\_DCT will do 1D-DCT with 8 points data )

Diagram

Description automatically generated

Figure 3.9 1D-DCT architechture

1. TRANSPOSE State

Diagram

Description automatically generated

Figure 3.10 TRANSPOSE algorithm

***Explain:***

* x[0:63][31:0] : 64 point data memory
* row\_cnt: count number of rows
* col\_cnt: count number of columns

## **3.4 Timming Analysis**

Table

Description automatically generated

Figure 3.11 Timming

## **3.5 Verilog code**

Text

Description automatically generated

Figure 3.12 DCT module code

Text

Description automatically generated

Figure 3.13 Controller module code

A picture containing text, monitor, screen

Description automatically generated

Figure 3.14 Datapath module code

A picture containing text

Description automatically generated

Figure 3.15 One\_dct module code

CHAPTER 4: SIMULATION

## **4.1 Simulation**

* Step 1: Write testbench verilog code (figure 4.1)
* Step 2: Generate input datas ( 20 input data files )
  + Generate input data with C++ code
  + Save input data to text file with format as figure 4.2
* Step 3: Simulate with designed module for each input file
* Step 4: Save output data to text file
* Step 5: Repeat step 3 and step 4 with other input files

Text

Description automatically generated

Figure 4.1 Testbench module code

Graphical user interface, text

Description automatically generated

Figure 4.2 Input data

CHAPTER 5: RESULT AND COMPARE

## **5.1 Result**

Diagram

Description automatically generated

Figure 5.1 Model Sim wave

Graphical user interface, text, application

Description automatically generated

Figure 5.2 Output data

## **5.2 Compare**

* Step 1: Do 2D-DCT with C++ code with the same input file
  + Convert encode input to real number
  + Do 2D-DCT with real number
* Step 2: Export output datas to files
* Step 3: Compare result with verilog output
  + Convert encode output to real number
  + Compare with each point data
  + Save compare result to text file
* Step 4: Repeat with other files

Graphical user interface, text

Description automatically generated

Figure 5.3 Output data of C++ code

Graphical user interface, text

Description automatically generated

Figure 5.3 Compare result

## **5.2 Comment**

The comparison results show that the average error rate per point of ouput data in hardware compared to software is very small, averaging about 0.0003%

CHAPTER 6: CONCLUSION

The Discrete Cosine Transform is one of the most widely transform techniques in digital signal processing. In addition, this is also most computationally intensive transforms which require many multiplications and additions. Real time data processing necessitates the use of special purpose hardware which involves hardware efficiency as well as high throughput. Many DCT algorithms were proposed in order to achieve high speed DCT.

In this project we use an algorithm based on the traditional formula of 2D-Dct. The system is not small and fast but it works relatively accurately.

**REFERENCES**

DESIGN OF 2D DISCRETE COSINE TRANSFORM USING CORDIC ARCHITECTURES IN VHDL - A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF Master of Technology

https://ieeexplore.ieee.org/document/7348130