**ELEC 4320 Final Project Report**

Student 1: XUE Hongjia UST ID: 20761325

Student 2: LI, Zhengdong UST ID: 20693346

*Topic 5. Two-dimensional Streamed and Pipelined Discrete Cosine Transformation (DCT) Kernel (Difficulty Level-5)*

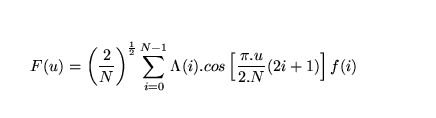
*In this project, you are required to design a kernel that can achieve DCT. The application takes in 64 input (for 2-dimensional, it is an 8-by-8 matrix) with 16-bit or higher precision. Proper pipeline stages between arithmetic operations should be well considered and you should also consider shifting the operands before and after multiplication to preserve data precision. The streamed design requires taking eight input data (i.e., a row or a column) at a clock cycle after “start” signal is given to initialize the kernel execution.*

**1. The background of the project**

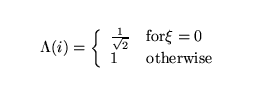
Introduction about DCT:

DCT expresses a finite sequence of data in terms of a sum of cosine functions oscillating at different frequencies[1]. Usually it is used in image compression to reduce the size by eliminating the high frequencies. It helps to separate the image into parts with different weightings. The idea is, we first divide the image in a block of 8-by-8 matrix with pixel value (in integer) from 0 to 255. After we apply 2-D DCT, we can find that most of the data are concentrated on low frequencies. That’s why we can compress the image by throwing away the high frequencies to achieve size reduction. [1] Among various transforms, DCT is the most popular and effective one in image and video compression. So far, it has been adopted by many international standards like JPEG, MPEG1, MPEG2, MPEG4, and H26x, etc.

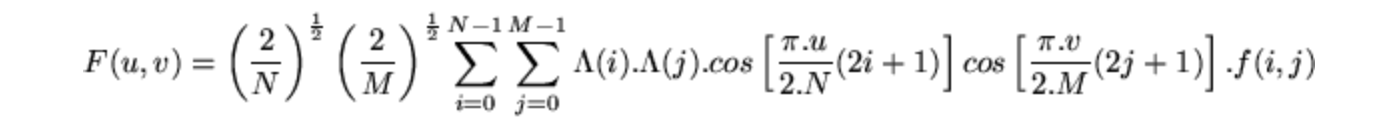
Formula for 1-D DCT: [2]



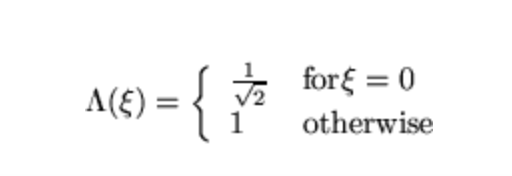
Where



Similarly, for 2-D DCT [2]:



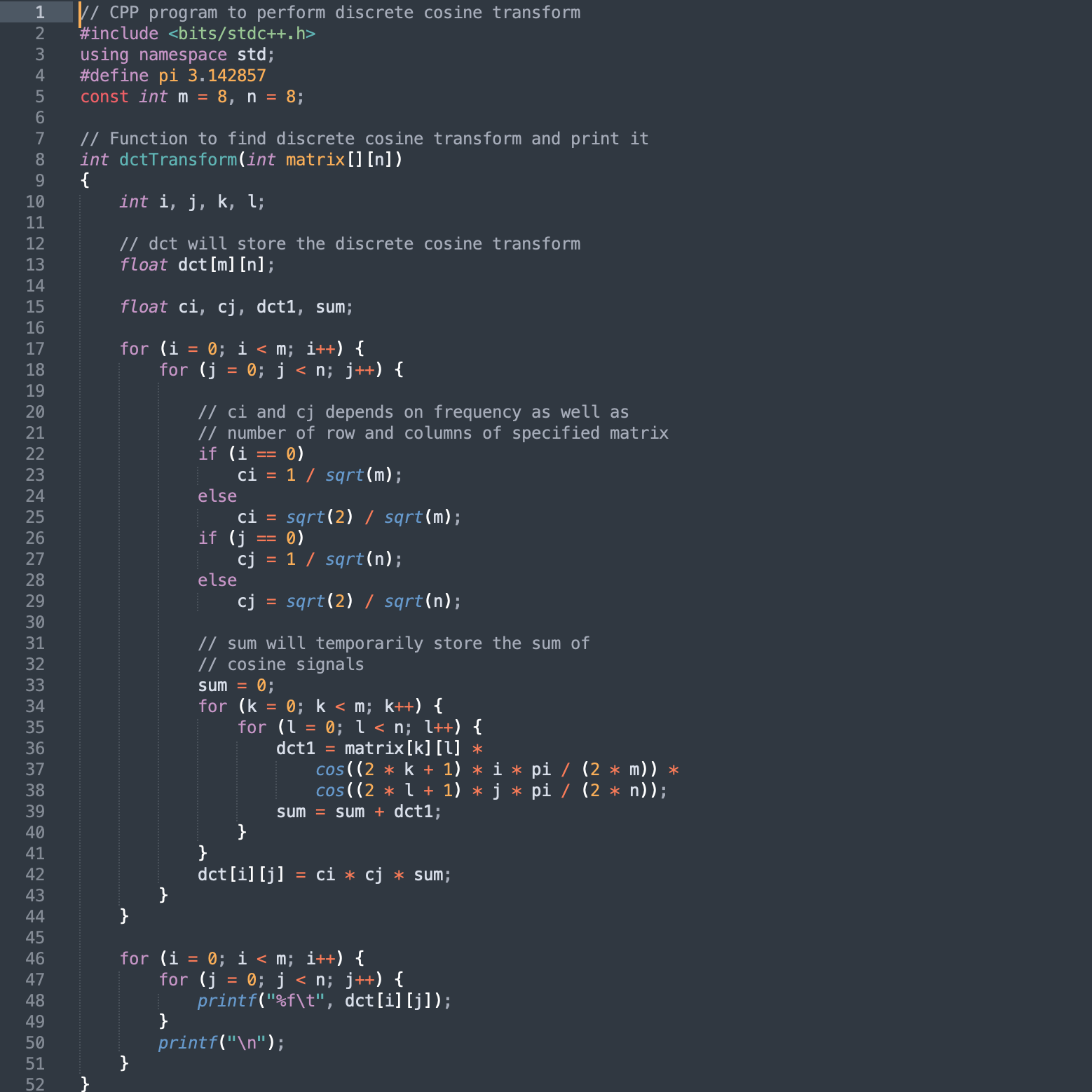
Where



In our design, we will first implement 1-D DCT for row first, and then 1-D DCT for column so that we can achieve a 2-D DCT for a 8-by-8 matrix.

**2. Existing solutions**

If the algorithm is directly implemented with the nested loop formulas for two-dimensional DCT, its time complexity would be O(N^4). For example, if we perform a 2-D DCT using C++ as below [3]



We need 4 nested loops to do so the time complexity is O( N^4 ) ,( in this case, N = 8 ) which is not convenient.

Hence, in our design, we perform an optimization using the pipeline such that some loops can be executed earlier and we can run them concurrently.

With the row-column computation method, it is allowed to perform optimization tricks such as pipelining and parallel computation, which will be discussed in detail in the solution flow section and the complexity would be O( N ). The major challenge of this design is increase in hardware usage and complexity of control flow.

**3. Proposed solution flow**

图示, 工程绘图

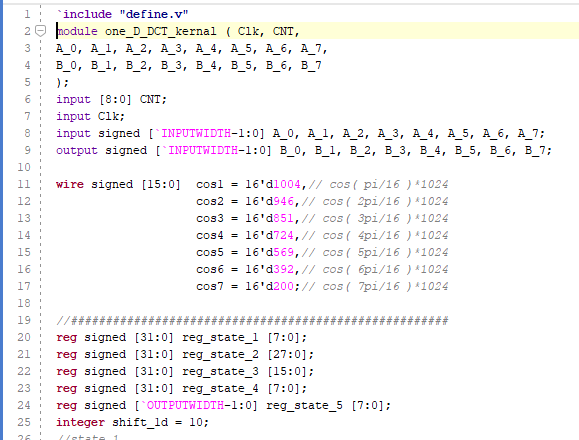
描述已自动生成

The flow of the project can be described in the above figure.

Register cycle\_cnt is used to count the number of cycles after the starting signal (at 1000ns). If minimum cycle numbers for different stages have already been achieved, the kernel will enter the next stage according to cycle\_count. We first perform the 1-D DCT for each row of the input Matrix\_A ( 8 rows in total ) and output the 1d\_DCT of different frequencies, which will be stored into temp\_Matrix\_B. After we finish all the rows, the kernel performs a 1-D DCT for each column of temp\_Matrix\_B and stores the values to temp Matrix\_C. Finally we can output the whole matrix\_C to Output\_RAM as the final result of 2-D DCT.

**4. Detailed implementation in EDA tools**

1D-DCT module:



and according to the equation in section 1, the input–output relation of this 1d\_DCT can be rearranged in the following table:

表格

描述已自动生成

and this matrix multiplication could be broken down into the following 5 stages:

#### *Stage 1*

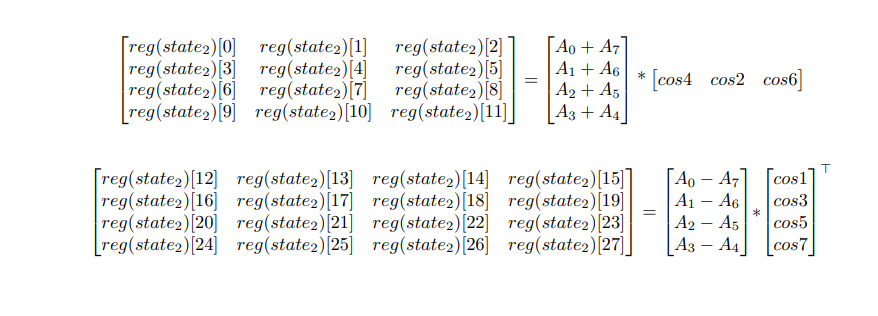
Calculate sum & difference

文本

描述已自动生成

#### *Stage 2*

Calculate the components for the even output and odd output according to following table:



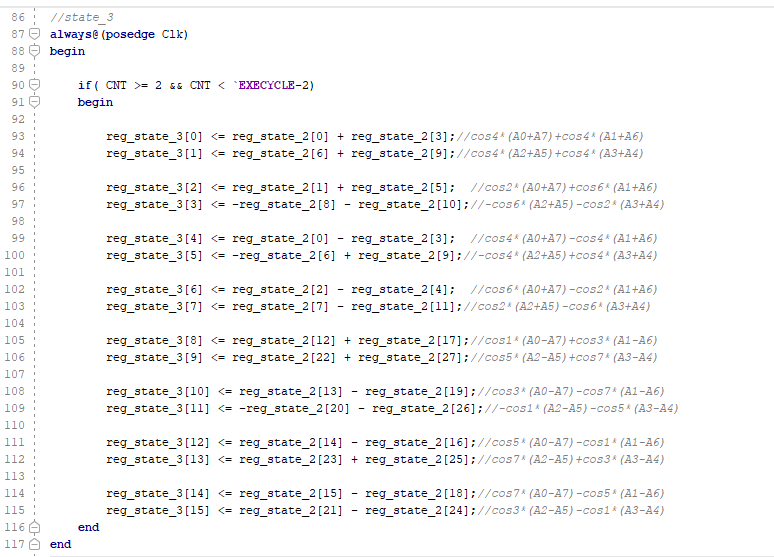
note that the cosine values here are the actual cosine values times 1024(2^10) since the hardware we used could only perform operation for integers

another special notice is that multiplication with A0 = 1/sqrt(2) can be achieved with the cosine factor cos( 4pi/16 )  ( further increase reusability )

since each output of DCT is the sum of 4 terms ( according to equation above ), 2 levels of addition/subtraction are needed, as is described below:

#### *Stage 3*

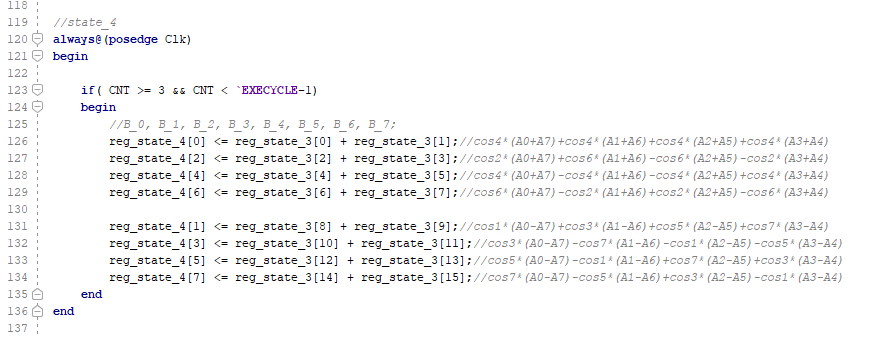
Level-one addition/subtraction



#### *Stage 4*

Level-two addition

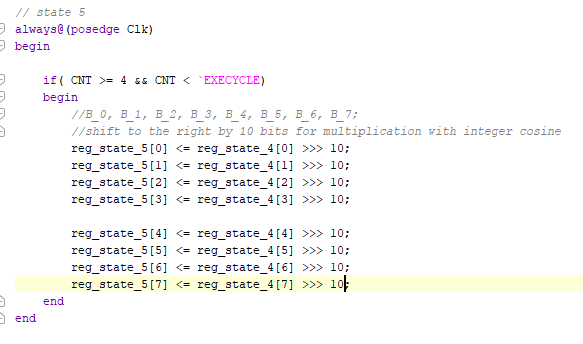
Simply add up the term in stage3



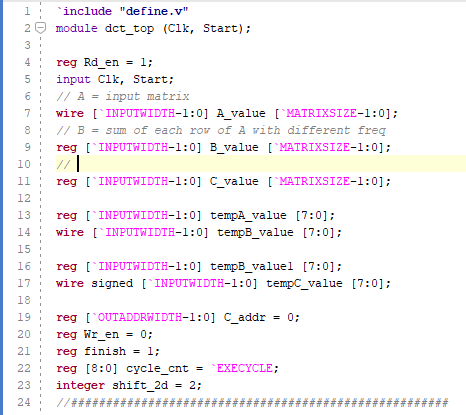
#### *Stage 5*

Shift to get the correct result

At this stage, the results obtained in stage 4 will be shifted to right arithmetically by 10 bits for the precision of the elementary cosine value and then assigned to the output registers



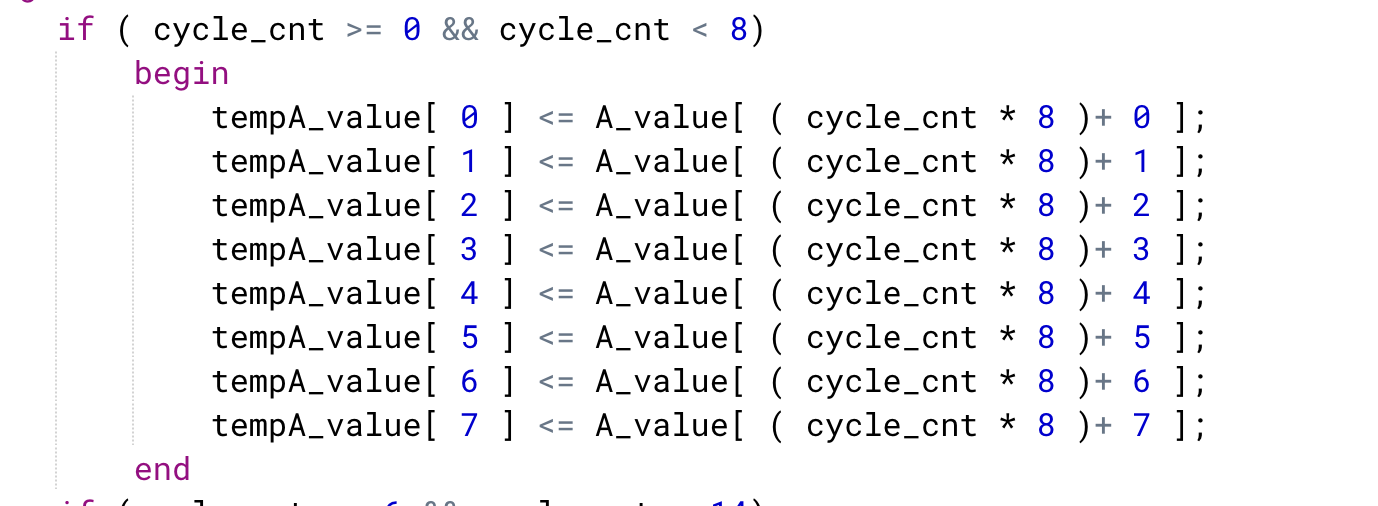
2D-DCT module:



In this design, array A\_value is the input 8-by-8 matrix while array C\_value is the output of our kernel.

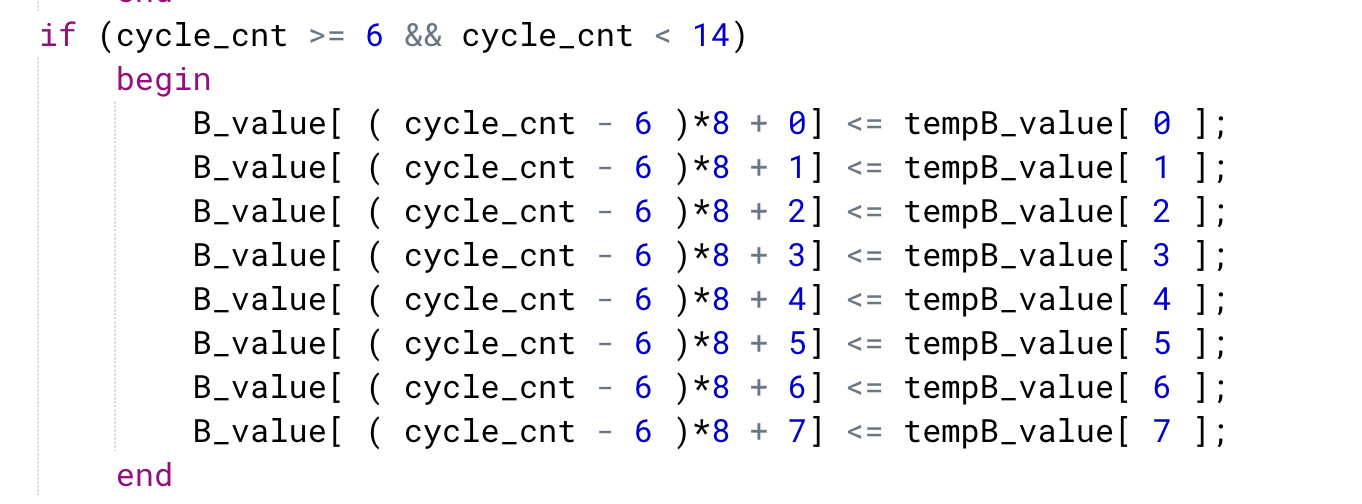
From cycle\_cnt = 0 to cycle\_cnt = 7:

The 1st row goes into the first 1\_d DCT module at cycle\_cnt = 1 and gets out at cycle\_cnt = 6 after 5 stages of pipeline, and similarly the 8th row will start at cycle\_cnt = 8 and finish at cycle\_cnt = 13. The kernel takes the input matrix (A\_value) row by row and stores them into array tempA\_value, which is the input of the first 1d\_DCT.



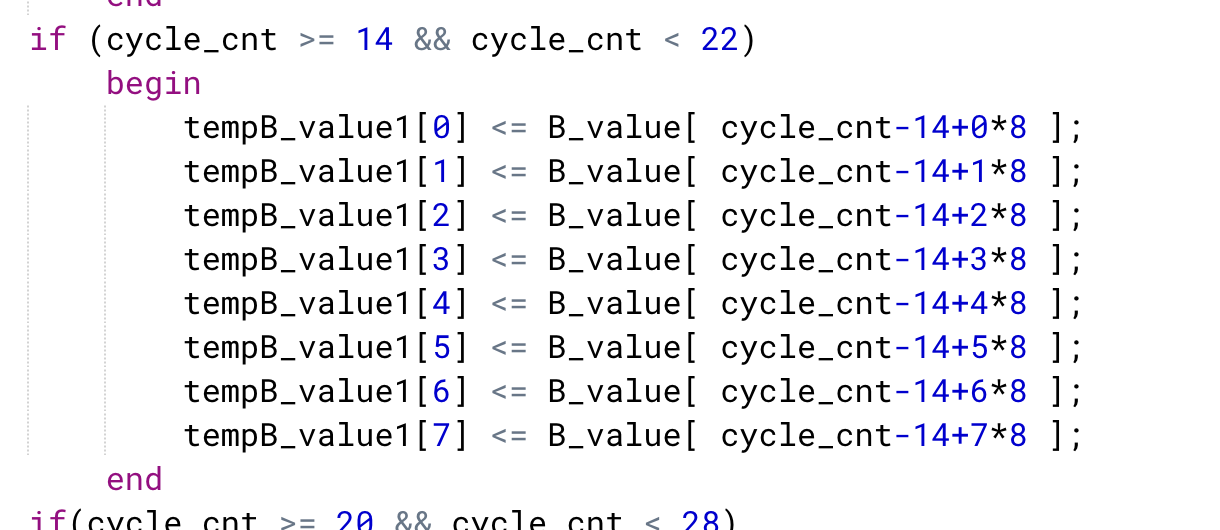
From cycle\_cnt = 6 to cycle\_cnt = 13:

Since the 1d\_DCT output of the 1st row finishes and is ready for writing into Matrix B\_value at cycle\_cnt = 6, we will store the values to array tempB\_value starting from cycle\_cnt=6, and transfer them to array B\_value row by row.



From cycle\_cnt = 14 to cycle\_cnt = 21:

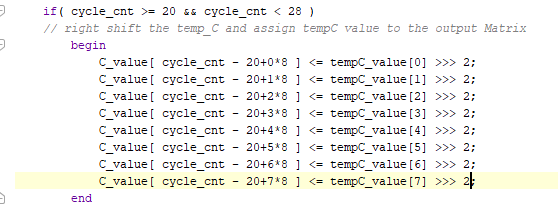
We are dealing with the column of DCT in this period. Since the DCT operations for all rows of Matrix\_A have already finished at cycle\_cnt = 13, hence we will start from cycle\_cnt = 14. The 1st column starts at cycle\_cnt = 14 and finishes at cycle\_cnt = 19, and similarly the 8th column will start at cycle\_cnt = 21 and finish at cycle\_cnt = 26.



From cycle\_cnt = 20 to cycle\_cnt = 27:

Since the 1st column ends at cycle\_cnt = 19, we can get the final result starting from cycle\_cnt = 20, hence, we will store the value from tempC\_value to C\_value starting from this period. We shift them to right by 2 bits according to the formula in Section 1. ( for in this case, N = M = 8, equal to divide by 4 )

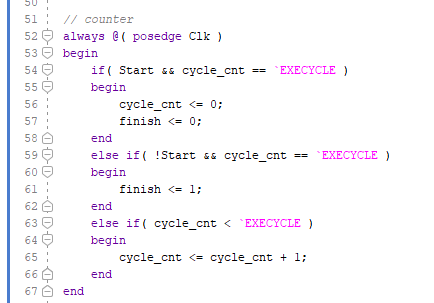


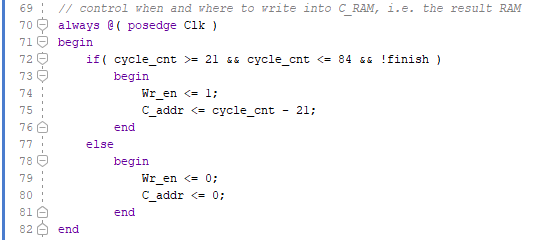


And starting from cycle\_cnt = 21 (to cycle\_cnt = 84):

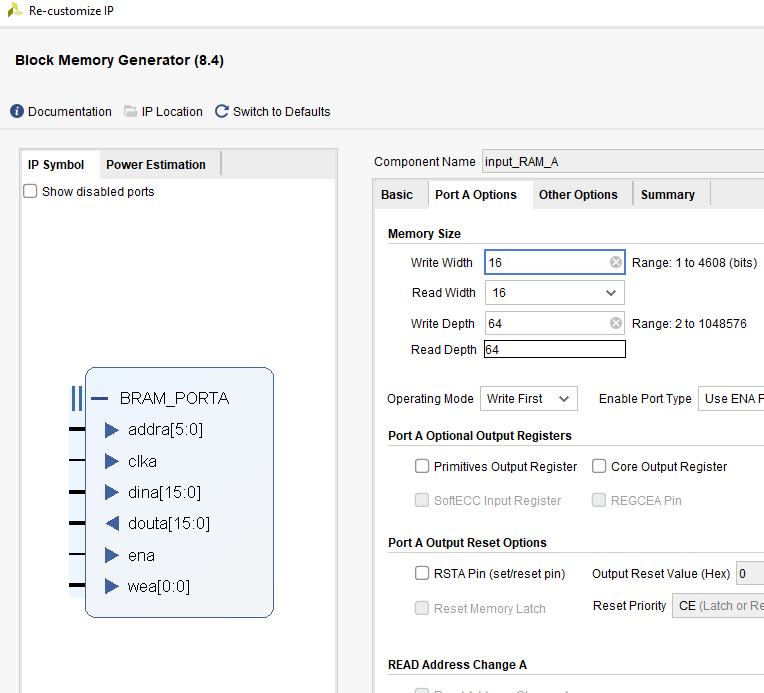
we transfer the C\_value to Output\_RAM as the final results by enabling Wr\_en ( set it to 1 ) and assigning proper address for output Matrix. If it is finished ( cycle\_cnt reaches 84 ), then we assign Wr\_en to 0 to stop it.

For input and output storage, this design has made good use of BRAM with specified width(16) and depth ( 64 )

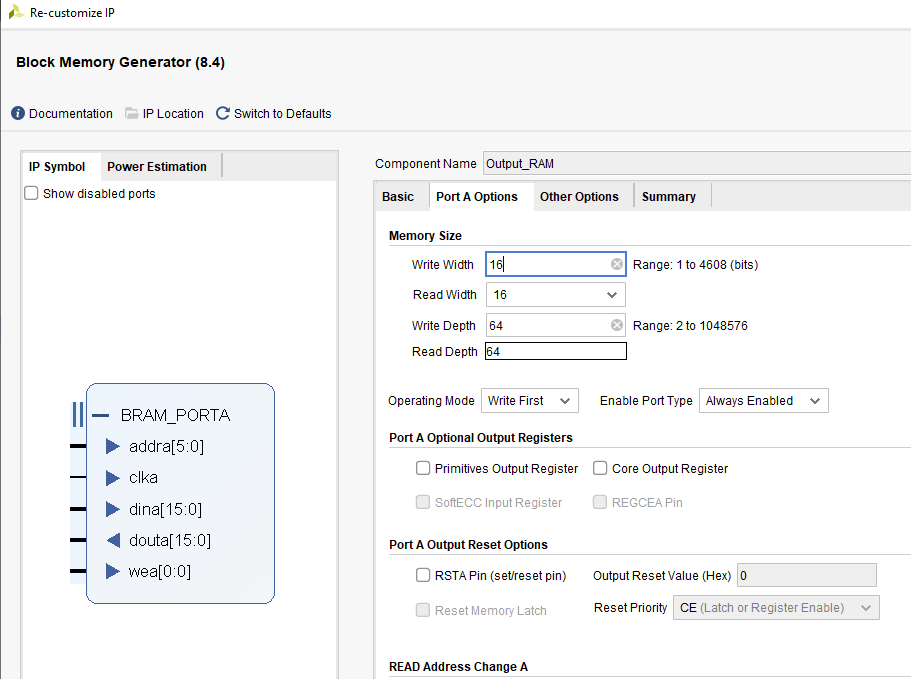




Input BRAM:

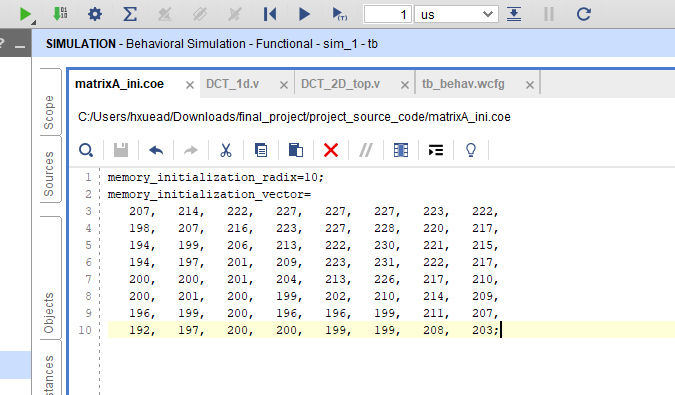


Output BRAM:



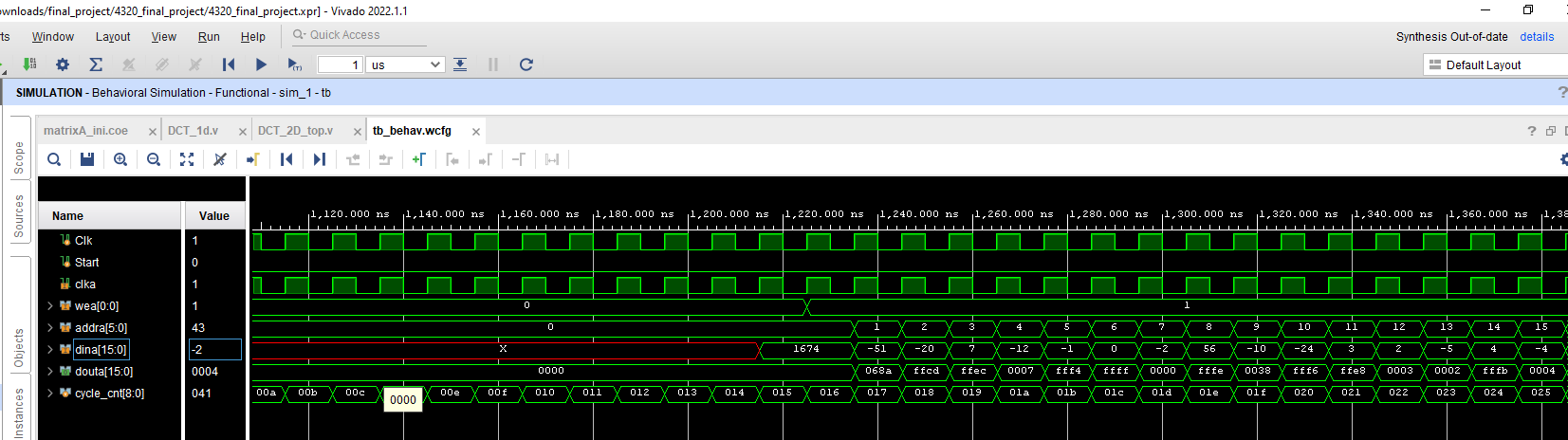
**5. Experimental results**

**Input matrix for simulation:**

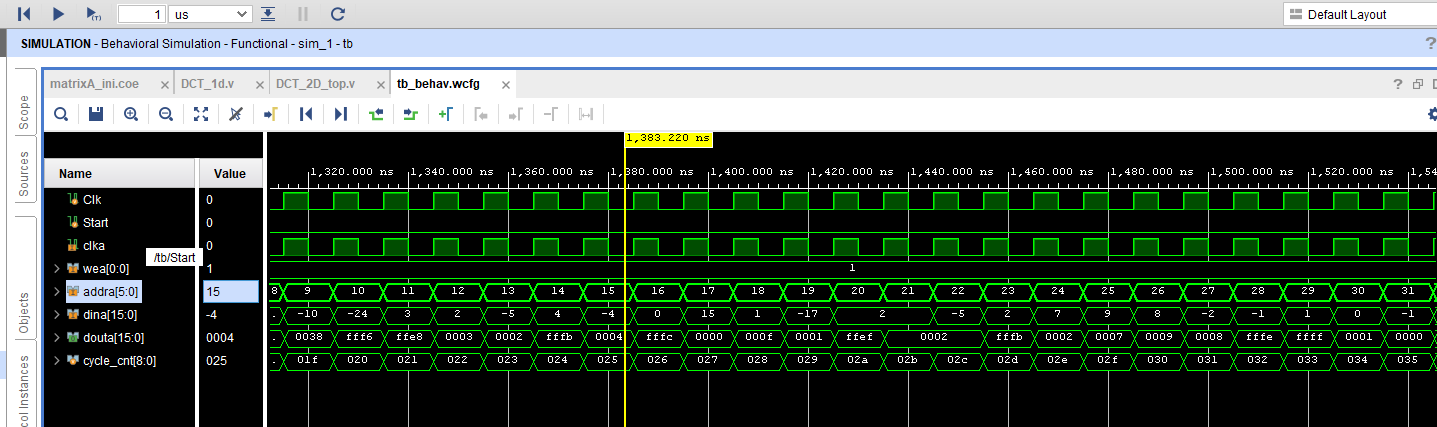
****

Behavioral simulation result:

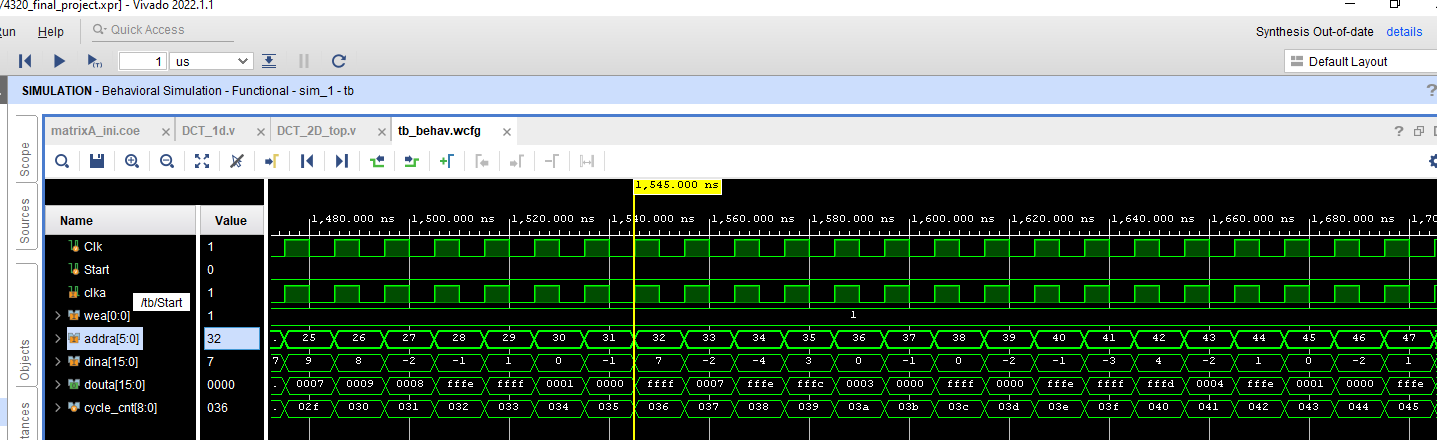
( output of the DCT module with write address 0-15 ):



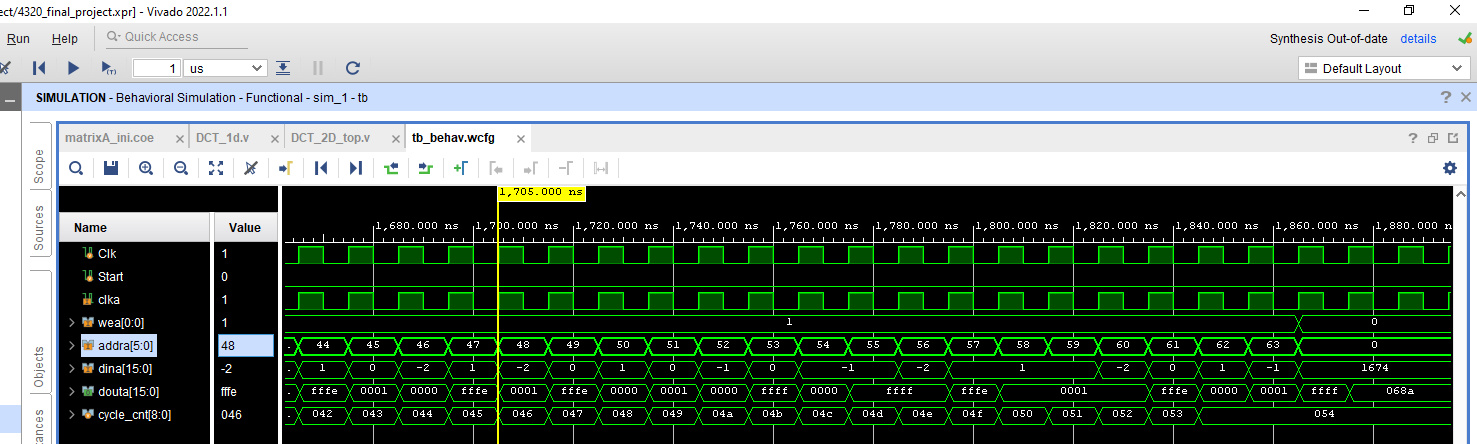
( written address 16-31 ):



( written address 32-47 ):



( written address 48-63 ):



At the same time, a matlab simulation is also conducted for DCT result comparison.

Matlab code:

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

A = [207, 214, 222, 227, 227, 227, 223, 222;

198, 207, 216, 223, 227, 228, 220, 217;

194, 199, 206, 213, 222, 230, 221, 215;

194, 197, 201, 209, 223, 231, 222, 217;

200, 200, 201, 204, 213, 226, 217, 210;

200, 201, 200, 199, 202, 210, 214, 209;

196, 199, 200, 196, 196, 199, 211, 207;

192, 197, 200, 200, 199, 199, 208, 203

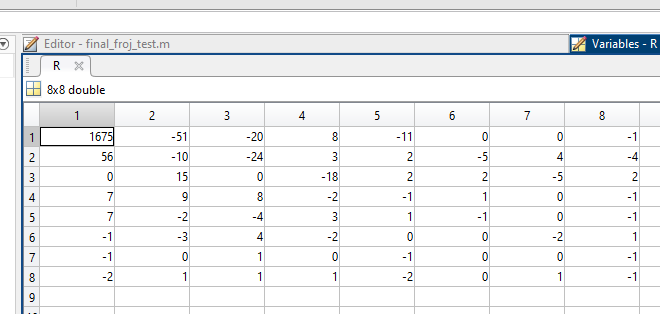
];

R = dct2(A);

R = floor(R);

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

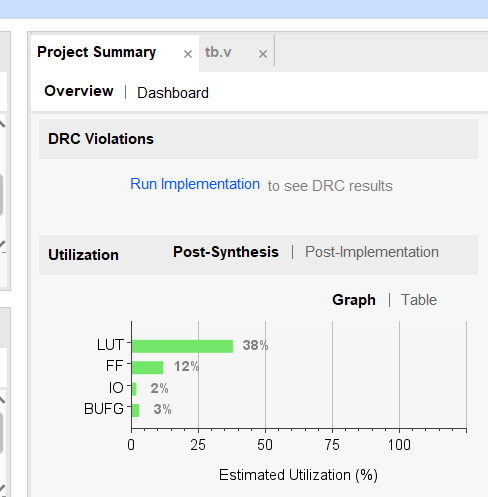
Matlab simulation result:



Post-synthesis resource usage:

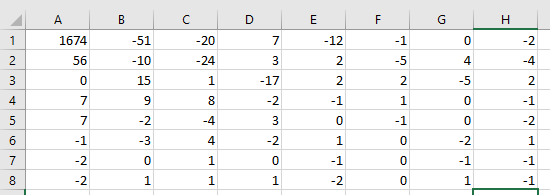
应用程序, 表格

中度可信度描述已自动生成



**6. Other interesting things you find in the project (optional)**

For clearer view, the resulting matrix by vivado:

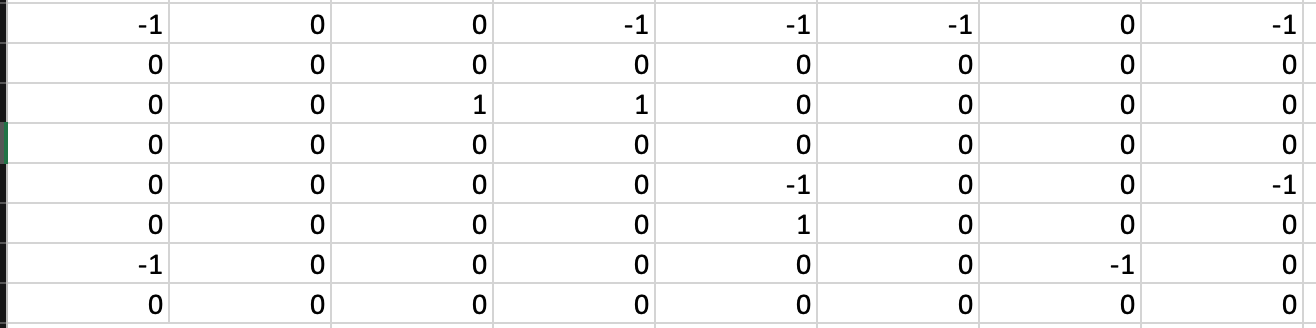




To compare the data precision,

we consider the **Absolute Error** = Va-Ve, ( where Va = measured value, Ve = exact value )

where Absolute error is shown below:



As it can be seen in section 5, in some of the results of the behavior simulation there are some small errors with magnitude less or equal to 1. But in terms of real application ( image compression ), such errors can generally be neglected as DCT is often followed by a Quantization process, [4] in which errors should be erased because of division.

A lot of codes are collected and reused from lab4a

References:

[1] <https://en.wikipedia.org/wiki/Discrete_cosine_transform>

[2] <https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node231.html>

[3] <https://www.geeksforgeeks.org/discrete-cosine-transform-algorithm-program/>

[4] https://en.wikipedia.org/wiki/Quantization\_(image\_processing)