Design and Implementation of an Image Co-Processor

Shihan Huang, Adrian Montero, Zhongyuan Ni, Akanksha Jain

ABSTRACT

In this paper we present an image co-processor for mobile and DSP applications used to accelerate discrete cosine transform (DCT), zigzag and run-length encoding. Computation expensive blocks, such as multiplications and quantization, are removed by storing pre-computed values in ROM lookup tables. The image co-processor was synthesized using a 130nm CMOS technology, resulting in an area of 3 mm^2. This design is capable of processing 720p at 100 frames per second and 1080p at 50 frames per second. The average power consumed is 370 mW, which results in an energy efficiency of 284 nJ per image block.

1. INTRODUCTION

As multimedia applications on mobile devices become more and more popular, supporting energy efficient image processing is a major concern. Implementing an image co-processor to which tasks are offloaded by a general-purpose processor provides a feasible solution to this problem. The area, performance, and energy efficiency can be independently optimized.

In this project, we design and implement a co-processor to accelerate DCT, zigzag and run-length encoding. The paper is organized as follows. Section 2 introduces basic concepts of DCT and the realized hardware implementation. Section 3 presents the hardware implementation of zigzag followed by run-length encoding in Section 4. Section 5 evaluates the area, performance, and energy efficiency of the design. Finally, the paper is concluded in Section 6.

2. DCT IMPLEMENTATION

2.1 DCT Basic

In this project, we implement a two-dimensional 8 point DCT, which is defined as:

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{i=0}^{7}\sum_{j=0}^{7}x(i,j)\cos\left[(2i+1)\frac{u\pi}{16}\right]\cos\left[(2j+1)\frac{v\pi}{16}\right] \quad (1)$$

Where $x(i,j)$ with $(i,j = 0,1,2,\dots,7)$ is the pixel value, $F(u,v)$ with $(u,v = 0,1,2,\dots,7)$ is the transformed coefficient and $C(0) = \sqrt{2}/2$, $C(u) = C(v) = 1$ if $u,v \neq 0$.

Note that this equation is a sum of products, a direct implementation would be a filter with 64 taps. And that is only for computing one $F(u,v)$. To calculate all 64 $F(u,v)$s, 64 such filters are needed. As you can see, this way of computing the transform leads to $8^4 = 4096$ additions and multiplications, which will occupy a large area.

In order to reduce the computation complexity, the algorithm is decomposed to two 1D DCT. Eq(1) can be broken into two parts:

$$F'(u,j) = \sum_{i=0}^{7}x(i,j)C(u)\cos\left[(2i+1)\frac{u\pi}{16}\right] \quad u,j = 0,1,2,\dots,7 \quad (2)$$

$$F(u,v) = \sum_{j=0}^{7}F'(u,j)C(v)\cos\left[(2j+1)\frac{v\pi}{16}\right] \quad u,v = 0,1,2,\dots,7 \quad (3)$$

Eq(2) and Eq(3) can be implemented identically while the Eq. (3) takes input from the output of Eq(2). Note that the 1D DCT can further be simplified as:

$$\begin{bmatrix}F(0)\\F(2)\\F(4)\\F(6)\end{bmatrix} = \begin{bmatrix}d & d & d & d\\ b & f & -f & -b\\ d & -d & -d & d\\ f & -b & b & -f\end{bmatrix}\begin{bmatrix}x(0)+x(7)\\x(1)+x(6)\\x(2)+x(5)\\x(3)+x(4)\end{bmatrix} \quad (4)$$

$$\begin{bmatrix}F(1)\\F(3)\\F(5)\\F(7)\end{bmatrix} = \begin{bmatrix}a & c & e & g\\ c & -g & -a & -e\\ e & -a & g & c\\ g & -e & c & -a\end{bmatrix}\begin{bmatrix}x(0)-x(7)\\x(1)-x(6)\\x(2)-x(5)\\x(3)-x(4)\end{bmatrix} \quad (5)$$

As the cosine function only takes a limited amount of results, we can pre-compute them instead of doing the cosine real-time.

2.2 Implementation

2.2.1 Pre-addition

We use one single adder to compute one of the addition result every cycle, and one of the subtraction result every two cycles. Therefore, it takes 12 cycles to complete the pre-addition. The reason we didn't paralyze this process is because the add accumulation is slower than the pre-addition. So there is no need to speed up this process but to keep area low.

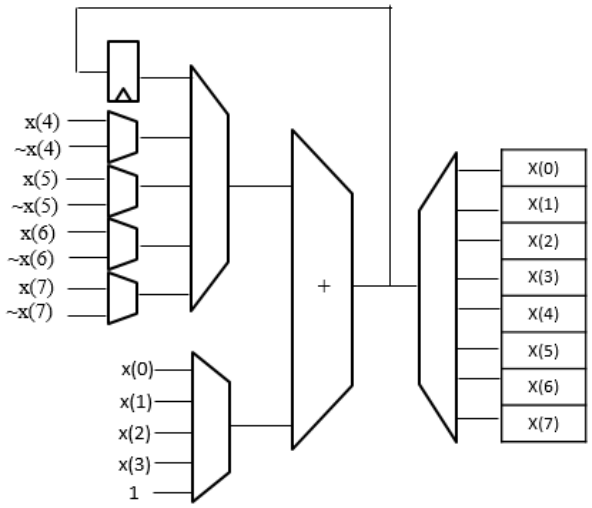

Figure 1. Pre-addition block diagram

2.2.2 Add Accumulation

In this project, 16 bits two's complement are used to represent one single data. Let b_k be the k th bit in the data x. Then

$$x = -b_0 + \sum_{k=1}^{15} b_k 2^{-k} \qquad (6)$$

Let $\boldsymbol{C} = [c(0)\ c(1)\ c(2)\ c(3)]$

and $\boldsymbol{X} = [x(0)\ x(1)\ x(2)\ x(3)]^T$, then

$$\boldsymbol{CX} = -\sum_{i=0}^{3} c(i)b_0(i) + \sum_{k=1}^{15}\left[\sum_{i=0}^{3} c(i)b_k(i)\right]2^{-k} \qquad (7)$$

Note that $\left[\sum_{i=0}^{3} c(i)b_k(i)\right]$ has only 16 possible outcomes given any combination of $[b_k(0)\ b_k(1)\ b_k(2)\ b_k(3)]$. Therefore, we can pre-compute the results, store them in a ROM and use $[b_k(0)\ b_k(1)\ b_k(2)\ b_k(3)]$ as look up address to retrieve the value. Then this calculation becomes an add-accumulation, which can be implemented in the following architecture:

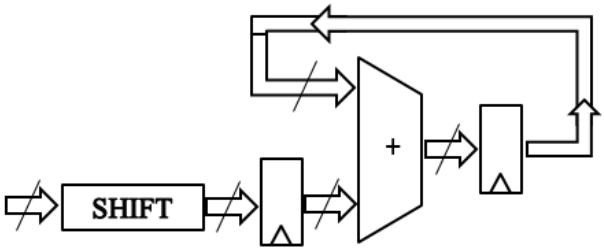

Figure 2. Add accumulation block diagram

It will take 16 cycles to add accumulate the final result. As there are 8 independent add accumulation calculations, we can process them in parallel.

2.2.3 1D DCT Architecture

As shown in Figure 3, a 1D DCT architecture consist of one pre-addition block, two bit extraction block, eight ROMs with different pre-computed results and eight add accumulation blocks.

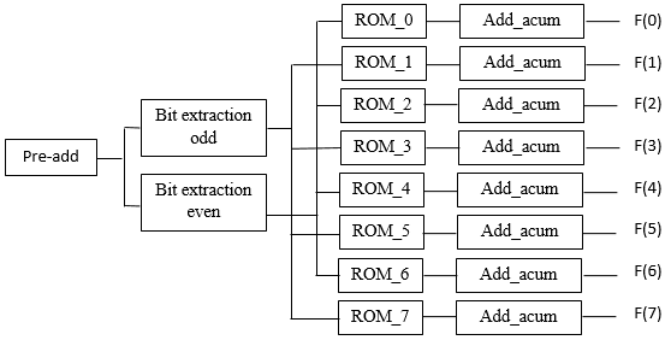

Figure 3. 1D DCT block diagram

2.2.4 2D DCT Architecture

To get the 2D DCT transform result, input pixels are fed into a 1D DCT block column wisely. And then the output of the first 1D DCT will be stored in a transpose RAM before all eight columns are calculated. Then the intermediate result in the RAM will be read out row wisely and fed into the second 1D DCT block, which will output the final transformation result.

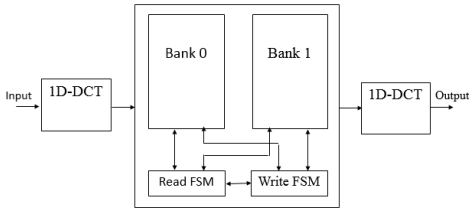

Figure 4. 2D DCT block diagram with transpose RAM

Because of the latency of reading out the data from RAM to the second DCT, the data of one set of 8x8 input pixel has to stay extra time in the RAM after it is all written. So we use interleaving technique to prevent stalls in the pipeline.

3. ENTROPY ENCODING

After DCT, the data need to be transmitted. In order to reduce bandwidth requirement for wireless transmission, the data needs to be compressed as much as possible. Along with data compression, data integrity also needs to be preserved, and hence we have done Entropy Encoding which is a form of lossless data compression. It comprises of two parts: Zigzag Ordering and Run Length Encoding (RLE).

3.1 Zigzag Ordering

Quantization is done on the DCT matrix which divides each coefficient of DCT matrix by a constant which is a predefined in Quantization Matrix. Quantization reduces high frequency components to zero and hence lower triangular matrix of Quantized matrix contains mostly zeroes. If the matrix is then read in Zigzag order rather than row or column wise then all the zeroes can be grouped together giving an opportunity to compress data using Run Length Encoding. In the design implementation, coefficients are read from interposed RAM and then stored into a buffer in Zigzag order which is used by RLE.

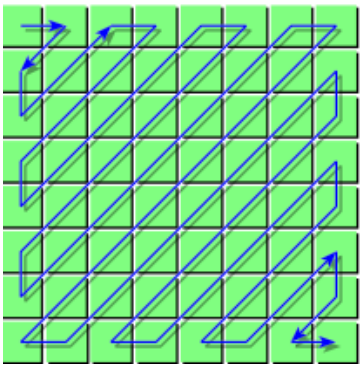

Figure 5. Zigzag traversing of 8X8 DCT Matrix

3.2 Run Length Encoding (RLE)

Run Length Encoding is a lossless technique to compress the image data and is the final stage of JPEG image compression. The input vector from zigzag is a 64 number vector and contains a lot of zeros. In order to use as few bits as possible to represent those numbers, the number of zero before the next non-zero coefficient is counted, namely run length. The values in the vector are represented by pairs of (Runlength, Value), and DC and AC coefficients are treated differently.

3.2.1 DC coefficient Encoding

Because there is only one DC coefficient per block and it varies greatly across a block but may not vary much against its neighbors, the difference of DC coefficients which needs much fewer bits to represent are stored instead of actual DC coefficients. The DC coefficient representation of Run Length Encoding is a pair of (Size, Amplitude).

3.2.2 AC coefficient Encoding

In terms of AC coefficient, the zero runs are counted before each non-zero AC coefficient, so the large number of bits to represent zeros are eliminated. If the run length is larger than 15, then a (15, 0) marker will be made, followed by the remaining zeros. When there is no more non-zero coefficients in the vector, an End of Block Marker (0, 0) will be attached the end of the output stream.

3.2.3 Negative value treatment

The negative value will be performed one's complement. The size of the amplitude of AC coefficients and difference of DC coefficients are found from look up table. After doing that, use the same way of treating AC coefficient to treat the amplitude. The number of bits to represent the sizes of AC coefficients and the difference of DC coefficients is four.

So is the number of bits to represent the run length. The Run Length Encoding representation is pairs of (Runlength, Value), where the Value is simply a (Size, Amplitude) pair.

3.2.4 Hardware Implementation

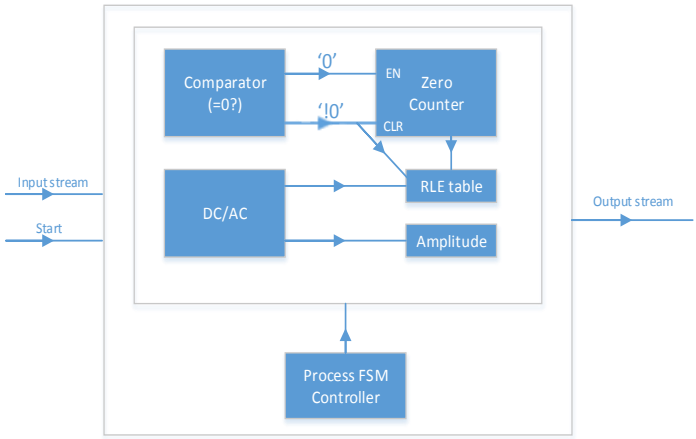

Figure 7. Block diagram of Run Length Encoding hardware implementation

The comparator is used to determine if the number in the input vector is zero or not, and the output is fed into the zero counter to count the number of zeros. The DC/AC block will differentiate DC coefficient and AC coefficients and determine the sizes. All the results will be fed into the RLE table.

The output stream is formed by grouping each entry of RLE table and the amplitude coming out of Amplitude block.

4. ANALYSIS

The complete system was implemented in Verilog and synthesized using a 130nm technology from IBM. The ROM look-up tables and transposed RAM were modeled in Verilog and characterized by CACTI 4.1.

The ROM look-up table stores pre-computed values used by DCT transform. It has a total size of 4 Kbits and requires 16 read ports. The best simulation results (minimum power and area) were obtained when the ROM was divided into four independent ROMs with 4 read ports each. The ROM look-up table dominates the power consumption of the entire design. The transposed RAM has a total size of 2 Kbits with independent read/write ports. Table 1 below summarizes CACTI simulation results.

Table 1 – ROM look-up tables and Transposed RAM simulation results

	(4) 1024 bits ROM	(1) 2048 bits RAM
Banks	1	2
Read Ports	4	1
Write Ports	0	1
Area	0.95168 mm^2	0.118072 mm^2
Power	308.5 mW	27.29 mW
Access Time	0.741125 ns	0.730582 ns

The DCT transform block, zigzag, and run-length encoding were synthesized in Verilog with a 5 ns clock period. Processing the initial 8x8 image block takes 436 cycles, and thereafter 144 cycles per block. Multiplying the cycle time by the clock period results in latency of 2.18 us and a throughput of 0.72 us. Table 2 below summarized synthesized results.

Table 2 – DCT, zigzag, run-length encoding synthesis results

	DCT/Zigzag/RLE	Comments
Area	1.953834 mm2	
Power	27.289 mW	Dynamic/leakage
Cycle time	5 ns	Fmax = 200 MHz
Latency	2.18 us	
Throughput	0.72 us	

The total area of the design is around 3 mm^2, which is dominated by the synthesized Verilog code. The total power consumed is 369 mW and as mentioned previously is dominated by the ROM look-up table.

The energy efficiency of the image co-processor can be calculated by multiplying the time it takes to process an 8x8 image block by the total power consumed by the system.

$$E_{latency} = 2.18\ \mu s * 368.967\ mW = 806\ nJ/block$$

$$E_{throughput} = 0.72\ \mu s * 368.967\ mW = 284\ nJ/block$$

In order to compare the energy efficiency of the image co-processor, we implemented equivalent C code and ran batch jobs in an Intel Xeon X5660. The average energy throughput per image block was found to be 1 mJ/block. The previous is orders of magnitude greater than our results.

5. WORK DISTRIBUTION

The work for this project was evenly distributed among team members. Shihan and Adrian worked on implementing the 1-dimensional DCT, ROM look-up table, and transposed RAM. Akanksha and Zhongyuan worked on implementing zigzag and run-length encoding respectively.

6. CONCLUSION

In this paper we designed and implemented an image co-processor to accelerate two-dimensional DCT transform, quantization, zigzag and run-length encoding. This image co-processor can be used with general purpose CPU or DSP chips. The design uses a DCT architecture based on distributed arithmetic and embedded uniform quantization. As a result, expensive computation steps requiring multiplication and division are completely eliminated. The total average power consumed by the realized system is 369 mW with 3mm^2 area overhead. Additionally, the system can process 8x8 input pixel blocks with an energy efficiency of 284 nJ/block.

7. REFERENCES

[1] L. Fanucci, R. Saletti, and F. Vavala, "A low-complexity 2-d discrete cosine transform processor for multimedia applications". *Electronic, Circuits and Systems*, 1999.

[2] P. Samadi, H. Wu, and M. Ahmadi, "The 2-d quantized DCT with distributed arithmetic". *Canadian Conference on Digital Object Identifier*, 2006.

[3] K. Nguyen-Phi, A. Docef, and F. Kossentini, "Quantized discrete cosine transform: A combination of DCT and scalar quantization". *IEEE International Conference*, 1999.