Image Compression System on an FPGA

Group 1
Megan Fuller, Ezzeldin Hamed
6.375

Contents

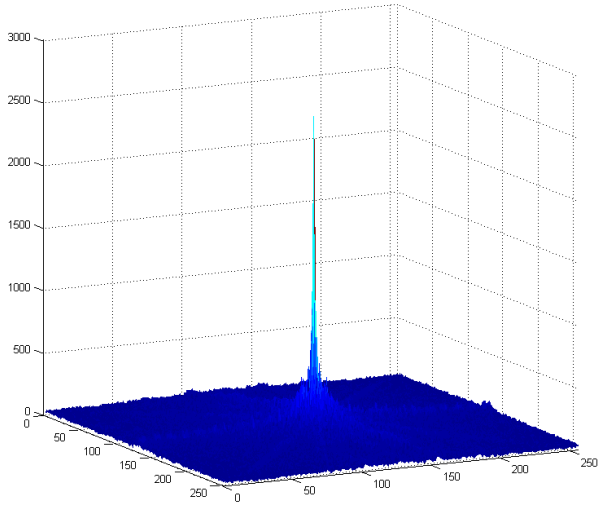
1 Objective

This project consists of a basic image compression system implemented on the FPGA. An image compression system requires the computation of a transform and then the selection of certain transform coefficients to store or transmit. This system takes in a 256x256 black and white image from the host PC, computes the transform, and returns all coefficients above a certain threshold to the host PC.

2 Background

(a) A Typical Image

(b) Image reconstructed from 25% of the DFT coefficients



(c) Square root of the magnitude of the DFT of the image

Figure 1: Illustration of the energy compaction property of the DFT

Image compression systems exploit what is known as the energy compaction property of certain transforms. This property is illustrated in Figure 1 for the Discrete Fourier Transform. As can be seen in the figure, there are very few high energy coefficients. The human visual system is insensitive to the loss of the low energy coefficients, and so these can be thrown away and

image intelligability maintained, as demonstrated by Figure 1b. (Note that Figure 1c is showing the square root of the magnitude of the DFT so low energy coefficients that are nevertheless nonzero are visible.)

There are two transforms that we will consider for compressing the image: the Discrete Fourier Transform (DFT) and the Discrete Cosine Transform (DCT). Some mathematical background on both transforms will now be presented, followed by a discussion of the advantages of implementing these transforms on an FPGA.

2.1 The DFT

An image can be considered a two dimensional spatial signal, and so it is reasonable to compute the 2D-DFT of an image. The 2D-DFT is a straightforward extension of the 1D-DFT and is defined by:

$$X(k_1, k_2) = \sum_{n_2=0}^{N_2} \sum_{n_1=0}^{N_1} x(n_1, n_2) W_{N_1}^{k_1 n_1} W_{N_2}^{k_2 n_2} \tag{1}$$

where N_1 is the width of the image, N_2 is the heigth of the image, and W_N is $e^{-j\frac{2\pi}{N}}$.

$X(k_1, k_2)$ can be comupted separably–that is, by computing the 1D-DFTs of the rows followed by the 1D-DFTs of the resulting columns. This can be shown by rearranging Equation 1 to be

$$X(k_1, k_2) = \sum_{n_2=0}^{N_2} W_{N_2}^{k_2 n_2} \sum_{n_1=0}^{N_1} x(n_1, n_2) W_{N_1}^{k_1 n_1} \tag{2}$$

The inner summation is simply the 1D-DFTs of the rows of $x(n_1, n_2)$. Call this $f(k_1, n_2)$. The outer summation is the 1D-DFTs of the columns of $f(k_1, n_2)$, and so the separable computation of the 2D-DFT is valid.

If all the computations were done by the "brute force" method (that is, not taking advantage of the FFT), computing the DFT separably would reduce the number of multiplications from $N_1^2 N_2^2$ to $N_1 N_2 (N_1 + N_2)$. Using a radix-2 FFT to implement the DFT will reduce this even further to $O(N_1 N_2 log_2(N_1 N_2))$ multiplications.

2.2 The DCT

Another commonly used transform in image compression is the 2D-DCT, defined by

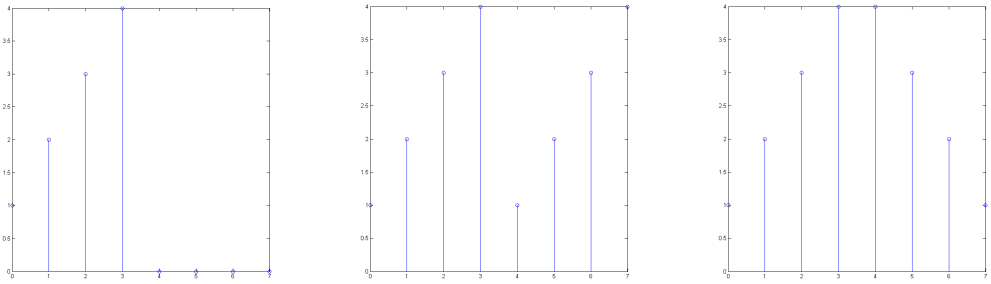
$$C_x(k_1, k_2) = \sum_{n_2=0}^{N_2} \sum_{n_1=0}^{N_1} 4x(n_1, n_2) cos\left(\frac{\pi}{2N_1} k_1 (2n_1 + 1)\right) cos\left(\frac{\pi}{2N_2} k_2 (2n_2 + 1)\right) \tag{3}$$

As with the 2D-DFT, the 2D-DCT can be computed seperably, as a series of 1D-DCTs. Furthermore, it can be shown[1] that if we define the 2N-point sequence $y(n) = x(n) + x(2N - 1 - n)$, the DCT of x(n) can be written in terms of the DFT of y(n):

$$C_x(k) = \begin{cases} W_{2N}^{\frac{k}{2}} Y(k) & 0 \leq k \leq N-1 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Equation 4 gives some insight into the relationship between the DCT and the DFT. The DFT is a single period of the Fourier Series coefficients of the sequence x(n) made periodic with

[1] Lim, Jae S. *Two-Dimensional Siganl and Image Processing.* Englewood Cliffs: Prentice Hall, 1990.



(a) x(n) (b) x(n) made periodic in N (c) y(n)

Figure 2: Illustration of the relationship between the DFT and the DCT

period N, as shown in Figures 2a and 2b. The DCT, in contrast, is a twiddle factor multiplied by the first N Fourier Series coefficients of the sequence y(n), which is only x(n) reflected about its ending point and made periodic in 2N, as shown in Figure 2c. This eliminates the discontinuity of the sequence in Figure 2b and generally allows for better energy compaction.

Computing the DCT according to Equation 4 is wasteful because it requires the computation of the 2N-point DFT of y(n) and then throws half the values away. Fortunately, this can be overcome by defining v(n) = y(2n) and computing the N-point DFT V(k). Then it can be shown that

$$Y(k) = V(k) + W_{2N}^{-k} V(-k). \tag{5}$$

Because of the extra multiplies, the DCT is slightly more computationaly intensive than the DFT. However, it also gives, in general, better energy compaction and therefore higher compression. This trade-off will be explored later in the project.

2.3 Advantages of an FPGA

It is useful to implement this system in hardware because taking a 2D-DFT (or 2D-DCT) is a computationally intensive operation. It can be done by dedicated hardware much more quickly than it can be done by software running on a general purpose computer. Also, because of the structure of the 2D-DFT, many of the computations could theoretically be done in parallel, further increasing the speed with which the image could be compressed. (In practice, we found we were limited in memory accesses and were not able to exploit the potential of parallelism the equations seem to promise.)

3 High-Level Design

The high level idea in our implementation is that we recieve an image from the host PC, store the image in the BRAM inside the FPGA, then calculate the 2D-DFT or 2D-DCT of this image. After that the magnitude of the DFT or DCT coefficients is compared against a threshold. This threshold is a dynamic parameter provided by the user at runtime. If the coefficient is greater than the threshold, the coefficient and its location will be saved. Otherwise, it will be discarded. These saved coefficients and their locations comprise the compressed file, which will be sent back to the PC for verification. By changing the threshold, the user can change the level of compression. This architecture is illustrated in Figure 3.

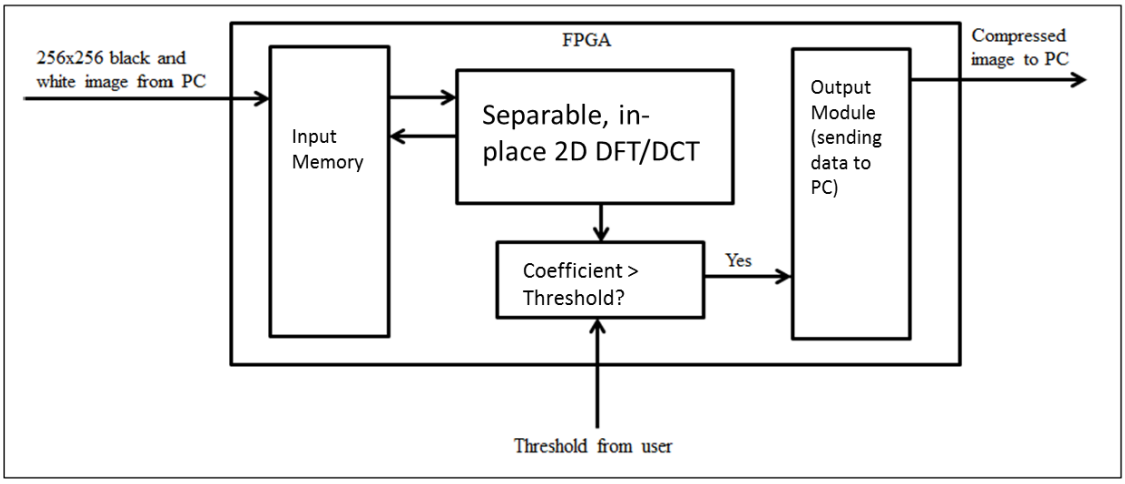


Figure 3: High-level Microarchitecture

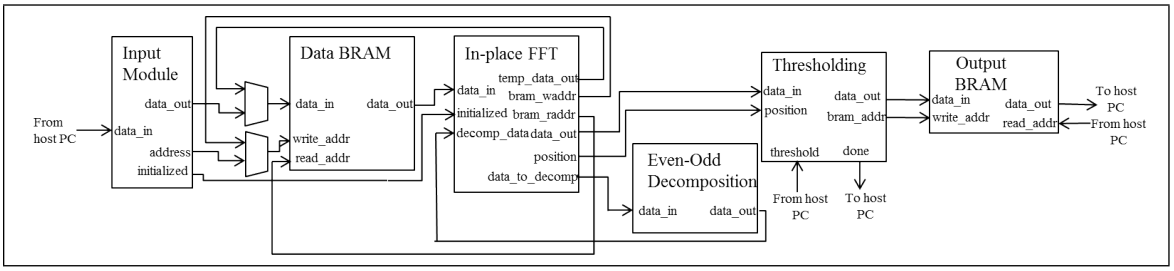


Figure 4: Detailed Microarchitecture

4 Test Plan

Matlab has built-in functions to read in images and compute the 2D-DFT or the 2D-DCT. It is also very easy to perform thresholding. Our baseline is a floating-point version of the image compression sytem. Obviously, this is not bitwise accurate to what the system on the FPGA returns, but it serve as a useful baseline to compare against.

In our testing environment, we reconstruct the image (decompression is simply an inverse 2D-DFT or 2D-DCT, which, again, is simple in Matlab) and compute the mean square error between the decompressed image and the original image.

5 Description of Microarchitecture

Our design consists of six modules: the input module, the data BRAM module, the FFT module, the even-odd decompostion module, the thresholding module, and the output BRAM module. The architecture is illustrated in Fig 4. Each of these will now be described in detail.

The host PC sends the pixels from left to right and then from top to bottom to the FPGA over SceMi. The input module receives these numbers and converts them to two's complement numbers by subtracting 128. Image pixels are unsigned numbers–converting them in this way serves the dual purpose of formatting them for the two's complement arithmetic units and

removing the large DC component, which will be added back in when the image is decompressed. The input module then sends the samples to the data BRAM according to the ordering needed by the selected transform. Once the entire image has been received, formatted, and stored, the input module will send an "initialized" signal to the FFT module, indicating it is time to start the compression.

5.1 Data BRAM Module

The data BRAM module consists of two BRAMs. It has two read and two write ports, one read and one write port for each BRAM. Note that, in Fig 4, each of these two ports is only shown once to avoid making the figure too complicated. The reason for dividing the memory into two blocks is to reduce the memory access time when we are performing the even/odd decomposition.

5.2 FFT Module

There are many issues to consider here. As stated in the high-level architecture, we implemented a seperable 2D-DFT/2D-DCT. In the DFT case, we first calculate the FFT of each row individually, and then calculate the FFT of the resulting columns. In the case of DCT, the same thing is done but an extra twiddle factor multiplication is required after taking each FFT. In general, this would require the computation of 256 x 2=512 FFTs. However, because the input image is real, this number can be cut in half by the use of two tricks, which will now be discussed, followed by an explanation of how the FFT is actually implemented.

5.2.1 Reducing the Number of Rows

This is almost the same in both the DFT and DCT cases. The FFT of a real signal is conjugate symmetric. That is, X(k) = X*(-k). (In this context, k should be taken to mean k mod N, where N is the length of the signal.) This means the real part of X(k) is even, and the imaginary part of X(k) is odd. Therefore, if there are two real signals, say a(n) and b(n), for which the FFT needs to be computed, it can be done by combining them into a third signal, c(n) = a(n)+jb(n). The FFT of c(n), C(k), is then computed and seperated into its conjugate symmetric and conjugate anti-symmetic parts, which are the FFTs of a(n) and b(n), respectively. This can be done by computing

$$A(k) = \frac{1}{2}(C(k) + C^*(-k)) \tag{6}$$

$$B(k) = \frac{1}{2}(C(k) - C^*(-k)) \tag{7}$$

And, instead of doing two FFTs (O(Nlog(N)) complex multiplies and adds), we do one FFT and 2N complex additions (or subtractions). We use this to compute the 1D-DFTs of the rows two at a time, and so we only have to compute 128 FFTs for the rows. And in the case of DCT we need to multiply the result by a twiddle factor to get the coefficients of the 1D-DCT of the rows.

5.2.2 Reducing the Number of Columns

In the DCT case the output of the 1D-DCTs of the rows is real, so the same trick that was done with the rows can be repeated here with the columns. This is not the case with the DFT. Because the 1D-DFTs of the rows are generally complex, the 1D-DFTs of the columns cannot be

combined in the same way the rows can. However, looking at Equation 1, there is still symmetry which can be exploited. Specifically, if the input $x(n_1, n_2)$ is real,

$$X^*(k_1, k_2) = \sum_{n_2=0}^{N_2} \sum_{n_1=0}^{N_1} x(n_1, n_2) W_{N_1}^{-k_1 n_1} W_{N_2}^{-k_2 n_2} \tag{8}$$

and

$$X^*(-k_1, -k_2) = \sum_{n_2=0}^{N_2} \sum_{n_1=0}^{N_1} x(n_1, n_2) W_{N_1}^{k_1 n_1} W_{N_2}^{k_2 n_2} \tag{9}$$

The right-hand side is simply $X(k_1, k_2)$, so only the "positive" values of k_2 need to be computed. (Again, positive and negative k are taken mod N.)

5.2.3 Reducing the Bit Width

In our implementation we tried to reduce the bit width as far as possible. There are two main issues that we ran into once we tried reducing the bit width. The first issue is the quantization noise, this was slightly improved when we used rounding instead of truncation. The idea here is that when we do the multiplication and addition the bitwidth of the accumulator register is slightly larger than the bitwidth of the data stored in the memory. When we want to write the data back to the memory we just throw away those extra bits below our least significant bit. The way we implemented rounding here is by initializing the accumulator register by 1 at the bit below our least significant bit. This way we have the same effect as rounding without any extra hardware.

The second issue that we faced, trying to reduce the bit width in fixed-point arithmatic, is overflows. We need to avoid overflow through out our implementation. To do this, if we are performing N additions, we need to have $log_2(N)$ extra bits to absorb possible overflows. In order to reduce this extra bit width overhead, one idea was to shift the data to the right (divide by 2) whenever we expect an overflow. This means that with every addition we shift the data to the right by one bit, whether the overflow occured or not. This way we have a higher effective bit width at the early stages of the computation and we do gain some accuracy. However, as we approach the very last stage, the location of the fraction point becomes the same as the fixed point case.

A much better idea is to shift only when we really need to shift, which is when an overflow occures. To do that without running into the complexity of a floating point implementation, we implemented what we call here dynamic scaling.

5.2.4 Dynamic Scaling

The main idea of dynamic scaling is to keep track of overflows throughout our computation and shift the data only when an overflow occures. In order to avoid saving an exponent value for each sample, we fix the location of the fraction point for each 1D-FFT. This means that if an overflow is detected, all the samples of the current 1D-FFT will be shifted. This way we avoid the problem of doing operations on samples with different exponent values. Implementing this dynamic scaling does not affect the critical path of the design. A few multiplexers are added to the write back stage and some extra registers are introduced to save the exponents of the 1D-FFT frames. As will be shown in the results section, the overhead of dynamic scaling is very small compared to the improvement in the accuracy of the computation.

5.2.5 Implementation

We implemented a single radix 4 butterfly, doing an in-place FFT with dynamic scaling. It takes on the order of $(\frac{N}{2} + \frac{N}{2})N \log_4 N$ clock cycles and requires one complex multiplier. The other option was to implement a single pipelined (streaming) FFT, but as we were interested in characterizing the performance of dynamic scaling, and it was not possible to use dynamic scaling inside the pipelined FFT, so we decided to implement the in-place FFT.

Twiddle factors are precomputed and stored in a ROM. The FFT block waits until the input module set the initialized signal, and then does the computation, reading from and writing to the data BRAM and sending data to the even-odd decomposition module as needed. The inputs are bit-reversed in both the column and row orientations, so the outputs will be in order. Once the computation is completed, it sends the coefficients with their locations to the thresholding module.

5.3 Even-Odd Decomposition Module

The even-odd decomposition module takes as its input two complex samples, which are added and subtracted according to equations 6 and 7. The FFT block provides the inputs such that the outputs corrospond to the 1D-DFTs of the rows, as explained previously.

5.4 Thresholding Module

The thresholding module is used for two purposes. First, it communicates with the host PC by accepting a threshold, sent over SceMi, and by writing a SceMi-readable register indicating that the compression has finished and the compressed file may be read.

Second, it takes as its input the coefficients produced by the FFT module and their locations. It will compare the coefficients against the threshold. If a sample passes the test, both the sample and its location will be sent to the output module. If it does not, it will be discarded. Thresholding is done on the magnitude squared of the complex sample in the DFT case, and is done directly on the real output sample in the case of DCT.

5.5 Output Module

The output module accepts as its input a complex sample and a position from the thresholding block. It stores these samples in a FIFO until they can be read by the host PC.

6 Test Results

We ran experiments in simulation to explore the following design space: Number of bits used to quantify the coefficients, whether the computations were done with rounding, whether the computations were done with dynamic scaling, and whether the DCT or the DFT was used. Based on the results of these simulations (discussed below) several of the designs were synthesized, and the results of synthesis will be discussed in the next section.

The quality metric used was the mean squared error between the image reconstructed from the compressed file and the original image. This metric was chosen because it is simple to compute and understand. However, as will be discussed in more detail later, it is not always a good measure of picture quality. This is because the human visual system responds differently to different kinds of artifacts. Picture quality is an inherently subjective thing, and therefore extremely difficult to quantify. For our experiments, we found the types of degredations to be

Figure 5: Original Image

generally similar for 12 and 16 bit coefficients, but quite different for 8 bit coefficients. Therefore, the remainder of this section will treat these two cases separately.

The original image used in all experiments is shown in Figure 5.

6.1 12 and 16 Bits

There are eight possible cases to consider here (each with and without rounding):

1. 12 bits, no dynamic scaling, DFT
2. 16 bits, no dynamic scaling, DFT
3. 12 bits, dynamic scaling, DFT
4. 16 bits, dynamic scaling, DFT
5. 12 bits, no dynamic scaling, DCT
6. 16 bits, no dynamic scaling, DCT
7. 12 bits, dynamic scaling, DCT
8. 16 bits, dynamic scaling, DCT

As will be shown in more detail, in the case of the DFT, 16 bits without dynamic scaling is as good as floating point, and so the case with dynamic scaling need not be considered. Furthermore, in the case of the DCT without dynamic scaling, 12 bit quantization is too coarse to produce an acceptable image at any level of compression, so this case also will not be considered. To summarize, the six cases that will be considered are

1. 12 bits, no dynamic scaling, DFT
2. 16 bits, no dynamic scaling, DFT
3. 12 bits, dynamic scaling, DFT
4. 16 bits, no dynamic scaling, DCT
5. 12 bits, dynamic scaling, DCT
6. 16 bits, dynamic scaling, DCT

6.1.1 Rounding

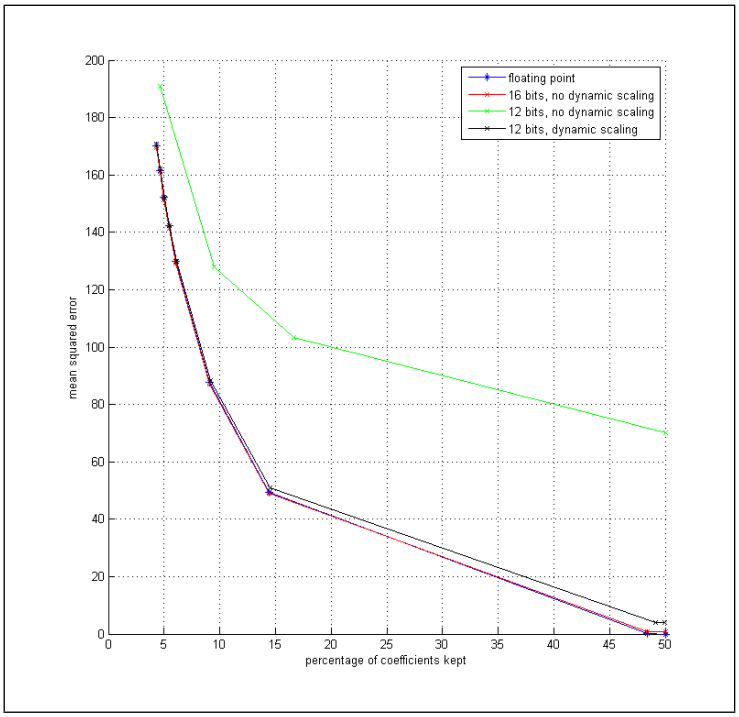
Each of the cases listed above was tested both with and without rounding in the computation of the transform. The result of adding rounding was, in general, a constant decrease in mean squared error at any compression ratio. The results are summarized in Table 1. In hardware,

Table 1: The Effect of Rounding

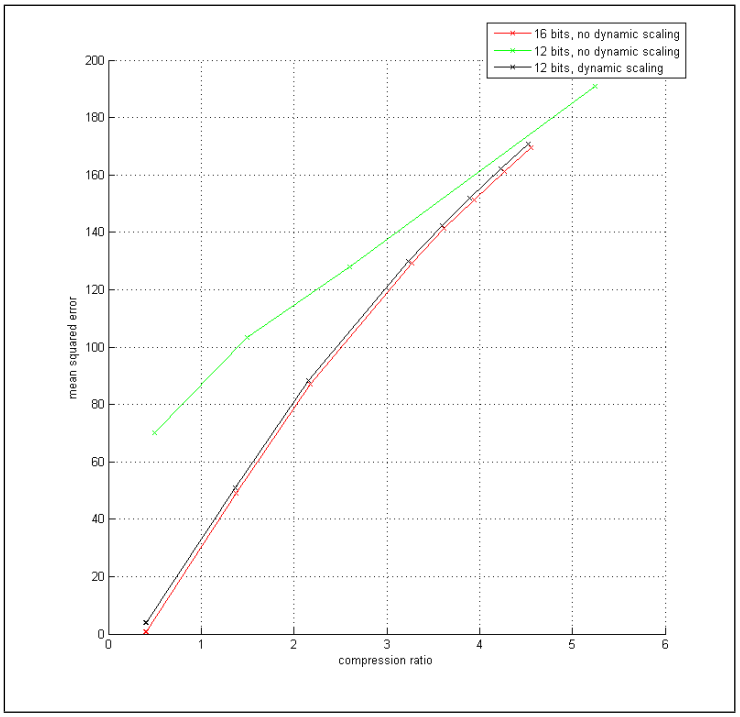
Design	MSE Decrease With Rounding
12 bits, no dynamic scaling, DFT	20
16 bits, no dynamic scaling, DFT	0
12 bits, dynamic scaling, DFT	2
16 bits, no dynamic scaling, DCT	0
12 bits, dynamic scaling, DCT	2
16 bits, dynamic scaling, DCT	0

rounding consists of a register initialization, and therefore is free and should always be done–it never hurts and often helps. However, it is most important when there is no dynamic scaling and the coefficients are more coarsely quanitzed. All remaining results consider only the case when rounding is done.

6.1.2 Dynamic Scaling



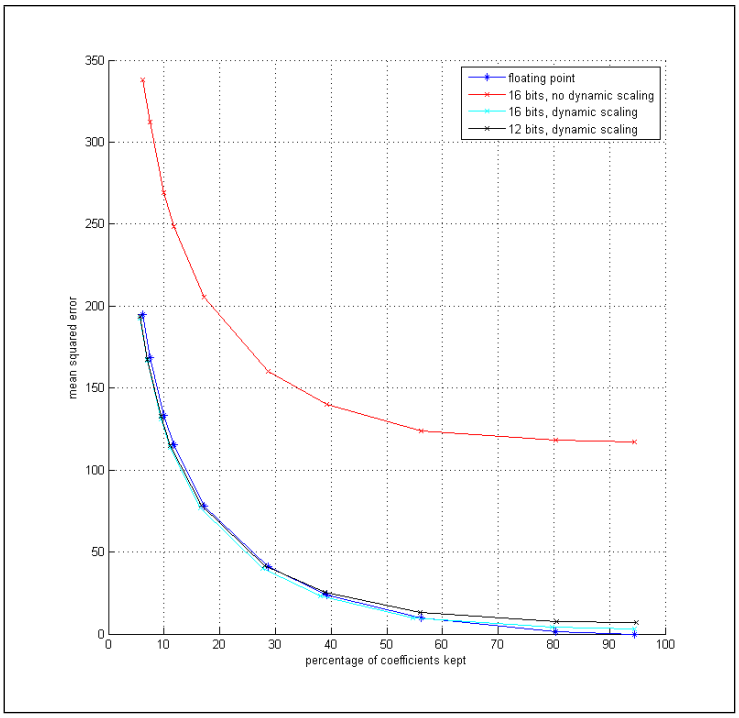
(a) MSE vs. Percentage of Coefficients Kept



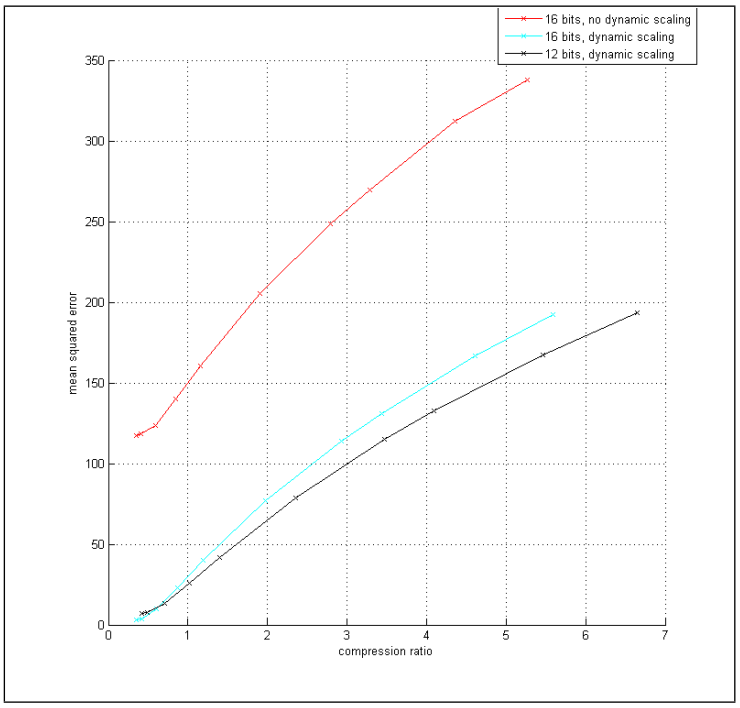
(b) MSE vs. Compression Ratio

Figure 6: Data for several methods of computing the DFT

Figure 6a compares the mean squared error to the percentage of coefficients retained for the three interesting DFT cases (16 bits without dynamic scaling, 12 bits with and without dynamic scaling). Also, the results for floating point precision (obtained from Matlab) are included for reference. From this, it can be seen that 16 bit coefficients are as good as floating point coefficients, and that dynamic scaling dramatically improves the 12 bit compression results,



(a) MSE vs. Percentage of Coefficients Kept



(b) MSE vs. Compression Ratio

Figure 7: Data for several methods of computing the DCT

bringing them almost to the level of the floating point results. Note that, because of the symmetry of the 2D-DFT, saving 50% of the coefficients is sufficient for perfect reconstruction.

Figure 6b shows the same data as Figure 6a, but compares MSE to the compression ratio, which takes into account the reduced number of bits required when storing only 12 bit coefficients, and also takes into account the fact that dynamic scaling requires a few extra bits to save the scaling factor. Note that floating point numbers no longer make sense in this comparison and so have been left out. Note also that, for a given compression ratio, the minimum mean squared error is produced when a larger number of bits are used for the coefficients. Finally, note that, again, dynamic scaling dramatically improves the 12 bit case, and the overhead is almost insignificant.

Figure 7a compares the mean squared error to the percentage of coefficients retained for the three interesting DCT cases (16 bits without dynamic scaling, 12 and 16 bits with dynamic scaling). Again, the floating point results from Matlab are included for reference. When dynamic scaling is done, both the 12 and 16 bit cases achieve results very close to floating point. Without dynamic scaling, 16 bits does not do nearly as well, and 12 bits does so poorly as to not even merit inclusion in the analysis.

Note, however, that the artifacts in the reconstructed image when dynamic scaling is not done are different and less visually obnoxious than the artifacts when dynamic scaling is done. This is illustrated in Figure 8. This means the case without dynamic scaling can have a higher mean squared error and still appear to be about the same quality to the human viewer. Taking this into account, dynamic scaling is still useful, but not as useful as it appears to be based only on the raw mean squared error numbers.

(a) Image reconstructed from 5.7% of the DCT coefficients, computed with dynamic scaling. MSE = 193

(b) Image reconstructed from 6.1% of the DCT coefficients, computed without dynamic scaling. MSE = 338

Figure 8: Comparison of DFT Artifacts

Figure 7b shows the dame data as Figure 7a, but compares MSE to the compression ratio. Unlike the case of the DFT, using a lower bitwidth can actually give a lower MSE for a given compression ratio. Again, the overhead of the scaling bits is negligable compared to the performance gain.

6.1.3 DCT vs. DFT

For comparison purposes, a few of the best cases of the DCT and DFT compression methods are shown on the same plot in Figure 9. If performance is the only concern, the 12 bit DCT with dynamic scaling is superior to all other options. However, implementation constraints may require the use of a different system. (A comparison of the physical performance of each system on the FPGA is given in the next section.)

6.2 8 Bits

The artifacts in the image when the error is due to coarse quantization are very different from those produced withn the error is due to too few coefficients being retained. For an illustration of this, see Figure 10. Because of this, the 8 bit case should not be compared to the 12 and 16 bit cases on the basis of mean squared error.

The 8 bit case produces intelligable images only when dynamic scaling is used in the computation of the coefficients, and therefore all cases considered use dynamic scaling.

6.2.1 Comparison of Systems

Figure 11 shows mean squared error as a function of compression ratio for the four interesting 8 bit cases (DFT and DCT, with and without rounding). Unfortunately, at these very large mean squared error values, the measurement is almost useless. To illustrate this, four reconstructed pictures are shown in Figure 12. These images all have about the same compression ratio, but very different mean squared errors. It is difficult to argue that the image quality follows mean squared error in this case. (This may also explain why, in the case of the DFT, rounding

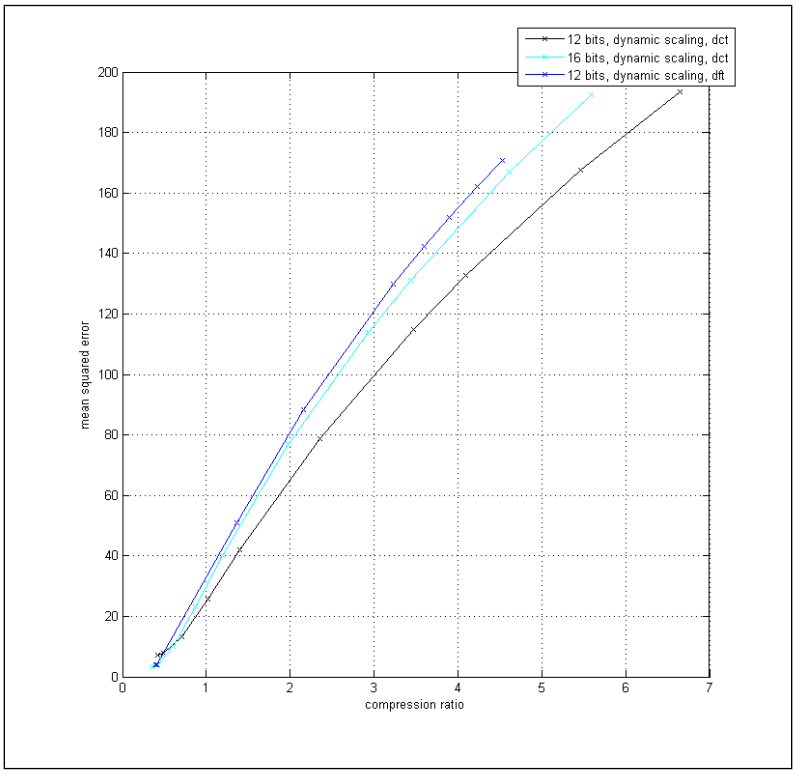


Figure 9: Performance of the DCT and DFT

(a) Image reconstructed from 50% of the DFT coefficients, computed with 8 bits, using dynamic scaling. MSE = 452

(b) Image reconstructed from 6% of the DFT coefficients, computed with 16 bits. MSE = 129

Figure 10: Comparison of quantization artifacts and thresholding artifacts

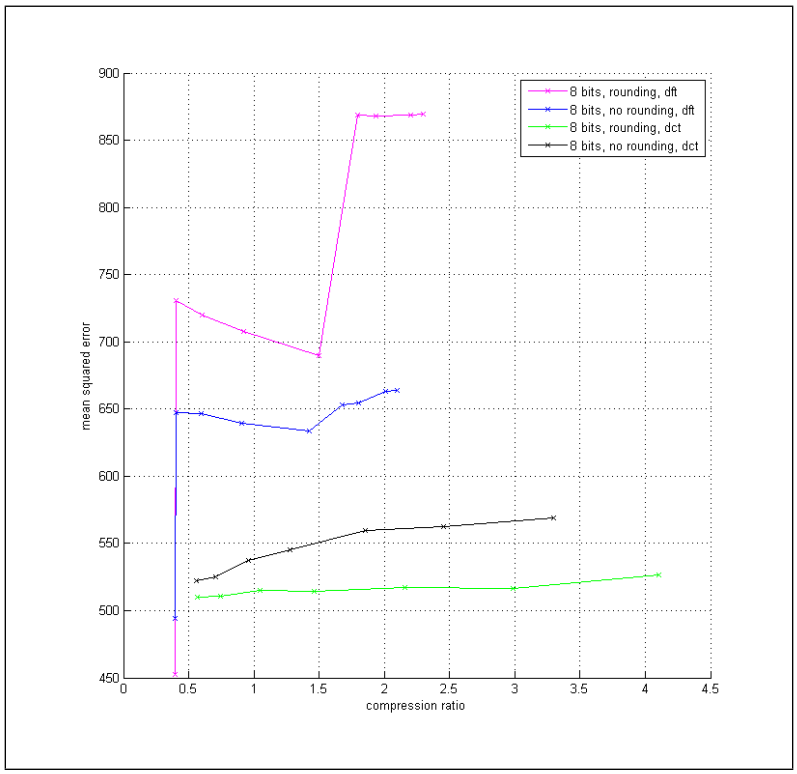


Figure 11: Performance of 8 Bit Systems

(a) 8 bit DFT coefficients, computed with rounding. Compression ratio = 2.3, MSE = 869

(b) 8 bit DFT coefficients, computed without rounding. Compression ratio = 2.1, MSE = 664

(c) 8 bit DCT coefficients, computed with rounding. Compression ratio = 2.2, MSE = 517

(d) 8 bit DCT coefficients, computed without rounding. Compression ratio = 2.4, MSE = 563

Figure 12: Comparison of pictures reconstructed from 8 bit systems

Table 2: Synthesis Results

design	critical path	slice registers	slice LUTs	bram	DSP48Es
DFT, 16 bits, no dynamic scaling	11.458ns	16%	23%	29%	7
DFT, 16 bits, dynamic scaling	11.763ns	17%	24%	29%	7
DFT, 12 bits, no dynamic scaling	11.273ns	15%	22%	24%	7
DFT, 12 bits, dynamic scaling	11.464ns	16%	23%	24%	7
DFT, 8 bits, dynamic scaling	11.287ns	15%	22%	18%	6
DCT, 16 bits, dynamic scaling	11.458ns	19%	26%	29%	10
DCT, 12 bits, dynamic scaling	11.273ns	18%	25%	24%	10
DCT, 8 bits, dynamic scaling	11.066ns	17%	23%	18%	8

Table 3: Latency

Component	Latency (clock cycles)
Initialization	870,000
DCT	263,900
DFT	262,200

increases mean squared error–it is not a limitation of rounding, but of mean squared error as a quality measurement.)

What is clear is that the 8 bit results are never as good as the 12 or 16 bit results. The 8 bit system should not be used unless circuit size is a critical consideration.

7 Physical Considerations

The previous section discussed only image quality. We will now look at the physical costs of certain features.

7.1 Synthesis Results

The design was synthesized for five DFT configurations: 16 bits and 12 bits, both with and without dynamic scaling, and 8 bits with dynamic scaling; and three DCT configurations: 16, 12, and 8 bits with dynamic scaling. Rounding is simply a register initialization and does not take any additional hardware resources. The other cases mentioned did not perform well enough to be included in synthesis. The results are show in Table 2. In general, the DCT requires more FPGA resources than the DFT, but it also produces better compression results.

Note that the critical path is about the same for all designs. This is as expected. The timing constraints we gave the synthesizer were limited by what the SceMi interface could handle. We expect that the rest of the design could meet tighter constraints if they were given.

7.2 Latency

We measured the latency of both the DFT and DCT in simulation. None of the other design parameters would change this. We looked at the input initialization latency separately from the compression latency because the initialization latency is dependent on the host system. For our tests, this was from a host PC via SceMi, but another application would give different results. For this same reason, we measured the latency of the transform as though the host application

could accept each output sample immediately. In our test setup, this was not actually the case, but it was a limitation of the SceMi interface, not the image compression system. The results are shown in Table 3.

With some slight modifications, the image compression system could have initialization and compression pipelined, so that we could read in a new image while computing the transform of the current image. If the host application could keep up with the transform (not the case in our test setup), so that the transform were the limiting factor in latency, and if the application were running on the 50MHz clock of the FPGA, the DCT would be able to transform about 189 images/second, and the DFT would be able to do about 191 images/second.

8 Future Work

The current image compression system works only for fairly small images (256x256 pixels). This constraint would be lifted if we were to store the image in DRAM, rather than in BRAM as we are currently doing.

Other possible extensions include support for color images, support for rectangular images of arbitrary dimensions (currently, only square images with lengths that are a power of 4 are handled), and combining the DFT and DCT systems into a single core that could compute either transform, as selected by the user at runtime.