Major Shrinkage!
Video Compression

18-551
Group 11 Fall 2009

Samir Singh (samirsin@andrew.cmu.edu)
Ryan Conaghan (rconagha@andrew.cmu.edu)
Nikola Ljuboja (nljuboja@andrew.cmu.edu)

Table of Contents

1. The Problem

In today's technologically advancing world, videos can be viewed through many different mediums. For most devices that display videos, such as DVDs, multimedia devices, cell phones, and cable boxes, bandwidth and/or memory is limited. As a result, it became necessary to develop a methodology to render high quality video with limited processing power and/or storage space.

The methodology that was developed was video compression. Video compression reduces the quantity of data used to represent digital video images. The majority of video compression schemes are lossy, meaning that the original cannot be reconstructed from the compressed file. Initially, it may seem that the lossy video compression will lead to extremely low quality videos; but this is not the case because we could lose a relatively large amount of data before a difference is noticed. Depending on the medium in which the video is being processed, one algorithm may be more appropriate than another.

2. The Solution

Our project encodes and compresses the raw video data to an MPEG-2 format. We used the DSK to make an MPEG-2 encoder. We remained compliant to the NTSC MPEG-2 standard so that our encoded file can be played on any standard MPEG-2-capable media player (VLC Media Player, Windows Media Player, etc).

The MPEG-2 format is an extremely common video format, and is used on TV broadcast, cable, and satellite, as well as DVD media. For video coding, it supports both interlaced and progressive video. For video, MPEG-2 compresses each raw video frame into one of three frames: an intra-coded frame (I-frame), a predictive-coded frame (P-frame), or a bidirectionally-predictive-coded frame (B-frame). The I-frame is a basic compression of a raw image, the P-frame contains just the changes between itself and the previous frame, and the B-frame contains the differences between itself as well as both the preceding and following frames.[1] We will discuss how the frames are determined and when they are used later in the paper.

Our MPEG-2 encoder encodes our frames by breaking down each frame into 8 pixel x 8 pixel blocks and performing a fixed-point discrete cosine transform (DCT) on the block in order to quantize the values in the matrix in order for lossy compression to be done on each block. To keep to the common standard, we use Huffman encoding. These 8x8 compressed blocks are then commonly formed into macroblocks, which are usually 16 pixels x 16 pixels large. See Section 4.3 for details.

Each pixel in our frames contains a 4-bit luminance number (how bright the pixel is), and two 4-bit chrominance numbers (the color of the pixel). As the human eye cannot detect many of the nuances of differing chrominance values, we were able eliminate some of them in order to save space without negatively impacting the video quality to the naked eye by subsampling each pixel. Raw video has a format of 4:4:4 (luminance: red chrominance: blue chrominance). MPEG-2 format supports 4:4:4, 4:2:2 (half of chrominance numbers removed), and 4:2:0 (three quarters of chrominance numbers removed). We started with a 4:2:2 format, and moved to a 4:2:0 format once our encoder worked in a 4:2:2 format.

Another way we could have saved space would have been to use interlaced video rather than the progressive video we did use. Interlacing would involve only encoding half of the picture per frame. This would be done by every other frame encoding the even rows of pixels, and the other frames encoding the odd rows. However, we did not have time to implement interlacing, as the code is very complicated when combined with motion prediction.

We use Level low and Profile main. We chose to use the main profile because it is the simplest implementation that accounts for bi-directional motion prediction

(B-frames). We also chose level low because it supports a low max bit rate (4 Mbit/s), so it would implementable on the DSK

3. Data Flow/Role of the DSK

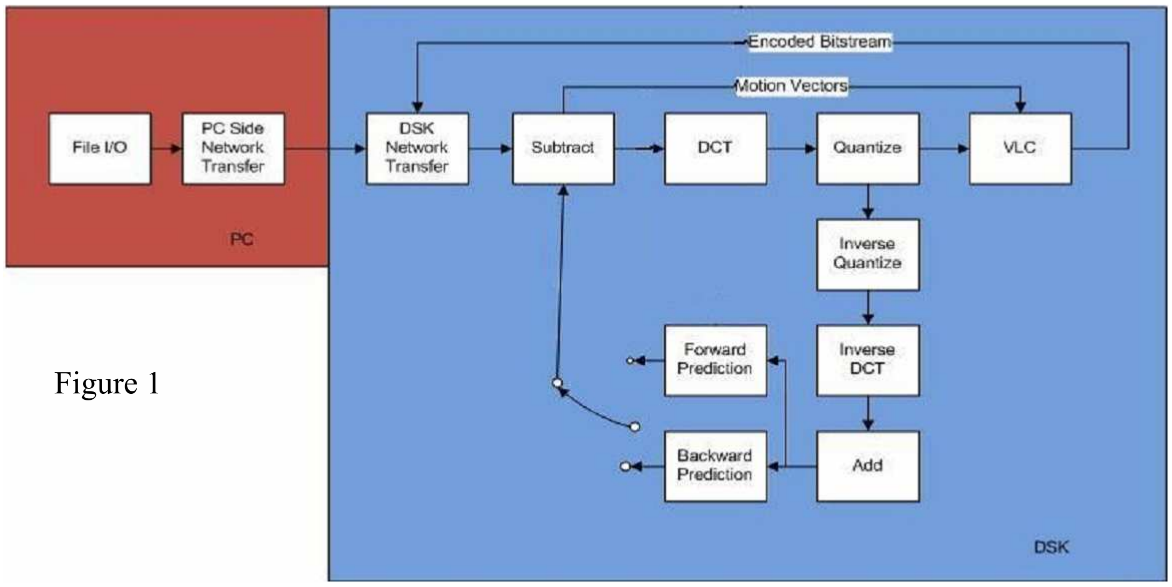


Figure 1

The PC side simply just sends the input video for reading, and writes the output bitstream to a new .m2v file.

The DSK does the majority of the work performing the algorithms shown in Figure 1 and described in Section 4. Since there is relatively high latency for PC→DSK transfer, we wanted all system work to be done on the DSK.

4. Algorithms

4.1 - Discrete Cosine Transform

The Discrete Cosine Transform (DCT) is an essential part to lossless compression of images and videos. The DCT has the same functionality as the Discrete Fourier Transform (DFT) in that it transforms a signal from time domain to the frequency domain. However, while the DFT utilizes a sum of sine functions to represent a signal, the DCT expresses the frequency as a sum of cosine functions. The cosines are actually critical for compression because cosines only have real values, thus fewer coefficients are needed to compute a signal's frequency, making DCTs more efficient.[2]

The DCT subdivides the image or video frame into 8x8 blocks. To perform the DCT, each row then column must transform through a one dimensional DCT. The row or column would just be eight values which represent the amplitudes of a signal and eight different points in time. The output array would then contain the same number of values as the original array. The first value is the DC which is the average of all the values in the input array. The other values in the output array are the AC coefficients which are the amplitudes of specific frequency components in the input array. The culmination of the two dimensional DCT algorithm can be expressed with the formula in Figure 2.[3]

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cos\left[\frac{\pi}{N_1}\left(n_1+\frac{1}{2}\right)k_1\right] \cos\left[\frac{\pi}{N_2}\left(n_2+\frac{1}{2}\right)k_2\right].$$

Figure 2

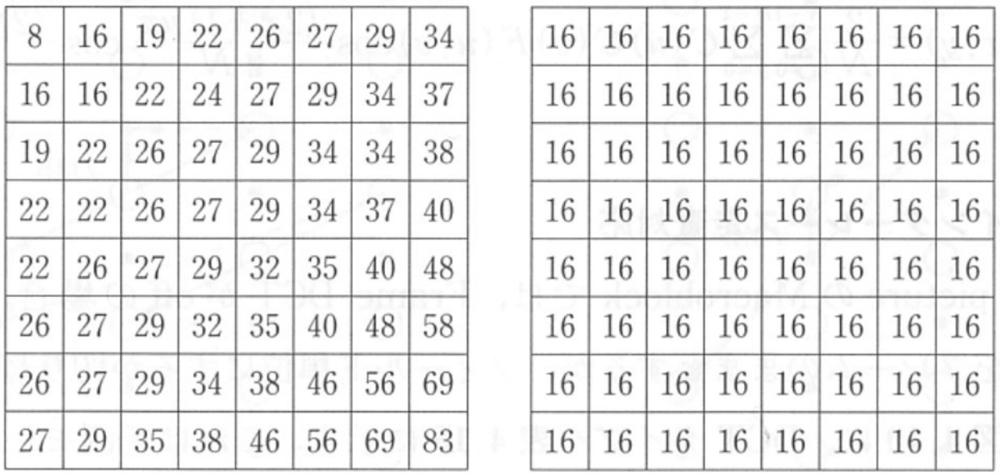
We then perform the DCT on each subdivision of each video frame. DCT compression is lossless, meaning that the original signal can be exactly reconstructed. The following steps show how further techniques can be used to increase the compression ratio but subsequently introducing lossy data.

4.2 - Quantization

Quantization is a lossy compression method that compresses a range of values to a single quantum value. Quantization also turns our fixed-point numbers from the DCT into easily calculable integers, which we can do as the human eye should not be able to check the loss in precision. Applying quantization will allow us to further reduce the size of the output.

The objective of quantization is to minimize the number of bits which must be transmitted to the decoder, so that it can perform the inverse transform and reconstruct the image. Reduced quantization accuracy reduces the number of bits which need to be transmitted to represent a given DCT coefficient, but increases the possible quantization error for that coefficient as we are taking the fixed-point number passed in from the DCT and quantizing it to an integer.[2]

In order to undergo transformations using quantization, we have a quantization matrix which is designed to preserve certain frequencies. This matrix can either be customized or a standard one depending on the property of the transformation. The output matrix would be each element of the DCT coefficient matrix divided by its corresponding element in the quantization matrix and then that result rounded to the nearest integer. The size of the matrix will be an 8x8 matrix. A matrix is used because it is defined in the MPEG-2 standard.



8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16

（a） Intra Figure 3 （b） Non-Intra

We will be using quantization to filter out high frequency components because humans are poor at determining the exact strength of high frequency brightness variations. After using a common quantization matrix and performing the steps in the previous paragraph, we will get an output matrix with many of the higher frequency components rounded to zero.[3] For our intra-coded frames (I frames), the values in the matrix are arranged such that the lowest values are in the top-left of the matrix, and the highest values are in the bottom right as show in Figure 3a. Intra-coded frames only do DCT and quantization for compression that is similar to straight JPEG compression. Non-intra-coded frames (B and P frames) will all have a constant value as shown in Figure 3b(16-bit in our project) as these frames only contain motion vectors only. Fixed point operations were not necessary for quantization because only whole values were needed so remainders were discarded.

4.3 - Macroblocking

In typical video compression algorithms, a 16x16 region in the video frame is known as a "macro block". As shown in Figure 4, this macroblock taken from a slice of an I frame, contains four 8x8 luminance blocks, one 8x8 color red block and one 8x8 color blue block. All the frames are assorted in Groups of Pictures (GOP). This refers to the number of frames between two I-frames. In between the I-frames, we have the predicted frames which occur in the order shown below. Macro blocks are implemented after quantization and used to predict the next macro block in motion prediction.[2]

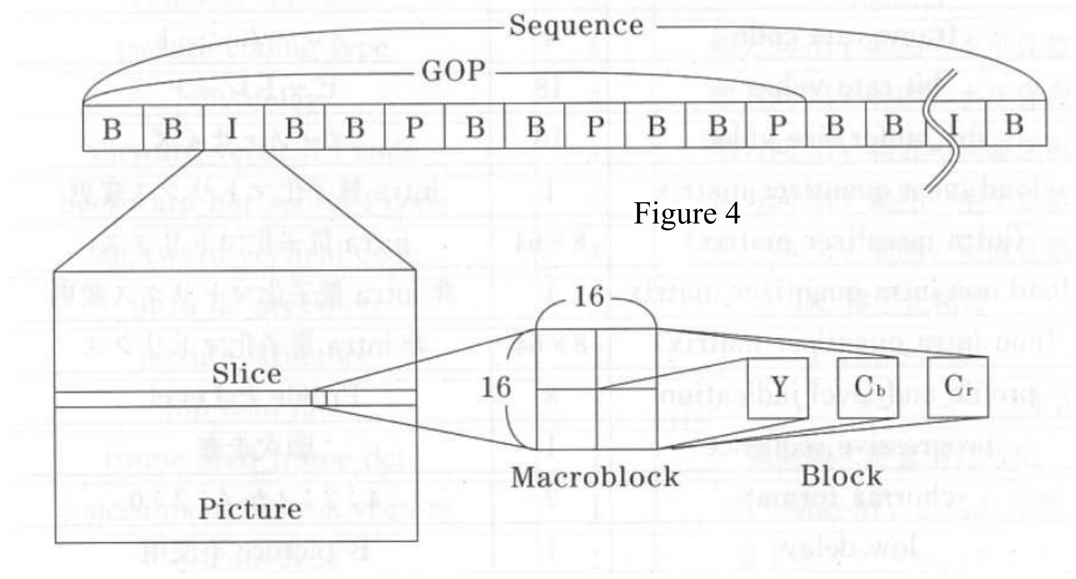


Figure 4

4.4 - Motion Prediction

Motion prediction is the key component that separates the video compression algorithm from the image compression algorithms. Since a video contains many frames of images, further compression can be rendered by reducing the redundancy of frequency through motion prediction. The simplest intra-frame prediction is those where no spatial reposition occurs. Thus, the macro block for the next frame is exactly the same. However, when motion occurs is when the predictions start to get tricky. The way to predict the motion of an object in a macro block is to create a bunch of trial offsets and test to see which one has the least amount of error between the block coded and the prediction.

The MPEG-2 algorithm uses a feedback system to perform the motion prediction. The coder subtracts the error picture from the original source picture to generate a prediction error picture. This picture is then combined with the DCT, quantized and then coded with the VLC (discussed in the next section).

As shown in Figure 5, for every macroblock, we do a 32x32 pixel search for the minimum error motion transpose relative to the previous or future reference frame. The difference between the reference and the minimum difference is called the prediction error. For this minimum error search, we follow the heurisitics of source code. Sometimes, the macroblock won't move and so there's no motion vector. Conversely, motion might be too "great" or outside the 32x32 search area which leads the encoder to giving up, which keeps the motion same, causing pixilation in the video.

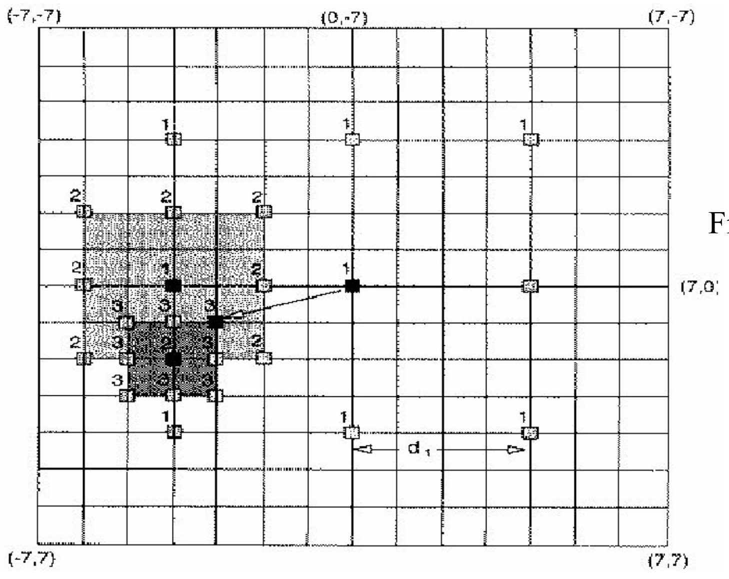


Figure 5

There are three different types of picture frames and a couple of ways they can be predicted. Intra pictures (I) are coded without a reference to another picture or block. They are only useful when no motion is applied to an object. Non-Intra Predictive pictures (P) rely on the I and P picture for motion compensation and can only go one way directionally. Non-Intra Bidirectional pictures (B) offer the greatest compression because they use next I and previous P pictures for motion compensation.[2] We utilize the B pictures by averaging the forward and backward direction pictures in order to perform the highest quality of compression. Of the three frame types B-frames provide the most compression.

Figure 6 shows how the prediction error from the minimum error search to create motion vectors from the predicted frames and the reference frames. Motion vectors are only encoded in B and P frames. The figure below is bi-directional prediction. Frame n uses the translation from n-1 to find the motion vector shown in the image. Then, using the motion vector we extrapolate to the n+1 frame to find the forward motion vector.

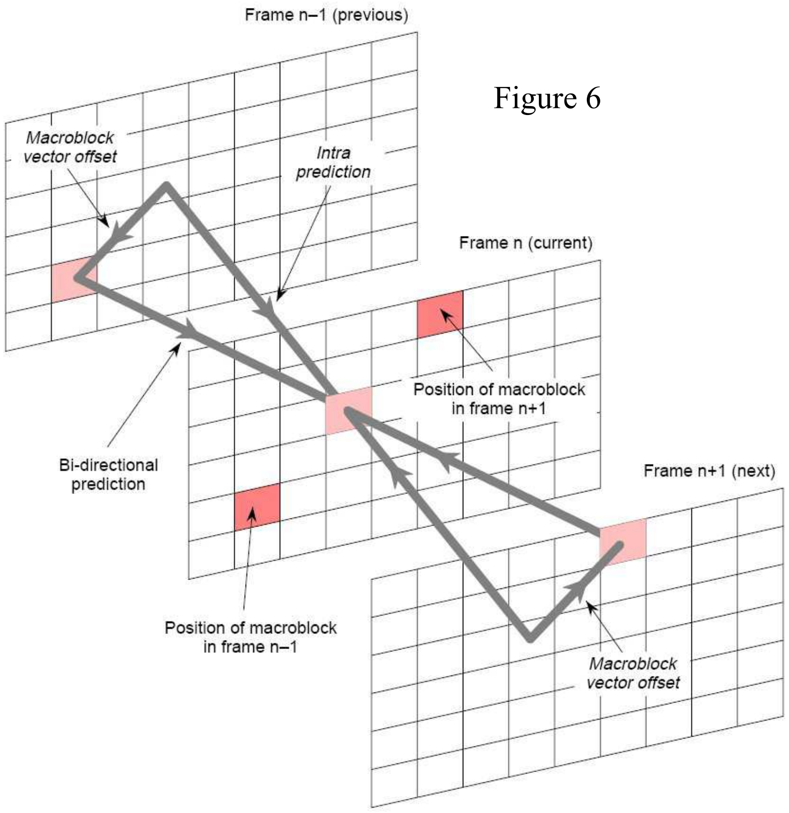


Figure 6

4.5 - Variable-Length Coding

Variable Length Coding (VLC) is a statistical coding technique. It uses short code words to represent values which occur frequently and long code words to represent values which occur less frequently and stores these code words into a lookup table for the decoding process. This then produces a file containing this encoded data as well as a file for the lookup table. While the presence of the lookup table means that encoding is not practical for small amounts of data, it is extremely useful for large files like a video.[7]

There are three different components of each macroblock that will be encoded in VLC. The first one is the VLC for the motion vector. It consists of (1) Huffman code, (2) difference motion vector, (3) fixed length search area. Figure 7 shows the VLC for luminance. On the right, we have the luminance value after DCT and quantize and on the left we have the encoding. This is a standard table that we follow and used as a look up table in our code.[7]

Variable Length Code	dct_dc_size_luminance
100	0
00	1
01	2
101	3
110	4
1110	5
1111 0	6
1111 10	7
1111 110	8
1111 1110	9
1111 1111 0	10
1111 1111 1	11

Figure 7

The next VLC is for the DC component. It consists of (1) size Huffman Code, (2) value of luminance and color.[7]

The last VLC is for the AC component. It consists of the same encoding as DC component except that it has an extra component called run which ranged between 1 and 64 and notifies the decoder of which AC component it is.[7]

MPEG-2 has an established dictionary of lookup tables at www.mpeg.org. We plan on using these tables as Group 10 in Fall 2006 found them to be more efficient than any tables they were able to manually produce.

4.6 - Fixed Point Notation

The fixed point notation is a number representation having a set number of digits after the radix point. This is opposed to floating point, where the radix point can move, giving it the ability to represent a greater number of possible fractions. Generally, fixed point units are cheaper and smaller than floating point units, so many microprocessors and embedded systems eschew using a FPU if possible. Also, fixed point allows for lower power dissipation. While the increased precision offered by floating point numbers make it preferable a growing number of applications, fixed-point remains very useful for video. This is because the human eye is easily fooled, meaning high-precision floating points are not necessary; furthermore, the high data rate of video lends itself well to fixed-point calculations.[5]

The upper bound or lower bound of a two's complement fixed-point number is the upper bound or lower bound of its integer type, divided by its scaling factor. This means that in binary, our upper bound and lower bound can be calculated as $(2^{b-1} - 1) / 2^f$ and $-(2^{b-1}) / 2^f$ respectively, where f is the scaling factor (number of digits after the radix point) and b is the total number of bits in the number.[5]

Addition and subtraction with fixed-point numbers involves simply adding the two numbers together. Multiplication involves multiplying the two numbers together, then dividing the result by the scaling factor 2^f. Division involves dividing the two numbers, then multiplying the result by the scaling factor. Overflows in division and multiplication are dealt with by keeping *I* least significant integer bits, and *Q* most significant fractional bits, where *I* and *Q* are the number of integer and fractional bits in the original notation. [5]

Our project is implemented entirely in fixed point. Originally, we thought we would need fixed point calculations throughout the program but fortunately, we found a fixed point assembly DCT in the TI image library. Also, the quantizer does not need to be implemented in fixed point because it rounds the values to the nearest whole number so the remainder does not need to be accounted for.

5. Sizing

The vides are in YUV format with a 352x288 picture size; there is 352x288 for the luminance, 176x144 for red color, and 176x144 for blue color. Each frame's size totals to 152, 064 bytes. In our code we have seven buffers for forward, backward, and error prediction as well as motion data and macroblocks in 2 byte arrays for fixed pointer calculations totaling at 1,100,184 bytes. In our implementation we have 9 groups of pictures at a time, which allows room for 36 frames on external memory, which means 6,574,488 bytes are transferred to the DRAM at a time. This does not include the encoded output bit stream; to be safe, we allocate the same amount of memory as input, making the total amount of heap used to be 12,048,792 bytes. As a result, all 256Kb of internal memory will be used. After optimization, we took up 91,808 bytes on internal memory. The total size of the encoded file depends on how much motion there is in the video, the less motion the more compression.

6. Data Rate

Our network transfer will be similar to Lab 3. We will send 1400 bytes at a time from PC→DSK because the DSK network receive buffer is small. It does this at a rate wavering around 2 MB/sec. The encoded bitstream will be sent all at once from DSK→PC with a rate wavering around 10 MB/sec. Initially, without any optimizations, the encoding took about 2 minutes per second of raw video. After optimizations were turned on in the compiler (-o0), encoding took about 1 minute per second of raw video. After DMA Paging, encoding took about 40 seconds per second of raw video.

7. Optimizations

There's definitively much room for optimizations in our project. We mostly worried about abiding to the MPEG-2 standard so that a media player could decode it rather then focus on speed. As expected, motion prediction takes the most amount of time because that's where the bug of the calculations takes place in the algorithm. The motion 32x32 motion search plus the creation of the motion vectors involves complex computations so we decided to work on optimizing the DCT and quantizer. So, before DCT and IDCT in the transform function, we page the macro blocks from external to internal memory. The same thing is done for quantization and inverse quantization in the quant and iquant functions. Figure 8 shows the number of cycles before paging and after paging for major functions.

Figure 8

Before Table

Function	Incl Cycles
motion	588160287
predict	8438348
transform	43786958
itransform	50817422
quant.	50339889
iquant.	50339904

After Table

Function	
motion	588788259
predict	8867372
transform	12967103
itransform	17248946
quant.	20676498
iquant.	20676544

8. Differences from Other Groups

Our group approached the same problem as group 10 of Fall 2006. However, instead of using Hadamard Transforms (HTs) we followed the original MPEG-2 standard and use Discrete Cosine Transforms (DCTs). Group 10 used Hadamard Transforms because of the fact that the transforms yield values of just -1's and 1's, which was useful for the hardware implementation they used. DCTs are used in MPEG-2 because they are able to convert spatial variations into frequency variations in a small amount of coefficients and no information is lost, meaning that the exact signal can be reconstructed from the DCT. Group 10 of Fall 2006 used AAC and some VLC while we are going to be only using VLC. Group 15 in Spring 2003 also did video compression, but they used the H.264 algorithm. We implemented the MPEG-2 standard using fixed-point arithmetic.

9. Source Code and Test Data

We modified source code taken from www.mpeg.org/MPEG/MSSG. This source code implements the MPEG-2 standard entirely in C. We modified the code so that it works on the DSK and met our specifications. This entailed doing the DCTs and IDCTs in assembly code using fixed point operations, DSK memory allocation, latency, and network transfer. We also removed any steps related to the interlaced format, since we are using the progressive format. Most significantly, any rate control parameters were removed as they would only be needed in a time-sensitive application such as broadcasting.

Also we use the image library taken from http://focus.ti.com/lit/ug/spru400b/spru400b.pdf. This gives us the FDCT and IDCT in assembly code.

Our input vides are raw videos in .YUV format from the internet. Specifically, we used http://media.xiph.org/video/derf/, which is a collection of video samples specifically for video processing and computer vision. They had a good number of videos at our resolution in the .YUV format with a wide variety of high motion and low-motion videos to compare our efficiency in various types of video.

10. Key Algorithm

The key algorithm that allows for the most compression is motion prediction. This is the most complicated algorithm that we used and required the most time. Details of the algorithm are in Section 4.4.

We first implemented forward prediction since that is necessary for the P-frames. We then also implemented backward prediction. The combination of forward prediction and backward prediction is called bi-direction prediction and is necessary for the B-frames. These B-frames will allow for the most compression.

11. Schedule and Demonstration

Week	**Start Date**	**Task**
0	October 12, 2009	Network transfer: Nik PC side I/0: Ryan PC side encoding: Samir
1	Oct 19, 2009	DCT and quantization: Nik Motion and prediction: Ryan Min. search and VLC: Samir
2	October 26, 2009	Debugging above: everyone
3	November 2, 2009	Putting together I-frame: Nik Putting together P-frame: Ryan Putting together B-frame: Samir
4	November 9, 2009	Debugging above: everyone
5	November 16, 2009	Optimization Design: Nik Profiling: Ryan Profiling: Samir
6	November 23, 2009	Thanksgiving
7	November 30, 2009	Optimization debugging: everyone

In, the final demo we took a raw .YUV video from our video repository on the PC, sent that video to the DSK over the LAN, encoded that video on the DSK, and sent back the output to the PC to be played on Windows Media Player. We then used the Elecard suite to do a frame-by-frame comparison of the original .YUV video against our video. Elecard allowed us to see the differences in each frame, what type of frame we are on (I frame, P frame, or B frame), and the size and data of each frame. See Figures 9 and 10 for the output of the Elecard suite.

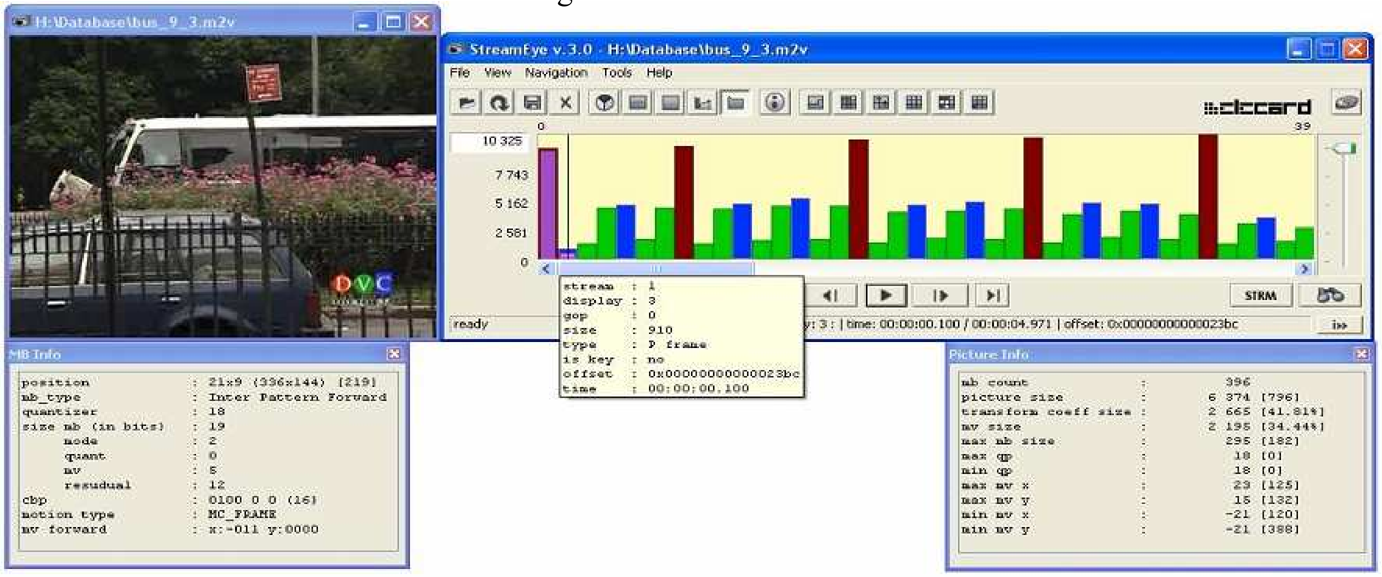
12. Results

Our input files were raw videos in .yuv format and the output encoded bit stream is .m2v format (video, no audio). As expected, videos with less motion have a much higher encoding rate due to image redundancy. The compression ratio for our video file is better than expected; a 22MB video with a lot of motion encodes to about 450KB, about 98% compression. A video with much less motion encodes more because the P and B frames were much smaller; we had a minimal video which was 44MB compress to 550KB, about 98.7% compression. A single I frame after DCT, quantization, and VLC (no motion prediction) was compressed from 152KB to 9KB.

The figure below shows screenshots from the Elecard MPEG-2 Analyzer. Figure 9 has a video of a bus driving across the screen with a lot of motion. Figure 10 has a man talking to the camera with little motion occurring in the video. The analyzer shows the .m2v bit stream in order of the encoded video file. The red bars are the I-frames, the blue bars are the P-frames, and the green bars are the B-frames. The size of the bar denotes the size of the frame in bytes. We can see after juxtaposing the two videos that the bus has P and B frames that are larger due to all the motion. The non-intra frames in the video with the man talking are much smaller because of all the image redundancy due to the lack of motion.

Also, we can see in the two videos the order of the GOP. The first set of GOP has only 7 frames and no initial B frames because there are no n-1 frames to base the motion vectors. After that, the frames occur in the regular GOP order. Also, we can see a pattern where the first B frame is always smaller then the second B frame. This is due to the face that the second B frame is based off of the first one using bi-directional prediction. Thus, the first frame is decoded, since the encoder has a built in decoder, and used for the second B-frames motion prediction which has less image redundancy and results in a larger encoded frame most of the time.

Figure 9



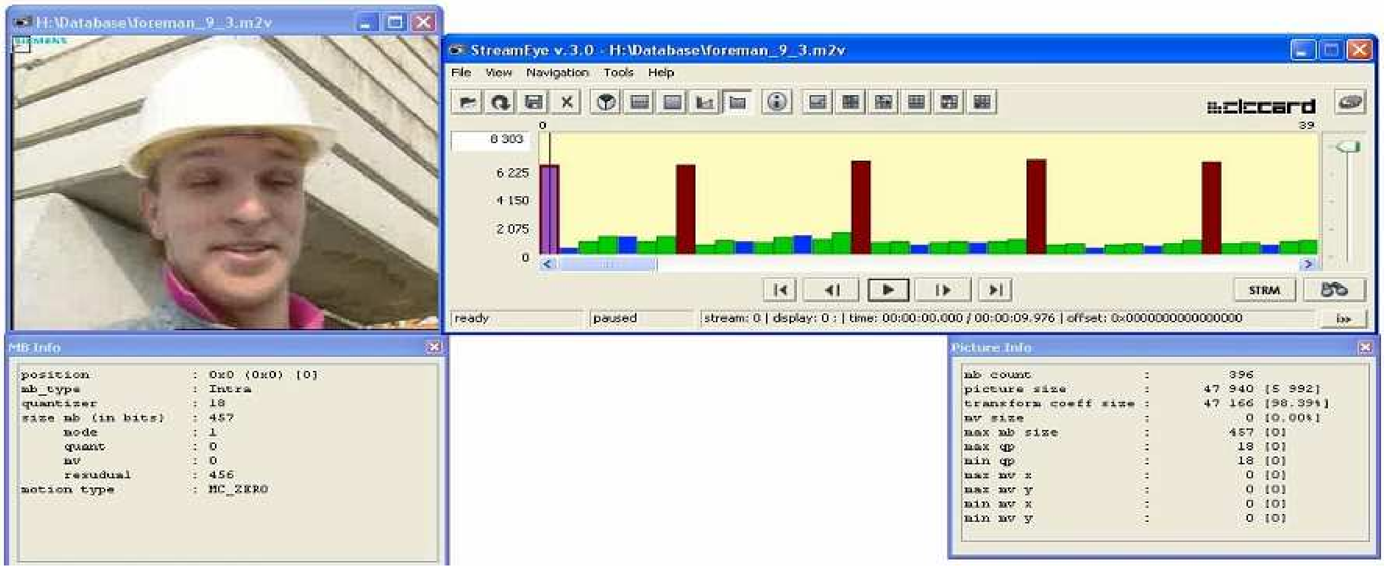
```
y: 3 : | time: 00:00:00.100 / 00:00:04.971 | offset: 0x00000000000023bc
```

```
stream   : 1
display  : 3
gop      : 0
size     : 910
type     : P frame
is key   : no
offset   : 0x00000000000023bc
time     : 00:00:00.100
```

```
position              : 21x9 (336x144) [219]
mb_type               : Inter Pattern Forward
quantizer             : 18
size mb (in bits)     : 19
      mode            : 2
      quant           : 0
      mv              : 5
      resudual        : 12
cbp                   : 0100 0 0 (16)
motion type           : MC_FRAME
mv forward            : x:-011 y:0000
```

```
mb count              :       396
picture size          :     6 374 [796]
transform coeff size  :     2 665 [41.81%]
mv size               :     2 195 [34.44%]
max mb size           :       295 [182]
max qp                :        18 [0]
min qp                :        18 [0]
max mv x              :        23 [125]
max mv y              :        15 [132]
min mv x              :       -21 [120]
min mv y              :       -21 [388]
```

Figure 10


H:\Database\foreman_9_3.m2v
StreamEye v.3.0 - H:\Database\foreman_9_3.m2v
File View Navigation Tools Help
elecard
8 303
6 225
4 150
2 075
0
39
STRM
ready
paused
MB Info
Picture Info

```
stream: 0 | display: 0 : | time: 00:00:00.000 / 00:00:09.976 | offset: 0x0000000000000000
```

```
position              : 0x0 (0x0) [0]
mb_type               : Intra
quantizer             : 18
size mb (in bits)     : 457
      mode            : 1
      quant           : 0
      mv              : 0
      resudual        : 456
motion type           : MC_ZERO
```

```
mb count              :       396
picture size          :    47 940 [5 992]
transform coeff size  :    47 166 [98.39%]
mv size               :         0 [0.00%]
max mb size           :       457 [0]
max qp                :        18 [0]
min qp                :        18 [0]
max mv x              :         0 [0]
max mv y              :         0 [0]
min mv x              :         0 [0]
min mv y              :         0 [0]
```

13. Video Database

File Image	File Name	# Frames	File Size (kB)	Compressed Size (kB)
	bus_cif.yuv	150	22,809	482
	foreman_cif.yuv	300	45,619	558
	coastguard_cif.yuv	300	45,619	510
	Flower_cif.yuv	250	38,016	743
	Hall_cif.yuv	300	45,619	403

14. Problems and Issues

The biggest issue was that the decoder expected the frames to be in a different order than the frames in our encoded bitstream. Figure 11a shows our encoded bit stream in display order. Figure 11b shows the transmission order, the way the decoder wants the ordering of the frames. We expected the decoder to change the positions of the frames to the transmission order; however, it did not do that so when the video is played in a media player, the motion is not fluid. This is not due to bad motion prediction but due to the different order of the P and B frames that the decoder expects. Given another week, we would have buffered the P and B frames to put them in transmission order.

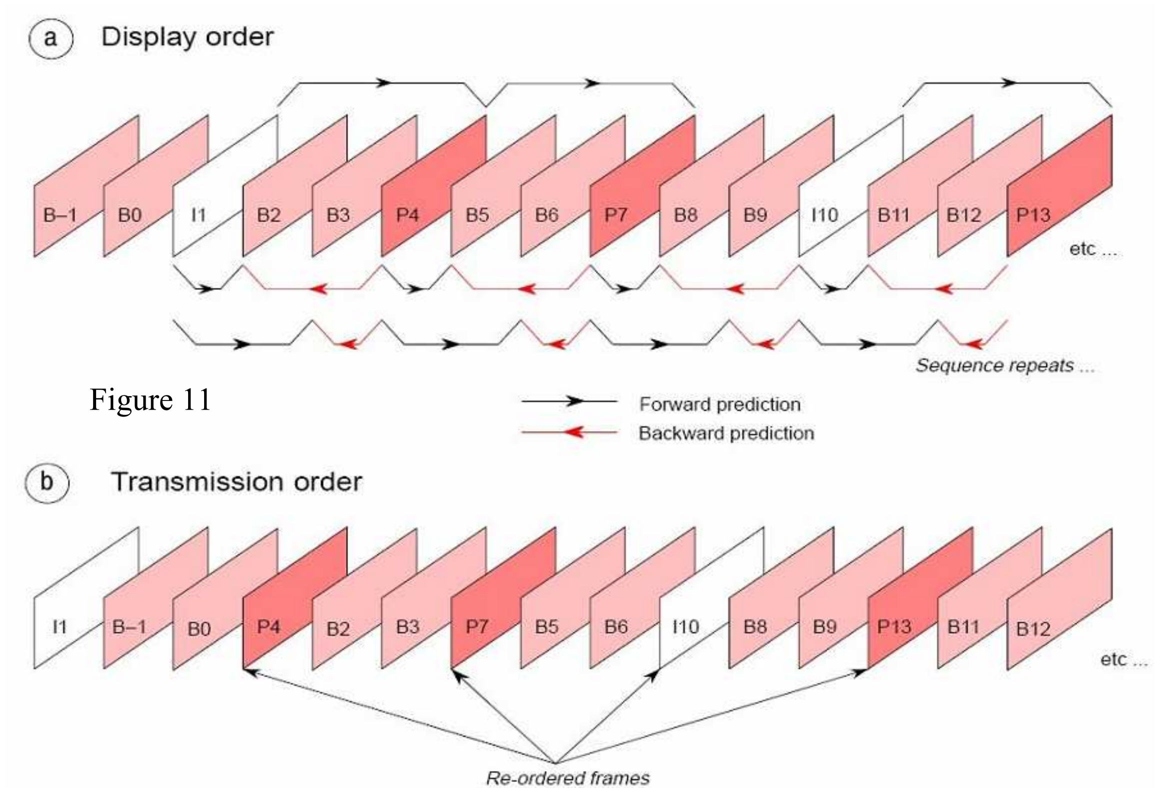


Figure 11

We also did not implement rate control, vbv buffer delay, field frames, and time code. We used progressive frames instead of filed frames because field is usually used for TV and is a harder algorithm because it switches between even and odd frames. Rate control and vbv buffer delay is also used for broadcasting to control the bit stream rate and would also be additional algorithms to implement. The time code was found to be unnecessary so we did not include it.

15. References

1. http://en.wikipedia.org/wiki/Mpeg_2
   - Provided an initial and basic understanding of MPEG-2 compression
   - Crucial to develop background knowledge
   - Concise
   - Contains Figure 2 and Figure 3
2. http://www.engr.usask.ca/classes/CME/462/
   - Provided in depth descriptions and figures for MPEG-2 compression
   - Gave us a complete understanding of MPEG-2 compression
   - Comprehensive throughout all steps of encoding
   - Contains Figure 4, Figure 5, Figure 6, and Figure 11
3. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=490024&isnumber=10491
   - Good article by leading video signal analysts, P.N. Tudor
   - Also provided a good explanation of MPEG-2
   - More comprehensive than Wikipedia, but less than the above source
4. www.mpeg.org/MPEG/MSSG
   - Provided source code in C
5. http://focus.ti.com/lit/ug/spru400b/spru400b.pdf
   - Provided image library in ASM
6. http://media.xiph.org/video/derf/
   - Provided all our sample raw .YUV files
7. http://www.xilinx.com/support/documentation/application_notes/xapp616.pdf
   - Provided an in-depth explanation of VLC
   - Contains Figure 7
8. http://en.wikipedia.org/wiki/Fixed-point_arithmetic
   - Provided basic overview of fixed-point notation and arithmetic

A. Memory Map

```
***********************************************************************
*******
          TMS320C6x COFF Linker PC Version 4.36
***********************************************************************
*******
>> Linked Sun Nov 08 19:31:47 2009

OUTPUT FILE NAME:   <./Debug/dsk_side.out>
ENTRY POINT SYMBOL: "_c_int00"  address: 0001b140


MEMORY CONFIGURATION

                  name            origin    length      used    attr
fill
         ----------------------  --------  ---------  --------  ----  -
-------
         ONCHIP_RAM              00000000   00038000  00034ab9  RWIX
         SDRAM                   80000000   01000000  01000000  RWIX


SECTION ALLOCATION MAP

 output                                  attributes/
section   page    origin      length       input sections
--------  ----  ----------  ----------   ----------------
.sysmem    0    80000000    01000000     UNINITIALIZED

.bss       0    00000000    00000000     UNINITIALIZED

.data      0    00000000    00000000     UNINITIALIZED

.text      0    00000000    0001b4e0

.nettextfast
*          0    0001b4e0    000075a0

.nettextslow
*          0    00022a80    00006100

.const     0    00028b80    00000711

.cinit     0    00029298    000011c4

.switch    0    0002a45c    00000024

.cio       0    0002a480    00000120     UNINITIALIZED

.stack     0    0002a5a0    00008000     UNINITIALIZED

.far       0    000325a0    00002520     UNINITIALIZED
```

B. Parameter File

```
MPEG-2 Test Sequence, 30 frames/sec
bus_cif    /* name of source files */
-       /* name of reconstructed images ("-": don't store) */
-         /* name of intra quant matrix file     ("-": default matrix)
*/
-         /* name of non intra quant matrix file ("-": default matrix)
*/
stat.out  /* name of statistics file ("-": stdout ) */
1         /* input picture file format: 0=*.Y,*.U,*.V, 1=*.yuv, 2=*.ppm
*/
150       /* number of frames */
0         /* number of first frame */
00:00:00:00 /* timecode of first frame */
9        /* N (# of frames in GOP) */
3         /* M (I/P frame distance) */
0         /* ISO/IEC 11172-2 stream */
1         /* 0:frame pictures, 1:field pictures */
352       /* horizontal_size */
288       /* vertical_size */
1         /* aspect_ratio_information 1=square pel, 2=4:3, 3=16:9,
4=2.11:1 */
4         /* frame_rate_code 1=23.976, 2=24, 3=25, 4=29.97, 5=30
frames/sec. */
2000000 /* bit_rate (bits/s) */
29       /* vbv_buffer_size (in multiples of 16 kbit) */
0         /* low_delay  */
0         /* constrained_parameters_flag */
4         /* Profile ID: Simple = 5, Main = 4, SNR = 3, Spatial = 2,
High = 1 */
10         /* Level ID:   Low = 10, Main = 8, High 1440 = 6, High = 4
*/
1         /* progressive_sequence */
1         /* chroma_format: 1=4:2:0, 2=4:2:2, 3=4:4:4 */
2         /* video_format: 0=comp., 1=PAL, 2=NTSC, 3=SECAM, 4=MAC,
5=unspec. */
2         /* color_primaries */
2         /* transfer_characteristics */
4         /* matrix_coefficients */
352       /* display_horizontal_size */
288       /* display_vertical_size */
0         /* intra_dc_precision (0: 8 bit, 1: 9 bit, 2: 10 bit, 3: 11
bit */
0         /* top_field_first */
1 1 1     /* frame_pred_frame_dct (I P B) */
0 0 0     /* concealment_motion_vectors (I P B) */
1 1 1     /* q_scale_type  (I P B) */
1 1 1     /* intra_vlc_format (I P B)*/
1 1 1     /* alternate_scan (I P B) */
0         /* repeat_first_field */
1         /* progressive_frame */
0         /* P distance between complete intra slice refresh */
0         /* rate control: r (reaction parameter) */
0         /* rate control: avg_act (initial average activity) */
```

```
0         /* rate control: Xi (initial I frame global complexity
measure) */
0         /* rate control: Xp (initial P frame global complexity
measure) */
0         /* rate control: Xb (initial B frame global complexity
measure) */
0         /* rate control: d0i (initial I frame virtual buffer
fullness) */
0         /* rate control: d0p (initial P frame virtual buffer
fullness) */
0         /* rate control: d0b (initial B frame virtual buffer
fullness) */
2 2 11 11 /* P:  forw_hor_f_code forw_vert_f_code search_width/height
*/
1 1 3  3  /* B1: forw_hor_f_code forw_vert_f_code search_width/height
*/
1 1 7  7  /* B1: back_hor_f_code back_vert_f_code search_width/height
*/
1 1 7  7  /* B2: forw_hor_f_code forw_vert_f_code search_width/height
*/
1 1 3  3  /* B2: back_hor_f_code back_vert_f_code search_width/height
*/
```