Demo: Video Compression and Encryption

Diwakar Panchamgam, TAMU
Course Instructor: Professor Deepa Kundur

Objectives of this Lab

- To introduce video compression using Discrete Cosine Transform (DCT) and encryption.
- To implement video compression and encryption in Simulink and in Real-time on the hardware.
- To provide exposure to the **optimizations** needed for the Real-time implementation on the board.

Prelab

- Prior to beginning, you must carefully read over this lab in order to plan how to implement the tasks assigned. Please highlight all the parts you need to show or answer for the TA, so that you do not miss any graded points during the in-lab component or for the report.
- For simulation:
  - Please download the vipmem_Y.avi.file available on the course webpage.
- For Real-time implementation:
  - A CMOS camera.
  - An analog TV to view the output from the board.
  - Cables to connect camera, board and TV.

Deliverables

- Please show the TA all requested in-lab components for full points.
- After the lab, each individual must submit a separate report answering all the questions requested by the TA and asked in this Lab (see Questions section). For full points, please make sure you address all the parts in the lab that are required for the report.

Grading and Due Date

- Please note STRICT DEADLINE for report on the course web site.

Introduction and Background

In this age of dramatic technology shifts, one of the most significant has been the emergence of digital video as an important aspect of daily life [1]. Digital cameras and camera-equipped cell phones are enabling easy capturing, storing, and sharing valuable moments through digital images and video. Set-top boxes are being used to pause, record, and stream live television signal over broadband networks to different locations, while smart camera systems are providing peace of mind through intelligent scene surveillance. Of course, all of these innovative multimedia products would not have materialized without efficient, optimized implementations of practical signal and image processing algorithms on embedded platforms, where constraints are placed not only on system size, cost, and power consumption, but also on the interval of time in which processed information must be made available[2].

A common theme in real-time image/video processing systems is how to deal with their vast amounts of data and computations [2]. For example, a typical digital video camera capturing VGA-resolution quality, color video (640 × 480) at 30fps outputs 27 million pixels per second. If we consider a gray scale image, each pixel is represented by 8bits and so we need a bandwidth of 216 Mbps to transmit VGA-resolution video. Therefore, video compression is frequently used to decrease the amount of data transmitted over a channel. Discrete Cosine Transform (DCT) is one of the techniques used in compressing video.

Discrete Cosine Transform (DCT)

In this lab we use the DCT block from the Video Processing Library in Simulink. Still, it is useful to understand the various steps in computing a DCT on an image as given below.

1. First divide the image (N by M pixels) into blocks of 8x8 pixels as shown in Figure 1.

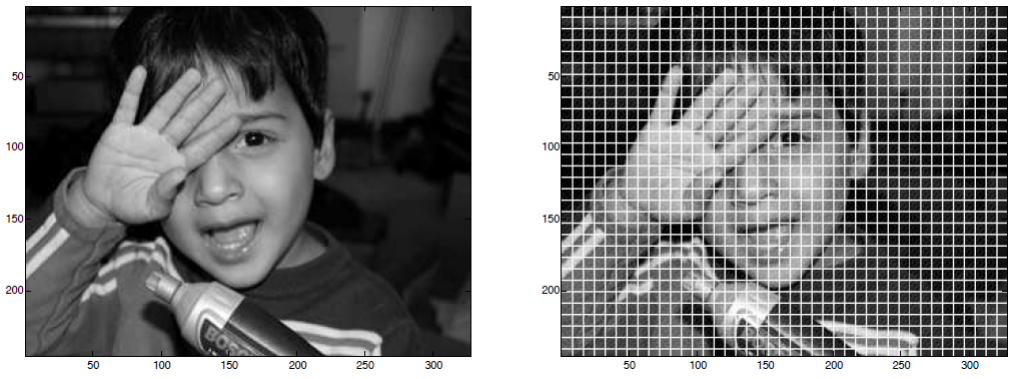

Figure 1: Original Image and Image divided into 8x8 blocks [3].

2. Compute the DCT on the 8x8 blocks using the formula given below. Please note that in this lab we use the DCT block from Video Processing Library in Simulink and not the formula given below.

$$\mathcal{I}^{B}_{DCT}(k,l) = \sum_{m=0}^{7}\sum_{n=0}^{7} I^{B}(m,n)\cos\left[\frac{\pi}{8}\left(n+\frac{1}{2}\right)k\right]\cos\left[\frac{\pi}{8}\left(m+\frac{1}{2}\right)l\right]$$

Figure 2: Formula to calculate DCT on an 8x8 block image.

3. Select the high-frequency components via RxR mask. Please note that both the images are displayed in a log-amplitude scale in Figures 3(a) and 3(b) below.

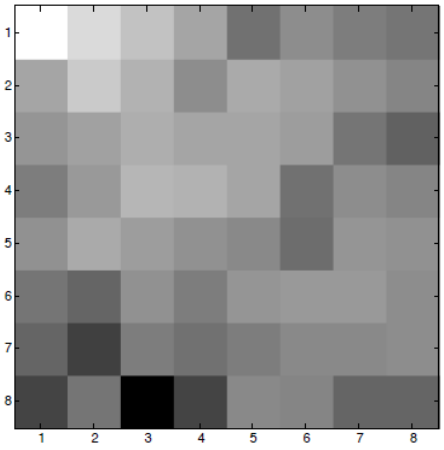

Figure 3(a): 8x8 block image in log-amplitude scale [3].

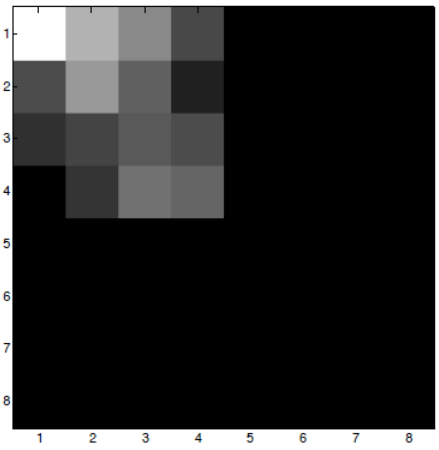

Figure 3(b): After applying RxR mask on the image where R=4 [3].

4. Compute the 8x8 block IDCT (Inverse DCT) on the compressed DCT coefficients using the formula given below. Please note that in this lab we use the IDCT block from the Video and Image Processing Blockset in Simulink and not the formula given below.

$$\tilde{I}^{B}(m,n) = \sum_{k=0}^{7}\sum_{l=0}^{7}\alpha(k)\alpha(l)\tilde{\mathcal{I}}^{B}_{DCT}(k,l)\cos\left[\frac{\pi}{8}\left(n+\frac{1}{2}\right)k\right]\cos\left[\frac{\pi}{8}\left(m+\frac{1}{2}\right)l\right]$$

Figure 4: Formula to calculate IDCT on a 8x8 block image [3].

5. Figures 5(b), 5(c) and 5(d) show lossy compression results using DCT with R=3, R=2 and R=1. Please note that R=3 means that we are selecting and transmitting only a 3x3 block from a 8x8 block resulting in a compression of 86%. As the value of R decreases, compression increases but picture quality decreases too.

Figure 5(a): Original Image [3].

Figure 5(b): R=3 (86% compression) [3].

Figure 5(c): R=2 (93% compression) [3].

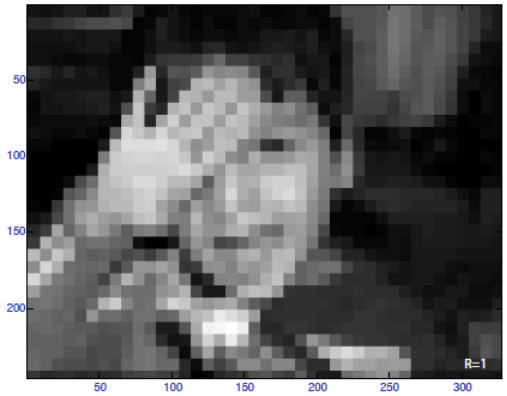

Figure 5(d): R=1 (98% compression) [3].

Encryption

For encrypting the video we perform additional actions after Step 3 (discussed on Page 3). Just for recap, in Step 3 we select an RxR matrix from the computed 8x8 block of DCT coefficients. To add encryption to the block of RxR DCT co-efficients, we multiply it with a scrambling matrix as shown in Figure 6 below. Please note that this is **element-wise** matrix multiplication and not the usual matrix multiplication.

$$RxR\ block\ of\ DCT\ co-ef\ f = \begin{bmatrix} A1 & A2 \\ A3 & A4 \end{bmatrix}$$

$$Scrambling\ matrix = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$$Scrambled\ RxR\ block\ of\ DCT\ co-ef\ f = \begin{bmatrix} A1 & -A2 \\ -A3 & A4 \end{bmatrix}$$

Figure 6: Scrambling process.

Decryption

For decrypting the video we perform additional actions before Step 4 (discussed on Page 3). Just for recap, in Step 4 we apply IDCT on the RxR block of DCT co-efficients. Before computing IDCT we multiply the scrambled RxR block of DCT co-efficients with the de-scrambling matrix as shown in Figure 7. Please note that the scrambling matrix and the descrambling matrix are the same in our case. Also note that we are doing **element-wise** matrix multiplication and not the usual matrix multiplication.

$$RxR\ block\ of\ DCT\ co-ef\ f = \begin{bmatrix} A1 & A2 \\ A3 & A4 \end{bmatrix}$$

$$Scrambling\ matrix = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$$Scrambled\ RxR\ block\ of\ DCT\ co-ef\ f = \begin{bmatrix} A1 & -A2 \\ -A3 & A4 \end{bmatrix}$$

$$Descrambling\ matrix = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

$$Descrambled\ RxR\ block\ of\ DCT\ co\text{-}\ ef\ f = \begin{bmatrix} A1 & A2 \\ A3 & A4 \end{bmatrix}$$

Figure 7: Scrambling and De-scrambling process.

Design and Implementation

First we implement video compression and encryption in Simulation and then in Real-time on the C6437 board.

Video Compression and Encryption (Simulation)

The goal of this lab is to, in part, build and test the following Simulink model shown in Figure 8.

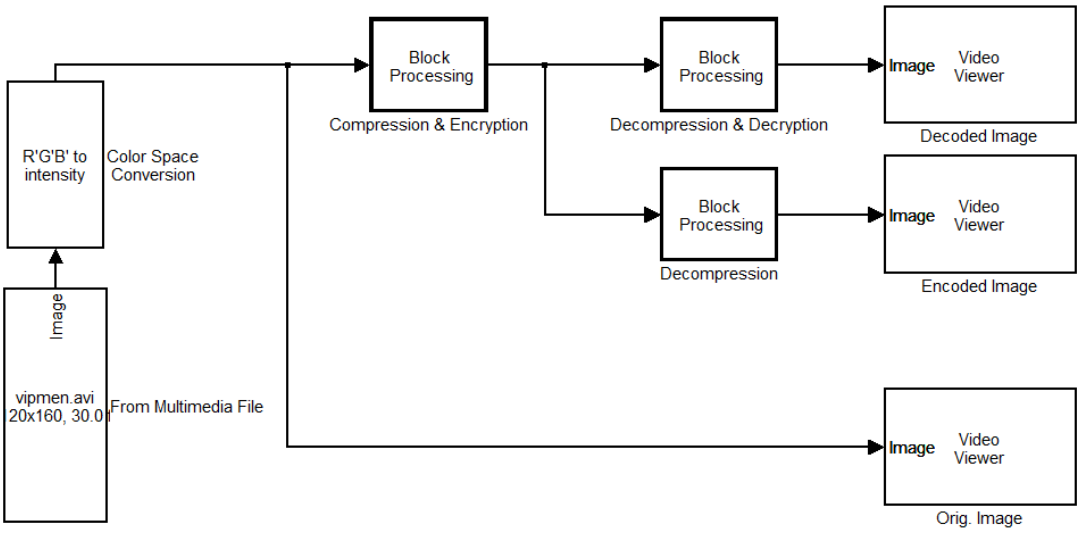

Figure 8: Video compression and Encryption Simulink Model to be implemented.

Building the Simulink Model

Input blocks:

1. Add a source file from Video and Image Processing Blockset → Sources → From Multimedia File. Set the filename to "vipmem.avi", the sample time to "inf" and the output data type to "Single."
2. Next convert the input color video from the file "vipmem.avi" to grayscale using Video and Image Processing Blockset → Conversions → Color Space Conversion. Set the conversion to "R'G'B' to intensity." Set the image signal to "One multi-dimensional signal."

Processing blocks:

3. First we create the Compression & Encryption subsystem as shown in Figure 9. To create a place holder for this subsystem in the main Simulink file, we add a Block Processing block from Video and Image Processing Blockset → Utilities.

   Remember in steps 1-4 of DCT discussed above on Page 2, we need to extract an 8x8 block from the source image and perform different steps on it. This block extracts sub-matrices of a user-specified size from the input matrix. Then it sends each sub-matrix to a subsystem for processing, and then reassembles each subsystem output into the output matrix.

   In the parameters for the Block Processing block, set the Number of inputs to "1" and the Number of outputs to "1." Set the Block size to "{[8 8]}" and Overlap to "{[0 0]}." Set the Traverse order to "Row-wise." Now click on the Open Subsystem button and then create the Simulink model shown in Figure 9.

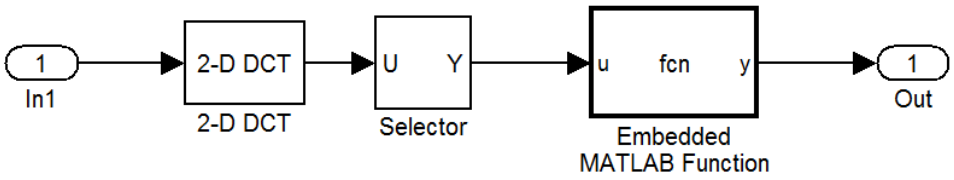

Figure 9: Compression and Encryption Simulink Model to be implemented.

a) Add the 2-D DCT block from Video and Image Processing Blockset → Transforms. Do not change any of the parameters in this block and leave them at their default values. Now we have computed the DCT on an 8x8 block of pixels.

b) Next we have to select a RxR block from the 8x8 block of DCT co-efficients. In this lab we set R=4 i.e. 50% compression. We use the Selector block from Simulink → Signal Routing. Set the Number of input dimensions to "2" and Index mode to "Zero-based." Also set the Index option to "Starting index (dialog) and Index to "0." Finally set the Output Size to "4."

c) In this step we multiply the 4x4 block of DCT co-efficients with the Scrambling matrix. This could be done using Matrix Multiply block, but instead we use Embedded MATLAB Function block from Simulink → User-Defined Functions. This is to introduce the concept of embedding MATLAB code in a Simulink model.

In the Embedded MATLAB function block we can write normal MATLAB function with an input variable and an output variable. The input variable 'u' in our case would be the block of 4x4 DCT co-efficients. The output variable 'y' in our case would be the product of input variable with our scrambling matrix. The code for the function is given in Figure 10.

```
function y = fcn(u)
MASK=  [ 1 -1 1 -1;
       -1 1 -1 1;
        1 -1 1 -1;
       -1 1 -1 1;];
y = u.*MASK;
```

Figure 10: MATLAB code for scrambling the 4x4 block of DCT co-efficients

4. In this step we create the Decompression & Decryption subsystem block as shown in Figure 11. To create a place holder for this subsystem in the main Simulink file, we add a Block Processing block from Video and Image Processing Blockset → Utilities. In the parameters for the Block Processing block, set the Number of inputs to "1" and the Number of outputs to "1." Set the Block size to "{[4 4]}" and Overlap to "{[0 0]}". Set the Traverse order to "Row-wise." Now click on the "Open subsystem" button and then create the Simulink model shown in Figure 11.

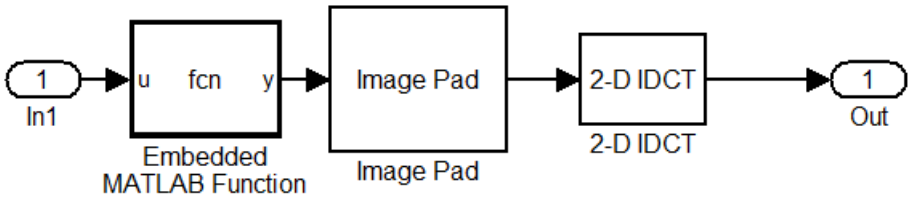

Figure 11: Decompression and Decryption Simulink Model to be implemented.

a) First we add the Embedded MATLAB function block where we write MATLAB code to descramble. The input variable 'u' in our case would be the block of 4x4 scrambled DCT co-efficients. The output variable 'y' in our case would be the product of input variable with our descrambling matrix. The code for the function is given below in Figure 11.

```
function y = fcn(u)
MASK=  [ 1 -1 1 -1;
       -1 1 -1 1;
        1 -1 1 -1;
       -1 1 -1 1;];
y = u.*MASK;
```

Figure 11: MATLAB code for descrambling the 4x4 block of scrambled DCT co-efficients

b) Previously in Step 3 on Page 6, we used a Selector block to select a 4x4 block from an 8x8 block of DCT co-efficients. Here we do the opposite i.e., we pad the 4x4 block of descrambled DCT co-efficients to get the 8x8 block of DCT co-efficients. Add the Image Pad block from Video and Image Processing Blockset → Utilities. Set the Method to "Constant" and Pad value source to "Specify via dialog." Set the Pad value to "0" and Specify to "Pad size." As we would be padding the 4x4 block to the right and bottom we set Pad rows at to "Right" and Pad columns at to "Bottom." We set the Pad size along rows and Pad size along columns to "4."

c) Add the 2-D IDCT block from Video and Image Processing Blockset → Transforms. Do not change any of the parameters in this block and leave them at their default values. Now we have computed the IDCT on a 8x8 block of pixels

5. Finally, we create the Decompression subsystem. We need this subsystem to display the scrambled video. To create a place holder for this subsystem in the main Simulink file, we add a Block Processing block from Video and Image Processing Blockset → Utilities. In the parameters for the Block Processing block, set the Number of inputs to "1" and the Number of outputs to "1." Set the Block size to "{[4 4]}" and Overlap to "{[0 0]}." Set the Traverse order to "Row-wise." Now click on the Open Subsystem button and then create the Simulink model shown in Figure 12. We repeat Steps 4 (b) and 4(c) on Page 7 to add the Image Pad and the 2-D IDCT blocks. Please note that Step 4(a) is not used here as we do not want to descramble the video here.

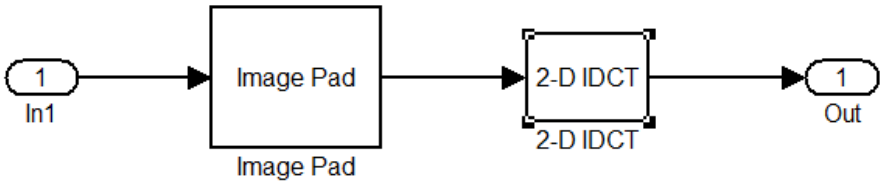

Figure 12: Decompression subsystem.

Output blocks:

6. Add three Video Viewer blocks from the Video and Image Processing Blockset → Sinks. Connect them to Decompression & Decryption subsystem, Decompression subsystem and to the input source as shown in Figure 8. Figures 13 (a), (b) and (c) show the original video, compressed video and scrambled video respectively. We are compressing the video by 1/2 in this lab as we transmitted only a 4x4 block from an 8x8 block of DCT co-efficients. Please note that in the compressed video the edges are blurry as the compression scheme we used is a lossy one. As we can notice in the scrambled video, our scrambling scheme is able to distort the original video making it not fit for normal viewing purposes.

Figure 13 (a): Original video.

Figure 13 (b): Compressed video.

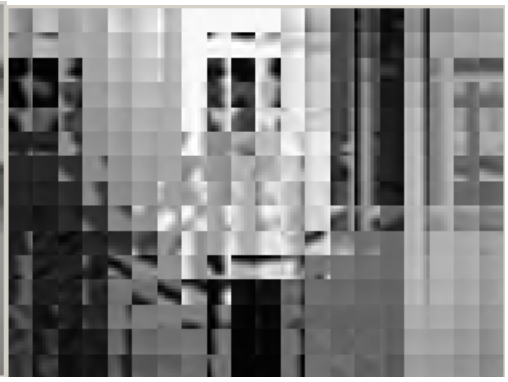

Figure 13 (c): Scrambled video.

Video Compression and Encryption (Real-time implementation)

Here we implement video compression and encryption in real-time on the C6437 board. The real-time implementation would be conceptually similar to the simulation, but would have significant changes to make it run in real-time on the board.

To understand the limitations of the board, let us compare the processing capabilities: In simulation we are running it on a personal computer having a quad-core processor running at 2GHz whereas the DM6437 has a single core processor running at 500MHz. One naïve solution would be to go for a board with higher processing capabilities say with a processor running at 2GHz. Then not only have we increased the cost of the board due to the upgrade but also it increases the power consumption drastically. In case of mobile devices it would be a fatal flaw. For example, how would like if you have to charge your cell phone every 4 hours to use it?

Thus we have an engineering problem at hand: *coming up with a solution within the given constraints*. The solution would be to **optimize** our original simulation to make it run in real-time on the board. In some cases where our best optimization is not sufficient to make it run in real-time, then we have to make some judicious **trade-off decisions**. These two principles would be further explained in the steps below where we create the Simulink model given in Figure 14. Please note that when your project is not working it would appear that getting a powerful board is the *obvious solution*, but it is not the right solution for reasons mentioned above. The current processor DM6437 is powerful enough and is being used in many real-time applications such as rear-view camera assisted parking in current cars.

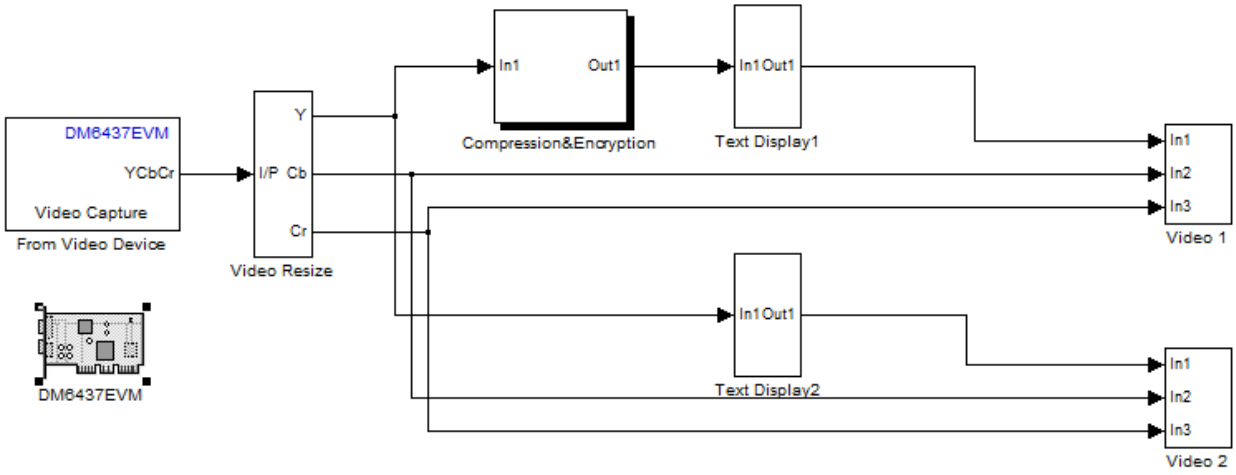

Figure 14: Real-time implementation of Video Compression and Encryption.

Input blocks:

1. Go to the Simulation → Configuration Parameters, change the solver options type back to "Fixed step", and the solver to "discrete (no continuous states)."
2. Start with your Simulink Model for video compression and encryption and delete all the input and output blocks.
3. Add DM6437 EVM block under Target Support Package → Supported Processors → TI C6000 → Board Support.

4. Add Video Capture block from Target Support Package → Supported Processors → TI C6000 → Board Support → DM6437. Set Sample Time to "-1."

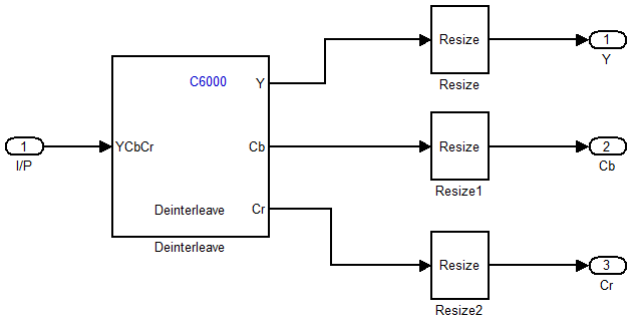

Figure 15: Video Resize subsystem.

5. Next we create the Video Resize subsystem as shown in Figure 15. After adding all the blocks shown in Figure 15, we have to select all of them and right click and select Create Subsystem option. Later rename the subsystem to "Video Resize."

Image-scaling operation is one of the most commonly used video and imaging processing functions. Resizing the raw video reduces the computing load on the processor. If we resize the video by ½ in both X and Y directions, then the load on the processor decreases by ¼. To give an idea of the massive data to be processed per second, please refer back to Page 2 to see that VGA video contains 27 million pixels per second. The DM6437 board actually has a separate hardware resizer module which can be used to offload the resizing, leaving the processor free for other tasks [4].

Tradeoff tip: The camera and the TV used in this lab have a resolution of 720x480 pixels. In the final display on TV (720x480) we would show two windows: first window (352x256) displaying the compressed video and the second window (352x256) displaying the original video. It would be a waste of precious computing power if we process the full size video and resize it to ½ in the end. Actually to decrease the load on the processor we would resize the input video to ¼ and do the processing on it and finally zoom it by a factor of 2 without any noticeable decrease in quality of picture. The board only allows the dimensions of video to be in multiples of 32 to take advantage of the 32bit internal memory bus. The actual dimensions would be given in the steps below.

a) Add the Deinterleave block from Target Support Package → Supported Processors → TI C6000 → Board Support → DM6437. We need this as we would be processing the Luminance component(Y) of input video (YCbCr).

b) Add three Resize blocks from Video and Image Processing Blockset → Geometric Transforms and attach them to the Y, Cb, Cr outputs from the Deinterleave block added before. For the Y component resizing block set the Specify parameter to "Number of output rows and columns" and Number of output rows and columns to "[176 128]." Set the Interpolation method to "Bilinear." Uncheck Perform antialiasing and Enable ROI processing parameters. For the Cb, Cr resizing blocks set the Specify parameter to "Number of output rows and columns" and Number of output rows and columns to "[88 128]." Set the Interpolation method to "Nearest neighbor." Uncheck Perform antialiasing and Enable ROI processing parameters.

Tradeoff tip: Please note that we selected Nearest neighbor interpolation method for Cb, Cr components as it is faster. Nearest neighbor method is less accurate but it is fine as we do not process Cb, Cr components in this lab. The Y component is interpolated using the more accurate but slower Bilinear interpolation method as we do processing on it in this lab.

Processing blocks:

6. Next we create the Compression & Encryption subsystem as shown in Figure 16. After adding all the blocks shown in Figure 16, we have to select all of them and right click and select Create Subsystem option. Then rename the subsystem to "Compression & Encryption."

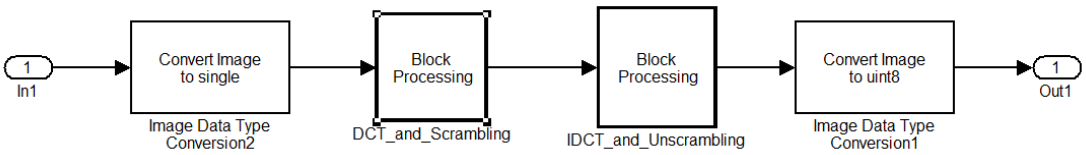

Figure 16: Compression & Encryption subsystem.

a) First, we need to convert the image data to "Single." Add Image Data Type Conversion block from Video and Image Processing Blockset → Conversions. Set the Output data type to "Single." The raw data coming from the camera is of unsigned integer (uint) type but we need to convert it to single for accuracy while computing DCT co-efficients.

Optimization tip: Please do not select double data as it would double the memory and computing requirements without any added advantage. Remember, double the memory per each of the 27 million pixels!

b) At the end after all processing has been done, we need to convert the data type back to uint as it takes less memory space than single. Add Image Data Type Conversion block from Video and Image Processing Blockset → Conversions. Set the Output data type to "uint."

c) Next we create the DCT_and_Scrambling subsystem as shown in Figure 17. To create a place holder for this subsystem in the main Simulink file, we add a Block Processing block from Video and Image Processing Blockset → Utilities. In the parameters for the Block Processing block, set the Number of inputs to "1" and the Number of outputs to "1." Set the Block size to "{[4 4]}" and Overlap to "{[0 0]}." Set the Traverse order to "Row-wise." Now click on the Open Subsystem button and then create the Simulink model shown in Figure 17. Please note that here we chose a 4x4 block of pixels to compute the DCT, whereas in Simulation we chose an 8x8 block of pixels.

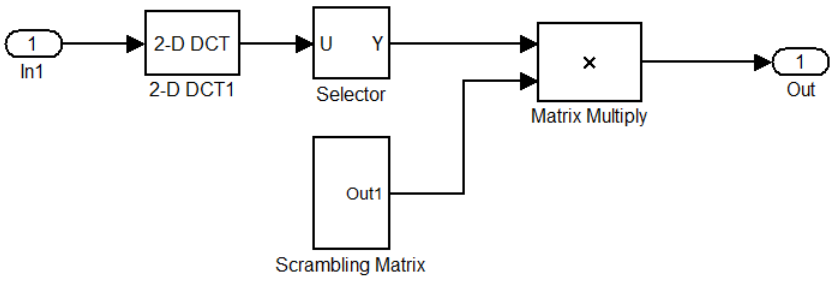

Figure 17: DCT and Scrambling subsystem

- Add the 2-D DCT block from Video and Image Processing Blockset → Transforms. Do not change any of the parameters in this block and leave them at their default values.

- Add the Selector block from Simulink → Signal Routing. Set the Number of input dimensions to "2" and Index mode to "Zero-based." Also set the Index option to "Starting index (dialog) and Index to "0." Finally set the Output Size to "3." Thus, we are going to choose a 3x3 block to transmit from the 4x4 block of DCT co-efficients computed, giving us a compression of 25%.
- Next we have to create a scrambling matrix to multiply the selected block of 3x3 DCT co-efficients as shown in Figure 18. Add two Constant blocks from Simulink → Commonly used blocks and set their values to "1" and "-1." Make sure to set the Output data type to "Single." Add Create 3x3 matrix block from Aerospace Blockset → Utilities → Math operations.
- Add Matrix Multiply block from Signal Processing Blockset → Math Functions → Matrices and Linear Algebra → Matrix Operations. Set Number of inputs to "2" and Multiplication to "Element wise." Set the Output data type to "Single."

  Optimization tip: Instead of writing our code in an Embedded MATLAB function block as we did in Simulation, we directly multiplied the 3x3 block with a Scrambling matrix. The reason is this would easier for the Simulink compiler to optimize as we are using predefined blocks.

  Optimization tip: We could greatly optimize the matrix multiplication if we use the Matrix Multiply blocks given in Target Support Package → Supported Processors → Texas Instruments C6000 → Optimized Blocks → C64X DSP Library. Please note that the matrix dimensions should be a multiple of 32 to take advantage of the 32bit internal memory bus. This optimization is left to the student to be implemented.

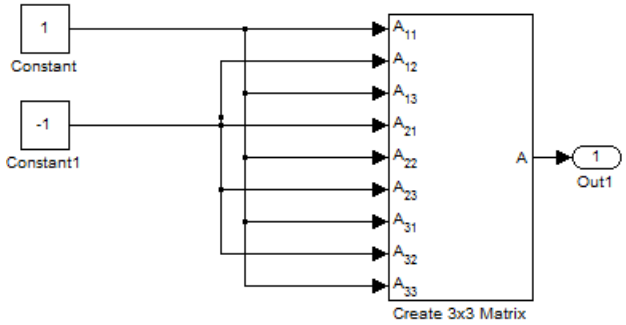

Figure 18: Scrambling matrix subsystem

d) Next we create the IDCT_and_Unscrambling subsystem as shown in Figure 19. To create a place holder for this subsystem in the main Simulink file, we add a Block Processing block from Video and Image Processing Blockset → Utilities. In the parameters for the Block Processing block, set the Number of inputs to "1" and the Number of outputs to "1." Set the Block size to "{[3 3]}" and Overlap to "{[0 0]}." Set the Traverse order to "Row-wise." Now click on the Open subsystem button and then add the blocks as shown in Figure 19.

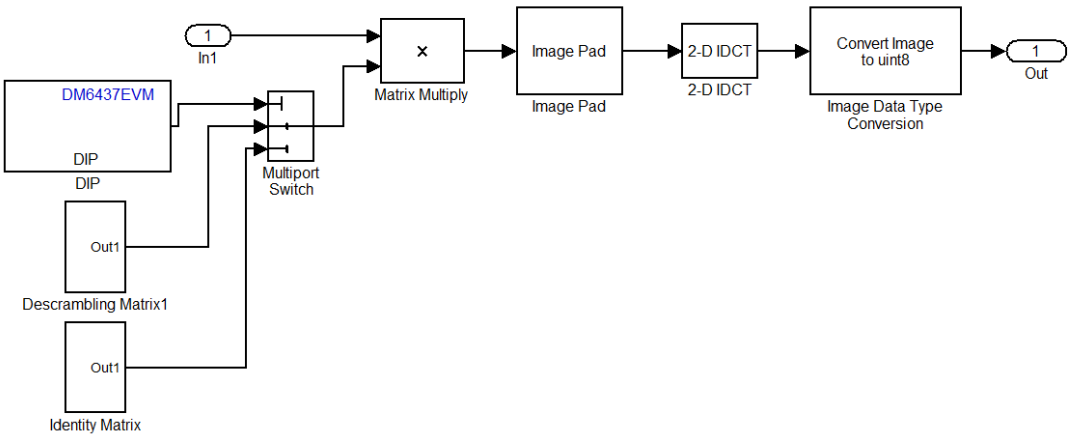

Figure 19: IDCT and Unscrambling subsystem.

- Add DIP block from Target Support Package → Supported Processors → Texas Instruments C6000 → Board Support. Set DIP switch to "SW4 (0)". We use the DIP switch to switch our output display between CompressedVideo and ScrambledVideo.
- As shown in Figure 18, we create a Descrambling Matrix block. Please note that both Scrambling Matrix block discussed in Figure 17 and this block are the same. Add two "Constant" blocks from Simulink → Commonly used blocks and set their values to "1" and "-1." Make sure to set the Output data type to "Single." Add Create 3x3 matrix block from Aerospace Blockset → Utilities → Math operations.
- Repeat the above step to create an Identity Matrix block. Here we use only one constant with value set to "1".
- Add a Multiport Switch block from Simulink → Commonly used blocks. Set Number of Inputs to "2" and Output data type to "Single."
- Add Matrix Multiply block from Signal Processing Blockset → Math Functions → Matrices and Linear Algebra → Matrix Operations. Set Number of inputs to "2" and Multiplication to "Element wise." Set the Output data type to "Single."
- Add the Image Pad block from Video and Image Processing Blockset → Utilities. Set the Method to "Constant" and Pad value source to "Specify via dialog". Set the Pad value to "0" and Specify to "Pad size". As we would be padding the 3x3 block to the right and bottom we set Pad rows at to "Right" and Pad columns at to "Bottom". We set the Pad size along rows and Pad size along columns to "1".
- Add the 2-D IDCT block from Video and Image Processing Blockset → Transforms. Do not change any of the parameters in this block and leave them at their default values. Now we have computed the IDCT on a 4x4 block of pixels
- At the end after all processing has been done, we need to convert the data type back to uint as it takes less memory space than single. Add Image Data Type Conversion block from Video and Image Processing Blockset → Conversions. Set the Output data type to "uint.

Optimization tip: Please note that the DIP switch selects either the Descrambling matrix or the Identity matrix in the multiplication with the 3x3 block of scrambled DCT co-efficients. We intentionally designed in such a way that the operations done in both cases (switch is on or off) are very similar i.e., in both cases we multiply the scrambled co-efficients with a matrix as it would help the compiler in optimizing the underlying code.

Output blocks:

7. We need to add text displaying either "CompressedVideo" or "ScrambledVideo" on the screen as shown in Figure 23. We create this subsystem by adding the following blocks shown in Figure 21.

   a) Add DIP block from Target Support Package → Supported Processors → Texas Instruments C6000 → Board Support. Set DIP switch to "SW4(0)." We use the DIP switch to switch our output display text between "Compressed Video" and "Scrambled Video."

   b) Add a Multiport Switch block from Simulink → Commonly used blocks. Set Number of Inputs to "2" and Output data type to "unit."

   c) Add two Constant blocks from Simulink → Commonly used blocks and check their Interpret vector parameters as 1-D. Set the first constant block's Constant Value to "[67 111 109 112 114 101 115 115 101 100 86 105 100 101 111]" which is the ASCII equivalent for "CompressedVideo." Set the second block's "Constant Value" to "[83 99 114 97 109 98 108 101 100 86 105 100 101 111 0]" which is the ASCII equivalent for "ScrambledVideo."

   d) Add the Insert Text block from Video and Image Processing Blockset → Text & Graphics. Set the Text parameter to '%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c', please note that there are 15 '%c' characters above which correspond to a string of length 15. Set Color value source: to "Specify via dialog" and Color value to "[0 255 255]." Set Location source to "Specific via dialog" and Location [row column] to "[10 50]". Set Opacity source to "Specify via dialog" and Opacity to "1." Set Image signal to "One multidimensional signal" and check the Input image is transposed (data order is row major). Set Font size (points) to "10."

      Optimization tip: First, we look at an **inefficient design** as shown in Figure 20. The length of "CompressedVideo" string is 15 and the length of "ScrambledVideo" string is 14. The Insert Text1 block needs to have the Text parameter set exactly to 15 "%c" characters. The Insert Text2 blocks needs to have the Text parameter set exactly to 14 "%c" characters. Now when the program is running on the board, it checks if the DIP switch is on or off. If it is on then it inserts "CompressedVideo" string on each frame of the video. If it is off then it inserts "ScrambledVideo" string on each frame of the video. So every time a new frame comes from the camera this "if…else" block would have to flush the pipeline causing a heavy penalty.

      The **efficient implementation** is shown in Figure 21. Now when the program is running on the board, it checks if the DIP switch is on or off. Text1 string will be either "CompressedVideo" or "ScrambledVideo" based on if the switch is on or off. Here also there is a pipeline flush but the penalty would be lower compared to the implementation above as we are switching on text and not on the full frame of data. Please note that to accommodate both the strings ("CompressedVideo" and "ScrambledVideo") we have set the Text parameter of Insert Text block to 15 "%c" characters. We append a null to "ScrambledVideo" string to make its length equal to15.

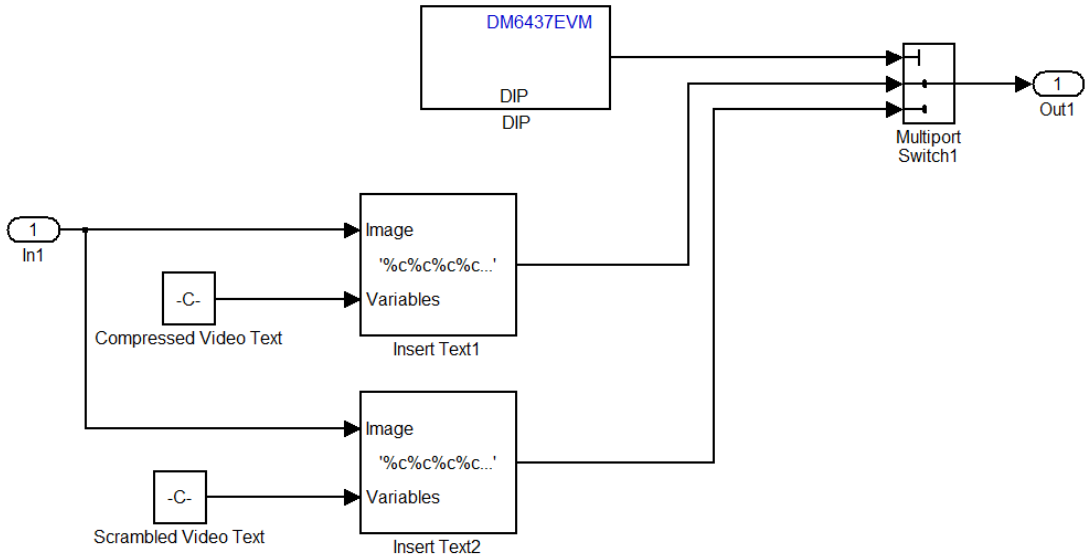

Figure 20: Inefficient implementation of Text Subsystem.

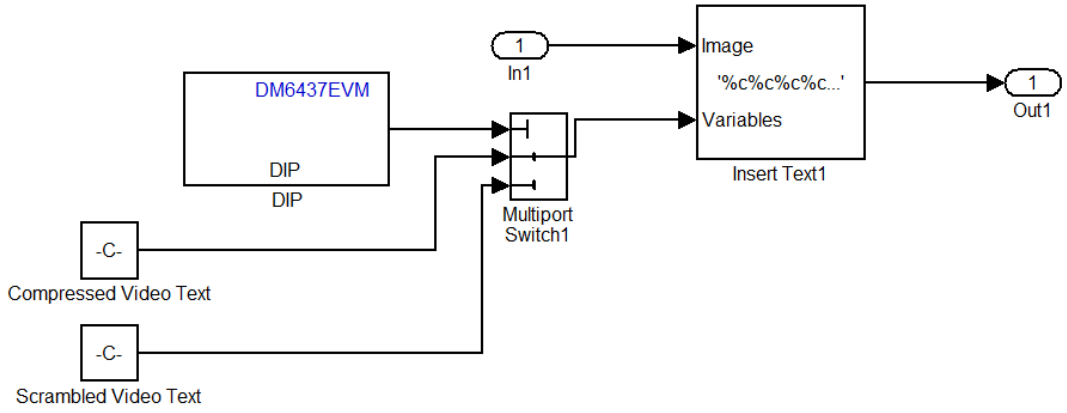

Figure 21: Optimized implementation of Text Subsystem.

8. Repeat the above step (Step 7) and make relevant changes to add "OriginalVideo" string in the other the window.
9. Add two Video Display blocks. For the first one, set Video Window to "Video 0", Video Window Position to [0 0 704 256], Horizontal Zoom to 2x, Vertical Zoom to 2x; for the second one, Set Video Window to "Video 1", Video Window Position to [352 0 352 256], Horizontal Zoom to 2x, Vertical Zoom to 2x. Add two Interleave blocks from DM6437 EVM Board Support and link the two Interleave blocks each with a Video Display block as shown in Figure 22. Link Cb and Cr channels between Video Resize block, Text display blocks and Video Display blocks as shown in Figure 14. Connect the output of the Text display blocks to Video Display blocks as shown in Figure 14.
10. Connect the video output (the yellow port) of the camera with the video input of the board and output of the board with the monitor.

11. Go to Tools → Real Time Workshop → Build the model.

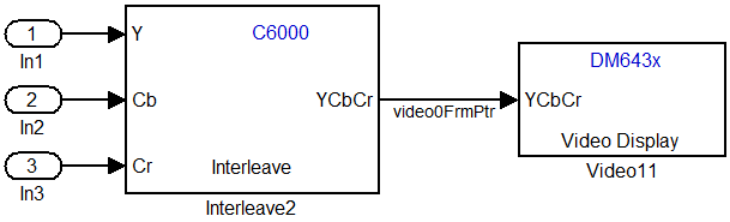

Figure 22: Video display subsystem

Results:

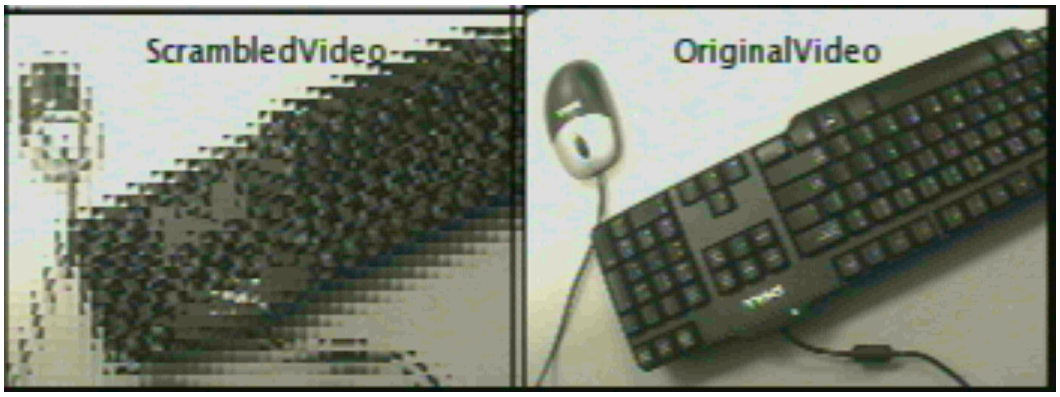

Figure 23 (a): Screenshot of TV with Scrambled Video & Original Video windows

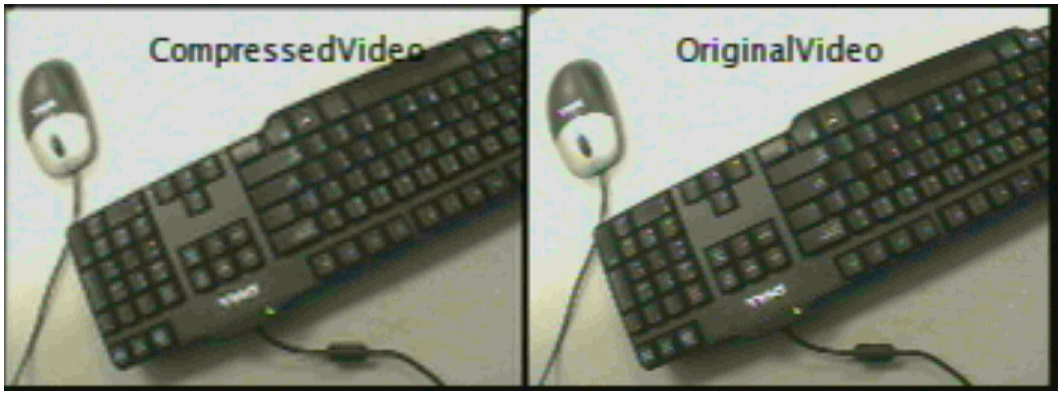

Figure 23 (b): Screenshot of TV with Compressed Video & Original Video windows

References

[1] A. C. Bovik, The Essential Guide to Video Processing, Elsevier Inc, 2009: 1

[2] N. Kehtarnavaz and M. Gamadia, Real-Time Image and Video Processing From Research to Reality, 1st ed. Morgan & Claypool, 2006: ix

[3] D. Kundur, Texas A&M University, "ECEN 448 Real-Time Digital Signal Processing." Last modified Dec 11, 2011. Accessed Jan 02, 2011. http://www.ece.tamu.edu/~deepa/ecen448/.

[4] X. Fu, Texas Instrumentns, "Understanding the DaVinci Resizer", Last modified July 2008, Accessed Jan 02,2011.

http://www.ti.com/lit/an/spraai7b/spraai7b.pdf