JPEG DECODER IMPLEMENTATION ON APX DSP PLATFORM

Miroslava Drazic 1, Dragana Todorovic 1, Aleksandar Zivkovic 1, Nemanja Popov 1, Zeljko Lukac 2
1MicronasNIT, Novi Sad,
2University of Novi Sad, Faculty of Engineering

I INTRODUCTION
JPEG is a standard image compression mechanism. JPEG stands for Joint Photographic Experts Group – the committee that wrote the standard [1]. JPEG is designed for compressing either full-colour or grey-scale images of natural scenes.

This paper represents the implementation of JPEG decoder on APX (Advanced Processor with eXpandable architecture) DSP (Digital Signal Processing) processor, developed by the company Micronas [2], with minimal resources consumption. The goal was to reorganize the relevant parts of code, to implement the nonexistent functionality of the decoder and to achieve real time decoding on APX processor. The structure of whole JPEG decoder will be presented shortly, and in more detail the parts of the decoder that were interesting for this implementation.

II JPEG ALGORITHM
JPEG can achieve very large compression ratios. The compressed image can be as much as one hundred times smaller then the original image. The lower the level of detail and the fewer abrupt colour or tonal transitions, the more efficient the JPEG algorithm becomes. This is possible because JPEG algorithm discards irrelevant data as it compresses the image, and it is thus called “lossy” compression technique. This means that once an image is compressed using JPEG compression, data is lost and cannot be recovered from the compressed image.

The operation of the JPEG algorithm can be divided into three basic stages:
• The removal of the data redundancy by means of the DCT.
• The quantization of the DCT coefficients, using weighting functions optimized for the human visual system.
• The encoding of the data to minimize the entropy of the quantized DCT coefficients. The entropy encoding is done with a Huffman variable-word-length encoder.

Although colour conversion is a part of the redundancy removal process, it is not part of the JPEG algorithm. It is the goal of JPEG to be independent of the colour space. JPEG handles colours as separate components. Therefore, it can be used to compress data from different colour spaces such as RGB, YCbCr, and CMYK. However, the best compression results are achieved if the colour components are independent (not correlated), such as in YCbCr, where most of the information is concentrated in the luminance (luma) component and less in the chrominance (chroma) component. Another advantage of using the YCbCr colour space comes from reducing the spatial resolution of the Cb and Cr chrominance components. Because chrominance does not need to be specified as frequently as luminance, every other Cb element and every other Cr element can be discarded. Therefore the JPEG algorithm usually first transforms the image from RGB to luminance/chrominance (Y-Cb-Cr) colour space and then down samples the colour components and leaves the brightness component alone.

III PROJECT REQUIREMENTS
Starting point for the project was a stand alone JPEG library (jpeglib) with built in resize options, designed to work on PC platform. The main goal of the project was to modify the library so it could:
- satisfy requested functionality,
- be easily ported to different platforms,
- enable program code modularity,
- have low memory footprint (code and data),
- decode images in real time.

Requested functionality implies that JPEG decoder library should support baseline and progressive JPEG decoding. Supported source formats should be encoded/compressed RGB and encoded/compressed YCbCr (also called YUV). YUV colour space is usually subsampled to different formats like: 4:2:2, 4:2:0 and 4:1:1. If YUV colour space is subsampled decoder assumes that subsampled YUV is always centered pixel positioning (format 4:4:4 is not...
The JPEG decoder library should also support JFIF, EXIF, PICT/JPEG and TIFF input file format. It could choose between decoding the whole picture or the thumbnail, if the thumbnail exists and is large enough. Output formats which should be supported are:

- RGB888
- RGB565
- YUV444 – interleaved
- YUV444 – planar
- YUV422 – planar
- YUV422 – interleaved UYVY
- YUV422 – interleaved YUYV
- YUV420 – planar
- YONLY

Library should also support zoom option. In this case input picture positioning should be used. Only MCUs (Minimal Coded Unit) that are needed should be decoded and MCUs that are not necessary should be skipped. JPEG library should not support:

- Hierarchical storage,
- Lossless JPEG,
- Arithmetic entropy coding,
- DNL marker,
- No integral subsampling ratios.

To be portable to APX platform, library should meet the requirements which concern the specified APplication Interface (API) and memory limitations. JPEG library API exports functions to access the JPEG structure (see figure 2). These function calls are used to communicate between application and JPEG library:

- JpegGetImageWidth - returns the width of the image being decoded (full size or thumbnail) in pixels.
- JpegGetImageHeight - returns the height of the image being decoded (full size or thumbnail) in pixels.
- JpegSetTargetSize - sets target size for picture to be decoded.
- JpegSetTargetPosition - sets target position for picture to be decoded.
- JpegSetTargetColor - sets target colour format for output picture.
- DecodePicture - decodes the whole image.
- DecodeSlice - decodes one slice (in baseline JPEG images it is same as image).
- DecodeScan - decodes one slice only and turns back the control to the top level application.
- FindHeader - searches the bit stream for JPEG sync word.
- HeaderParser - Parses the JPEG header and extracts the image information.
- WriteToOutput - packs decoded and scaled data in desired output format.

The starting point for the implementation of TIFF Reader module libtiff, a library that supports manipulation of TIFF image files both contributed by IJG (Independent JPEG Group).

In order to fulfil the low data memory consumption JPEG decoder and built in resizer should work on slices. Resizer should deliver scaled slices to function WriteToOutput which should pack decoded and scaled data in desired output format.

### IV IMPLEMENTATION OVERVIEW

The major problem to the implementation of JPEG library, which will fulfil proposed requirements, was the realization of resizer functionality. As it is sad above, the resizing function should work on slices instead of on the whole decoded image, like it was in the starting library. The resizing function is realized according to two strategies – 'decode option' and 'resize option'. The decode option defines a decoding strategy for the decoding of an 8x8 block. In the normal mode the entire 8x8 block is decoded, downsize modes are 4x4 block (16/64 pixels are decoded), 2x2 block (4/64 pixels are decoded) or DC only (1/64 is decoded). The decode option will be used only if after it is performed image size is still higher or equal of the desired (scaled) size on both dimensions (width and height). Decode option size is not the same for all colour components in case of different YUV sampling factors in input image. Table 1 shows the different cases and how they are handled.

<table>
<thead>
<tr>
<th>Scale Factor</th>
<th>Y UV</th>
<th>UV</th>
<th>UV</th>
<th>UV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upscale</td>
<td></td>
<td>UV</td>
<td>UV</td>
<td>UV</td>
</tr>
<tr>
<td>1:1 &lt; 1:2</td>
<td>8x8</td>
<td>8x8</td>
<td>8x8</td>
<td>8x8</td>
</tr>
<tr>
<td>1:2 &lt; 1:4</td>
<td>8x8</td>
<td>8x8</td>
<td>8x8</td>
<td>8x8</td>
</tr>
<tr>
<td>1:4 &lt; 1:8</td>
<td>4x4</td>
<td>8x8</td>
<td>8x8</td>
<td>8x8</td>
</tr>
<tr>
<td>1:8 &lt; 1:16</td>
<td>2x2</td>
<td>4x4</td>
<td>4x4</td>
<td>4x4</td>
</tr>
<tr>
<td></td>
<td>1x1</td>
<td>2x2</td>
<td>2x2</td>
<td>4x4</td>
</tr>
</tbody>
</table>

Table 1 DCT option for different input YUV formats.

The resize option is in fact resizing the remaining picture (after rescaling with the DCT decode option) to generate exact dimensions of the final image that cannot be emitted only by decode option. Due to small memory consumption resizer is limited to emit no more than 16 lines per call. This is needed in case of upsampling when memory consumption rises rapidly according to high scale factor. This limitation had produced very large complexity of the resize function. Resizer function had to have a built-in logic to resolve different cases. For example, in cases of high scale factors, the resizer logic had to calculate how many decoded lines should be used per one resizer call in order to emit the desired lines in multiple resizer calls. In the other case when the scaling factor is very small, the resizer should wait for decoder to decode the enough number of lines for one resizer call. During all this time it should calculate the error that is accumulated and to compensate it when is possible.

In YUV colour space number of pixels of Y component is usually 2 or 4 times larger then number of pixels in UV components (YUV 4:2:2, 4:2:0 or 4:1:1 formats). Input picture positioning in case of zooming option is limited to the multiple of sampling factor (step of two pixels for 4:2:2 and 4:2:0; step of 4 pixels for 4:1:1). In a case of RGB and
YUV444 there are no limits for input picture positioning (picture can start at any sample).

JPEG Decoder library is capable to decode JPEG images stored in TIFF file format. TIFF Reader module parses an input TIFF file and manages the decoding process of JPEG image stored inside the TIFF file. TIFF Reader module is integrated into JPEG Decoder library. Its functionality is not exported to the library client. The client uses library API in the same manner as if the image had JFIF file format. TIFF Reader conforms to TIFF 6.0 specification. TIFF Reader supports JPEG compression scheme 6 and 7. In general case, JPEG image data stream in a TIFF file format is organized in tiles. A tile is an image block that can be decoded as a standalone JPEG image. If a tile width equals to the image width it is called a strip. TIFF Reader supports only TIFF images with data organized in stripes. HeaderParser module is adapted to get relevant JPEG information when an input file is in TIFF format. Therefore an input file is not being searched for sync word, but JPEG information is rather read directly from the TIFF directory structure. The decoding process starts by decoding a first slice of an input image. In case of a TIFF, TIFF Reader sets appropriate values to the decoder main decompressions object (e.g. pointers to JPEG tables) and positions input data stream pointer to the first image strip. The decompression process is continued from DecodeJPEG module. If the next slice to decode is a part of another strip, the DecodeJPEG module is reset, as well as the main decompression object and the appropriate data stream pointer is set to the beginning of the next strip. Therefore, an input image is decoded in cycles, repeating the decoding process for each strip. DecodeJPEG module reuses the same memory space for each strip. Library outputs the raw data which can be transferred to hardware display or stored in a file.

JPEG decoder library supports maximum DCT size of 8x8 pixels, maximum sampling factor 4, maximum MCU height is 16 and data precision is 8-bits.

V RESULTS
Whole size of data RAM needed for decoding is size of decoder instance. Decoder instance size changes according to coding mode of input picture (progressive or non-progressive). DATA RAM overview is given in table 2.

Table 2 – Memory allocation for decoder instance in baseline mode

<table>
<thead>
<tr>
<th>Input picture WidthxHeight</th>
<th>Input picture Format</th>
<th>Output picture WidthxHeight</th>
<th>Decoder instance Size[Kb]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024 x 768</td>
<td>4:4:4</td>
<td>720 x 576</td>
<td>67</td>
</tr>
<tr>
<td>1024 x 768</td>
<td>4:2:0</td>
<td>720 x 576</td>
<td>81</td>
</tr>
<tr>
<td>2048 x 1024</td>
<td>4:4:4</td>
<td>720 x 576</td>
<td>95</td>
</tr>
<tr>
<td>2048 x 1024</td>
<td>4:2:0</td>
<td>720 x 576</td>
<td>105</td>
</tr>
<tr>
<td>2560 x 2048</td>
<td>4:4:4</td>
<td>720 x 576</td>
<td>51</td>
</tr>
<tr>
<td>2560 x 2048</td>
<td>4:2:0</td>
<td>720 x 576</td>
<td>72</td>
</tr>
</tbody>
</table>

Memory consumption depends also on sampling factors for UV components of the input image because it defines an MCU size.

For testing purposes an APX simulator was used to profile the decoder performance. Results for program code memory requirements are obtained by compiling C code with APX compiler. Results for both program code and data memory requirements are shown in Table 3.

Table 3 JPEG Decoder Library Memory Consumption

<table>
<thead>
<tr>
<th>Component</th>
<th>Program code [KB]</th>
<th>Data [KB]</th>
<th>Memory allocation [KB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDK</td>
<td>10</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Wrapper to JPEG Decoder Library</td>
<td>6</td>
<td>11</td>
<td>17</td>
</tr>
<tr>
<td>JPEG decoder library</td>
<td>154</td>
<td>105/5120</td>
<td>260/5274</td>
</tr>
<tr>
<td>Total</td>
<td>170</td>
<td>117/5132</td>
<td>288/5302</td>
</tr>
</tbody>
</table>

VI CONCLUSION
In this paper it is presented one implementation of JPEG decoder for DSP APX architecture. During the implementation of the decoder low memory consumption was important task and it is achieved regarding program code and data memory reduction. Library API enabled easily porting to MDED platform with MIPS processor, used in digital television. For further improvements reduction of processor time (speed-up) will be needed. Therefore, optimization of C code functions as well as writing the most demanding functions in APX assembler is necessary.

REFERENCES


Abstract: This paper presents JPEG decoder implementation on APX DSP platform. Main target was to realise real time decoder implementation with maximal capabilities and with minimal memory and processor (MIPS) requirements. Paper presents short overview of an entire JPEG decoder capabilities, but essential parts during implementation were explained in more details.
JPEG DECODER IMPLEMENTATION ON APX DSP
PLATFORM, Miroslava Drazic, Dragana Todorovic,
Aleksandar Zivkovic, Nemanja Popov, Zeljko Lukac.