A Hardware Implementation of Real-Time Video Deblocking Using Shifted Thresholding

Martin Hansen
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada
Email: mdbhanse@uwaterloo.ca

Alexander Wong
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada
Email: a28wong@uwaterloo.ca

William Bishop
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada
Email: wdbishop@uwaterloo.ca

***Abstract*— Video compression has become very important as demand has increased for the storage and transmission of digital video content. Popular video compression schemes like MPEG encoding make use of block-transform coding techniques which are susceptible to blocking artifacts. Recently, an efficient deblocking algorithm based on the concept of shifted thresholding has been proposed. This algorithm uses only integer arithmetic and replaces division operations with bit shifting. This paper proposes a new hardware architecture for the implementation of video deblocking using shifted thresholding. A prototype system for high performance video deblocking using a FPGA (field programmable gate array) board is described. The prototype system leverages the reduced hardware complexity of the shifted thresholding algorithm to cost-effectively implement video deblocking on a FPGA board.**

I. Introduction

With the ever increasing need for efficient storage and transmission of digital video content, video compression has become an active area of research. Video compression is essential for applications ranging from high definition video broadcasting to the wireless transmission of video content to portable entertainment systems. Popular video compression schemes such as MPEG [1] and recent video compression schemes such as H.264/AVC [2] make use of block-transform coding, where blocks of pixels are processed independently to reduce computational and storage requirements.

A significant drawback of block-transform video coding is that blocking artifacts are introduced at block boundaries. These artifacts noticeably degrade video quality, particularly if the video content is compressed at a high compression rate. To improve video quality, a process known as video deblocking is used to reduce the impact of blocking artifacts.

A large number of video and image deblocking methods have been introduced. These methods have been categorized [3] as follows:

1) Projections onto convex sets (POCS) methods,
2) Spatial block boundary filtering methods,
3) Wavelet filtering methods,
4) Statistical modeling methods,
5) Constrained optimization methods, and
6) Shifted transform methods.

Traditionally, methods based on spatial block boundary filtering have been used in real-time video decoding due to their low computational complexity. However, interest has grown recently into the use of shifted transform methods. Such methods typically deliver improved deblocking quality. However, the computational complexity of shifted transform methods have traditionally been very high due to the need for a large number of floating-point calculations.

Recently, an efficient deblocking algorithm based on the concept of shifted thresholding was proposed [3]. This algorithm uses only a fraction of the computations required by traditional shifted transform methods. Furthermore, it requires only integer computations and uses bit shifting to replace division operations. Despite these simplifications the algorithm still achieves image quality that is competitive with other methods in its class. The algorithm is also ideal for implementation using inexpensive hardware.

This paper presents an efficient hardware architecture for video deblocking using shifted thresholding. The proposed architecture is described and explained in detail in Section 2. The hardware complexity of the proposed architecture is analyzed in Section 3. A prototype design is presented in Section 4 along with experimental results. Conclusions are drawn in Section 5.

II. Proposed Architecture

The proposed hardware architecture implements a shifted thresholding algorithm for video deblocking [3] on an Altera DE2 board [4]. The shifted thresholding algorithm transforms an initial decompressed image into a deblocked, decompressed image. The algorithm performs six distinct operations as illustrated in Fig. 1. The proposed hardware architecture assumes greyscale bitmap images of dimensions 640×480, with each greyscale pixel represented by 8 bits. However, the architecture could be easily modified to support larger images and larger colour representations.

A 6 stage pipeline architecture was chosen for the hardware design. It should be noted that in the interest of hardware optimization, the pipeline stages deviate slightly from the 6 distinct operations described previously. The first pipeline stage loads image data from memory in blocks of 8×8 pixels,

0840-7789/07/$25.00 ©2007 IEEE

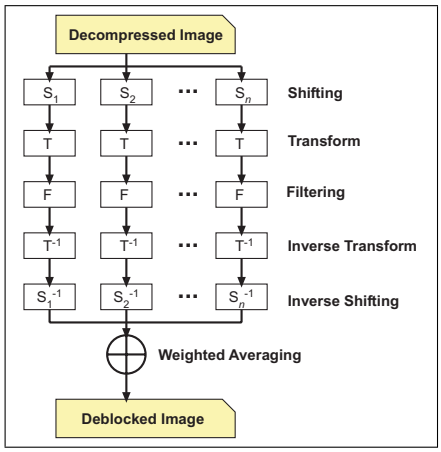

Fig. 1. Overview of shifted transform deblocking [3]

and manages the coordinate shifting that is necessary to produce the four shift patterns. These blocks are passed one at a time to the second pipeline stage which implements the two matrix multiplications required by the Discrete Cosine Transform (DCT). The third pipeline stage performs scalar multiplication, thresholding, and another multiplication. The fourth pipeline stage implements two matrix multiplications for the inverse DCT operation. Finally, the blocks are multiplied again (weighted averaging) and saved back to memory in pipeline stages five and six, respectively. The final pipeline design is shown in Fig. 2.

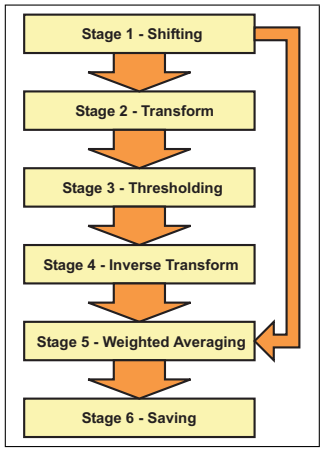

Fig. 2. Pipeline architecture for shifted transform deblocking

The strength of this deblocking architecture lies in its computational simplicity. All of the mathematical calculations are performed using integer values, eliminating the complexities of floating-point arithmetic. The architecture also avoids division operations through the use of bit shifting. The most complicated operation used is integer multiplication which can be easily pipelined and optimized to manage area and latency.

A. Stage One - Shifting Component

The first stage of the pipeline loads the pixel data from the SDRAM on the Altera DE2 board and manages the block shifting. It forms the 8×8 blocks of pixels to be manipulated by subsequent pipeline stages. Fetching each 64 pixel block requires between 64 clock cycles and 80 clock cycles, depending upon the address of the block in memory. The control logic processes the blocks row by row and processes the entire image a total of five times. The first iteration retrieves the original (unshifted) image. The following four iterations shift the image based on a set of four shift patterns: $(\Delta x, \Delta y) = (-3,-3),(-1,-1),(1,1),(3,3)$ to produce a total of four shifted images.

B. Stage Two - Transform Component

The operations necessary to accomplish the DCT [5] are two matrix multiplications followed by a scalar multiplication and a right shift of 18 bits [3]. This can be expressed mathematically as follows:

$$\mathrm{Y} = (\mathrm{PXP}^{\mathrm{T}}) \otimes (\mathrm{E} \otimes \mathrm{E}^{\mathrm{T}}) >> 18 \qquad (1)$$

where X is the input block, P is the 8×8 integer transform matrix, and E is a constant matrix used for scaling. The first of these operations, the double matrix multiplication $(\mathrm{PXP}^{\mathrm{T}})$, is implemented in the second pipeline stage with the remainder of the DCT performed in the third pipeline stage. The double matrix multiplication can be simplified for the purpose of hardware design using the following mathematical property $(\mathrm{AB})^{\mathrm{T}} = \mathrm{B}^{\mathrm{T}}\mathrm{A}^{\mathrm{T}}$ as follows:

$$\mathrm{PXP}^{\mathrm{T}} = (\mathrm{PX})\mathrm{P}^{\mathrm{T}} = (\mathrm{X}^{\mathrm{T}}\mathrm{P}^{\mathrm{T}})^{\mathrm{T}}\mathrm{P}^{\mathrm{T}} = \Phi\left[\Phi\left[\mathrm{X}\right]\right] \qquad (2)$$

where $\Phi\left[\mathrm{X}\right] \equiv \mathrm{X}^{\mathrm{T}}\mathrm{P}^{\mathrm{T}}$. Matrix transposition is a zero-cost operation which can be accomplished using routing resources in hardware. Therefore, equation 2 is simpler in the sense that, with no additional hardware resources, the calculation can be implemented as the same operation performed twice. This simplification reduces the amount of physical hardware required to implement the calculation. However, the time required for the operation increases since less parallelism is possible. It should be noted that the pipeline can be designed to accomodate this latency without any significant effect upon the overall system.

Due to FPGA logic cell constraints, each matrix multiplication is split into two computations. First, the upper half of the result is computed followed by the lower half. Again, the latency can be easily absorbed into the pipeline, which must allow for slow memory access times. Each operation requires 1 clock cycle per column, and 2 additional clock cycles for overhead, resulting in 40 clock cycles for the entire second pipeline stage.

C. Stage Three - Thresholding Component

The third pipeline stage begins by computing the remainder of the DCT operation using scalar multiplication and right bit

shift. This is followed by a thresholding operation. The thresholding required is a simple comparison operation. Following the thresholding is another scalar multiplication, this time in preparation for the inverse DCT that follows in stage four of the pipeline.

Both scalar multiplications required are identical, and therefore the same hardware can be used for both. The calculations are implemented using on-chip multiplication units, which run asynchonously. The thresholding is accomplished using a comparison operation between each matrix value and its corresponding threshold value, stored in a ROM within the on-chip M4K memory blocks. Because of the slow memory access time to fetch these values, 1 clock cycle is required per pixel. With 5 clock cycles per multiplication and with the shift operation being a zero-cost operation, this pipeline stage requires a total 74 clock cycles.

D. Stage Four - Inverse Transform Component

The inverse DCT in the fourth pipeline stage is performed identically to the operations in the second pipeline stage. With the scalar multiplication already completed by the third pipeline stage, the two matrix multiplications are performed again, now using the inverse transform matrix $\mathrm{Q} = \mathrm{P}^{\mathrm{T}}$, in place of P. This pipeline stage takes 40 clock cycles, including a right shift of 18 bits at the end of the stage, which completes the transform requirements of the algorithm.

E. Stage Five - Weighted Averaging Component

With the main matrix manipulations complete, the fifth pipeline stage assigns a weight to each pixel within the block based on pixel position. This is done using an on-chip M4K ROM lookup table to reduce usage of FPGA logic cells. Each pixel is fetched from the table based on its 8 bit value, its position (which maps to one of eight different scaling factors), and its block type (non-shifted or shifted). Therefore, the ROM requires $2^{(8+3+1)} = 4096$ bytes.

In addition, any shifted blocks must have this shifting taken into account since the pixel locations are relative to the initial position of the pixel in the image and not the location within the shifted version. Prior to the weighting process, the signed pixel data must also be truncated, since calculation errors can result in pixel values less than 0 or greater than 255. Fetching from the ROM table requires 1 clock cycle per pixel. Adding in the 4 clock cycle overhead results in a total time of 68 clock cycles for this pipeline stage.

F. Stage Six - Saving Component

In the sixth pipeline stage, the blocks are weighted again. Non-shifted blocks are given a relative weight of 4 (a shift left by 2 bits) and the 4 shift patterns are each given a weight of 1 (no shift) for a combined weight of 8 (which is then normalized by a shift right of 3 bits). Now the pixel values are added to those already in memory at their appropriate locations. This process involves a memory fetch, an addition, and a memory write operation. The minimum time required for the sixth pipeline stage is between 32 clock cycles and 80 clock cycles.

G. Interfaces

The off-chip resources currently used for this project are an SDRAM (for the initial image) and an SRAM (for the final image). Additionally, the FPGA itself (discussed in Section 3) has internal M4K memory blocks, which are used to implement ROM modules. The hardware development platform used contains various other useful components, such as a Universal Serial Bus (USB) port, an Ethernet port, and a Video Graphics Adaptor (VGA) port. In the future, additional logic could be added to transfer the input and output images using these components.

Both the SRAM and on-chip ROM interfaces are quite simple and consist of basic control signalling and obvious timing relationships. The SDRAM is controlled by a finite state machine (FSM) which manages initialization, row selection, precharging, and the other operations required by this memory device. A custom interface was made to implement this FSM for the purpose of translating simple load commands into the complex sequence required by the SDRAM.

H. Memory Usage

The SDRAM and SRAM external memories are both used in this design. Both are required to minimize the bottleneck created by memory access. Both devices are 16-bit memory devices that can store 2 image pixels at each physical address. This width is beneficial for throughput, but introduces additional complexity for accessing the shifted blocks, since the shifts are all by an odd number of pixels and thus result in pixel accesses being out of line with the boundaries of each memory word. Furthermore, this misalignment also crosses row boundaries in the SDRAM, requiring additional clock cycles for fetching data in these cases.

The initial decompressed image is stored on the SDRAM as a 640×480 bitmap, with one byte per greyscale pixel. This size gives a total of 307 200 bytes to be processed. Obviously this value is dependent on the number of pixels in the image as well as on the resolution of each pixel. The SDRAM word access is well within the speed of the 20 ns clock provided on the board, but this memory has the added burden of precharging and other operations, all of which add to the latency. The SDRAM also requires a 3 ns clock lead from the rest of the design.

The deblocked final image is stored in the SRAM, and always constitutes the same size as the original. SRAM access is quite fast. No additional latency is introduced by the SRAM. It is possible to store two bytes (one word) per clock cycle.

III. Hardware Complexity

The system was implemented as a VHDL design on an Altera DE2 board [4]. This board includes an Altera Cyclone II FPGA as well as various on-board components, including several memories, USB port, Ethernet port, VGA port and audio connectors, and other miscellaneous devices. For the purpose of our design, the devices used on the Altera DE2 board other than the USB programming interface for the FPGA were the SDRAM, the SRAM, and the FPGA itself.

The goal of the hardware implementation was to reduce hardware complexity and latency over other similar algorithms for video deblocking. Area usage was not a primary consideration in the implementation, especially in the un-optimized design. The current area usage on the FPGA is 31 585 logic blocks, or 95% of this specific chip.

This large area usage is partly due to the amount of data being processed in parallel by the pipeline stages. Each stage concurrently manipulates a 8×8 block of pixels, with each pixel represented by a minimum of one byte (in stages one, five, and six) and a maximum of four bytes (in stages two through four). This storage requires approximately 8 kbits of on-chip memory which contributes significantly to logic block usage. This is a conservative estimate, as some stages require storage space for more than one block at a time.

The on-board clock speed used for the FPGA is 50 MHz. The design easily achieves this speed. The longest path delay for the FPGA design is rated at 15.6 ns, which results in a theoretical maximum clock speed of 64.1 MHz.

Some optimizations of this design should be undertaken to reduce area usage. Due to the upper bounds on throughput inherent when accessing memory, the deep parallelization of the design may not be necessary. The middle pipeline stages, requiring only 40 cycles to complete, are not ideal if they provide no speed benefit with the cost of added area. It is estimated that there can be area reductions of 5% obtained by further optimizing the current pipeline.

IV. Current Results and Future Work

For the purpose of testing, a set of test images have been compressed heavily using the Joint Photographic Experts Group (JPEG) standard [6] to introduce blocking artifacts. These images have been used as an artificial test suite for the deblocking architecture. The output of the hardware design has been verified against results produced by a software version of the video deblocking algorithm.

The prototype implementation uses the 50 MHz clock signal provided on the Altera DE2 board. The throughput of the hardware design is limited by the slowest pipeline stage. An event-triggered pipeline is used. The performance of the pipeline is dependent upon memory access times so throughput is not a constant. Each 64 pixel block requires between 74 clock cycles and 80 clock cycles to process. Given a 640×480 image and given that 5 iterations over the image are required, the minimum processing time is 37.8 ms (see Equation 3), allowing 26.4 frames per second (fps) (see Equation 4) to be deblocked. This rate exceeds the real-time video processing rate of 24 fps so it is possible for the prototype implementation to deblock a video stream in real-time.

$$t_{\min} = \frac{1\left(\frac{640\times480}{64}\,(74)\right) + 4\left(\frac{640\times480}{64}\,(80)\right)}{50\,000\,000} = 37.8\,\text{ms} \quad (3)$$

$$f_{\max} = \frac{1}{37.8} = 26.4\,\text{fps} \quad (4)$$

Currently, the prototype design acheives a rate of 18.3 fps. Dummy states are still present in the design for testing purposes. It is projected that the maximum speed is readily acheivable after further optimizations. However, this design is not yet fast enough for consumer applications such as HDTV that require a minimum resolution of 720×480. The prototype design is only capable of delivering 23.5 fps (see Equation 5 and Equation 6) on the Altera DE2 board. There are several avenues that can be pursued in the future to acheive 24 fps at high-definition resolutions.

$$t_{\min} = \frac{1\left(\frac{720\times480}{64}\,(74)\right) + 4\left(\frac{720\times480}{64}\,(80)\right)}{50\,000\,000} = 42.6\,\text{ms} \quad (5)$$

$$f_{\max} = \frac{1}{37.8} = 23.5\,\text{fps} \quad (6)$$

One potential solution that would allow the processing of HDTV frames would be to use more suitable external memories. Using external high-speed SRAM devices for both input and output would eliminate the bottleneck created by row selection in the input SDRAM. Also, if the output memory allowed dual-port access, the delays introduced by reading and writing in the sixth pipeline stage would be dramatically reduced. With these improvements and further optimizations to the third pipeline stage, it is projected that the design could theoretically deliver 29.8 fps at 640×480 and 26.5 fps at 720×480.

V. Conclusion

In this paper, we have introduced a high performance hardware architecture for video deblocking on a FPGA board. The hardware architecture has been designed such that it is suitable for real-time video deblocking on a low-cost FPGA device. The current prototype design delivers 18.3 fps but the design is theoretically capable of delivering 26.4 fps for video frames of size 640×480 pixels. The design is suitable for real-time video deblocking at this frame size and smaller sizes. With a slightly more expensive FPGA device, it is projected that real-time video deblocking should be possible on high-definition video frames of size 720×480 pixels. It is our belief that this design can be easily implemented in consumer-grade set-top boxes, digital movie playback devices, and other digital multimedia systems given further optimization and testing.

References

[1] ISO/IEC 13818-2, Generic coding of moving pictures and associated audio information, Part 2: Video, November 1994.

[2] Joint Video Team of ITU-T and ISO/IEC JTC 1, Draft ITU-T recommendation and final draft International Standard of Joint Video Specification (ITU-T Rec. H.264 — ISO/IEC 14496-10 AVC), May 2003.

[3] A. Wong and W. Bishop, "Efficient deblocking of block-transform compressed images and video using shifted thresholding," in *Proceedings of the Eighth IASTED International Conference on Signal and Image Processing*, August 2006.

[4] Altera Corporation, *DE2 User Manual*, Version 1.4, 2006.

[5] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Transactions on Computers*, pp. 90-93, Jan 1974.

[6] G. Wallace, "The JPEG still picture compression standard," *Communications of the ACM*, vol. 34, no. 4, pp. 30-34, 1991.