Motion Estimated Frame Interpolation

S Chetty[†] R MacArthur[‡] S Wood[§]
EEE4084F Class of 2016
University of Cape Town
South Africa
[†]CHTSHA024 [‡]MCRROS001 [§]WDXSEA003

***Abstract*—A concept design description for an implementation, within the realm of high performance computing hardware, of motion estimated frame interpolation. The resulting concept design is embedded in a capable output screen and receives input from a sub-par frame-rate HDMI video stream. A prototype is developed that demonstrates a simplified proof of concept involving frame averaging as a method of interpolation. The prototype is employed on an FPGA with a VGA output.**

I. Introduction

Frame interpolation involves generating intermediate video frames between existing ones in order to make the animation more fluid and to compensate for motion judder; ultimately producing a higher frame rate. There are many different methods that exist to do this. The simplest of them is frame repetition, which simply copies the frame. Another basic method is frame averaging, where an intermediate frame is obtained by taking the average of two consecutive frames [1].

The algorithms that produce the most visually appealing results involve estimating motion vectors in the image, calculating intermediate frames based on these motion vectors and then masking any artifacts which may have arisen in the process.

Video frame interpolation is often used for high quality video applications. High definition video processing can be resource intensive which means the algorithms involved lend themselves to an accelerated implementation. This paper describes the proposed implementation of frame interpolation algorithms on an ASIC target architecture. In addition, a prototype FPGA implementation is developed.

II. Overview

The interpolation algorithm is pipelined in nature. Firstly, areas of motion must be identified. This information is then fed through an interpolation algorithm which must process the generated motion vectors to result in a new frame. Finally, artifact masking must be applied. Stages of the pipeline are described further in the following subsections as well as in Section III.

A. Motion estimation

Motion vectors describe the position changes from one frame to another. Many methods exist to produce these motion vectors, for example the phase correlation of two frames [2]. This is where each frame is divided into blocks of $n \times n$ pixels each. The motion vectors are obtained for each block by correlating that block with the following frame. The peak value in the resulting correlation is used to calculate the motion vector. An example of this can be seen in Figure 1.

B. Frame interpolation

Interpolated frames are calculated by drawing on the results of the motion estimation by placing each block at the midpoint of it's corresponding motion vector. This method, however, could potentially produce results with artifacts in the form of "holes" (regions where there is no available image data).

C. Artifact masking

In order to produce visually appealing results the artifacts produced in the interpolation algorithm process must be removed. The artifacts considered will be image holes and block boundary discontinuities. Image holes can be repaired by filling in these pixel values from the corresponding pixel values of either the previous or next frame (or both) in the interpolation. However, this process itself may result in block discontinuities. Block discontinuities can be dealt with by use of a deblocking filter. This filter works by smoothing the edges of a block with filter co-efficients suited to the block sizes. Many other sophisticated methods of deblocking currently exist, some of which are used in existing video codecs such as H.264 and H.265.

III. Detailed Design

A full concept diagram for the product is shown in Figure 2. The following sections outline the detailed design and explanation of the arrival to this diagram. The final device is be embedded in a 120 Hz 4K-capable output screen and receive up to 4K video at 60 Hz. The pipeline described will double the frame rate of the video stream.

A. Architecture

It was decided that the device should be embedded in the screen because the latest HDMI standard (2.0b) cannot output 4K video at a frame rate greater than 60 fps [3]. Due to the resource intensive nature of the interpolation algorithm, the amount of graphics processing power needed and the fact that the device is embedded in the screen the design lends itself most heavily to an ASIC architecture. A CPU architecture cannot match the parallel processing power of an ASIC which is essential for this particular processing task. A GPU architecture struggles to compete with the power and

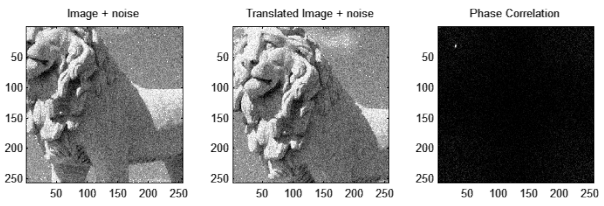

Fig. 1. Result of phase correlating two frames

space efficiency of an ASIC and does not necessarily suit the pipelined nature of the algorithm. An ASIC provides the best flexibility in terms of both parallelism and pipeline architecture styles both of which are important for motion estimated frame interpolation. An ASIC has the strength of specificity and although an it has a long design time, the algorithm is granular enough for sections to be developed separately on FPGAs.

B. Interpolation pipeline and buffering

The principle behind the utilisation of the buffers resides in the advancement of frames from the buffers on the left hand side of Figure 2 through to the right hand side. The original frames stored in the buffers will follow the concerned stage block in the pipeline. This means that the corresponding original frames need to be at the position of the concerned stage block before the stage block begins to operate on them. This poses the issue of *pipeline stall* due to the inability of the pipeline to make use of the buffers while the buffers are being advanced (relocated and refilled). This issue was mitigated by including a second row of buffers of this configuration (buffer rows A and B). The two row buffer configuration means that, at any given point, one row of buffers will be being advanced in the pipeline while the other row is being used for interpolation allowing processing of a pair of original frames to commence immediately after that of the previous pair. Together with the designed fact that row A will be one frame ahead of row B (evident in the alternate connection of the input DEMUX to the first two buffers), this arrangement also means that the frames apparent to the input side of the output MUX will be in an order convenient for the control of output frames. It is important to note that advancement of buffers involves copying any particular buffer's contents into the buffer two places down the pipeline since each individual row of buffers is used every two pipeline cycles.

Figure 2 also shows the intermediate buffers between each pipeline stage block. These are used to allow the concurrent operation of each stage in the pipeline.

C. Motion estimation

The motion vectors between two frames will be obtained by use of a two dimensional correlation algorithm. However, a naive implementation of time domain correlation for an $M \times N$ image has a computational complexity of O(M^2N^2). In order to reduce the computational costs involved, the correlation will be computed by use of the Fast Fourier Transform (FFT) which has a computational complexity of O($MN \log_2(MN)$) [4].

An estimate of the amount of floating point operations to compute the 1D FFT is given by $5N \log_2(N)$ this estimate factors in the complex multiplications and additions that need to be performed [4]. Given this, the amount of floating point operations per second that need to be performed in order to find the motion vectors for 4K 60 fps video is estimated as: $5 \cdot 5 \cdot 3840 \cdot 2160 \cdot 60 \cdot \log_2(3840 \cdot 2160) \approx 286$ GFLOPS [4]. Due to this high computational cost, a custom ASIC will have to be designed to perform the FFTs.

The algorithm to obtain the motion vectors involves taking the FFT of the first image and multiplying it with the complex conjugate of the FFT of the second image. The inverse FFT of this result will then produce the global correlation of the two images where the peaks in the image will be used to find motion vectors.

The second image is divided using quad-tree partitioning based on the correlation result. It is divided such that there is only one strong peak corresponding to its partition. Each partition with a strong peak is then correlated with the first image to determine that block's motion vector. This correlation involves unequally sized images and will therefore be performed by interpolative upsampling in the frequency domain. Only the FFT of the current block would have to be calculated as the FFT of the first frame has already been calculated in the first global correlation step. As a result, the FFT of each block would involve less computations than if upsampling were done in the time domain [5]. Figure 3 shows a function block diagram of this process. The vectors found as a result of this process are stored in the correlation buffer for access by the interpolator Block.

D. Frame interpolator

After the correlation stage of the pipeline, the resulting motion vectors are read in by the interpolator block from the correlation buffer. A diagram of this block has been omitted due to the simplistic nature of its functioning. The interpolator locates the midpoints of each vector between the position of the block in the first frame and this destination point, computed using a simple division by two of the difference between the X and Y coordinates, and writes the block image data into the interpolation buffer at this midpoint position. This is executed for all subdivisions of the first frame, concurrently, in an attempt to create a new frame which represents the midpoint of all motion between the two original frames. At the same time, a second buffer, the interpolation map, is written to with ones at positions representing pixels of the intermediate frame to formulate a map indicating the areas in the frame that have been filled by the interpolator. This map will be used by the artifact masker to fill in the gaps that are left out during the interpolation process.

E. Artifact masker

As discussed, the result of the frame interpolator contains artifacts that need to be removed. The most significant type

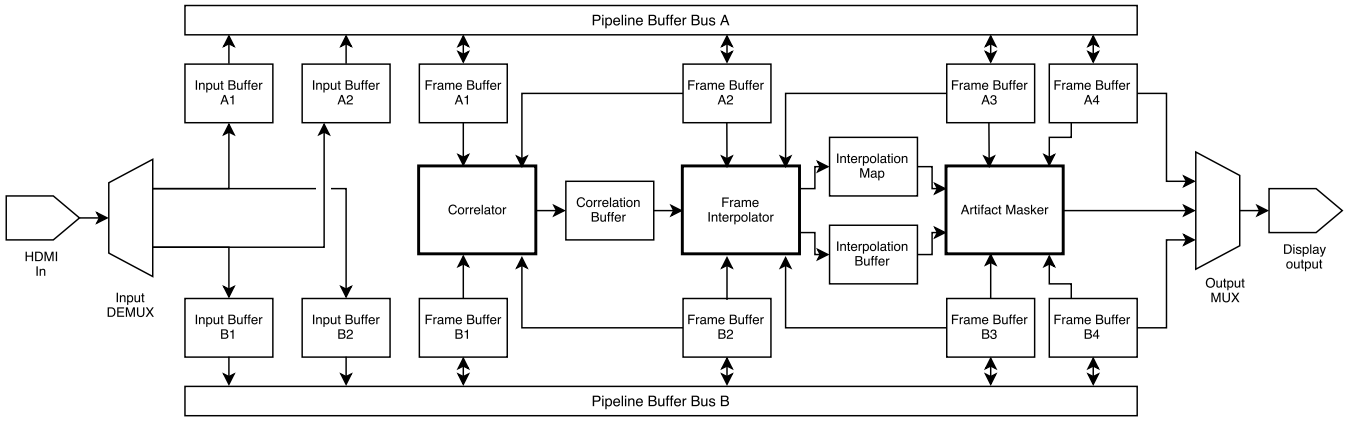

Fig. 2. High-level circuit diagram showing the entire concept design

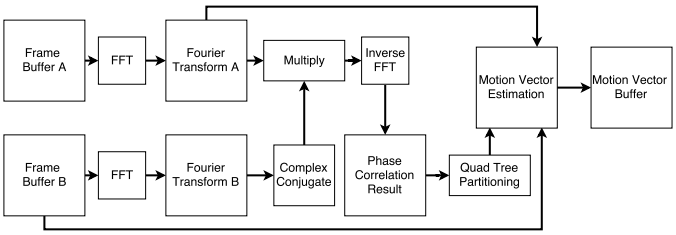

Fig. 3. Block diagram showing phase correlation motion detection

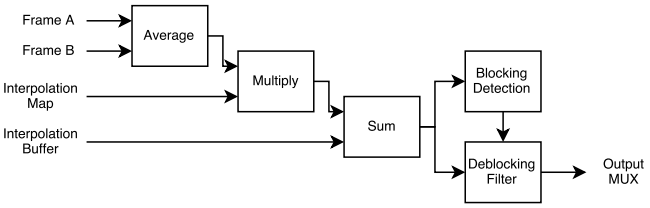

Fig. 4. Block diagram of Artifact Masker Block

of artifact to remove is the holes in the frame at which the interpolator did not fill. The artifact masker block will fill these holes by copying averaged pixels from the two original frames into the intermediate frame at the coordinates found in the interpolation map generated by the interpolator. This is achieved by summing, element-wise, the intermediate frame in its current state with the average of the two original frames multiplied by the inverse of the interpolation map. Being element-wise operations, the summation, averaging and multiplication operations can all be implemented using parallel arithmetic cores.

A significant set of block discontinuities in the frame will have resulted after the interpolation and hole filling procedures. To reduce the effect of this final set of artifacts, the frame is subjected to a deblocking filter which operates in a similar fashion to that which is part of the H.264 encoding standard [6]. The H.264 standard consists of blocks that are predefined in size and position, thus the deblocking effect is applied without knowledge of the position of the discontinuities being required. The blocks in each of the interpolated frames in our case are not consistently positioned, thus the frame needs to undergo edge detection before it is filtered. A candidate block edge detection algorithm is proposed in [7] which leverages estimated relative quantisation error calculated in the frequency domain. Once the edges have been identified, the frame is filtered to smooth the edges of the blocks.

IV. Methodology

An implemented prototype was developed and demonstrated on a Nexys 4 Artix-7 FPGA. Due to time, hardware and manpower constraints, the prototype follows a simple design.

Two frames were stored in two 512×384 block ROMs on the FPGA. These frames were then subjected to interpolation consisting of real time frame averaging. A multiplexer was used to cycle through both the stored original frames and the averaged frame and to output that particular frame to the VGA interface.

A. Block ROMs

As mentioned, the block ROM provided on the FPGA was used to store the two original frames that were to be interpolated. The Artix-7 100T has 4860 kbits of fast block RAM. We aimed to store two 512×384 frames with 12 bits of colour each and thus needed $2 \cdot 512 \cdot 384 \cdot 12 \approx 4719$ kbits of ROM. This meant we used 97% of the total available block ROM on the device.

B. Frame averaging

The simplest method of frame interpolation which was considered, as a proof of concept, was the process of frame averaging. Observe, in Figure 6, how this differs from motion estimated frame interpolation (shown in Figure 5). Frame averaging involves adding two consecutive frames, element-wise, and halving the resultant values. The results produced from this method are not ideal for applications where

image quality is important. This is because frame averaging often produces undesired artifacts in the interpolated frames. However, it was sufficient for the prototype demonstration.

The prototype performed real time frame averaging of the two original frames stored in the ROMs. Each corresponding pixel from each frame needed to be averaged element-wise. In a hardware implementation, this is done with a simple add and a single bit-wise left shift which translates to a floored division by 2. This was done in a combinatorial manner.

C. VGA interface

A single Video Graphics Array (VGA) control block was employed to coordinate the frame output and the current pixel on the screen. The VGA output consisted of 14 pins: an HS and VS pin for screen synchronisation and 12 pins for the 12 bit RGB colour. Video with a resolution of 1024×768 at a 60 Hz screen refresh rate has a 65.0 MHz pixel clock (horizontal refresh rate) and a 48.36 kHz vertical refresh rate [8]. Modern VGA monitors are "multi-sync" so they can accommodate non-standard frequencies but use of a standardised frequency is recommended. The hardware block had two counters, one for the horizontal direction and one for the vertical direction. The counters counted the addressable pixel values and appended the required front porch, sync and back porch times before repeating the cycle. This is demonstrated in Figure 7. A clock management tile, with a phase-locked loop (PLL) on the FPGA was used to generate the 65.0 MHz clock from a 100 MHz input.

The ROM blocks in the FPGA were sized such that the dimensions were powers of two and the initialised values were be padded such that the horizontal and vertical pixel value can simply be concatenated to form the address of the pixel location. The frames stored in block memory were stretched to fit the 1024×768 resolution by duplicating each pixel in both the vertical and horizontal axes.

D. Frame rotation

The output cycled between original frame A, the interpolated frame, original frame B and the interpolated frame again at 4 Hz so the result could be seen by the human eye. This is the equivalent of a 2 Hz frame rate upscaled to 4 Hz.

V. Results

A. Expected results of concept design

The stages of our concept design pipeline are easily parallelisable. For example, there are already many well established divide and conquer methods of computing the FFT in hardware [10]. The multiplications, block shifting and artifact masking are also simple to parallelise. Because of this, the conceptual design is expected to perform frame interpolation in real time. The only time delays expected to occur are those associated with the latency of the pipeline. In this case, a 3 frame latency is expected which is roughly 50 ms at a refresh rate of 60 Hz.

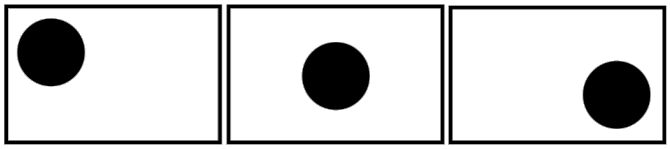

Fig. 5. Diagram showing motion estimated frame interpolation

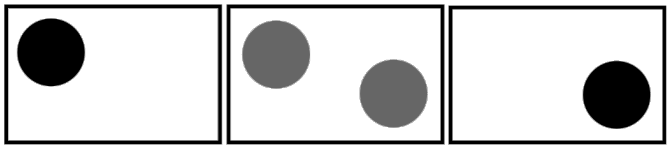

Fig. 6. Diagram showing frame averaged interpolation

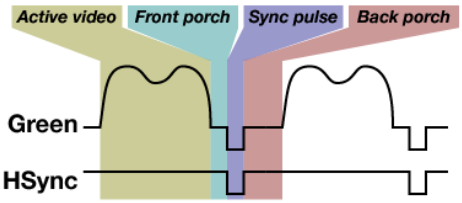

Fig. 7. Diagram for VGA timings [9]

B. Prototype implementation

Figure 8 shows a block diagram of the final prototype design implementation. Figure 9 displays the test case used on the prototype design, in which the top two images are two consecutive frames from a sample video. The prototype cycled through these frames with the averaged frame inserted in between and included an option to display the frames without the interpolated frame to demonstrate the contrast. In addition, the prototype had an option to cycle between the two original frames without the interpolated frame. This was done in order to highlight the difference between the original video stream and an interpolated one.

VI. Conclusion

In this paper a concept design implementation for a motion estimated frame interpolation device is proposed. The product is to be embedded in a 120 Hz 4K-capable output screen and receive up to 4K video at 60 Hz to interpolate with motion compensated frames. A detailed design and explanation of the proposed pipeline and algorithm implementation is discussed.

In addition, a demonstrative prototype implementation on a Nexys 4 FPGA was developed that shows frame averaging of two stored frames using a VGA output. This projects hopes to contribute to the fast growing modern display device industry and gain an understanding of the algorithmic approach and high processing nature of frame interpolation.

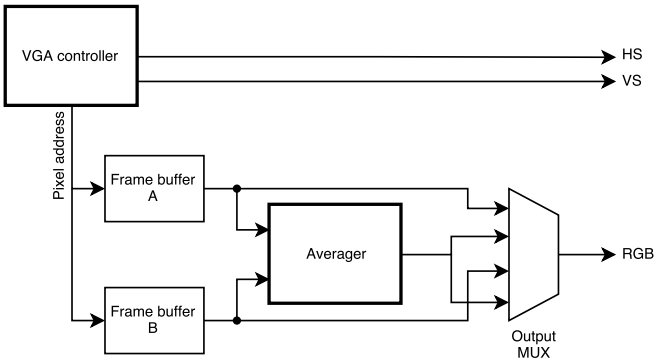

Fig. 8. Prototype design block diagram

Fig. 9. Prototype design test case showing the two video frames and the averaged frame

REFERENCES

[1] J. Zhai, K. Yu, J. Li, and S. Li, "A low complexity motion compensated frame interpolation method," in *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*. IEEE, 2005, pp. 4927–4930.

[2] S. Ertürk, "Digital image stabilization with sub-image phase correlation based global motion estimation," *Consumer Electronics, IEEE Transactions on*, vol. 49, no. 4, pp. 1320–1325, 2003.

[3] "HDMI Overview," http://www.hdmi.org/manufacturer/hdmi_2_0/.

[4] "NMath Premium: FFT Performance," http://www.centerspace.net/nmath-premium-fft-performance.

[5] V. Argyriou and T. Vlachos, "Motion estimation using quad-tree phase correlation," in *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, vol. 1. IEEE, 2005, pp. I–1081.

[6] V. S. Rosa, A. A. Susin, and S. Bampi, *An HDTV H.264 Deblocking Filter in FPGA with RGB Video Output*. IEEE, 2007. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4402519

[7] G. Triantafyllidis, D. Tzovaras, and M. Strintzis, "Blocking artifact detection and reduction in compressed data," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 10, pp. 877–890, 2002.

[8] "VGA Signal Timing," http://www.tinyvga.com/vga-timing.

[9] N. Ickes, "Vga video output," http://www-mtl.mit.edu/Courses/6.111/labkit/vga.shtml.

[10] G. D. Bergland, "Fast fourier transform hardware implementations–an overview," *Audio and Electroacoustics, IEEE Transactions on*, vol. 17, no. 2, pp. 104–108, 1969.