A Reconfigurable Frame Interpolation Hardware Architecture for High Definition Video

Ozgur Tasdizen and Ilker Hamzaoglu
Faculty of Engineering and Natural Sciences, Sabanci University
34956, Tuzla, Istanbul, Turkey
tasdizen@su.sabanciuniv.edu, hamzaoglu@sabanciuniv.edu

Abstract — **Since Frame Rate Up-Conversion (FRC) is started to be used in recent consumer electronics products like High Definition TV, real-time and low cost implementation of FRC algorithms has become very important. Therefore, in this paper, we propose a low cost hardware architecture for real-time implementation of frame interpolation algorithms. The proposed hardware architecture is reconfigurable and it allows adaptive selection of frame interpolation algorithms for each Macroblock. The proposed hardware architecture is implemented in VHDL and mapped to a low cost Xilinx XC3SD1800A-4 FPGA device. The implementation results show that the proposed hardware can run at 101 MHz on this FPGA and consumes 32 BRAMs and 15384 slices.**

Keywords: Frame Rate Up-Conversion, Frame Interpolation, Hardware Implementation, FPGA.

I. INTRODUCTION

Frame Rate Up-Conversion (FRC) is the conversion of a lower frame rate video signal to a higher frame rate video signal. LCD panels used for High Definition TV (HDTV) have a frame rate up to 240 Hz, whereas video signals are usually recorded at 24 Hz, 25 Hz, or 30 Hz. Therefore, FRC is required in order to display the HDTV video signals in the LCD panels. FRC can be done by interpolating a new frame between every two consecutive original frames like in 25 Hz to 50 Hz conversion, and it can be done by interpolating three new frames between every two consecutive original frames like in 25 Hz to 100 Hz conversion. In the case of 24 Hz to 60 Hz conversion 3:2 pull-down technique is used [1]. FRC for 1:4 conversion ratio is illustrated in Fig. 1. The dashed frames in this figure are the interpolated frames.

Simple FRC techniques like frame repetition and Linear Interpolation (LI) are used in some consumer electronics products. But, these techniques often produce artifacts to which human eye is very sensitive. Frame repetition results in motion jerkiness and LI causes blurring at object boundaries [2,3]. To overcome these problems, FRC algorithms using motion information between consecutive frames are developed [2,3]. For example, Motion Compensated Averaging (MCA) technique performs frame interpolation by using the Motion Vectors (MVs) found by the Motion Estimation (ME) process.

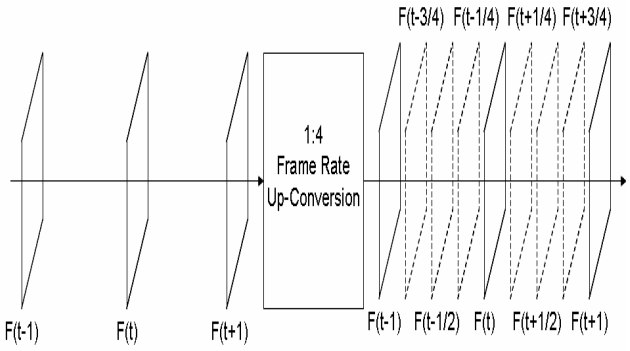

Figure 1. Frame Rate Up-Conversion

The LI and MCA techniques perform frame interpolation as shown in Eq. (1) and Eq. (2) respectively. In these equations, "t" is the time instance the frame "F" belongs to, "$\vec{x}$" is the spatial location of the current pixel in the frame and "τ" is the time slot the interpolated frame belongs to. For the conversion ratio 1:2, τ will be 0.5 for both interpolated frames, and for the conversion ratio 1:4, τ will be 0.25, 0.5, and 0.75 for the three interpolated frames.

$$F_{LI}(\vec{x}, t-\tau) = (1-\tau)F(\vec{x}, t-1) + \tau F(\vec{x}, t) \quad (1)$$

$$F_{MCA}(\vec{x}, t-\tau) = \frac{1}{2}\left[F(\vec{x}+\tau\vec{v}, t-1) + F(\vec{x}-(1-\tau)\vec{v}, t)\right] \quad (2)$$

ME is not only used for FRC. It is also used in video compression standards such as MPEG4 and H.264, and in video enhancement applications such as de-interlacing and de-noising. However, the motion information required for FRC is not the same as the one required for video coding. While ME for video coding finds the MVs providing the smallest prediction error, ME for FRC requires finding the MVs reflecting the true motion among consecutive frames.

Among the ME techniques, block matching is the most preferred method due to its simplicity. Block matching partitions the current frame into non-overlapping rectangular blocks and tries to find a block from a reference frame in a given search range that best matches the current block. Block matching algorithms cause blocking artifacts. Several FRC algorithms are proposed to reduce these blocking artifacts [3].

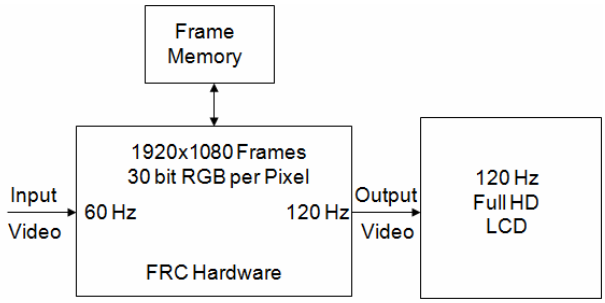

Figure 2. An Example FRC System

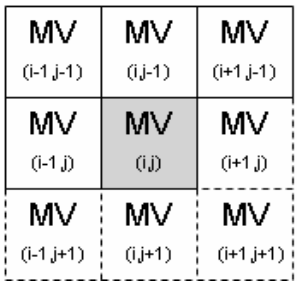

Figure 3. MVs Required to Interpolate Current MB(i,j)

In this paper, we assume that the true MVs required by the proposed FRC hardware will be obtained by another hardware, which includes a block matching ME hardware like the one we proposed in [4], and provided to the proposed FRC hardware.

An example FRC system is shown in Fig. 2. Analyzing the off-chip memory bandwidth requirement of this FRC system clearly shows that FRC systems require significant data transfer from the off-chip frame memory. Since this FRC system implements a 1:2 conversion ratio, it will interpolate new frames by using one MV per Macroblock (MB) and accessing one MB from the current frame and one MB from the reference frame. Since each color channel is 10 bits, the RGB values of a pixel take 30 bits which can be stored in a 32 bit word in memory. A Full HD frame has 1920x1080 (1.98M) pixels which take 7.92MB. Therefore, 2x7.92MB = 15.84MB have to be accessed from the off-chip frame memory in order to interpolate one frame. The received input frame and the interpolated frame will be stored in the frame memory and they will be sent to the LCD display from the frame memory. Therefore, 47.52MB per frame will be accessed from the off-chip frame memory. As it can be seen from this example, FRC systems require significant off-chip memory bandwidth.

FRC algorithms such as Adaptive Motion-Compensated Interpolation and Overlapped Block Motion Compensation proposed in [5-10] produce good quality results. However, for interpolating a MB, these algorithms do not only access the MBs in the current and previous frames pointed by the MV for the current MB, they also access the MBs pointed by the MVs of the eight spatially neighboring MBs of the current MB. The MVs required for interpolating MB(i,j) is shown in Fig. 3. In the figure, "i" and "j" denote the x and y coordinates of a MB, respectively. The dark shaded MB is the current MB(i,j) and dashed MBs are its non-causal neighboring MBs. Therefore, these FRC algorithms access 9 MBs from current frame and 9 MBs from reference frame for interpolating a MB. This significantly increases the off-chip memory bandwith requirement of an FRC system.

Even though the off-chip memory bandwidth required by these FRC algorithms can be reduced by using a large on-chip memory as proposed in [11], real-time implementation of these FRC algorithms for HDTV is very difficult and they require a significant area for the on-chip memory.

Therefore, in this paper, we propose a low cost hardware architecture for real-time implementation of frame interpolation algorithms requiring much lower off-chip memory bandwidth; LI, MCA, Static Median Filtering (SMF), Dynamic Median Filtering (DMF), Soft Switching (SS) and Cascaded Median Filtering (CMF) [3]. The proposed hardware architecture is reconfigurable and it allows adaptive selection of these frame interpolation algorithms for each 16x16 MB.

The proposed hardware architecture is implemented in VHDL and mapped to a low cost Xilinx Spartan XC3SD1800A-4 FPGA using Xilinx ISE 9.2.04. It is verified with RTL simulations using Mentor Graphics Modelsim. The implementation results show that the proposed hardware can work at 101 MHz and it consumes 15384 slices and 32 Block RAMs (BRAMs).

Several complete FRC hardware implementations including these frame interpolation algorithms are proposed in [12-15]. However, they do not specify the details of the frame interpolation part of their hardware, and they do not propose a reconfigurable hardware architecture for implementing these frame interpolation algorithms.

The rest of the paper is organized as follows. Section II explains the frame interpolation algorithms implemented by the proposed FRC hardware. Section III describes the proposed reconfigurable FRC hardware architecture. Section IV concludes the paper.

II. Frame Interpolation Algorithms

FRC by repetition of the original frames results in motion jerkiness and LI causes blurring at object boundaries. MCA is used to overcome these artifacts. However, it introduces blocking artifacts. Blocking artifacts occur at object boundaries when a block contains multiple objects with different motions. An appropriate solution to these local problems is graceful degradation [3].

Graceful degradation methods are SMF, DMF, SS, and CMF. Their equations are shown in (3), (4), (5), and (6) respectively. Their advantages and drawbacks are discussed in detail in [3]. In general, SMF produces good results for stationary scenes; however it fails for detailed parts of the video. DMF performs better for these parts of video. The drawback of DMF is its tendency to cause serration of edges in highly detailed areas. The block diagrams of SMF and DMF are shown in Fig. 4 and Fig. 5, respectively.

SS is an alternative to the rapid switching of DMF between LI and motion compensated pixels. SS takes the weighted average of motion compensated and non-motion compensated pixels. As a result, switching between LI and MCA becomes softer. As shown in Eq. (5), the weighting mechanism is controlled by a factor "*k*" which shows the reliability of the MVs. For reliable MVs, MCA will be preferred and for unreliable MVs, LI will be preferred.

SS may result in local motion jerkiness or local blur. CMF combines the strengths of SMF, DMF, and SS by taking the median of these methods. CMF can overcome the problems of these individual methods if controlled carefully.

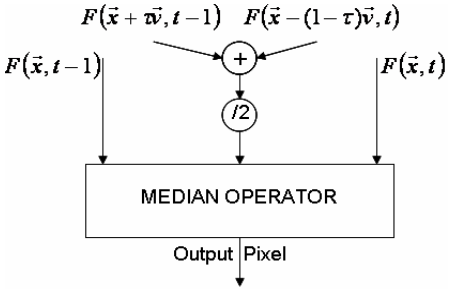

Figure 4. SMF

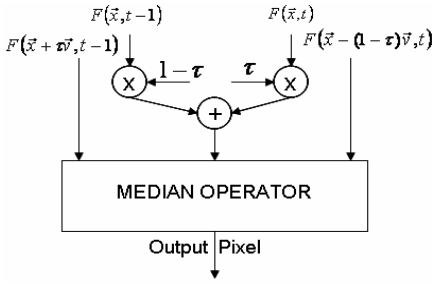

Figure 5. DMF

$$F_{SMF}(\vec{x},t-\tau)=median(F(\vec{x},t-1),F(\vec{x},t),F_{MCA}(\vec{x},t-\tau)) \quad (3)$$

$$F_{DMF}(\vec{x},t-\tau)=median(F(\vec{x}+\tau\vec{v},t-1),F(\vec{x}-(1-\tau)\vec{v},t),F_{LI}(\vec{x},t-\tau)) \quad (4)$$

$$F_{SS}(\vec{x},t-\tau)=kF_{LI}(\vec{x},t-\tau)+(1-k)F_{MCA}(\vec{x},t-\tau) \quad (5)$$

$$F_{CMF}(\vec{x},t-\tau)=median(F_{SMF}(\vec{x},t-\tau),F_{DMF}(\vec{x},t-\tau),F_{SS}(\vec{x},t-\tau)) \quad (6)$$

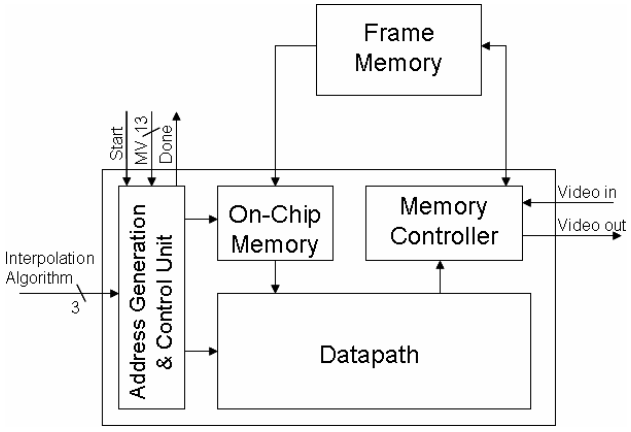

Figure 6. Top-Level Hardware Architecture

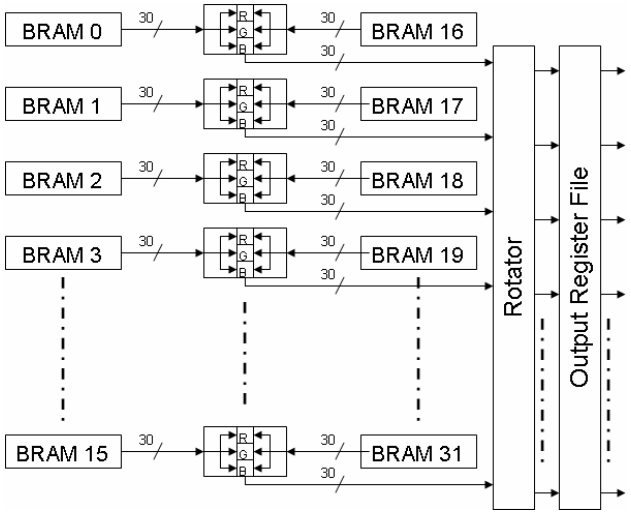

Figure 7. On-Chip Memory and Datapath

III. PROPOSED HARDWARE ARCHITECTURE

The top-level block diagram of the proposed frame interpolation hardware architecture is shown in Fig. 6. The proposed hardware architecture implements LI, MCA, SMF, DMF, SS and CMF frame interpolation algorithms and it allows adaptive selection of these algorithms for each MB. The proposed hardware interpolates frames MB by MB. It takes the selected interpolation algorithm and the MV for each 16x16 MB as inputs and performs the frame interpolation. In this paper, we implemented the on-chip memory and the datapath part of this hardware shown in Fig. 7.

The input MV to the frame interpolation hardware points to a MB in the current frame and a MB in the reference frame in a range of (±48, ±24) pixels. MVs used in the interpolation process correspond to a larger search range in the ME process. For example, for the conversion ratio 1:2, the MVs with a range of (±48, ±24) pixels used in the interpolation process correspond to a search range of (±96, ±48) pixels in the ME process.

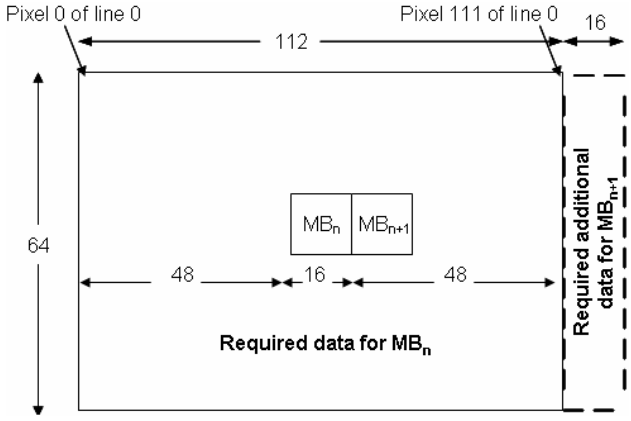

Figure 8. Data Stored in the On-Chip Memory

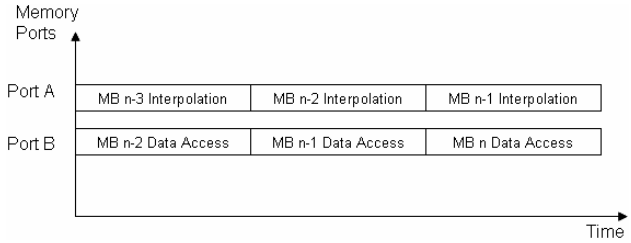

Figure 9. MB Schedule

As shown in Fig. 7 and Fig. 8, the on-chip memory consists of 32 BRAMs, and it is used to store 112x64 pixels from the current frame and 112x64 pixels from the reference frame. BRAM 0 to BRAM 15 are used to store the proper area from the current frame and BRAM 16 to BRAM 31 are used to store the proper area from the reference frame. Since each color channel (R, G, B) is 10 bit wide, BRAMs are configured as 448x32-bit, and each BRAM is used to store 4 lines of the required area from the corresponding frame.

As shown in Fig. 8, most of the data that should be stored in the on-chip memory for two consecutive MBs are the same. Therefore, for the next MB only the 64x16 pixels non-overlapping area, shown with the dashed lines in Fig. 8, can be accessed from the frame memory by using data re-use methodology. In addition, since the BRAMs in the FPGAs have dual ports, the interpolation of a MB can be overlapped with accessing the non-overlapping area required by the next MB from the frame memory as shown in Fig. 9. However, this requires storing additional 16 pixels per line in each BRAM and it increases the complexity of the address generation module.

The proposed datapath includes 48 Processing Elements (PEs). The boxes named as “R”, “G”, and “B” in Fig. 7 represent the PEs. Each PE performs the interpolation of a color channel. Therefore, the datapath interpolates R, G, B channels of a pixel in parallel and it interpolates 16 pixels in each clock cycle. The Rotator consists of 30 identical rotators each 16 bits long. They are used to align the interpolated pixels to match with their original positions where they must be in the current MB. The Output Register File has 256 registers each 30 bits long. The interpolated MB will be stored in this register file, and it will be sent to the frame memory by the memory controller.

The block diagram of a PE is shown in Fig. 10. In the first clock cycle of the interpolation process, the previous pixel $F(\vec{x},t-1)$ and the current pixel $F(\vec{x},t)$ will be stored in 10 bit registers “Reg. P.” and “Reg. C.”. In the second clock cycle, motion compensated values of previous pixel $F(\vec{x}+\tau\vec{v},t-1)$ and current pixel $F(\vec{x}-(1-\tau)\vec{v},t)$ will be stored in the 10 bit registers “Reg. P. MC” and “Reg. C. MC”. “Reg. SMF”, “Reg. DMF” and “Reg. CMF” include three 10 bit registers. In the second cycle, outputs of “Reg. P.” and “Reg. C.” will be added and the least significant bit will be discarded so that their average will be calculated and stored in the register “Reg. DMF”. Similarly, in the third cycle MCA value will be calculated and stored in the register “Reg. SMF”. “Reg. CMF” stores the outputs of SMF, DMF and SS.

SS value is calculated by the Soft Switching module. The block diagram of the Soft Switching module is shown in Fig. 11. This module takes LI and MCA, and multiplies them with “*k*” and “*(1-k)*”. In order to save area, no multiplier or divider is used in this module. By only using one adder/subtractor and one multiplexer, multiplying with the “k” and “(1-k)” values 24/32:8/32, 20/32:12/32, 18/32:14/32, 16/32:16/32 can be realized. For example, multiplying with the “k” value 20/32 can be realized by adding the result of “<<2” (x4) operation to the result of “<<4” (x16) operation and multiplying with the “(1-k)” value 12/32 can be realized by subtracting the result of “<<2” operation from the result of “<<4” operation. The least significant 5 bits of the adder-subtractor result is discarded to divide it by 32.

The Median module is shown in Fig. 12. It takes three 10 bit inputs “A”, “B”, and “C”, and finds the median of these inputs. The Median module has three comparators and four 2-to-1 multiplexers. In order to increase its clock frequency, pipelining registers are used at its input and output. First, the median value for SMF is calculated. Then, the median value for DMF is calculated in the next clock cycle. Finally, the median value for CMF is calculated. In order to calculate CMF, the result of the Median module for SMF and DMF are stored in “Reg. CMF” together with the result of Soft Switching module.

The “Output Mux” is used to select the result of the interpolation algorithm specified by the Interpolation Algorithm input. This multiplexer selects either results of LI, MCA, SS or the result of Median module. The results of LI, MCA and SS will be ready in the second, third, and fourth clock cycles, respectively. SMF, DMF, and CMF results will be ready in the 5th, 6th, and 8th clock cycles, respectively. When operated in LI, MCA, SMF, DMF, or SS modes, there is no need to stall the pipeline, but CMF

mode requires stalling the pipeline for two clock cycles.

The proposed hardware architecture is implemented in VHDL and mapped to a low cost Xilinx Spartan XC3SD1800A-4 FPGA using Xilinx ISE 9.2.04. It is verified with RTL simulations using Mentor Graphics Modelsim. The implementation results show that the proposed hardware can work at 101 MHz and it consumes 15384 slices and 32 BRAMs. A PE consumes 222 slices. Soft Switching and Median modules consume 27 and 35 slices, respectively. When operated in any mode except CMF, the proposed hardware interpolates a 16x16 MB in 16 clock cycles after the first result is ready. When operated in CMF mode, it interpolates a 16x16 MB in 48 clock cycles after the first result is ready.

IV. CONCLUSIONS

In this paper, a low cost reconfigurable hardware architecture for frame interpolation of HD frames is presented. The proposed hardware architecture implements the LI, MCA, SMF, DMF, SS, and CMF frame interpolation algorithms and it allows adaptive selection of these algorithms for each MB. The proposed hardware architecture is implemented in VHDL and mapped to a low cost Xilinx XC3SD1800A-4 FPGA. The implementation results show that the proposed hardware can run at 101 MHz on this FPGA and it consumes 32 BRAMs and 15384 slices.

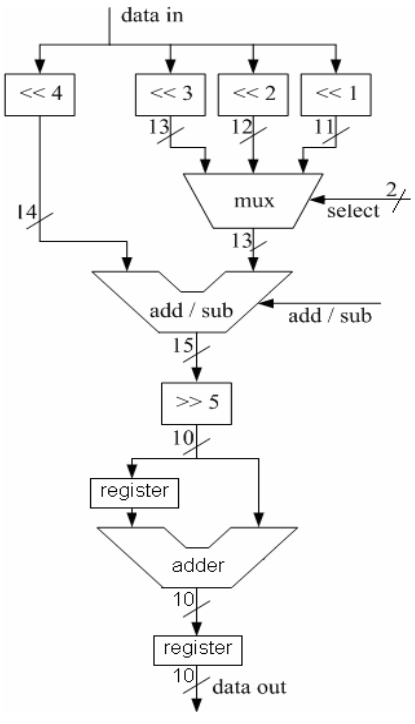

Figure 11. Soft Switching Module

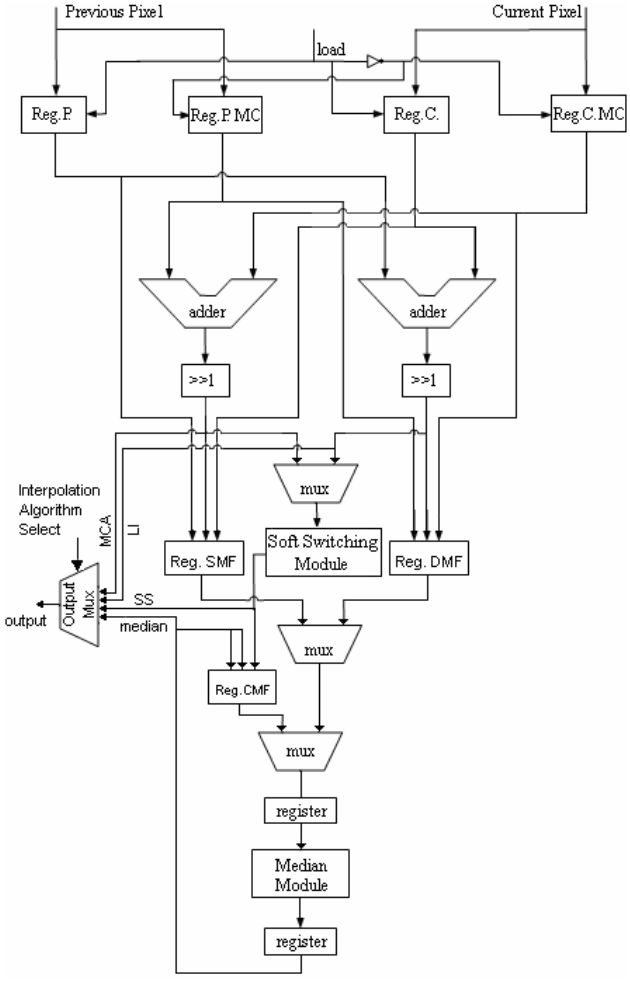

Figure 10. Processing Element

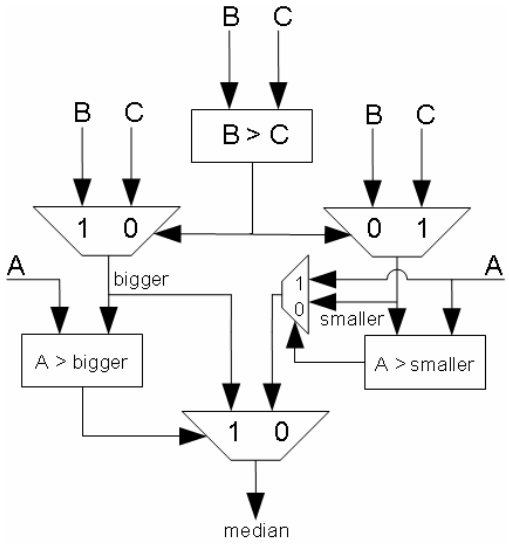

Figure 10. Median Module

ACKNOWLEDGEMENTS

This work is supported in part by TUBITAK (The Scientific and Technological Research Council of Turkey).

REFERENCES

[1] Bugwadia, K. A., Petajan, E. D., Puri, N. N., "Progressive-Scan Rate Up-Conversion of 24/30 Hz Source Materials for HDTV", *IEEE Trans. on Consumer Electronics*, vol. 42, no. 3, pp. 312-321, Aug. 1996.

[2] Castagno, R., Haavisto, P., Ramponi, G., "A Method for Motion Adaptive Frame Rate Up-Conversion", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no.5, pp. 436–442, Oct. 1996.

[3] Ojo, O. A., De Haan, G., "Robust Motion-Compensated Video Upconversion", *IEEE Trans. on Consumer Electronics*, vol. 43, no. 4, pp. 1045-1056, Nov. 1997.

[4] Tasdizen, O., Kukner, H., Akin, A., Hamzaoglu, I., "High Performance Reconfigurable Motion Estimation Hardware Architecture", *DATE Conference*, Nice, France, Apr. 2009.

[5] Ha, T., Lee, S., Kim, J., "Motion Compensated Frame Interpolation by new Block-based Motion Estimation Algorithm", *IEEE Trans. on Consumer Electronics*, vol. 50, no. 2, pp. 752-759, May 2004.

[6] Zhai, J., Yu, K., Li, S., "A Low Complexity Motion Compensated Frame Interpolation Method", *IEEE ISCAS*, pp. 4927-4930, Kobe, Japan, May 2005.

[7] Choi, B. D., Han, J. W., Kim, C. S., Ko, S. J., "Motion-Compensated Frame Interpolation Using Bilateral Motion Estimation and Adaptive Overlapped Block Motion Compensation", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no. 4, pp. 407–416, Apr. 2007.

[8] Yang, Y. T., Tung, Y. S., Wu, J. L., "Quality Enhancement of Frame Rate Up-Converted Video by Adaptive Frame Skip and Reliable Motion Extraction", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 17, no.12, pp. 1700–1713, Dec. 2007.

[9] Lee, S. H., Shin, Y. C., Yang, S., Moon, H. H., Park, R. H., "Adaptive Motion-Compensated Interpolation for Frame Rate Up-Conversion", *IEEE Trans. on Consumer Electronics*, vol. 48, no. 3, pp. 444-450, Aug. 2002.

[10] Lee S. H., Kwon, O., Park, R. H., "Weighted-Adaptive Motion-Compensated Frame Rate Up-Conversion", *Trans. on Consumer Electronics*, vol. 49, no. 3, pp. 485-492, Aug. 2003.

[11] Beric, A., Van Meerbergen, J., De Haan, G., Sethuraman, R., "Memory-Centric Video Processing", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 18, no.4, pp. 439–452, Apr. 2008.

[12] De Haan, G., Biezen, P. W. A. C., Ojo, O. A., "An Evolutionary Architecture for Motion-Compensated 100 Hz Television", *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, no.3, pp. 207–217, Jun. 1995.

[13] De Haan, G., Kettenis, J., Löning, A., De Loore, B., "IC for Motion-Compensated 100 Hz TV with Natural-Motion Movie-Mode", *IEEE Trans. on Consumer Electronics*, vol. 42, no. 2, pp. 165-174, May 1996.

[14] De Haan, G., "IC for Motion-Compensated De-Interlacing, Noise Reduction, and Picture-Rate Conversion", *IEEE Trans. on Consumer Electronics*, vol. 45, no. 3, pp. 617-624, Aug. 1999.

[15] Beric, A., De Haan, G., Sethuraman, R., Van Meerbergen, J., "An Efficient Picture-Rate Up-Converter", *Journal of VLSI Signal Processing*, vol. 41, no. 1, pp. 49-63, Aug. 2005.