Low Complexity Pipelined Architecture for Real-Time Generic Video Scaling

Asmar A. Khan

***Abstract*—A five stage pipelined architecture is proposed to target real-time video resolution conversion in spatial domain. This low complexity design is based on pre-computed memory mapping which computes the geometric position of interpolated pixels and their gray level values based on the scaling factor. The memory map has been conceived using nearest neighbor interpolation technique which simplifies the implementation. The scaling ratio is provided as an input to the hardware architecture before conversion. The design is capable of converting any video frame size in real-time. The memory requirement in this operation has been significantly reduced in comparison with earlier hardware based schemes. The results have been validated on a Xilinx Spartan FPGA running at 100 MHz Conversion times for different scaling ratios have been reported**

I. Introduction

Video size scaling is an important operation in the paradigm of multimedia transcending and communications. The advancements in biomedical imaging and presence of heterogeneous devices have increased the demand of image and video conversion and scaling. Nowadays, consumer multimedia devices have diverse resolutions and often communicate other devices in different scenarios. This communication is subjected to real-time video scaling operations.

Many scaling and resolution conversion techniques have been proposed in the literature [2-9]. Most image resolution conversion techniques reported are thus far are software based [6] and meant for off-line processing. Software schemes are generally not capable of meeting processing requirements for real-time video scaling operations. Also, software resolution conversion is encoder and platform dependent and for fixed target size. Thus there is a need for dedicated hardware architecture that can achieve real-time performance for any given scaling factor. Recently some hardware based image scaling designs have been proposed [2, 3, 10]. These schemes target only a fixed image size for which the ratio of size conversion is a whole number. Two important issues can be identified in the existing hardware schemes. (i) The designs cater for fixed size scaling in up-direction (zooming) [4] and (ii) Require excessive memory and complex control making them difficult to meet the stringent timing constraints in real-time. Some hardware based solutions for video scaling have been compared in [10].

Manuscript received April 19, 2011; revised July 25, 2011.

Asmar A. Khan, School of Computing and Communications, Lancaster University, LA1 4WA, UK (Email: a.khan3@lancaster.ac.uk)

A relatively large memory would be needed in these existing schemes to support non-integer scaling ratios. An important aspect of the work reported here is to provide a generic architecture that is memory-efficient and meets real-time image and video size scaling for arbitrary ratios. This work is an extension to our earlier reduced-memory architecture for fixed image sizes proposed in [11].

The rest of the paper is organized as follows: Section 2 describes the general image and video scaling background in context of re-sampling. This section also explains the 'up' and 'down' sampling operations and their memory requirements. The proposed memory-efficient architecture that achieves generic size conversion is explained in section 3. Results and analysis including FPGA implementation have been discussed in section 4. This is followed by the conclusion in section 5.

II. Re-Sampling and Interpolation

Video size scaling involves spatial scalability of the frames. This scalability is achieved by re-sampling the original image to the desired size. Decimation and interpolation are two key operations in re-sampling. A non-integer scaling factor results in a combination of decimation and interpolation operations. The discrete signals are then smoothed out with a low-pass filter in order to alleviate the aliasing effect. Nearest neighbor, Bi-cubic, Quadratic and Spline are some of the well known interpolation techniques which have been compared in [1]. When re-sampled to an integer factor, the signals require less memory as only an up-sampling or a down-sampling operation will suffice. On the other hand, when scaling factor is a non-integer value, the intermediate memory demand increases tremendously, as illustrated in figure 1.

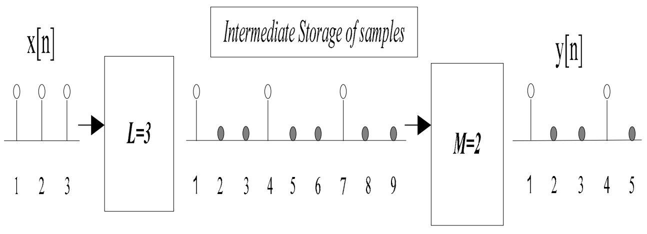

Fig. 1. Re-sampling by factor [L/M]

Fig. 1 depicts a re-sampling operation where the data points are up-sampled by a factor L and then down-sampled by M. These up and down sampling operations require a large amount of intermediate storage memory. A comparison of this intermediate memory requirement for some well known scaling conversions is shown in Fig. 2. The bars show the intermediate memory needed for size conversion in horizontal and vertical directions respectively.

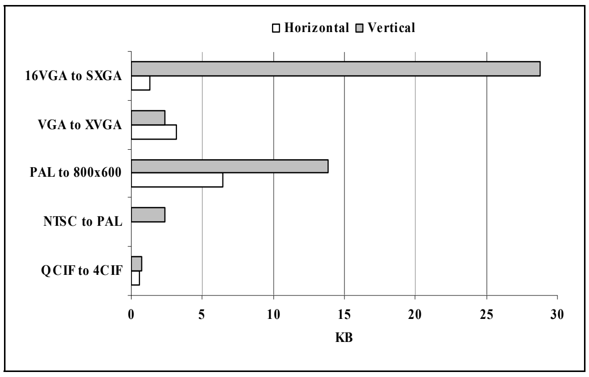

Fig. 2. Re-sampling intermediate memory requirement for different spatial resolution conversions

A. Interpolation

Interpolation is a key operation in the process of re-sampling that approximates the blank pixels that have been introduced by up-sampling. Different interpolation techniques use different number of pixels to reconstruct the missing pixel value. The comparison in [1] shows that Nearest Neighbor interpolation is the simplest in terms of implementation cost whereas Gaussian scheme gives the best quality. However, Gaussian technique is orders of magnitude complex in terms of computational requirements [10].

B. Fixed Size Image Resizing

Although some hardware based up-scaling and down-scaling techniques have been proposed recently [10, 12], yet all these techniques consider a fixed target image size in their design. None of these schemes address a generic scaling method that can convert an image or a video frame to any given spatial resolution. Calculating the position of missing indices in the operation remains a challenge both in terms of computations as well as the hardware implementation.

C. Memory Requirement for Re-sampling

The idea of incorporating a 'reduced memory map' for a fixed-size conversion was introduced in [11] to map the input image pixels to their corresponding output positions. This reduces the intermediate memory requirement significantly and also makes the design scalable. The memory map was generated using a given scaling factor. A non-integer scaling factor results in a large memory, whereas an integer scaling factor requires either up or down sampling only and lesser memory. This direct mapping can be achieved in one single operation. In this work, architecture has been developed that reduces the memory requirements for real-time video scaling operation where the conversion ratios are generic and they are specified at run-time. The details of proposed design are given below. generate

III. Proposed Methodology

As the objective of this study is to achieve generic size scaling, an algorithm has been developed to construct the flexible memory map in hardware. The 'map generator' module takes original and desired resolution as input parameters. The desired resolution is then divided by the original resolution and the value of *remainder* is stored in a register. This *remainder* is added to itself until the *remainder-sum* becomes equal to input resolution parameter specified. A small threshold value 'ε' is used to round-off the *remainder-sum*. Counts '*X*' and '*Y*', respectively representing the original and new mapping, are incremented by one each time the *remainder* is added to itself. In this way, an algorithm similar to greatest common divider (GCD) has been developed. The output from this module provides the '*X*' to '*Y*' mapping of original and new pixels. Thus a non-integer scaling size conversion can be performed by consuming very less memory. The same process is repeated for the other (horizontal or vertical) dimension. The algorithm is summarized using the flow chart in Fig. 3.

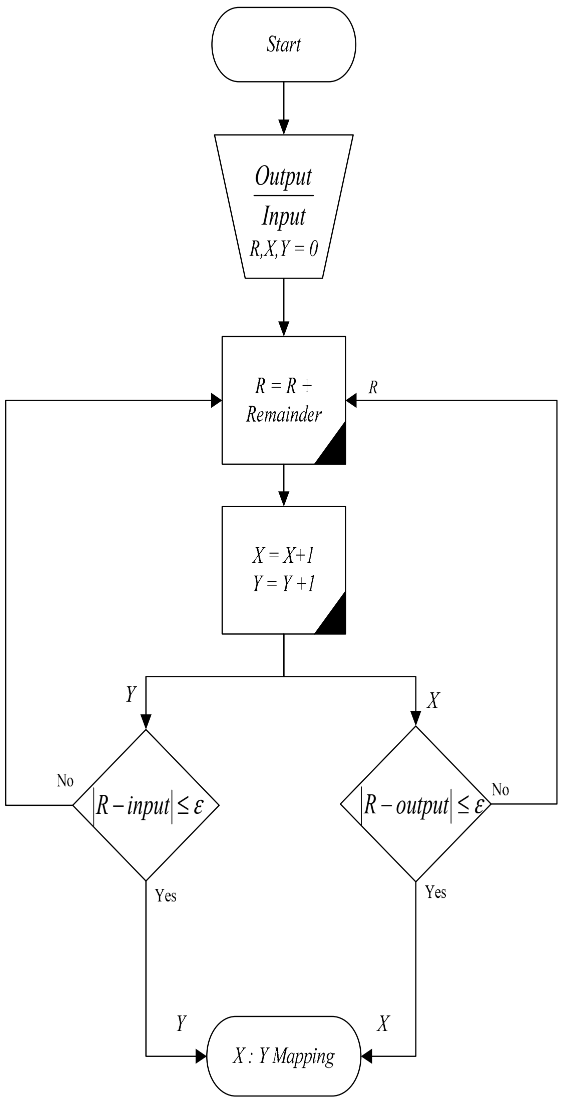

Fig. 3. Flow chart for generic scaling hardware

The concept of direct mapping in a single dimension is explained in Fig. 4. The up-sampling by a factor 'L' and down-sampling by 'M' is performed using the map explained earlier. The shaded samples are the ones reconstructed through Nearest Neighbor interpolation technique. This technique makes the mapping possible and reduces the cost of the circuit remarkably in the design

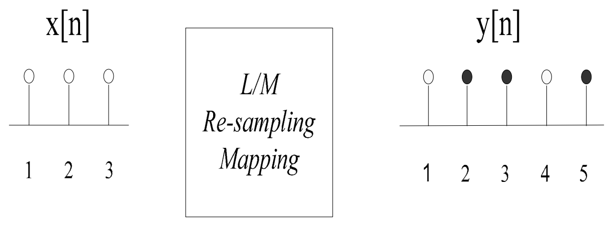

Fig. 4. Non-Integer Re-sampling & Mapping

A. Controller State Machine

The state machine based controller module utilizes the memory mapping of input ‘X’ samples to output ‘Y’ samples. The controller decides by looking at the net result of the conversion to determine whether the input samples require low-pass filtering or not. If the net result is ‘not a downscaling’ in horizontal or vertical direction then the low-pass filtering is bypassed. Otherwise a 7-tap low-pass filter for CCIR to SIF conversion [7] is applied. The controller then allocates the memory space from the on-chip ROM, based on the output of the ‘map generator’. The interpolated samples at the output are then post-processed in order to alleviate any artifacts caused by the interpolation operation. A 7-tap linear symmetric filter proposed in [7] is applied for the post-processing operation followed by the mapping of vertical samples. Vertical mapping is achieved by repeating the columns based on the output of ‘map generator’. The vertical mapping is accomplished separately and independently from the horizontal one.

The controller-based technique outperforms the earlier block-level technique presented in [2] and reduces the memory requirements to off-chip image storage only. The controller using Nearest Neighbor interpolation does not use any masking kernel for interpolation and hence does not require any dependent pixels information. This reduces the design complexity and makes it faster at the same time. The state transition diagram of the controller is shown in Fig. 5. The horizontal and vertical re-sampling is achieved separately making it possible to have different mapping and interpolation techniques for horizontal and vertical directions. Other higher-order interpolation techniques can also be embedded in this approach.

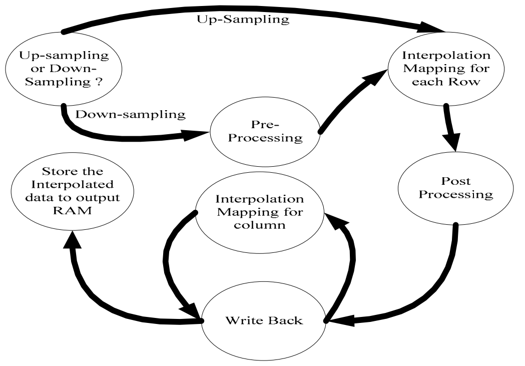
Up-Sampling
Up-sampling or Down-Sampling ?
Interpolation Mapping for each Row
Down-sampling
Pre-Processing
Store the Interpolated data to output RAM
Interpolation Mapping for column
Post Processing
Write Back

Fig. 5. State Transition Diagram for Controller

B. Pipelined Architecture

A five stage pipelined architecture has been developed to afford higher clock speed and data throughput, hence coping with the real-time processing issues. The block diagram of complete architecture is shown in Fig. 6. The image is read row-wise and fed into the serial data path comprising of different pipelined stages. ‘Read’ stage module works independently of any other module and reads ‘X’ amount of pixels from the input image, where ‘X’ is the number which was calculated by the ‘map generating’ hardware. When the next ‘X’ samples are read, the earlier ones are in pipeline at ‘pre-processing’ stage. The pipelined registers change their states at every clock edge provided by the controller state machine described in section 3.1. This design is fully pipelined and serial, hence reducing the memory required to a minimal level yet meeting the real-time processing.

At any clock cycle, the five stages of the systems have different ‘X’ samples for processing. The ‘controller’ works like the Central Processing Unit and provides each sub-module with the information and the clock signal it requires in order to perform the respective operation at appropriate time slot. The design proposed here is an enhancement over our prior work in [11] in that the design is pipelined and many delays due to distributed memory latencies are avoided.

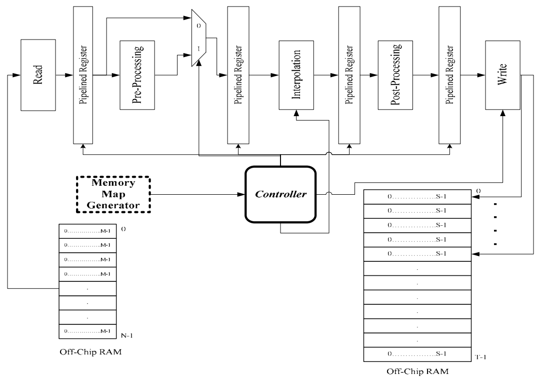

Fig. 6. Pipelined Architecture for Generic Scaler

IV. Results and Analysis

The design has been implemented on Xilinx Spartan FPGA running at 100 MHz clock. The pipelined registers have been implemented as on-chip ROM using lookup tables (LUT). The design was captured in Virology and verified on ModelSim simulator to check the timing delays in real-time scenarios. The FPGA clock at 100 MHz was chosen for the design and this made the scaling at frame rate of 30 fps possible. The memory count in our design is almost 10 times less than what was required in [2]. Although the design is based on the Nearest Neighbor technique, yet it performs well while preserving the edges [9]. Memory reduction has been achieved here because of a combination of an efficient controller and a memory map which reduces the intermediate memory during re-sampling operation.

A. Memory Requirement

The conversion performance (memory and fps) mentioned in [10] target only a fixed size scaling in different schemes. The memory consumption of proposed methodology has been compared with the previous designs in Table 1. It can be seen that a significant reduction of on-chip memory (up to around 80%) has been achieved in the proposed design. Nearest neighbor interpolation has reduced the design complexity and improved efficiency at the same time. The proposed design can cater for every scaling factor with much reduced memory consumption making it ideal for hand-held, portable and mobile devices.

TABLE 1. MEMORY REQUIREMENT COMPARISON

Scaling Conversion	**Memory Requirement as per [10] (10 x Bytes)**	**Proposed Memory Requirement (Bytes)**
QCIF to 4CIF	158.4	32
NTSC to PAL	-	48
PAL to 800x600	432	80
VGA to XVGA	256	80
16VGA to SXGA	1024	150

B. *Hardware Resources*

The gate count of the design comes out to be around 12K gates and 3500 LUTs including the pipelined registers. The gate count can be further reduced by using on-chip Block RAM in FPGA resources to replace the ROM in the design.

C. *Delay and Throughput*

The proposed design has been implemented on Xilinx Spartan FPGA running at 100 MHz clock speed. The five-stage pipelined design outputs different number of samples every clock cycle depending upon the scaling factor. The '$Y \div X$' scaling factor causes the read cycle to read 'X' pixels from input image and write 'Y' pixels to the output memory in each write cycle. A QCIF image when converted to VGA takes 11 samples in a row and maps them to 40 samples hence mapping every 3 samples in a column to 10. The complete image is processed in (25344/11 =) 2304 cycles. Thus each frame is processed in 0.02304 sec. This corresponds to a frame rate of around 44 in real-time video processing. The computation here has been made only for a luminance component of the image. Similar procedure could be applied to the chrominance components as well.

TABLE 2. CLOCK SPEED AND FRAMES PER SECOND COMPARISON

Scaling Conversion	Clock as per [10] (MHz)	Clock Proposed (MHz)	fps as per [10]	fps for Proposed
PAL to 800x600	46	100	25	44
VGA to XVGA	65	100	-	33
16VGA to SXGA	105	100	50	30

Table 2 above presents a comparison of clock speed and frames per second with the earlier presented techniques for some well-known size conversions. Our proposed design is capable of maintaining an acceptable rate of 30 fps at 100 MHz clock for the highest scaling factor. The frame rate in our scheme is limited only by the size of input video resolution. A higher speed FPGA or ASIC could be used to scale the performance even further.

V. CONCLUSIONS

The work presented in this paper presents a low complexity real-time video scaling technique for generic scaling factor. The hardware proposed is serial in nature and meets the real-time conversion requirements. The architecture is fully pipelined and requires reduced hardware and memory resources. The pipelined architecture has been verified on a Xilinx Spartan FPGA whereas this can be improved to a very high clock speed as available in Virtex-4. The circuit operates on decoded data and no encoder compatibility is required. Future improvements to the design can be made by implementing higher order interpolation techniques like bilinear or cubic spline.

ACKNOWLEDGEMENT

Support provided by Higher Education Commission of Pakistan and Lahore University of Management Sciences is gratefully acknowledged.

REFERENCES

[1] T.M. Lehmann, "Survey: Interpolation Methods in Medical Image Processing", IEEE Transactions on Medical Imaging, VOL 18, No. 11, November (1999)

[2] E. Aho, J. Vanne, T.D. Hämäläinen, K. Kuusilinna, "Block-Level Parallel Processing for Scaling Evenly Divisible Images", IEEE Transactions on Circuits and Systems VOL. 52, No. 12, December (2005)

[3] S. Ramachanran, "Design and FPGA Implementation of An MPEG Based Video Scalar with Reduced On-chip Memory Utilization", Journal of Systems Architecture, VOL 51, pp 435-450, September (2005)

[4] M. Aurelio, M.O. Arias, "Real Time FPGA–Based Architecture for Bi-cubic Interpolation: An Application for Digital Image Scaling", International Conference on Reconfigurable Computing and FPGAs (2005)

[5] T.C. Lin, T.K. Truong, "DCT-Based Image Codec Embedded Cubic Spline Interpolation with Optimal Quantization", IEEE International Symposium on Multimedia pp 2746-9, September (2006)

[6] L. Wang, Q. Wang, "A Fast Intra Mode Decision Algorithm for MPEG-2 to H.264 Video Transcoding", IEEE 10th International Symposium on Consumer Electronic pp 1-5, December (2006)

[7] MPEG-1, "Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to 1.5 Mbps", ISO/IEC 11172-2, November (1991)

[8] L. Wanrong, D. Bushmitch: "Design and Implementation of a High Quality DV50-MPEG2 Software Transcoder", International Conference on Consumer Electronics, ICCE, pp 142-143, June (2002)

[9] C.H. Kim, S.M. Seong, J.A. Lee, L.S. Kim, "Winscale: An Image-scaling Algorithm Using an Area Pixel Model", IEEE Transactions on Circuits and Systems for Video Technology, VOL. 13 No. 6, pp 549–553, (2003)

[10] E. Aho, J. Vanne, T.D. Hämäläinen, K. Kuusilinna, "Configurable Implementation of Parallel Memory based Real-time Video Downscaler", Microprocessors & Microsystems,VOL. 31 No. 5, pp 283-292, August (2007)

[11] A. A. Khan, S. Masud, "Memory Efficient VLSI Architecture for QCIF to VGA Resolution Conversion", Proceedings of Pacific-Rim Symposium on Image and Video Technology, Japan, LNCS 5414, pp 829-838, (2009)

[12] C. Lin, M. Sheu, H. Chiang, "The Efficient VLSI Design of BI-CUBIC Convolution Interpolation for Digital Image Processing", IEEE International Symposium on Circuits and Systems, ISCAS, VOL. 18, No. 21, May (2008)