VLSI ARCHITECTURE AND DESIGN FOR HIGH PERFORMANCE ADAPTIVE VIDEO SCALING

Arun Raghupathy *

Qualcomm Inc.,
San Diego, CA-92121

Pohsiang Hsu, K. J Ray Liu and Nitin Chandrachoodan

Electrical Engineering Department,
Univ. of Maryland, College Park, MD-20742

ABSTRACT

In this paper, we develop an efficient architecture for video scaling based on the adaptive image scaling algorithm [3],[4]. We then develop the design of the computation units and perform synthesis to show that the chip area required to perform scaling from QCIF to 4CIF is about $20mm^2$ using 0.5μ technology.

1. INTRODUCTION

In video applications, the data rate involved is extremely high. When the available bandwidth is limited, the image size is restricted. A high-performance real time scaling technique that can scale an image while introducing little distortion will allow us to transmit video data using a small bandwidth while maintaining perceptual quality. Another motivation for developing a good scaling algorithm is to enable display of lower resolution images on higher resolution monitors (For eg., an NTSC format picture on HDTV monitors). Simple scaling techniques (such as simple pixel replication, bilinear interpolation and cubic interpolation) cause visible effects such as jagged or blurred edges when the images contain sharp edges or thin lines. Non-linear model based interpolation techniques have been proposed in [2] and [3]. However, these technique were considered difficult to implement in practical systems because of the complexity of the classification and the filtering requirements.

We develop an architecture based on the algorithm in [3],[4]. Specifically, we consider an architecture for scaling QCIF images to 4CIF format at 30 frames/sec. A straightforward architecture cannot meet the speed requirements. We apply transformations to the data flow graph to obtain an efficient architecture that meets the throughput requirements. The ASIC area requirements are then estimated based on a synthesis starting from a Verilog description of the computation units.

2. ARCHITECTURE AND DESIGN

In this section, we develop an efficient hardware architecture for scaling by a factor of two in each dimension. The algorithm for scaling of 24-bit YUV images is summarized below. Simple bilinear interpolation is used for the U and V components. The data flow graph for the Y component is shown in Fig. 1(b). In Fig. 1(b), the 4 output pixels corresponding to pixel $p[0,0]$ in the original image are being computed. In Fig. 1(b), a[-1:2,-1:2] correspond to the orientation angles of the pixels p[-1:2,-1:2], respectively. The filter coefficients, used in computing the 4 output pixels corresponding to the input pixel $p[0,0]$, are represented by r[-1:2,-1:2]. The pixel memory stores the pixels of the original image. The Sobel block computes the gradients f_x and f_y in the x and y directions at $p[2,2]$ using the pixels within dotted lines in Fig. 1(a). The angle block computes edge orientation at $p[2,2]$ by calculating $tan^{-1}(-f_x/f_y)$ and quantizing the result into 8 possible orientations. The neighborhood is classified as oriented if there is a dominant orientation (in particular, if an orientation occurs more than 6 times within the 16 pixel neighborhood). Otherwise, the pixel's neighborhood is classified as non-oriented. Filters corresponding to the 8 possible orientations are pre-stored in ROM. If the neighborhood is oriented, then the appropriate oriented filter coefficients are loaded from ROM and the interpolated pixels. If the neighborhood is non-oriented, the Y components are interpolated bilinearly using the same filtering unit (using coefficients corresponding to the bilinear case).

From the dataflow graph in Fig. 1(b), the memory bandwidth required from the pixel memory is $9 + 16 = 25$ pixels for every input pixel (or every 4 output pixels). Similarly, the memory access rate per input pixel for the angle memory is 15. This memory bandwidth can be reduced by introducing some storage elements (i.e. registers) into the Sobel, histogram and filtering units. In particular, the memory access rate for the pixel data memory can be reduced to 7 per input pixel by providing storage for 9 pixels within the Sobel operator. Similarly, the memory access requirements for the angle memory can be reduced from 15 to 3. The pixel memory access requirements can be further reduced to 5 by sharing data available in the Sobel unit with the filtering unit (See [1]). Due to the structure of the data accesses for pixels and angles, delay lines can be used instead of random access memories. Also, since the graph is feedforward, we can pipeline the architecture at any feedforward cutset.

2.1. Sobel Computation

A straightforward implementation of the Sobel operator would require 10 adders to compute f_x and f_y. We can use sub-expression sharing to reduce the complexity of the Sobel operator to 8 additions (See Fig. 4(a) where $f(i,j)$ is the Y component pixel value at position (i,j)). The critical path, if the data flow graph is implemented as a combinational circuit, is given by 2 1-bit full adders and 1 12-bit

*This work was supported in part by the NSF NYI Award MIP9457397 and the ONR grant N00014-93-10566, and was done while this author was at University of Maryland, College Park.

0-7803-5471-0/99/$10.00©1999 IEEE

(corresponding to the output precision of 12-bits assuming 8-bit inputs) carry propagate adder (CPA). If the throughput of the system is not constrained by the Sobel operator, we can fold the data flow graph so that we use only 4 adders and reuse the adders over 3 clock cycles. Folding reduces the adder complexity in half at the cost of some registers and additional control.

2.2. Orientation Angle Computation

The most obvious approach to compute the quantized orientation is to compute the ratio f_x/f_y at an appropriate precision and follow that by a ROM table lookup. The angles $\phi_i = 11.25° + 22.5° \times i;\ i = 0, 1, \cdots 7$ correspond to the transition levels and the angles $\gamma_i = 22.5° \times i;\ i = 0, 1, \cdots 7$ correspond to the reconstruction levels of the quantization of $tan^{-1}(-f_x/f_y)$. For a 2° error in the specification of the transition levels, we need to choose the precision so that we can represent the difference $tan(11.25°+2°)-tan(11.25°) = 0.036$ (i.e. a precision of atleast 5 bits after the decimal point). This requires that the divider provide an output of $12+5 = 17$ bits (or, a ROM look-up of $2^{17} = 128K$ words of size 3 bits each). This leads to high hardware complexity.

The CORDIC processor in angle accumulation mode [5] can be used to find the angle $\theta = tan^{-1}(Y/X)$ corresponding to the initial point with coordinates (X, Y). The angle is found by applying a series of shift and add operations on the initial coordinates (X, Y). Note that the angle can be represented as a sequence of μ_i's (the angle is accumulated as $z(i+1) = z(i)-\mu_i a(i)$ where $a(i) = 2^{-s(i)}$). If we want to define the quantization regions with an accuracy of 2°, when using the shift sequence $s(i) = i$ we need atleast 6 stages of CORDIC followed by a look-up table (addressed by the vector $\mu_i;\ i = 0, 1, \cdots, n-1$). The look-up table has 2^6 words of size 3 bits. This procedure can be modified to perform the required computation using 5 stages followed by a look-up table of size 2^4 words of 3 bits. For example, we can write $78.75° \approx tan^{-1}(2^0) + (tan^{-1}(2^{-2}) + tan^{-1}(2^{-3})) + (tan^{-1}(2^{-3}) + tan^{-1}(2^{-4}))$. Similarly, we can express other transition angles using $s(i) = 0, 2, 3, 3, 4;\ i = 0, 1, 2, 3, 4$ (See [1] for details). Using appropriate control to decide whether to add or subtract at each stage, we can obtain the required quantized orientation angle. The complete architecture is shown in Fig. 2(a). This signal flow graph can be folded onto a single CORDIC-like processor to obtain an area efficient implementation or can be directly implemented as a combinational circuit.

2.3. Histogram-like Computation

We need to find the dominant orientation in the neighborhood as well as the number of times that orientation occurs. One efficient approach is the folded architecture shown in Fig. 3. The contents of the orientation angle registers are compared with the 3 bit-counter output every clock cycle. The outputs of the comparators are fed to a $(16, 4)$ compressor to obtain the total number of pixels that have the same orientation corresponding to the 3-bit counter contents. The dominant orientation and maximum count registers are both initialized to zero. Also, during each cycle the output of the $(16, 4)$ compressor is compared with the previous contents of the maximum count register. If the present output of the $(16, 4)$ compressor is larger than the previous contents of the maximum count register, both the maximum count and dominant orientation registers are updated. The maximum count register is updated with the output value from the $(16, 4)$ compressor, while the dominant orientation register is updated with the present 3-bit counter contents. Therefore, after 8 cycles the desired histogram data is obtained in the maximum value register and dominant orientation registers.

2.4. Oriented and Non-oriented Filtering

It was found (See [1]) based on an analysis of the required oriented filters and the resultant magnitude of quantization errors that 11 bits are sufficient to represent the filter coefficients. Clearly, the fastest implementation would require 16 multipliers (corresponding to the number of pixels in the neighborhood) per output pixel. This means a total of 48 multipliers. Since the filters corresponding to different orientations are different, general purpose (*not* fixed coefficient) multipliers are required. It would be too expensive to have such a large number of multipliers on a single chip. We need to apply some transformation so that the multiplier hardware can be reused. One approach is to fold the computation required for each output pixel onto a smaller number of multiply-accumulate (MAC) units (the folding factor can be determined based on the throughput requirements). The architecture for computing one interpolated Y component output is shown in Fig. 2(b). The partial sums of the filtering operation are accumulated in carry save form. The final addition is performed after the 16 MAC's have been completed.

3. SYSTEM DISCUSSION AND VLSI DESIGN

We consider here a design that can scale from QCIF (144 rows by 176 columns) to 4CIF (576 rows by 704 rows) at 30 frames per second. As suggested in [4], performing the scaling in 2 steps (i.e. from QCIF to CIF and then from CIF to 4CIF) rather than a direct scaling leads to better performance. The system architecture is shown in Fig. 4(b). Four input delay lines are needed for the Y component and two lines (of half the size) for the U and V components. In order to convert the output pixels to progressive scan order, output sync delay lines are required (4 lines are required for each of the Y, U and V components, out of which 2 lines are used to store the output from the filter/bilinear units when the other 2 lines are used to provide the outputs in progressive scan order). See [1] for details.

A QCIF video sequence at 30 frames/sec has a pixel rate of 7.6×10^5 pixels/s (a CIF video sequence has a rate 3.04×10^6 pixels/s). For QCIF to CIF (CIF to 4CIF) conversion we need to compute the 4 output pixels corresponding to one input pixel within $1315ns$ ($328ns$). In order to determine whether pipelining between units is required, we first estimate the number of clock cycles required by each unit to complete the computation. Based on the architecture shown in Fig. 2(b), atleast 17 clocks are needed to complete the filtering computation. Similarly, the folded Sobel computation, the folded histogram architecture and the folded angle computation require 3, 10 and 7 clock cycles, respectively. Finally, taking into account the interface between these units as well as with the output synchronization delay lines, about 42 clock cycles are required to compute the output pixels corresponding to one input pixel. This translates to a clock cycle of requirement of $328/42 \approx 7.81ns$ for

Unit	Std. Cell Count	Std. Cell Area (μ^2)	Clock Cycles	Timing
Sobel	702	911601.00	3	10.76ns
Angle	815	769414.50	7	10.81ns
Histogram	410	393295.50	10	8.04ns
Filter	841	815625.00	20	9.80ns
Bilinear	258	395838.00	2	9.12ns

Table 1: Cell Area and Timing from Synthesis

CIF to 4CIF conversion. We showed that the throughput can be attained in a real design by describing the various blocks in Verilog, synthesizing them and performing a static timing analysis to estimate the speed (See [1]). The synthesis was performed using a 0.8μ standard cell library with Cadence's Synergy tool. If a straightforward approach does not lead to the required throughput, then there are two possible approaches. The first approach is to apply pipelining to meet the throughput requirements. Alternately, we can assume a newer technology such as 0.5μ and perform appropriate scaling of our 0.8μ results. If we assume a linear scaling of propagation delay [6] with technology, then it is enough to obtain a critical path of $8 \times 0.8/0.5 = 12.8ns$ (in reality, close to linear scaling is possible [7]).

Various logic optimizations and re-organizations were applied to meet the throughput requirements. The results summarized in Table 1. In addition, we also need a controller (with cell count 2311 and cell area $2793393\mu^2$) that has all the required registers and provides appropriate commands for the computation units. Note that three copies of the filter unit are required to compute all three interpolated output pixels in parallel. We estimated the area based on an automatic layout of the standard cells using Cadence's Silicon Ensemble corresponding to the computation and control circuitry. The area of the delay lines and ROM were estimated based on the designs in [8], [9]. The computation and control parts required an area of $35.3mm^2$ at 0.8μ. The delay lines required an estimated area (based on [9]) of $\approx 43mm^2$ at 0.8μ. This implies a total area of $\approx 78.3mm^2$ at 0.8μ. Assuming that the areas scale by $(5/8)^2$, an area of about $30.6mm^2$ at 0.5μ. The results indicate that without pipelining between units we can achieve the throughput required to scale a QCIF image to a 4CIF image at 30 frames/sec using a 0.5μ technology.

We have shown that an efficient VLSI architecture and implementation can be obtained for QCIF to 4CIF conversion at 30 frames/sec. The total chip area for such an implementation was estimated to be about $20mm^2$ at 0.5μ. Further speed-up can be obtained by pipelining so that the scaling architecture can be used with larger image formats.

4. REFERENCES

[1] A. Raghupathy, "Low Power and High Speed Algorithms and VLSI Architectures for Error Control Coding and Adaptive Video Scaling," *Ph.D Thesis*, Univ. of Maryland, College Park, Dec. 1998.

[2] J. Salonen, "Edge and motion controlled spatial upconversion," *IEEE Trans. Cons. Elec.*, vol. 40, pp. 225–233, Aug. 1994.

[3] Y. Wang and S. K. Mitra, "Image representation using block pattern models and its image processing applications," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 321–336, Apr. 1993.

[4] P. Hsu and K. J. R. Liu, "Method and apparatus for adaptive image interpolation," *Odyssey Technologies-Confidential Report*, Nov. 1997.

[5] Y. H. Hu, "CORDIC-based VLSI architectures for digital signal processing," *IEEE Sig. Proc. Mag.*, vol. 9, pp. 16–35, July 1992.

[6] N.H.E.Weste and K. Eshraghian, *Principles of CMOS VLSI design.* Addison Wesley, 1993.

[7] B. Davari, R. H. Dennard, and G. G. Shahidi, "CMOS scaling for high performance and low power - the next ten years," Proc. of the IEEE, vol. 83, pp. 595–606, April 1995.

[8] K. Dejhan, F. Jutand, N. Demassieux, O. Colavin, A. Galisson, and A. Artieri, "A new high-performance programmable delay line IC," IEEE Trans. on Cons. Elec., vol. 35, pp. 893–899, Nov. 1989.

[9] H.-J. Mattausch, F. Matthiesen, J. Hartl, R. Tielert, and E. P. Jacobs, "A memory-based high-speed delay line with large adjustable length," IEEE J. Solid-State Circuits, vol. 23, pp. 105–110, Feb. 1988.

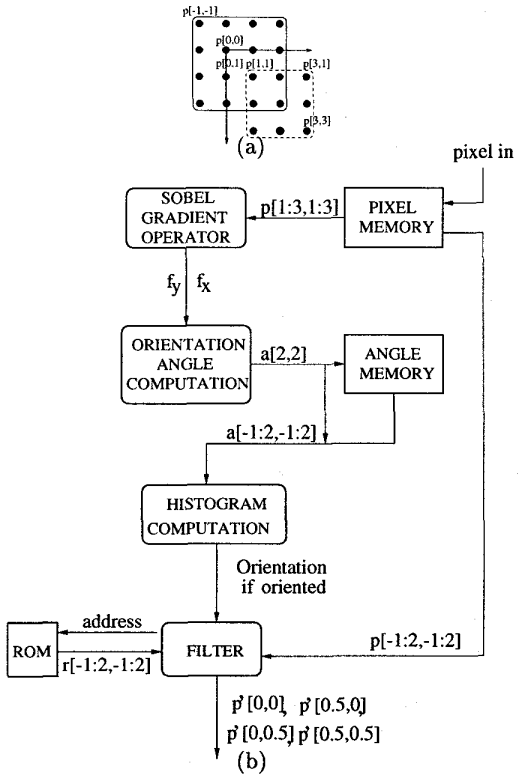

Figure 1: (a)Neighborhood of the present pixel block showing notation (b)Dataflow Graph of Computations in required in adaptive scaling

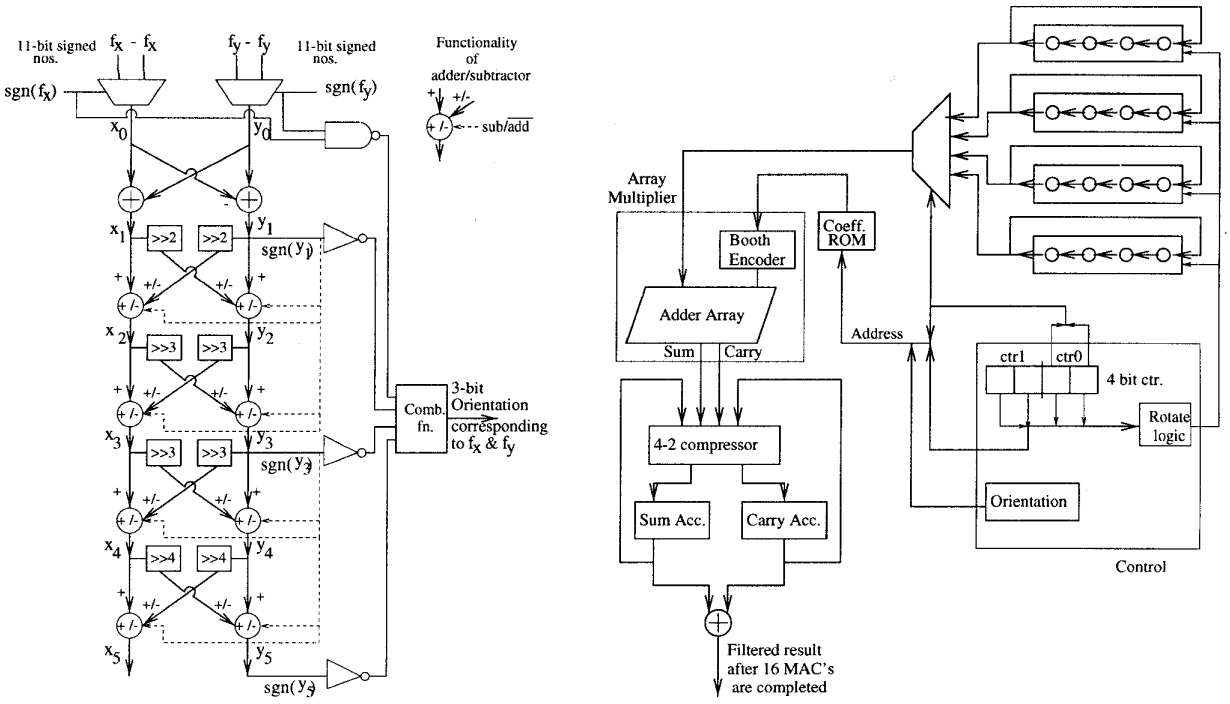

Figure 2: (a) Angle Computation using CORDIC-like Technique (b) Folded Filter Architecture

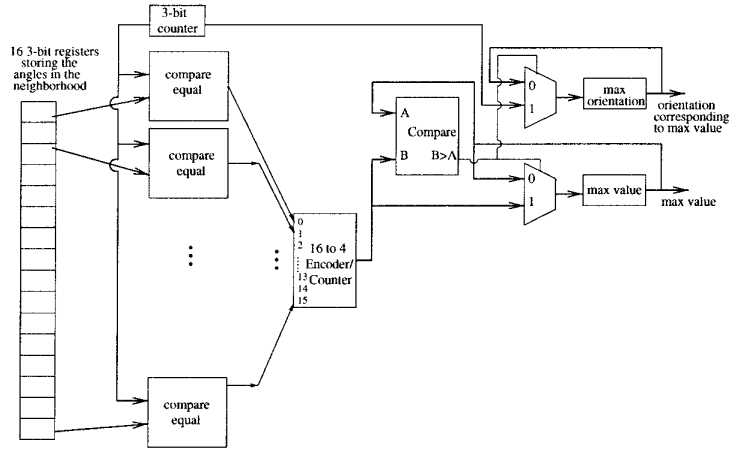

Figure 3: Folded Architecture for Histogram Computation

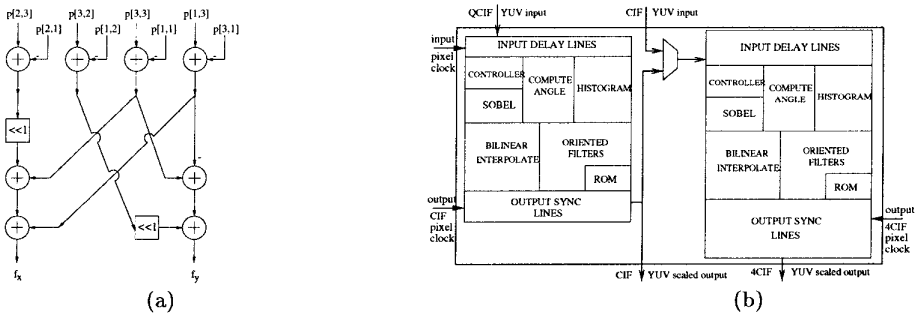

Figure 4: (a)Sobel Operator Computation b)Overall System Architecture