A Scalable Halftoning Coprocessor Architecture

Anders Kugler[1] and Roger-David Hersch[2]

Laboratoire de Systèmes Périphériques
École Polytechnique Fédérale de Lausanne
CH-1015 Lausanne - Switzerland

Abstract

Exact-angle superscreen dithering requires large dither tiles. Since storing precomputed screen elements for each intensity level would require too much memory, dithering must be executed on the fly at halftoning time. For this purpose, a dithering coprocessor is presented which generates halftoned images at high speed.

The proposed hardware architecture is based on a pipelined and scalable design which speeds up halftoning by a factor of twenty compared with modern RISC software-based solutions. We describe the architecture of the coprocessor and show to what extent it can be scaled for improving performances. The proposed coprocessor could find applications in digital color copiers which need to print scanned color images at high speed.

Keywords: Halftoning, parallel dithering, coprocessor architecture.

1: Introduction

Halftoning is the art of producing a binary image from a grayscale image such that the resulting image gives the impression of having grayscale tones. The halftoning technique has long been used in the graphic arts and the printing industry. Digital methods have replaced the traditional photographic halftone process. Due to the proliferation of bi-level desktop printers and photocomposers, converting grayscale images to halftoned black-and-white images remains an important issue.

The halftoning process which is developed here is based on a regular grid of dither tiles. Each tile contains N ordered cells and supports the imaging of N+1 gray levels. Dithering consists of comparisons where, for each binary pixel of the target image bitmap, the intensity level of the corresponding grayscale pixel in the source image is compared with the threshold level of the corresponding cell in the dither tile [1]. The transformation of a grayscale image into a binary image is slow, due to the very large number of comparisons involved.

In order to improve halftoning speed, output halftone patterns can be computed in advance for each intensity level of gray. This enables the halftoning process to be parallelized and pipelined [2]. For each gray level, the screen elements are precomputed and stored in memory. Although this method is applicable to small dither matrices, it becomes unsuitable for large dither matrices, where the space in storage required by the precomputed pixels may become very great. Typically, for superscreens [3], where each screen is made of a million threshold values, storing the halftone patterns for 256 gray levels would require 32 Megabytes of memory.

Dithering algorithms which involve large dither arrays [3], such as exact angle superscreening, may require the halftoning process to be performed on the fly. Since one comparison is necessary for each output image pixel, halftoning with large dither matrices turns out to be a slow task. Half-

(1) kugler@gris.informatik.uni-tuebingen.de
(2) hersch@di.epfl.ch

Proc. IEEE Intern. Conf. on Application Specific Array Processors,
(Eds. Capello et. al.), IEEE Computer Society Press, 1995, 76-84

toning speed is affected by the huge amount of data which needs to be processed before sending the pagemap to a high resolution printer. The task of the proposed dithering coprocessor architecture is to increase the speed of the imaging process.

A rather analogous architecture to the prototype presented here was developed by Adobe. Their coprocessor operates in the PostScript language environment and acts as a display-list rendering coprocessor [4]. It converts a stream of drawing commands into an array of bits at a given resolution. Aside from other image rendering tasks such as shape filling, it also speeds up halftoning. The architecture presented here is restricted to halftoning and is not bound to the PostScript page description language environment. It directly produces a bitmapped image from a grayscale image and a given dither tile.

2: The dither threshold array and the dithering algorithm

In order to emulate halftones in a bitmapped image, the screen cells are arranged in patterns which are repeated periodically to cover the dither plane. Dithering rules [5] govern the visual appearance of the halftoned image. The human eye and brain integrate the tiny features of each pattern into an average grayscale tone or color.

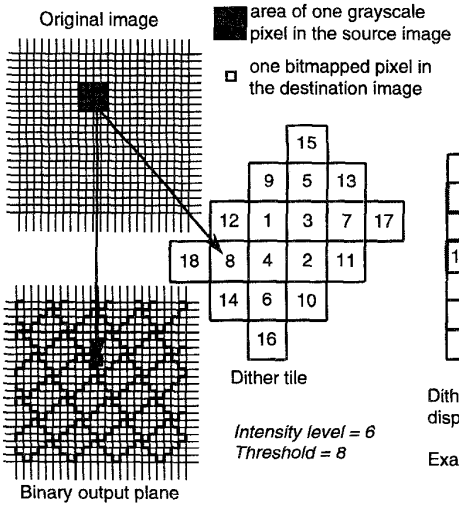


Figure 1 - The intensity level of the pixel in the original image is below the threshold and the corresponding output bitmap pixel is turned to be black.

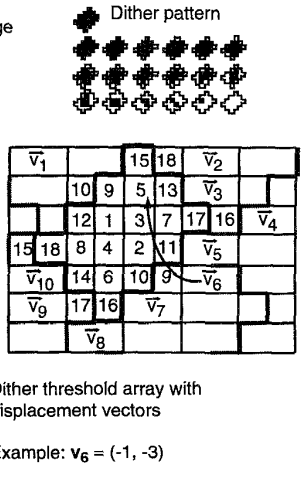


Figure 2 - The vectors outside the border of the dither tile point to threshold locations inside the dither threshold array (for example $\mathbf{v_6}$ points to cell 5 and $\mathbf{v_5}$ points to cell 15).

The halftoning process which is presented here is based on a regular grid and is known as ordered

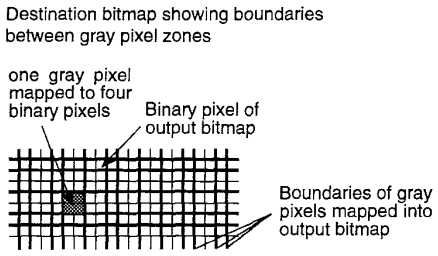
dithering. In the case of ordered dithering, the screen dot grows within a tile of limited size. A dither threshold array is associated to this tile. Each cell within the tile is assigned a threshold value: the intensity at which it will be turned on.

The halftoning process consists in scanning the output bitmap with a chosen dither threshold array and, for each output pixel, finding its corresponding locations both in the dither array and in the grayscale input image, comparing the two values, and writing the appropriate bitmapped pixel to the output image (Figure 1).

At runtime, the threshold values are taken from a dither threshold array. This array is divided into rows and columns which are scanned in the same way as the binary output plane. When moving from one pixel to its neighbour pixel in the output bitmap, we either move to the next threshold value on the right, or we jump from a position which is outside the tile to the appropriate position inside the dither tile. For this purpose displacement vectors are added to locations of the dither array positioned at the border of the tile. These vectors point to the position containing the next threshold value inside the dither array when moving out of the tile (Figure 2). The data structure for holding a screen tile forces the halftoning algorithm to cycle in the same tile during the halftoning process.

Let us recall that Holladay [6] proposed a rectangular representation for parallelogram shaped dither tiles. We have used a description for the dither threshold array which offers an economical solution for storing large dither tiles while providing an effective way of generating the output bitmap. It can be shown that Holladay's representation is equivalent to the description given here.

A grayscale pixel in the source image plane can be imagined as covering one or more bitmapped pixels of the binary output plane. Mapping the grayscale pixels to the corresponding binary pixels consists in finding the real number coordinates of the boundaries between grayscale pixels in the destination bitmap. All binary pixels whose pixel centers lie within the boundaries of a source pixel will be dithered with reference to this source pixel.



Source pixels	0	1	2	3	4	5	6	7	8	9	10	11	12	...
Destintion pixels	0	1	1	2	3	3	4	4	5	6	6	7	7	...
ε_N	-4	7	-1	10	2	-6	5	-3	8	0	-8	3	-5	...

Figure 3 - Example of the incremental method for mapping destination pixels to source pixels: d = 19, s = 11.

An incremental method is used for determining the grayscale pixel boundaries in the destination bitmap which is based on a Bresenham-like algorithm [7]. Finding the boundaries of gray pixels in the destination bitmap can be thought of as the more familiar problem of tracing a line in a raster display where the X axis represents the binary pixel locations in the destination image and the Y axis the grayscale pixel locations in the source image. The ideal mapping between binary pixels and grayscale pixels is represented by a straight line segment. When tracing a line on a discrete raster device, the computed grayscale pixel boundaries are either below, above or exactly on the drawn line. As we move from one destination pixel to the next one, we need to check if the next destination pixel corresponds to the current gray pixel or to the next gray pixel in the source image. In that case we have to read its intensity value, before comparing it with the current threshold value.

Supposing that the scaling factor (DestinationImageSize/SourceImageSize) is rational and can be reduced to an irreducible fraction d/s, let us define r = d-s. The iterative algorithm for computing the error term ε_N is the following:

$$\varepsilon_{N+1} = \varepsilon_N + s, \quad \text{if } \varepsilon_N < 0,$$
$$\varepsilon_{N+1} = \varepsilon_N - r, \quad \text{if } \varepsilon_N \geq 0.$$

Initially ε_0 is equal to -r/2. When moving from one destination pixel to the next one, ε_N is either incremented by s or decremented by r and the sign of ε_N is checked. If ε_N is strictly negative ($\varepsilon_N < 0$), the same gray pixel is used. On the other hand, if ε_N becomes positive or equal to zero, the next gray pixel's value in the source image is fetched from memory.

3: Hardware superscalar screening

Software-based screening typically requires that five to ten instructions be executed for each rendered device pixel. Even with carefully hand-optimized assembly code, the best that can be achieved with software-based solutions is five CPU instructions per rendered device pixel [8]. On RISC platforms, due to the smaller instruction set, the number of CPU instructions may rise to more than ten instructions per pixel. Our aim is to decrease the number of cycles which are required to produce one bitmapped output pixel.

FOR EACH scanline in the destination image DO
- initialize the pipeline
- make a copy of the start of scanline pointer (CopyThresholdArrayPtr)

FOR EACH ***COUPLE OF*** binary pixels in the current scanline DO

- read the information at the current position (ThresholdArrayPtr) in the threshold array
- if the information corresponds to a displacement vector, move to the position pointed at by the vector

- retrieve the ***GRAY LEVELS*** of the ***two*** corresponding pixels in the input image
- compare these two ***GRAY LEVELS*** with the two ***THRESHOLD VALUES***
- generate the ***two*** appropriate output bits
- write the output to the binary output image

END FOR

- update ThresholdArrayPtr to the position just below CopyThresholdArrayPtr

END FOR

Figure 4 - Pipelined tasks performed by the hardware architecture.

The basic coprocessor architecture performs two comparisons in parallel and simultaneously generates two bitmapped output pixels.

Reading a pair of threshold values and reading the grayscale pixels from a source scanline buffer can be done simultaneously (Figure 4). Once these values are known, they are compared, and the appropriate output pixels are generated.

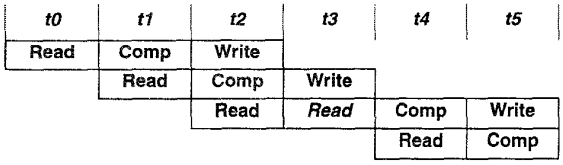


Figure 5 - Pipelined accesses to memory and comparisons.

In the worst case, the time taken by one iteration of the inner loop in the algorithm is two complete cycles: one cycle for reading the data from memory, comparing two gray levels with two thresholds and writing the output bits. One additional cycle may be required to read the threshold values if the value which has just been read corresponds to a displacement vector. By extending the mechanism described here, we could carry out multiple comparisons, such as four or eight comparisons simultaneously.

4: The operating environment

To synchronize the coprocessor with the host microprocessor and with the output device (the imaging engine), incoming and outgoing data is buffered in FIFOs (Figure 6).

The coprocessor works in parallel with a host microprocessor from which it receives the grayscale input image and a description of the dither matrix.

The current architecture requires an external dual-port memory of 2048 32-bit words for storing parts of the input image waiting to be processed. An external 8-bit wide FIFO is necessary for buffering the output data. The halftoning coprocessor interacts with the output device by signals indicating when the output buffer is half-full.

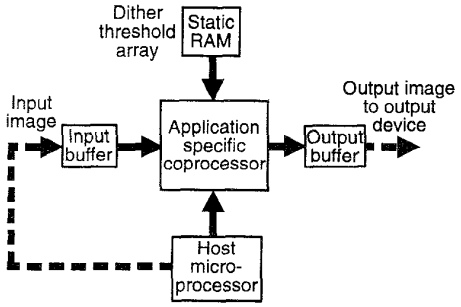


Figure 6 - General architecture.

5: The proposed hardware architecture

Before starting the halftoning process the host microprocessor stores the threshold array description in an external static memory. An 8-bit wide control register interfaces the coprocessor with the program running on the host microprocessor. The output image size (ImDstW, ImDstH) and a few other constants are sent to the coprocessor at initialization time. The source scanline buffer is a dual-port memory. This dual-port memory allows the host microprocessor to fill it with image data

without interrupting the work of the coprocessor.

At runtime, the coprocessor generates two binary output pixels in parallel by comparing a couple ($Gray_H$, $Gray_L$) of gray pixels from the source image with the current threshold values ($Threshold_H$, $Threshold_L$). A Threshold Buffer holds the next two threshold values fetched from the static RAM. If the fetched data corresponds to a displacement vector, the threshold values of the pointed location in the threshold array are retrieved. The boolean results of the two comparisons are loaded into shift registers. Once the shift registers are full, the result is written to a register before being sent to the output buffer (Figure 7).

Since the coprocessor may not work at the same speed as the host microprocessor or the imaging engine, handshake signals (InputBufferRequest, OutputBufferFull) from the input and output buffers result in correct synchronization between the coprocessor and the external devices. Two interruption requests (IRQ_A, IRQ_B) from the sequencer to the host microprocessor and to the output device are part of the synchronization primitives. IRQ_A asks the host microprocessor to

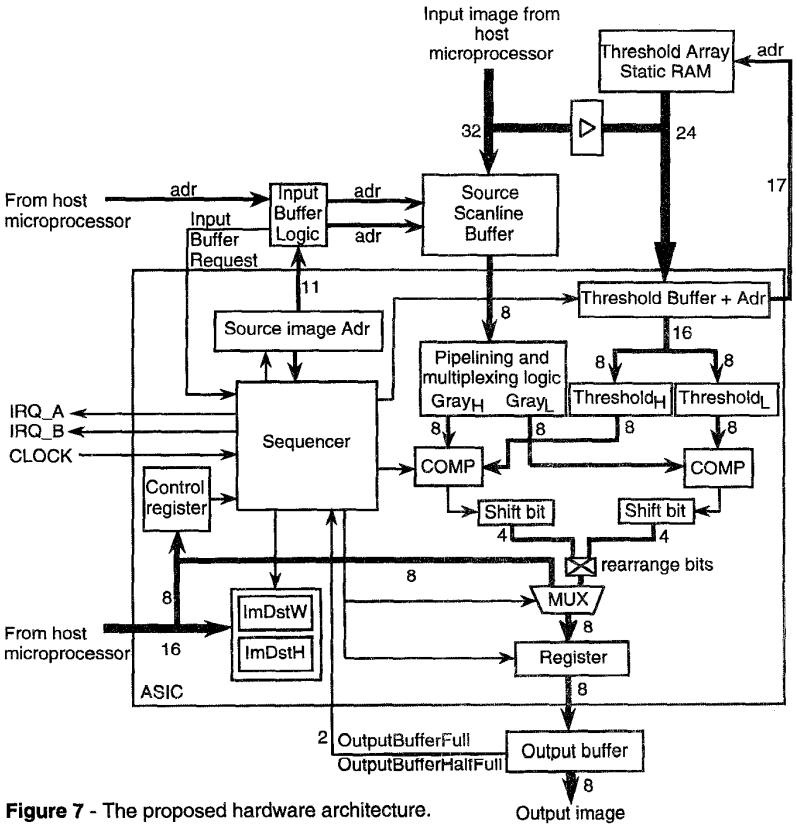


Figure 7 - The proposed hardware architecture.

put new data into the input buffer. IRQ_B tells the output device that data can be read from the output buffer.

The main operations (reading the source image pixels, reading the threshold values, comparing, writing the output bits) are pipelined and are executed in parallel.

A sequencer coordinates the flow of the main operations with the rest of the computations. It computes the boundaries of the gray pixels in the destination image, addresses the source scanline buffer and the threshold array and sends interrupt signals to the host microprocessor and to the output device.

The halftoning process involves three pipelined tasks: reading the grayscale pixels from the input buffer, reading the threshold values from the threshold array static RAM, comparing and writing the appropriate output bits.

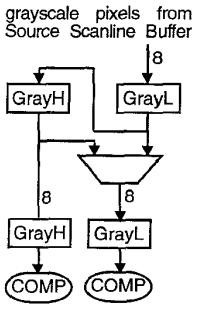
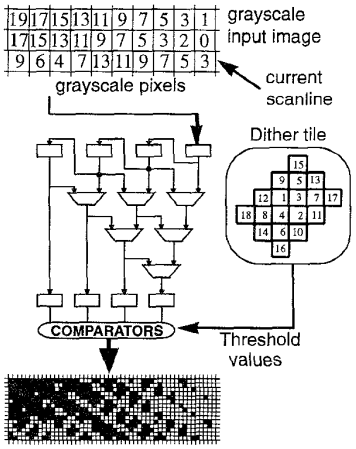


Figure 8 - The datapath followed by the grayscale pixels.

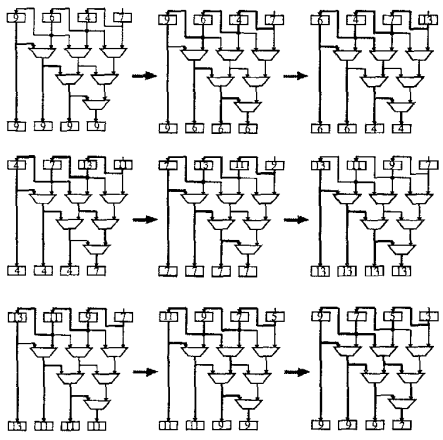
In order to guarantee that two bitmapped pixels are generated during the same cycle and to avoid a delayed cycle, the comparators are fed by a two level pipeline which buffers the upcoming grayscale values required for the next iteration (Figure 8). The pipeline stores the grayscale pixels before assigning them to the comparators.

6: A scalable architecture

Let us examine how the present architecture can be extended in order to carry out four or more simultaneous comparisons. Before comparison, the data from the input buffer must propagate through a pipelining and multiplexing logic, whose purpose is to map the grayscale pixels to their respective boundaries in the binary output image.



One input image pixel is mapped to 5x5 pixels in the output bitmap image.



Path followed by the grayscale pixels of the current scanline before reaching the comparators.

Figure 9 - Combinational logic controls the datapath before comparison.

The coprocessor buffers the upcoming grayscale pixels in a two-level pipeline. During the first stage of the pipeline, data is read from memory while, during the second stage of the pipeline, the registers are loaded with the data for the next set of comparisons. Figure 9 gives an example of the path taken by the grayscale pixels through the pipelining and multiplexing logic before comparison. An architecture with four comparators will only run at optimal speed if the average number of output pixels corresponding to one grayscale pixel is superior or equal to 4x4 binary pixels.

If one considers that one grayscale input pixel corresponds to 4x4 binary output pixels, the input buffer must be fast enough to feed the entry stage of the pipeline with one grayscale pixel per cycle. Assuming that one source image pixel covers 4x4 binary pixels, to simultaneously generate eight binary pixels using eight comparators, the input buffer either must work at twice the output buffer rate or must provide a larger bus (16-bit wide).

Let us model the performance of such a scalable architecture. Table 1 compares different architectures, each clocked at 20 MHz.

Comparators	Best-case number of cycles per binary pixel	**Average performance** generated output pixels per second
2	0.5	26 million
4	0.25	52 million
8	0.125	104 million

Table 1

The number of comparators which can efficiently work in parallel is limited by the throughputs of both the input stream and the output rate of the coprocessor.

Designing an architecture with eight comparators can be imagined if the target resolution is high (1200-2400 dpi) and the input resolution is low (150-300 dpi). The rule is to map 2x2 source pixels to one output clustered-dot screen element. When preparing an image scanned at 150 dpi for an output resolution of 600 dpi or more, where each screen element contains at least 8x8 pixels, it is worthwhile using an architecture with four comparators.

7: Current results and performance

The first prototype of a coprocessor has been designed to demonstrate the feasibility of such an architecture. It contains two comparators, working in parallel and has been implemented in a FPGA (Field Programmable Gate Array) from Xilinx. The circuit which was used is a XC3190 offering 320 CLBs (Combinational Logic Blocks), equivalent to 4000 gates. 150 pins were used to interface the circuit to the external memories as well as to the host microprocessor.

We have produced a slightly simpler, less efficient circuit than the fully pipelined architecture described here, but we give the results for the fully pipelined architecture, clocked at 20 MHz and extensively simulated with the schematics simulator provided by Viewlogic.

The best-case performance is two bitmapped pixels every 50 ns (one clock cycle). When the read cycle is lengthened (a vector is read instead of a couple of threshold values), the overall time for generating two output pixels becomes 100 ns (2 clock cycles). Thus, the average performance of the coprocessor reaches 26 million bitmapped pixels per second. Table 2 shows the execution times taken to halftone an A4 page at different output resolutions. An optimized but purely sequential software implementation of the halftoning algorithm was programmed in C.

Output image resolution	Two comparator based coprocessor	SUN Sparc-2 software implementation
300 dpi	0.25 s	4.9 s
600 dpi	1.0 s	19.7 s
800 dpi	1.8 s	35 s
1200 dpi	4.0 s	1 mn 19 s

Table 2

With a standard cell-based VLSI technology, the coprocessor could include input and output buffers. Such a scalable architecture could be remarkably cheap and would run much faster than current software-based solutions running on powerful RISC processors. It would be suitable for a printer optimized for printing color images at high speed. Such printers form an integral part of modern digital color copiers.

8: Conclusion

Exact-angle superscreen dithering requires large dither tiles. Since storing precomputed screen elements for each intensity level would require too much memory, dithering must be executed on the fly at halftoning time. For this purpose, a dithering coprocessor is presented which generates halftoned images at high speed.

The proposed hardware architecture is built on a pipelined and scalable design. To maximize processing speed, the proposed coprocessor uses separate FIFO memory input and output buffers, and provides a complete pipeline for performing one comparison per comparator per cycle.

When clocked at 20 MHz the coprocessor is able to halftone images at a rate of 26 million binary output pixels per second and outperforms a standard SUN Sparc-2 software implementation by a factor of 20.

An analysis of the design methodology reveals that making multiple comparisons in parallel decreases the time required for halftoning in a linear way. Based on an implementation with two comparators, a model of an architecture providing four or more comparators has been developed. The practical limits of such an architecture have been evaluated: the number of comparators able to work efficiently in parallel is limited by the ratio between input and output image sizes. We show that a four comparator architecture is well suited for exact-angle screening at high resolution.

9: References

1. J. Foley, A. van Dam, S. Feiner, J. Hughes, *Computer Graphics: Principles and Practice,* Addison-Wesley, Reading, Massachusets (1990).
2. M. Morgan, R.-D. Hersch, V. Ostromoukhov, Acceleration of Halftoning, *SID Digest of Technical Papers,* vol. 24, 151-154, (1993).
3. Peter Fink, *PostScript Screening: Adobe Accurate Screens,* Adobe Press, Mountain View (1992).
4. Adobe "PixelBurst" chip speeds RIPs, *The Seybold Report on Desktop Publishing,* 7(2), 39-42 (1992).
5. Robert Ulichney, *Digital Halftoning,* MIT Press, Cambridge, Massachusets (1987).
6. Thomas M. Holladay, An Optimum Algorithm for Halftone Generation for Displays and Hard Copies, *SID Digest of Technical papers,* vol. 21, 185-192 (1980).
7. Jerry R. Van Aken, Carrell R. Killebrew, Better bitmapped lines, *BYTE,* vol. 3, 249-253 (1988).
8. PixelBurst coprocessor (comments), *The Seybold Report on Desktop Publishing,* 7(5), 36-38 (1993).