Nimbus: An Integrated Display Chip
Bart Locanthi
Rae McLeIIan
AT&T Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

It appears that the race between circuit density and flat display size is being won by the semiconductors. With SRAM technology now delivering more than a million bits per chip, it is now feasible to integrate display memory with display control.

The circuit described here contains a two-port SRAM array, one port being serial access combined with duty cycle modulation for grey scale display on LCD's. The memory array itself is synchronous to the processor port, providing low latency high bandwidth access for image manipulation.

1. Introduction

Another pair of lines have crossed. Since the introduction of the video DRAM nearly a decade ago, memory size has been increasing faster than display size. While flat display resolution has been improving steadily, producing the high voltage D/A converter arrays necessary for analog grey scale on liquid crystal and electroluminescent displays (LCD/ELD's) is a continuing difficulty. A moderately large LCD might be 640x480, and 4 bits of grey scale is just beyond the limit of what can be achieved with duty cycle modulation. However, the resulting 1.2 Mbits certainly is within the range of what can be achieved in a single chip SRAM.

Managing displays has always been a messy business. To refresh its display, a display controller must either steal cycles from a system bus or manage a private display memory (and still provide access to the system processor). In either case, rising bandwidth requirements and the support chip market essentially require that display memory be composed of special memory chips.

Nimbus represents a new approach to the display problem. Combining a system bus interface with the actual display memory, shift out logic, and display control is a moderately challenging integration task, yet the result is a low pin count system solution to a perennially messy and onerous task.

A pleasant byproduct of this architecture is the high bandwidth access to the display memory available to the processor. Modern microprocessors such as Crisp [1] have synchronous, single cycle and block access buses. Providing memory access at bus speed allows the exclusion of architectural tricks such as block fill that adversely affect both density and power distribution as well as the ability of programmers to make use of such features. Nimbus presents no architectural barriers to simplicity and speed in graphics software.

2. Building Blocks

Inside Nimbus are two processes. The first process presents the display memory to the system bus. The second process runs the LCD and periodically accesses the display memory on its behalf. Although the display process must have priority in order to ensure uninterrupted information display, the memory control is synchronous to the system bus.
The display control comprises horizontal and vertical counters and limit registers. In principle the memory is accessed once per scanline pair. Instantaneous current considerations argue for more frequent, less global accesses. Balancing these factors is outside the scope of this paper.

Nimbus implements grey scale on bilevel displays through a combination of temporal and spatial dithering. On any given frame, the image sent to the display is a spatially dithered rendering of the original. The spatial dither function changes for each frame so that the average intensity of each pixel in the display corresponds to its value in SRAM.

A final Nimbus building block is the bit swizzling ALU, which performs pixelwise bit manipulation and arithmetic operations useful for grey scale graphics yet specialized enough to be unlikely additions to a microprocessor instruction set.

2.1. SRAM Array

An integrated display chip should contain as much storage as feasible. However, its architecture should provide for expansion and should be uniform across several display sizes. Physically, Nimbus is specified to be a 640x480 (maximum) window of a 1024x512x4 logical frame. Making the logical row size a power of two allows trivial separation of addresses into row and word (within row) parts:

<table>
<thead>
<tr>
<th>proc addr</th>
<th>row</th>
<th>word</th>
<th>byte</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It is convenient but not essential to make the address increment between scanlines a power of two. Our solution to the LCD memory organization problem is twofold. First, processor addresses are adjusted so that addresses for like locations in the two display fields differ only in the high address bit:

field::row = (row>HH) ? (row+256-HH) : HH

where HH is one half of the display height, for example 240 for a 640x480 display. Second, we move the field bit so that successive halfwords in the SRAM fabric are in different fields.

<table>
<thead>
<tr>
<th>sram addr</th>
<th>row</th>
<th>word</th>
<th>half</th>
<th>field</th>
<th>byte</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

These two measures allow Nimbus to present the illusions of contiguous addressing within and even spacing between rows, and simplicity in bitmap graphics software is preserved [3].

2.2. Memory Control

Nimbus memory control runs synchronous to the system bus and supports single and quad word transfers. The display section is presented with an asynchronous interface.

LCD timing differs from CRT timing in interesting and complementary ways. Having no resonant circuitry, horizontal sync/retrace needn't be a significant, or even constant fraction of the horizontal scan time.

With CRT displays it is sufficient to post a display request at the beginning of horizontal sync and expect a memory cycle to complete before the next scan. With LCD's the next scan can begin almost immediately. While in principle Nimbus could block the display process while performing an access for the system bus, in practice a small constant horizontal "sync" delay of a few memory cycle times is sufficient to ensure an uninterrupted pixel stream to the display.

2.3. Display Control

LCD control is relatively simple. A chain of counters keep track of horizontal and vertical scanning, making asynchronous requests to the memory control section to keep the shift out pipeline full.

2.4. Shift Out Logic

Liquid crystals are relatively slow devices with binary brightness control. In particular they are typically scanned much more rapidly than they can respond. This allows the implementation of grey scale through duty cycle modulation.

Consider a stream of 4 bit pixels being shifted out from the memory fabric. Applying a suitable time and space varying dither function to these intensity values can give the appearance of grey scale display without temporal or spatial artifacts.

Nimbus uses the standard 4x4 ordered dither matrix \(D(x,y)\) in conjunction with a permutation (bit reversal) of the frame count modulo 16 \(F(t)\). The displayed pixel value \(P(x,y,t)\) is a function of these and the intensity \(I(x,y)\):

\[ P = (I > ((D + F) \& 16)) \cdot 1 + 0 \]

This computation can be performed in parallel at low speed or pixel by pixel followed by serial to parallel con-
version. Use of the bit reversed frame count means that pixel values will be modulated at a frequency approaching the refresh rate rather than a constant fraction of the refresh rate. Use of the dither matrix means that neighboring pixels in a constant intensity display will be modulated out of phase with each other.

One 32 bit word from the SRAM fabric thus results in 1 bit intensities for 8 pixels, 4 in each of the two fields, as required by the physical interface to the LCD

<table>
<thead>
<tr>
<th>upper</th>
<th>lower</th>
</tr>
</thead>
</table>

The maximum internal clock rate used in feeding a 640x480 LCD refreshed at 60 Hz is about 24 MHz. The clock rate for the resulting byte stream of packed binary intensities is only 3 MHz.

2.5. Bit Swizzling and Arithmetic

In grey scale bitmap graphics we frequently need to perform operations on bitmaps that have different pixel depths. Pixelwise arithmetic is useful in implementing filtering as well as the grey scale analogies to [S^-D, S^1-D, S^S&D].

Swizzling bits and doing arithmetic with selectively broken carry chains are fundamentally difficult and time consuming things for general purpose processors to perform. More so than the typical bitblt() assist mechanisms like shifting and masking [5], these operations should be the first to be put in any hardware assist for grey scale bitmap graphics.

Nimbus includes a bit swizzling grey scale accumulator. This is an independent unit attached to the system bus, and as such is available for operations besides those having to do with the LCD.

2.5.1. Pixel Arithmetic

The TMS34010 [4] defines arithmetic codes for pixelwise addition, subtraction, addition and subtraction with clamping at 0 and all 1's, min, and max. These six operations must each be defined on both 2 and 4 bit pixel depths.

2.5.2. Bit Swizzles

Pick a display pixel depth, say four bits. In addition to using grey scale fonts of the same depth, we will also wish to display image data and use bitmap fonts. In general, we wish to be able to convert data between all possible pixel depths - and maybe some more.

The TMS34010 implements 1->N expansions in hardware; other systems use table lookup. The 1->2 case is unsurprising:

This definition presumes that users want bitmap input data to swing the entire output range. However, if the intent of the expansion is to open up some "computation room" in each pixel, the input data should wind up in the least significant bit of each output pixel. In this case the problem isn't serious - unwanted pixel replications can be masked off as required.

What should a 2->4 expansion look like? One possibility is

in which the full range to full range mapping of the previous example is preserved. Again, unwanted pixel replications can be masked off.

Pixel contraction requires a more serious policy decision, since the operation discards data. Assuming the desire is truncation and that we want to retain the high order bits of each pixel, the 4->2 contraction looks like

We can get the low order bits instead by shifting the source left two bits first. Rounding may be implemented by preceding the contraction by a saturated pixelwise add of 0x22222222 to the source.

Allowing for five pixel sizes (1, 2, 4, 8, and 16 bits) yields a requirement for twenty (5^2 - 5) swizzle types. Add to that the various pre- and post-processing operations that might be desired, and the result is an alarming number of possibilities.

Perhaps a more conservative approach would be useful. Once we admit to the possibility of additional pro-
cessing, it makes sense to re-examine the notion of bit swizzling with distillation in mind.

Consider a 4→2 swizzle that conserves bits:

One way of looking at this is that the result contains both high and low order bits with masking left to the user. Another way is to think of the result as 32 swizzled bits that could be the 4→2 contraction of 64 bits in two words x and y combined as

\[(x \& 0xCCCCCCCC \mid (y>>2) \& 0x33333333)\]

If nothing else, using pre-processing to pack source data can amortize the cost of getting to the Nimbus swizzle box from the CPU.

Even more striking is the realization that this conserving 4→2 swizzle is also a 2→16 swizzle. In fact, each of the twenty 32 bit swizzles has a different swizzle as its inverse.

We can reduce the number of swizzles further by noting that some of them are compositions of two or more swizzles. Perhaps someday we will be able to talk someone into putting a few swizzle instructions into their processor's instruction set.

2.5.3. Architecture

All accumulator operations are of the form \[ACC = ACC \text{ op SRC}\] where ACC is the previous value of the accumulator and SRC is written from the system bus. The operation to be performed is determined by the address supplied.

To the programmer, this box looks like a set of read-write registers, e.g.

```c
struct accum {
    ulong value;
    ulong swiz12;
    ulong swiz14;
    ulong swiz24;
    ulong swiz21;
    ... ulong sadd4;
    ulong sub4;
    ulong min4;
    ulong max4;
} *ACC;
```

and is invoked by sequences like

```
ACC->swiz21 = src;
dst = ACC->value;
```

Only one of these registers, value, actually contains data. The others merely front for it, either performing the swizzling for write data in, or swizzling its contents on reads. This latter function is useful for swizzle composition.

While trivial usage may appear somewhat clumsy, it should be remembered that the operations performed are anything but trivial. In addition to the simplicity of its implementation, this architecture adds but a single word of state to a processor's context.

3. Summary

Integrating storage and display control for LCD's brings some particularly happy architectural results. Since typical LCD environments emphasize both low power and low voltage, CMOS SRAM technology is the preferred medium for implementation. While serial access bandwidth and the resulting low latency processor access have so far been province of video DRAM technology, integrating the memory array and display logic together brings these advantages to the low power CMOS domain.

4. References