The Architecture of the LR33020 GraphX Processor: A MIPS-RISC based X-Terminal Controller

Sanjay Desai
Microprocessor Products Group, LSI Logic Corp.
1525 McCarthy Blvd.
Milpitas CA 95035

Abstract
This paper describes the architecture and implementation of a MIPS-RISC based application specific microprocessor designed specifically for X-Window Terminals. It consists of a graphics coprocessor integrated with a MIPS-I (R3000) compatible CPU core along with system level functions found in a typical X-Terminal. The design of the graphics coprocessor has been optimized for accelerating low level graphics operations typical in X Windows applications. It is implemented as coprocessor 2. A set of coprocessor 2 graphics instructions process pixel data fetched by dedicated DMA channels through a high bandwidth (64 bit) memory interface. The combination of the graphics coprocessor and high bandwidth memory interface result in a very high performance. The presence of system level functions on the same die as the CPU and graphics engine make for a very highly integrated processor - effectively a system on a chip.

1 Introduction
Leveraging the company's ASIC technology, the LR33020 "GraphX" processor is an application specific processor designed to provide a high integration, high performance solution for X Terminal applications. By integrating many of the peripheral functions needed in an X terminal design with a MIPS-I R3000 compatible CPU core (taken from an earlier chip (1)) and a high performance graphics coprocessor, the LR33020 is the world's first embedded processor specifically designed for X terminal applications.

The main motivation behind the project was to be the first to address the specific needs of a potentially big market for embedded microprocessors. The X-Terminal system is rapidly gaining popularity in terminal applications in both the technical and commercial marketplaces. Most X-Terminal systems to date have been designed using a general purpose CISC microprocessor along with some sort of graphics hardware acceleration to aid the X-Server in graphics intensive operations. Also needed are ASICs/logic to interface to a CRT display device. The LR33020 has sought to integrate all these separate functions in a single chip.

Since the primary function of an X-Terminal system is to run the X-Server and act as a display device for an application running on a remote machine, the graphics processing capabilities are of primary importance. However, since a typical X-Server is about 50% general purpose code and 50% graphics code, a fast general purpose CPU is also needed for high performance. The LR33020 addresses both these needs by providing a proven high-performance industry standard RISC CPU core with a dedicated graphics coprocessor and a high-bandwidth (64 bit) memory interface. Another important factor in a X-Terminal system is the cost. By integrating CPU, graphics and system functions the LR33020 provides a cost-effective solution.

2 Overall Chip Architecture
Fig.1 shows a block diagram of the LR33020 "GraphX" processor. It consists of a MIPS-I (R3000) compatible CPU core with 4K bytes of Instruction Cache and 1K bytes of Data Cache, a Graphics Coprocessor consisting of a bitblt processor and dedicated DMA channels, and integrated system functions. The CPU core is a standard cell implementation of the MIPS R3000 processor without the memory management functions. It has been taken from an earlier chip - the LR33000, a MIPS-RISC based embedded controller (1). The integrated system functions provide most of the peripheral functions needed in an X Terminal design. These include a video controller with video FIFO and a video DMA channel, a DRAM/VDRAM controller, a four word deep write buffer (which allows for single cycle CPU store-word operation), a bus arbiter, an SRAM and I/O controller, a PS/2 style serial (keyboard) port and two counter/timers. The video controller provides video signal generation for both a monochrome system and a color (RAMDAC based) system. The video FIFO serves to serialize the pixel data stream in a DRAM.
based frame buffer or can serve as a hardware cursor storage in a VRAM based frame buffer. The Graphics coprocessor, implemented as coprocessor 2, is described in detail in succeeding sections.

![Block Diagram of the LR33020](image)

### 3 Graphics Coprocessor

The Graphics Coprocessor is an integrated Bit-Blt processor and Graphics DMA Controller that provides hardware acceleration and support for common 2-D graphics operations like bit-bits. Instead of changing the base R3000 instruction set architecture, the LR33020 controls the graphics rendering with coprocessor 2 instructions. Opcode space left for coprocessor support in the R3000 instruction set is used to define the new graphics instructions. These instructions are collectively labeled as step instructions because they sequentially step through the pixel data in the frame buffer. This allows construction of complex, higher level graphics routines without requiring the extra hardware support for higher level graphics operations like line-draw.

Due to the tight coupling of the coprocessor and the CPU, coprocessor step instructions are issued in-line with the main CPU pipeline. Each step instruction goes through the first three stages of the CPU pipeline before it enters the coprocessor pipeline. The coprocessor pipeline is a two stage pipeline with an additional instruction buffer. This means that up to three step instructions can be in the coprocessor without having to stall the CPU. A fourth step instruction, however, will result in a CPU stall until at least one instruction is retired from the coprocessor.

The Graphics Coprocessor has an independent path to the external memory/frame buffer through the dedicated DMA channels. This, coupled with an on chip DRAM/VRAM controller and a high bandwidth (64 bit) external memory interface, ensures that the coprocessor is fed with the pixel data at speeds limited only by the memory/frame buffer access times while allowing the main CPU to run out of the on chip caches. The CPU, however, is stalled when there is a cache miss and a graphics DMA is in operation.

### 3.1 Coprocessor Datapath and Graphics DMA

The dedicated 32 bit datapath of the coprocessor, shown in fig.2, is designed to do pixel extraction and alignment, color expansion (by adding color to monochrome bitmaps such as text characters), transparency masking, masking of pixel data at object boundaries and pixel planes, and 16 different logical functions on source and destination pixel data. The functions in the datapath are controlled by special control registers. The datapath also implements direct VRAM interface support through a bit-swizzling function which ensures correct bit alignment during VRAM block writes. The source and destination DMA channels feed the datapath with the pixel data from the frame buffer and write the processed pixel data to the frame buffer under control of the step instructions.

![Graphics Coprocessor Datapath](image)

### 3.1.1 Graphics DMA
The DMA channels have to be initialized with the appropriate addresses and the pitch values before issuing any step instruction. Dedicated coprocessor 2 registers hold these values.

The source data graphics DMA channel fills the four word source FIFO directly from the frame buffer/memory (bypassing the on chip data cache) using burst mode read. Coprocessor 2 step instructions initiate this fill whenever the queue is empty. The DMA channel can calculate the source data address across lines and line wraps. Thus, during source to destination rectangular bit-blt the DMA automatically takes care of the pixel addressing once it has been initialized with the beginning address and the pitch.

The destination graphics DMA channel fetches (optionally) and stores destination pixel data from/to the frame buffer (bypassing the data cache and the write buffer). In VRAM based systems up to four words of pixel data can be stored in a single write operation using the VRAM block-write feature. Also, in DRAM based systems, a constant source word can be written to four consecutive locations with a burst-write instruction, which accelerates solid fill operations.

### 3.2 Graphics Instructions

Low level graphics routines are written using a combination of the general CPU instructions and coprocessor 2 instructions. The coprocessor instructions consists of the instructions for loading and reading the coprocessor registers and the newly defined step instructions.

**Loading and reading coprocessor registers:** The R3000 instruction set has six instructions for loading and reading coprocessor registers. These are LWC2, SWC2, MFC2, MTC2, CFC2 and CTC2. Using these instructions, software can transfer data between CPU registers and the coprocessor registers or between the data cache/external memory and the coprocessor registers.

**Step instructions:** Four dedicated coprocessor instructions control the pixel processing and movement. These instructions have been designed to most efficiently carry out the common case bit-bit operations yet provide flexibility to do more complex graphics operations. These instructions are as follows:

- sstep
- sbstep
- bstep s(b), l, r
- wstep s(b), l, r, (b)four

A brief explanation of the function of each instruction is given below. The sstep and sbstep instructions are used primarily for initialization. The wstep normally forms the core of a low-level coprocessor assisted graphics routine. The four step instructions include five optional flags which set various conditions. If the s or sb flags are set with the wstep or bstep instructions, the functions of the sstep or sbstep instructions are performed before the wstep or bstep functions. The l and r flags are used to apply the left and right boundary masks to the pixel data. They ensure that only the pixels in the desired region are processed and those outside the boundary of the object being drawn are left unchanged.

- **sstep**: Steps to the next word in the source queue. If the source queue is empty, initiates another DMA fetch of the pixel data to fill the source queue.
- **sbstep**: Clears the source queue, steps to the beginning of the next source line, and initiates DMA fetch of pixel data to fill the source queue.
- **bstep s(b), l, r**: Writes the processed destination pixel data back to memory and steps to the beginning of the next source line. If the s flag is set, a sstep instruction is also performed. If the sb flag is set a sbstep instruction is also performed. If the l flag is set, the left mask is applied. If the r flag is set, the right mask is applied.
- **wstep s(b), l, r, (b)four**: Writes the processed destination pixel data back to the memory. It optionally fetches the destination pixel data if a read-modify-write operation is to be performed (determined by a control bit). If the s flag is set, an sstep instruction is also performed. If the sb flag is set an sbstep instruction is also performed. If the l and r flags are set, the left and right masks are applied respectively. If the four flag is set, four (constant) words are written to the memory in a block fashion. If the bfour flag is set a VRAM block write operation is done.

**3.3 Programming the graphics coprocessor**

Programming the graphics coprocessor is a two step process. First, the coprocessor control registers are initialized. These include the skew value, the shift value, the foreground and background colors, the left and right mask values, the pixel format, the plane-mask value and the logical function value. The source and destination DMA channels are also initialized with the correct addresses and pitch. The second step in the programming process is writing the correct sequence of step instructions to achieve the desired pixel movement operation.

### 4 Performance

Due to the tightly coupled nature of the graphics coprocessor and the presence of a 64 bit data path to the
memory through the on chip DRAM controller, graphics routines can usually be coded to process and move the pixel data at the maximum speed available from the DRAMs. The step instruction pipeline allows the processor to work ahead of the actual memory transfers without stalling. Table 1 shows the relative performance of the LR33020 for common 2-D graphics operations compared to other industry standard systems. Note that in Table 1 the LR33020 is compared against full blown Unix workstations. What the table shows is that LR33020 offers equivalent or comparable performance at a much lower cost.

5 Integrated System Functions

The integrated system functions (fig.1) of the LR33020 bring all the system components in a X-Terminal system (with the exception of the network controller function) on the same die as the cpu. As shown in fig.3 this affords a very low chip count to implement an X-Terminal system. This example shows a monochrome X-Terminal system. A color X-Terminal system would have a RAMDAC to interface with the CRT.

6 Implementation

The chip is implemented in a standard cell based technology (1 micron drawn, 0.7 micron I-effective) with ASIC design tools. The CPU core (consisting of the integer pipeline and the caches) was taken from an earlier chip. The functions outside of the CPU core were designed and integrated with the CPU core to form the chip. Logic synthesis tools were used to design most of the chip. It has approximately 500,000 transistors with the CPU core taking up about 420,000 of those transistors. Memory compilers were used to generate all the memory structures on the chip. The chip was designed from concept to silicon by a team of five engineers in less than ten months.

7 Conclusion

Implementing non-standard functions as coprocessors, rather than changing the base processor architecture, is a clean and fast way of customizing existing RISC processors for specific applications. The presence of the graphics coprocessor coupled with a high bandwidth memory interface results in a significant graphics acceleration on the LR33020. Simulations have shown up to three times performance improvement in some of the inner loops of bit-blt routines and in character draw routines as compared to the same routines running on just the MIPS CPU without any coprocessor support.

8 Acknowledgements

The author wishes to acknowledge the significant contributions of Bob Caulk, James Clifford Wei, Jay Patel, Simon Moy, Mark Kwong and Ray Peck in the design and verification of the LR33020.

9 References

(1) The architecture of the LR33000: A MIPS compatible RISC processor for Embedded Control Applications; Moshe Gavrielov, et al., ICCD’91

(2) LR33020 GraphX Processor User’s Manual

<table>
<thead>
<tr>
<th>Object</th>
<th>LR33020-64</th>
<th>LR33020-32</th>
<th>Magnum</th>
<th>DEC68b</th>
<th>DEC68b</th>
<th>HP</th>
</tr>
</thead>
<tbody>
<tr>
<td>(obj/sect)</td>
<td>33Mhz</td>
<td>33Mhz</td>
<td>33Mhz</td>
<td>33Mhz</td>
<td>33Mhz</td>
<td>66Mhz</td>
</tr>
<tr>
<td>10x10 Solid Fill</td>
<td>147</td>
<td>128</td>
<td>39</td>
<td>88</td>
<td>145-213</td>
<td>278</td>
</tr>
<tr>
<td>10x10 Stippled Fill</td>
<td>101/100</td>
<td>95/100</td>
<td>2.6</td>
<td>34</td>
<td>96-155</td>
<td>136</td>
</tr>
<tr>
<td>10x10 Opaque Stippled Fill</td>
<td>105</td>
<td>34.9</td>
<td>31</td>
<td>96-129</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>10x10 Copy</td>
<td>55.4</td>
<td>10.3</td>
<td>1.8</td>
<td>1.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100x100 Solid Fill</td>
<td>8.5</td>
<td>1.8</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100x100 Stippled Fill</td>
<td>1.7</td>
<td>0.7</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100x100 Copy</td>
<td>2.1</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Relative Performance of the LR33020

Fig. 3 An example Monochrome X-Terminal System using the LR33020

Acknowledgements

The author wishes to acknowledge the significant contributions of Bob Caulk, James Clifford Wei, Jay Patel, Simon Moy, Mark Kwong and Ray Peck in the design and verification of the LR33020.