SOFTWARE TOOLS FOR MICROPROGRAMMED GRAPHICS PROCESSOR DESIGN

Lawrence D. Finkel

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

As bit map display systems strive for higher resolution, the ability to maintain a high bandwidth update rate is constrained by the need to refresh the screen. In order to make greater use of the available update memory cycles, a microprogrammed graphics processor, the BitMover, is being designed to execute high speed pixel calculations. Standard microprocessors are too slow to meet the design goals of this project.

Controlling the BitMover requires a horizontal microinstruction word of approximately 94 bits is required. Debugging microprograms at the hardware level with a microinstruction format of this size is difficult and costly. To circumvent this, software has been written to simulate the execution of microinstructions. It consists of a series of software routines in which the chip inputs and outputs comprise the parameters passed to the routines. A graphics mode has been implemented that eases the debugging process.

1. Introduction

A new bit mapped CRT terminal is being designed to achieve high resolution, and high graphics throughput at a moderate cost. Specifically, 1024 lines by 1280 pixels per line will be displayed. Supporting such a high resolution, non-interlaced image requires a video bandwidth of over 100 MHz. Since many of the available memory cycles will be spent refreshing the display, the time available for display memory updates will be severely constrained. A means must be found to make the greatest possible use of the available memory update cycles.

To effectively make use of update memory cycles, a graphics processor, the BitMover, is being designed to implement graphics algorithms. Perhaps the greatest allure the bit map display has is its ability to do high quality graphics. To achieve this, the bit map display stores intensity information for every pixel, irrespective of its neighbors. This is contrary to normal alphanumeric terminals, in which the smallest addressable region is a character cell. Bit map terminals can thus draw complex graphical entities by individually turning pixels on or off in the displayed image.

The BitMover off-loads host generated graphics commands and calculates the addresses of pixels requiring modification. Common algorithms for point, line, circle, arc, and area fill are provided in microcode. Also, the architecture was designed to allow fast area copy operations, to obtain high speed scrolling. Note that the size of display memory is 2K by 2K pixels. Non-displayed memory locations can be used for storage of fonts, or portions of obscured windows. An address counter giving the x,y location of the upper left hand corner will be used to determine which area of memory is currently being displayed.

Due to the high speed needed by the BitMover in order to use 100% of the available memory cycles, the use of conventional microprocessors was ruled out. At the other extreme, special purpose hardware would require a long and costly design time and would present a difficult environment in which to make changes to the system. So, a microprogrammed system, representing the best cost/benefit implementation was chosen. This can be designed using off the shelf parts, and yet still meet the speed requirements for our system.

Microprogramming, though ideally suited to dedicated device designs such as the BitMover, presents unique problems for code generation and debugging. Since the microcode development for the BitMover is going on in parallel with the hardware development, the algorithms cannot be tried on the BitMover itself. Even with the hardware

complete, debugging is difficult, often requiring special purpose test equipment. For these reasons, a microprogram simulator is being developed, not only to debug microcode, but also to test alternative machine architectures.

2. The BitMover Architecture

The general layout of the BitMover is shown in Figure 1. The microcode memory is a 4K by 94 bit writable control store implemented with fast static RAMs. An input fifo is loaded by the host system with the graphics opcodes and operands. The numerical value of these opcodes is then used as an index into a jump table of graphic subroutines. This jump table contains the starting addresses of the desired subroutines. Values from the BitMover can be returned to the host system by means of the output fifo. The microprogram controller being used is the Advanced Micro Devices (AMD) 2910-1 high speed sequencer. Multiway branch logic has been added to the address lines of the sequencer in order to implement four, eight, or sixteen way branching. A condition code multiplexor chooses one of the many possible test conditions for use by the next address logic of the sequencer. The ALU for the BitMover is the AMD 29116 bipolar microprocessor. A special MUX has been added to the instruction inputs of the 29116 to allow operations to have different source and destination registers. Also of interest are the AMD 2940 DMA generator chips, labeled as source and destination X,Y registers. These chips are being used as address counters for the graphics algorithms. The incremental nature of the graphics subroutines can be applied to the function of the 2940's to produce pixel addresses for the display memory. The chips allow an initial address to be loaded, and then incremented or decremented as required to compute the next pixel address. The 2940's can also be disabled for calculations in which the pixel address is unchanged. A done signal is generated by the 2940 when execution of an algorithm is complete. Lastly, a pipeline register is shown at the output of the microcode memory. This allows for overlapped fetch and execution, and faster throughput. The data path between the ALU, mapping RAM, source and destination registers, and fifo is provided for by the sixteen bit wide YBUS.

3. The Simulator

The architecture shown in Figure 1 is not necessarily the final design for the BitMover. As stated, a simulator should allow hardware modifications, and alternative design experiments with a minimum of software changes. To accomplish this, flexibility must be incorporated that separates the description of the machines structure from the simulation routines of its components. This separation in the BitMover simulator is achieved by using subroutine building blocks, in which each subroutine block simulates a given chip in the design. In some cases, notably the bit-slice cascaded 2940s, no attempt has been made to separate the individual slices. Using these device modules, different machine architectures can be tried merely by the order in which the modules are called. Care must be taken to insure that the architecture designed is realizable purely from a timing standpoint.

A key goal in designing the simulator was to minimize software changes that occur as a result of hardware design modifications. This is true for microinstruction format changes in which it is undesirable to have to change large amounts of software to investigate changes. The BitMover simulator requires changes to only one routine in order to change the microinstruction format. The pipeline register routine formats the binary microcode and maps that into microorders which the rest of the software then uses. While the microinstruction is 94 bits wide, the simulator allocates storage for 128 bits. This allows extra bits to be placed in certain fields to set breakpoint flags, signal instruction literals, and control other aspects of the simulator. A different microinstruction format is thus convenient to implement and extra bits are available for additions and diagnostics to be included.

The simulation routines can be thought of as modules, in which each module simulates a physical device, or logical hardware unit. This software architecture yields many advantages. A library of device descriptions can be created which can be used on many different designs. And, the structure of the microinstructions is correlated with the hardware units being controlled.

Since the internal gate level status of the BitMover's chips is unavailable for any kind of debugging, no attempt has been made to simulate circuit functions at this level. A convenient approach has been to declare global variables for the pinouts of the chips. The inputs and outputs of a chip or module are thus available at all times to the debugging routines.

4. User Interface

A typical debugging session begins with a file of manually produced BitMover microcode. The use of a microcode compiler was ruled out since algorithm speed was the most important factor. A microassembler then generates the binary microcode from the mnemonic assembler file. A postprocessing program is run on the resulting file to modify it to reflect features of the BitMover not provided for by the original assembler. The two features requiring postprocessing are instruction literals, which are operands placed on the instruction inputs of the ALU, and multiway branch address alignment, in which it is necessary for certain lines of microcode to be placed at even numbered addresses.

Because the format of the BitMover microcode differs from the simulator format, a reformatting step is executed when the simulator is first called. Even if the simulation is not run to completion, the final BitMover microcode is still available in a separate file. It is this file which is down-loaded into the BitMover and forms the microcode for the graphics algorithms.

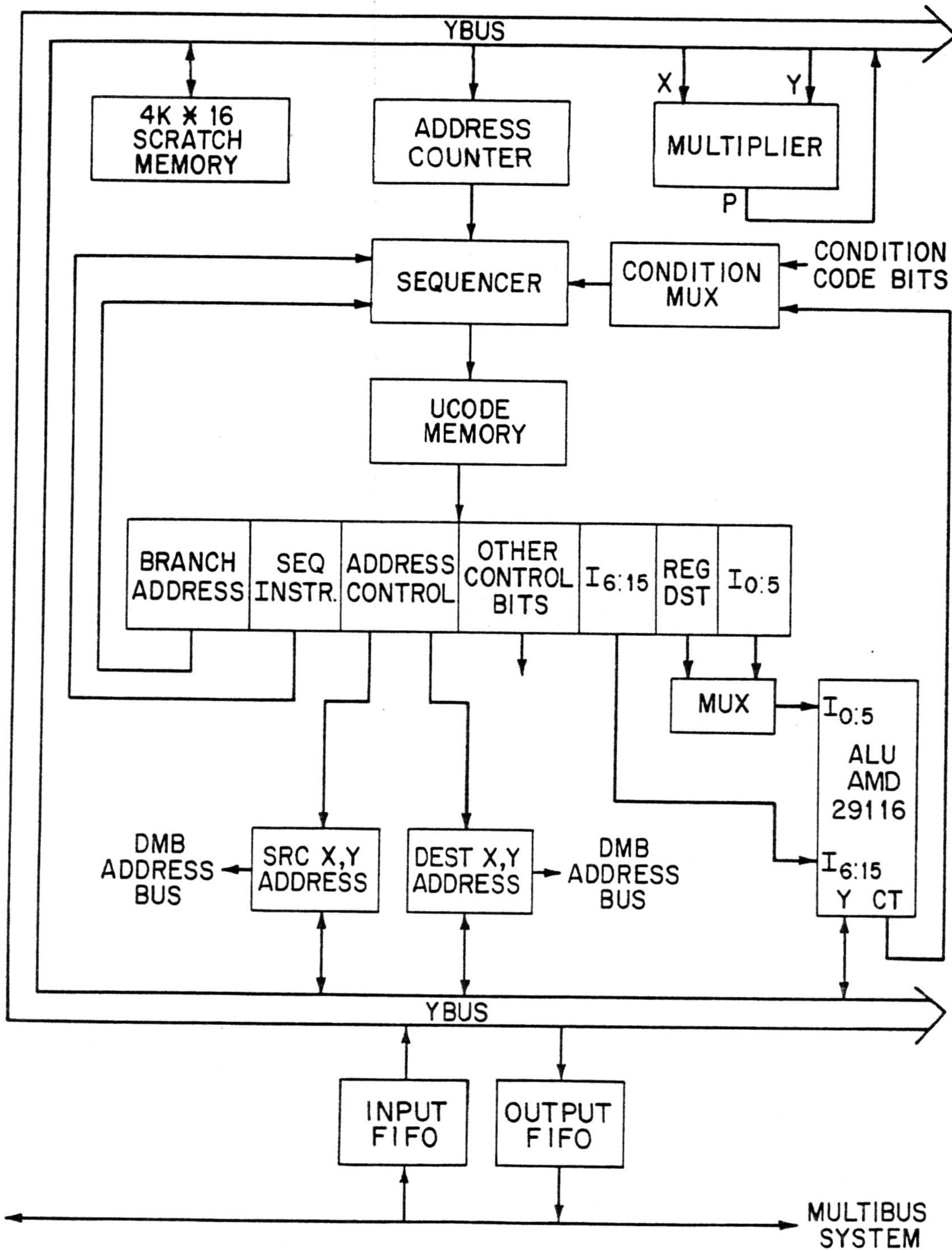
The simulation begins at address zero of the microcode. The first few lines of the microcode are a loop which waits for a graphic opcode to appear in the input fifo. When an opcode is available, a jump is made to the graphic routine specified, at which point the operands are loaded and execution of the algorithm begins. Output information is presented following each simulated cycle showing, the current address being executed, the next address to be executed, the microinstruction word at that address, the just completed sequencer and ALU instructions, and the status registers. Only those parts of the hardware which are being modified by the instruction currently being executed have their values displayed. This reduces the amount of debugging output which must be checked.

The simulator also supports breakpoints to ease the debugging process. The verbose mode described above, in which large amounts of status information are displayed, can be totally suppressed using this breakpoint feature. The verbose mode is entered upon reaching a microcode address containing a breakpoint on flag, and this mode can be disabled with a breakpoint off command embedded in the microcode.

Perhaps the most useful feature of the simulator is its ability to do graphic displays for debugging. In this debugging mode, the power of an already existing bit map display terminal is used. The process works by scanning the BitMover's display memory as maintained by the simulator. This memory is then displayed on the bit map terminal and the graphics primitives produced can be compared to the high level language implementations. If these two versions are the same, then the correctness of the microcode is reasonably assured. Also, being able to see the graphics output produced by incorrect microcode generally gives a better indication of the cause of the problem.

5. Conclusions

A microprogram simulator is a necessity in designs using complex microinstruction formats and highly parallel microcode. Use of hardware debugging techniques to inspect system status requires that the hardware design has been substantially completed. The time and cost of hardware debugging is thus limited to all but final system checkout. Because the entire state of the simulator can be checked at any time, early discovery of logic errors is a more straightforward procedure. Finally, hardware design changes can be tested easily by using logically self-contained software modules in a user-determined architecture.


YBUS
4K ✱ 16
SCRATCH
MEMORY
ADDRESS
COUNTER
X
Y
MULTIPLIER
P
SEQUENCER
CONDITION
MUX
CONDITION
CODE BITS
UCODE
MEMORY
BRANCH
ADDRESS
SEQ
INSTR.
ADDRESS
CONTROL
OTHER
CONTROL
BITS
I6:15
REG
DST
I0:5
MUX
I0:5
ALU
AMD
29116
I6:15
Y CT
DMB
ADDRESS
BUS
SRC X,Y
ADDRESS
DEST X,Y
ADDRESS
DMB
ADDRESS
BUS
YBUS
INPUT
FIFO
OUTPUT
FIFO
MULTIBUS
SYSTEM