III.4.3. RENDERER STRUCTURAL DESCRIPTION

III.4.3 – 1 Design Issues

Receive FIFO's

The first question to be addressed is whether or not the Renderer requires receive FIFO's (RxFIFO's) on its Ring ports. The IGC port clearly does, since incoming messages consist of IGC commands which take varying times to execute, and hence cannot be swallowed at a guaranteed rate.

It is tempting, however, to believe FIFO's are unnecessary on the Backing Store Port as long as one can guarantee that incoming messages can be received without interruption. In fact, it can be guaranteed that messages of incoming backing store data ("receive BS data" commands) are swallowed at 20 MHz without interruption. However, messages containing commands to transmit status or backing store data cannot, since they require that a channel be acquired. One might imagine a small FIFO to buffer these outgoing commands, with its half-full flag controlling RxReady. However, a "transmit BS data" command will tie up the VRAM system reading the outgoing data, so such commands must also de-assert RxReady (as the command is received), in case the next command is a "receive BS data" command. Such a scheme could be made to eliminate the need for RxFIFO's on the Backing Store Port, but at the cost of more complicated memory control (the VRAM system would have to respond *immediately* to a request to write incoming data), and probably system performance degradation due to the fact that "send BS data" commands could not be buffered up (each would have to de-assert RxReady until the transmission was completed, since it is not known whether or not the next message might be a "receive BS" command).

The next questions concern how destination address and tail-bits are handled. It is most desirable to make the limit on incoming message size as large as possible, for Ring efficiency, and to make the amount of data tranferred in "receive BS data" and "send BS data" commands a power of 2, so that a sector of BS data can be received using as few "receive BS data" commands as possible. Since commands consist of opcodes followed by data, and since maximum incoming message size tends to be $2^n + 1$, this means that a maximum size message containing one "receive BS data" command must contain the opcode and the data, and nothing else. This means that a separate word cannot be written into the RxFIFO to mark the tail-bit. It is also desirable to have the destination address treated as an opcode (for the BS port only), so that one Renderer may be instructed to send BS data to another Renderer. This means that the destination address should be written into the RxFIFO. Since it is desirable to have some way of distinguishing message boundaries, a 33-rd bit must be used to do this. Since no extra word

can be written into the RxFIFO for a tail-bit, and since it is impossible to tell that a data word is the last word of the packet before it is written into the FIFO (without adding some sort of pipeline latch between the Ring and the RxFIFO), this marker must be attached to the destination address of the next packet.

So the interface between the Ring and the RxFIFO writes one word into the RxFIFO for the destination address and each data word of the incoming packet, and sets a "Head" bit on a 33-rd FIFO bit with the destination address.

Video-RAM Timing

The VRAM memory system communicates with the Backing Store Port via the Corner-Turner (CT) chips. To avoid any system performance degradation, the CT's must sustain a rate of 20 MegaWords/sec transmitting or receiving data from the BS Port. Thus each CT must run at 20/(number of CT's) MHz. Because a CT must send or receive the 8 nibbles of a pixel sequentially, and because these 8 nibbles are necessarily contained within one VRAM chip, no interleaving of the VRAMs connected to a given CT chip is possible. This means that the VRAMs must operate at the same average frequency as the CT's. VRAMs are readily available to run at 10MHz page-mode, but none will run at 20 MHz. This means that at least 2 CT's are needed.

However, using 2 CT's, the VRAMs must sustain 10 MHz. This must include time to do RAS cycles to load new row addresses, transfer cycles for VRAM <-> EMC transfers, and refresh cycles. The timing is quite complex to do this, and probably impossible to do without some performance degradation (the 20 MHz rate is not quite mantained). By using 4 CT's, the memory timing becomes much simpler, with all memory controls defined on a 50ns pitch, and 20MHz data rates guaranteed (but with up to one macrocycle worth of startup time).

The Macrocycle

We define a *macrocycle*, consisting of thrirty-two 50 MHz cycles, with an overall length of 1600 ns. Each CT must sustain 5 MegaWords/sec. So within a macrocycle, a given CT must send or receive 8 pixels. The 1600 ns macrocycle allows time to load a row address, do 8 page-mode reads or writes, and then do either a CAS-before-RAS refresh or a transfer operation.

The macrocycle is divided into 2 phases: the IOPhase, cycles 0 through 25, and the TRPhase, cycles 26 through 31. The signal TRPhaseH defines which phase is active. IOPhase can be of 3 types: (1) idle, the default; (2) a *WrMacro*, 8 nibbles are clocked from each of the 32 nibble-

wide ports on each of the 4 Corner-Turners and written into the active bank of VRAM; or (3) a *TxMacro*, 8 nibbles are read from each of the 128 VRAMs in the active bank and clocked into the Corner-Turners, while simultaneously 32 words are clocked from the CT's into the transmit port. The TRPhase (for refresh/transfer) can be of 3 types: (1) do a CAS before RAS refresh cycle, the default; (2) an *MTSMacro*, do a VRAM transfer operation, moving one row of memory into the shifter; (3) an *STMMacro*, do a VRAM transfer operation, writing the data register into one row of memory; or (4) an *WMCMacro*, do a VRAM write mode control cycle operation. Macros of types STMMacro, MTSMacro, and WMCMacro, are collectively called *TfMacro*. Note that the terms WrMacro and TxMacro are mutually exclusive; they are "orthogonal" to the terms MSTMacro, STMMacro, and WMCMacro, which are also mutually exclsuive. Note also that the types STMMacro and WMCMacro are indistinguishable at the level of the Macro Generator; they differ only by the value of VRAM input SE as RAS falls, and SE is controlled directly by the Transfer Controller.

Performance of Backing Store IO Commands

Because of the "head room" gained by having 4 Corner-Turners, the Backing Store IO commands ("receive BS data" and "send BS data") can run at peak speed, that is, data can be read from and written to the Backing Store memory at the full 20 MegaWord/sec rate supported by the Ring. However, these IO commands have overhead times which prevent perfect performance. The overhead occurs at the beginning of each command. It is manifested as a startup latency for a single command, and as a degradation of the 20 MegaWord/sec throughput for a series of IO commands. Fortunately the overhead time is independent of the size (number of words transferred) of the IO command, and so it becomes insignificant if the commands can be made to move large amounts of data.

The "send BS data" command must execute a preliminary TxMacro, with the Corner-Turner outputs disabled, in order to load the first 32 words into the CT's. The next macrocycle actually begins sending the data onto the port TxData wires. Thus the overhead is the amount of time it takes to decode the command (a couple of cycles), acquire the outgoing channel, initiate the preliminary TxMacro, and execute the preliminary macro. The re-start feature of the Macrocycle Generator allows the preliminary macro to begin immediately, if the MG is within the IOPhase, or within about 6 cycles if it is in the Transfer Phase. The preliminary macro is begun, before attempt is made to acquire the channel, so the execution time of the preliminary macro is overlapped with the channel acquisition time. So assuming channel acquisition takes less than a macrocycle, the overhead is about 34-40 cycles (of the 20 MHz clock).

The "receive BS data" command must load 32 words into the Corner-Turners before executing the first macro. Thus the overhead consists of command decode time, 32 cycles to load the

CT's (assuming the RxFIFO never runs dry), and the time taken to initiate the first macro.

As startup latencies, these times cannot be reduced. However, it might be possible to overlap the overhead time of one command with the final macro of the previous command, and thereby reduce the degradation of through-put. The would require that the previous command "end" before it really ends; that is, the Backing Store FSM would have to begin looking at the next command after the final macro of the previous command has begun, rather than after it has eneded. For example, if the next command were a "receive BS data" command, the loading of the first 32 words of incoming data into the CT's would have to be overlapped with the final macro of the previous command.

Suppose we allow a "receive BS data" command to "end" after its last macrocycle has begun, so that we may begin executing the next command. If the next command is a "send BS data" command, we would have to wait for the final macro of the "receive" command to end, because the first step of the "send" command is the preliminary macro. So the only potential savings is to acquire the outgoing channel during the last macro of the "receive", but this time can probably be overlapped with the preliminary macro anyhow. However, if the next command were a "receive", we could begin loading data into the CT's, overlapping this with the last macro of the previous command; however, this would require adding a latch to hold the opcode, because the address bits could not be loaded into the Address Counter until the IOPhase of the last macro of the old command had ended. This would reduce the overhead between successive "receive" commands to just a few cycles, but at the cost of the extra latch and its control. Or perhaps, a "receive more BS data" command could be defined, which would not load the Address Counter, and just continue the same address sequence. Alternatively, the old "receive" could "end" after the IOPhase of its last macro, and the new "receive" could begin executing right away, but this would only save about 5 cycles of the overhead.

Suppose we allow a "send BS data" command to "end" after its last macrocycle has begun. If the next command were a "receive" it could not begin executing because it must begin loading data into the CT's and they are still busy with outgoing data. If the next command is also a "send", it could neither acquire a channel (because the old command still has the Tx port) nor do the preliminary macro (because the old command is still doing a macro). So only the command decode portion of the next command could be overlapped. Even this could not be done without providing a two-tiered RxLatch, with the opcode read from the first level, and the second level driving the BS bus; this is because the Corner-Turners would still have control of the BS bus (CTOE is asserted). So only a few cycles could be removed from the overhead, and at some cost, by ending a "send" command early. A "send more BS data" command could be defined, to avoid the preliminary macro, but this would still require the two-tiered RxLatch, and we would still have to wait for the last macro of the old command to end before acquiring

the new channel.

So the only feasible approach for attenuating the effect of overhead time on through-put seems to be to overlap successive "receive BS data" commands. Even this seems to be more trouble than it is worth. The best approach for increasing performance is clearly to make the "send BS data" and "receive BS data" commands move as much data as possible so that the overhead becomes insignificant. The amount of data moved by a command is limited primarily by the maximum allowable incoming message size for a given port. The Renderer probably has the smallest such limit, perhaps as little as 513 words. However, this limit can be waived if it is known that only one device is transmitting to the Renderer, as described at the end of Section III.4.1-3.

III.4.3 – 2 Renderer Structural Decomposition

The Renderer schematic is represented in a 3-level heirarchy.

At the top level, the Renderer schematic is divided into several sections. The EMCs and VRAMs are contained in 4 banks, *Banks 0 through 3*; each bank contains 16 EMCs and 16 VRAMs, along with the driver chips for their control signals and the terminators for these signals. The *Transfer* section contains the Transfer Controller, which controls movement of data between the EMCs and VRAMs. The *Backing Store Logic* section contains circuitry for controlling the Backing Store Port, the Corner-Turners, and the random IO port of the VRAMs. The *IGC Logic* section contains the IGC and related circuitry, including the circuitry to control the IGC Port. The LEDs and Errors section contains error handling circuitry and the board's indicator LEDs; logically related to it are the connections which form the status word from various Renderer signals. Finally, the *Clock Generator* section contains the ECL to TTL converters and various fanout parts for the 20 and 40 MHz clocks, and the terminators for the clock signals.

The EMC and VRAM Banks

Each of the four Banks are represented by a single D size sheet.

The 64 EMCs are numbered EMC0 through EMC63, and the 64 VRAMs are numbered VRAM0 through VRAM63. They are logically and physically arranged in pairs, eahc pair containing one EMC and the similarly numbered VRAM. Bank0 contains indices 0, 4, 8, ..., 60; Bank1 contains indices 1, 5, 9, ..., 61; and similarly for Banks 2 and 3. Normally each EMC represents 2 columns in the 128 x 128 pixel region associated with a Renderer, with EMC*i* containing columns *i* and 64+*i*. Thus the columns are interlaced across the Banks,

with any group of 4 adjacent columns falling in the 4 different banks. Each Bank is also divided into sub-Banks A and B. When the random IO ports of the VRAMs are accessed, only BankA or BankB is active on any given memory cycle. However, in accessing the VRAMs' serial ports, all VRAMs act in lockstep.

All 64 EMC chips on a Renderer receive the same control signals, logically speaking, with the exception of the pixel-memory address inputs Add<0:7>HLPR. The IGC has two sets of pixel-memory address outputs, AddrL<0:7>HSPR and AddrH<0:7>HSPR; the two sets of address are normally identical, except during configuration mode. In configuration mode, the two sets of signals are inverse to each other. The allows individual addressing of the EMCs (see EMC Chip documentation for details). In order to implement this, address input Add*j*HLPR of EMC*i* is connected to either AddrL*j* or AddrH*j*, depending upon whether bit *j* of *i* is a 0 or a 1. Thus when the address controls are inverse of each other, each of the 64 EMCs gets a different address.

Each EMC has 27 IGC-originating control signals, plus the IOCtl control signal for their IO register. Since each Bank contains 16 EMCs, 4 of the address lines must be paired. So each Bank requires 27 + 4 + 1 = 32 control signals, numbered EAB<0:31>. These signals are driven from four ACT574 edge-triggered latches.

Transfer Control

The Transfer Controller is contained on a single sheet. It contains a control PAL, RDTRF, a row counter, a nybble counter, a latch for the sector number and transfer direction, and parts for delaying and fanning out the VRAM serial clock signal SC.

IGC Port Control

The IGC Port is controlled by the Stream Parser FSM (SP).

It strips the destination address from an incoming message and begins parsing the IGC command stream. The Stream Parser reads words from the RxFIFO and writes them into the IGC with the correct register addresses attached. Based on the formatting information in the opcode, it determines which register addreses to attach and when the last word of the instruction has been reached. On the same cycle as writing the last word of the instruction into the IGC (or on the very next cycle, for instructions with no arguments), the Stream Parser also asserts the IGC input signal GoLLPR. This signals the IGC that the new instruction is

avaliable to be processed and causes the IGC to assert BusyHSP1. The Stream Parser does not begin writing the next instruction until BusyHSP1 is de-asserted.

The IGC Status Transmitter has only one function: to send out status messages, requested by the IGC, and to send out error messages, in response to any one of teh Renderer's error strobes being asserted.

Backing Store Port Control

The Backing Store Port also is controlled by one large state machine called the Backing Store FSM (BS FSM). This machine decodes incoming commands, controls the Macrocycle Generator to execute IO commands, controls the Semaphores directly, and controls the transmit port to generate outgoing messages.

For each incoming command, the 4 LSB's of the opcode define the command as follows:

D3	D2	D1	D0	COMMAND	SIZE OF COMMAND
0	0	0	0	BS_NOOP	opcode only
1	0	0	0	BS_SENDSTATUS	opcode + 2
0	0	1	0	BS_RECEIVE	opcode + 128 * no. scanline segments
0	0	1	1	BS_TRANSMIT	opcode + 1
0	1	0	0	BS_PBS	opcode only
0	1	0	1	BS_VIGC	opcode only
other codes				<RESERVED>	

For an IO command, bits 4-24 of the opcode define the amount and location of the data to be transferred, as shown in Figure III.4-1.

(Figure III.4-1 goes here)

The BS FSM latches the sector number, starting scanline number, and number of scanlines arguments into the Address Latch/Counter and Macro Counter.

For an "receive BS data" command, the BS FSM clocks 32 words of incoming data into the CTs (interleaving them, 8 into each CT), sets the trigger for a WrMacro, and waits for the macro to begin. As soon as the macro begins, it begins clocking the next 32 words from the RxLatch into the CTs. If there is no interruption of the incoming data (the FIFO never gets empty), another WrMacro will beging as soon as the trigger is asserted, so there is no interruption of the incoming data. Otherwise, if there was a wait, so that another idle macro began before the next 32 words had been clocked into the CTs, either the Macro Generator reset feature will cause the MG to reset to 0 and begin a WrMacro, or if enough cycles have elapsed so that the MG is in the TRPhase, the WrMacro begins as soon as the macro ends., BSRxCtrl clocks 32 pixels from RxFIFO into the 4 Corner-Turners

All Enables Are Off

There are several complications when using the AEO mechanism. First, since the IGC itself contains a pipeline of up to 3 commands, the end of an IGC Port message may be reached before execution of the last command has begun; hence the requirement of 4 commands after the last AEO-affetcing command before a flush-able message. Second, there is a lag of several 40 MHz cycles between the time an AEO-affecting command is executed and the time the AEO settles (and AEO may be unstable during this gap). This is not a problem for outgoing status messages, because there is enough delay between the time an IGC_SENDSTATUS command begins executing and the time the status bits are latched; similarly, the delay of executing an IGC_VBS, followed by completing a BS_VBS and then executing a BS_SENDSTATUS insures that BS status messages will contain correct AEO bits as well. It is a problem when executing AEO-conditional instructions in IGC microcode; probably need to insert an AEOWAIT instruction between last Enable-affecting instruction and first instruction which tests AEO.