MASTER READOUT MODULE

December 22, 1994

1.0 OVERVIEW OF MASTER READOUT MODULE 2
1.1 Description of the MRM 2
1.2 Block Diagram of the MRM 2
1.3 Subsections of the MRM 4
1.3.1 VXI Interface 4
1.3.2 Token Control 4
1.3.3 Token and RDY_BSY Time-out Logic 10
1.3.4 Fastout FPGA 11
1.3.5 Dual Port Ram 11
1.3.6 Pipeline Registers 12
1.3.7 VXI Local Bus 12
1.3.8 Differential Bus 13
1.3.9 Fiber Optic Interface 15
1.3.10 Pulser FPGA 15
1.4 Memory Map 17

2.0 TEST PROCEDURE FOR THE MASTER READOUT MODULE 21
2.1 VXI Interface Test 21
2.1.1 Verifying "resman" mapping 22
2.1.2 Testing Internal VXI Registers 24
2.2 Data Flow Through the MRM 26
2.3 Pulser FPGA Test 27
2.3.1 Interactive Pulser FPGA Test 27
2.3.2 Automatic Pulser FPGA Test 28

APPENDIX A: ABEL PROGRAM FOR U41 (22V10) IN MRM 30

APPENDIX B: TEST SOFTWARE 44

APPENDIX C: PARTS LIST AND DATA SHEETS 46

1.0 OVERVIEW OF MASTER READOUT MODULE

This section gives a brief overview of the MRM (Master Readout Module) which was designed and developed by Argonne National Laboratory's ECT Department (Electronics and Computing Technology). LBL (Lawrence Berkeley Laboratory) will be using the MRM for data acquisition on the GammaSphere Project.

1.1 Description of the MRM

The MRM is a C size VXI Module. Its main function is to collect data from the LRMs (Local Readout Modules) and any local GeBGO modules, and transmit it to the Event Builder. The LRM then transmits the data to the MRM through a differential ribbon cable bus. All of the LRMs and the MRM share the differential ribbon cable bus. A token passing scheme is used to determine which LRM may drive the differential bus. The GeBGO modules plugged into the MRMs VXI mainframe will transmit their data over the VXI Local Bus when they receive the token from the MRM. The MRM converts the 32 bit differential bus and the 32 bit VXI Local Bus into four serial bit streams, which are transmitted to the Event Builder over four fibers. The Event Builder then converts the four optical signals into a 32 bit wide differential ECL bus which clocks data into the VME Memory Cards.

1.2 Block Diagram of the MRM

On the following page is a block diagram of the MRM. Section 1.3 describes the function of each section shown in the block diagram. Note that the MRM consist of sections which are common with the GeBGO and the LRM.

MRM and GeBGO both have:

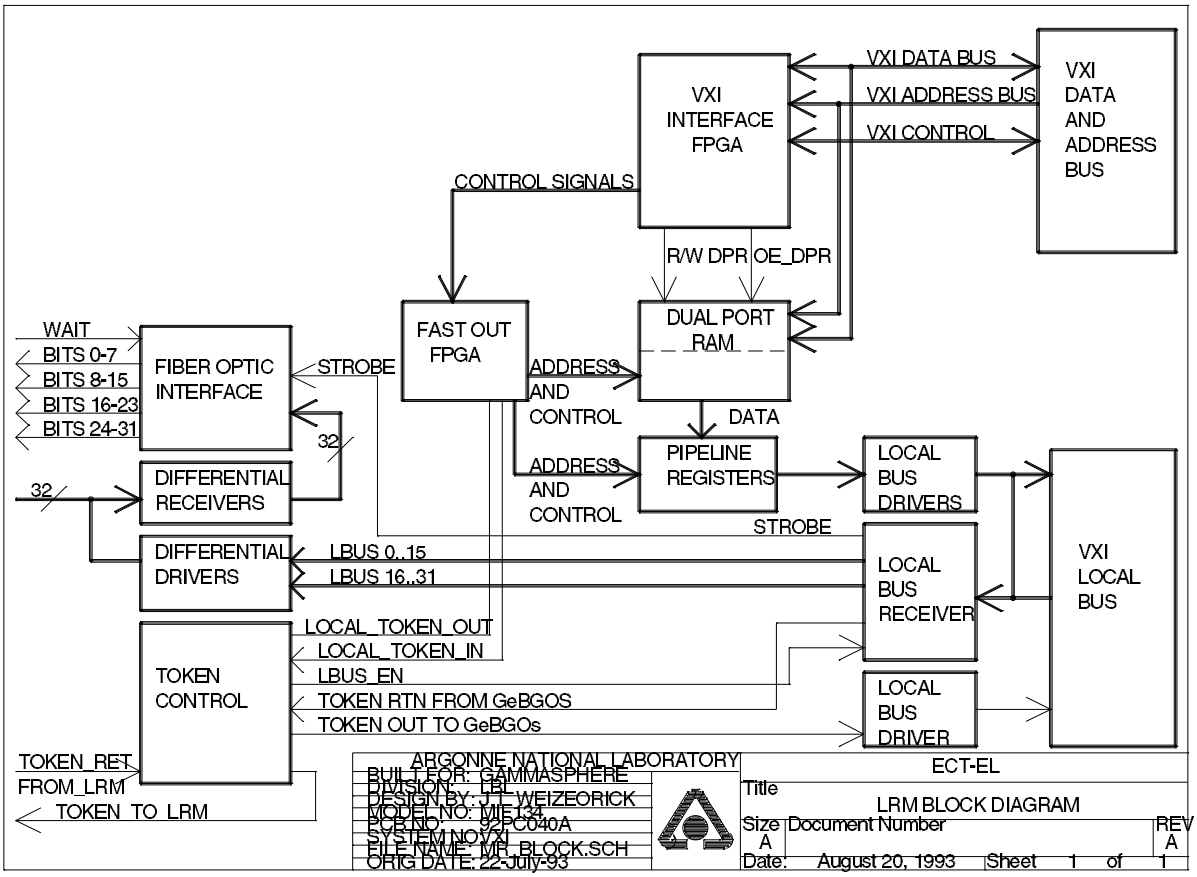
- VXI Interface
- FASTOUT FPGA
- Local Bus Drivers
- Dual Port Ram
- Pipeline Registers

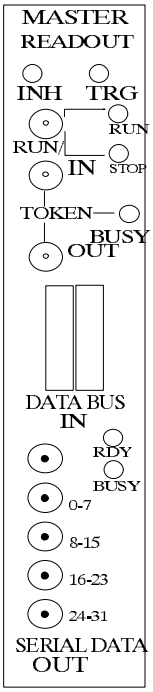
MRM and LRM both have:

- VXI Interface
- Token Control section
- Local Bus Receivers

MRM also has:

- Differential Receivers for the data coming form the LRM
- Fiber Optics to Event builder
- Pulser FPGA





MASTER READOUT FRONT PANEL

1.3 Subsections of the MRM

The Master Readout Module consist of the following subsections listed below.

- VXI INTERFACE
- TOKEN CONTROL
- FASTOUT FPGA
- DUAL PORT RAM
- PIPELINE REGISTERS
- VXI LOCAL BUS
- DIFFERENTIAL BUS INTERFACE
- FIBER OPTICS
- PULSER FPGA

1.3.1 VXI Interface

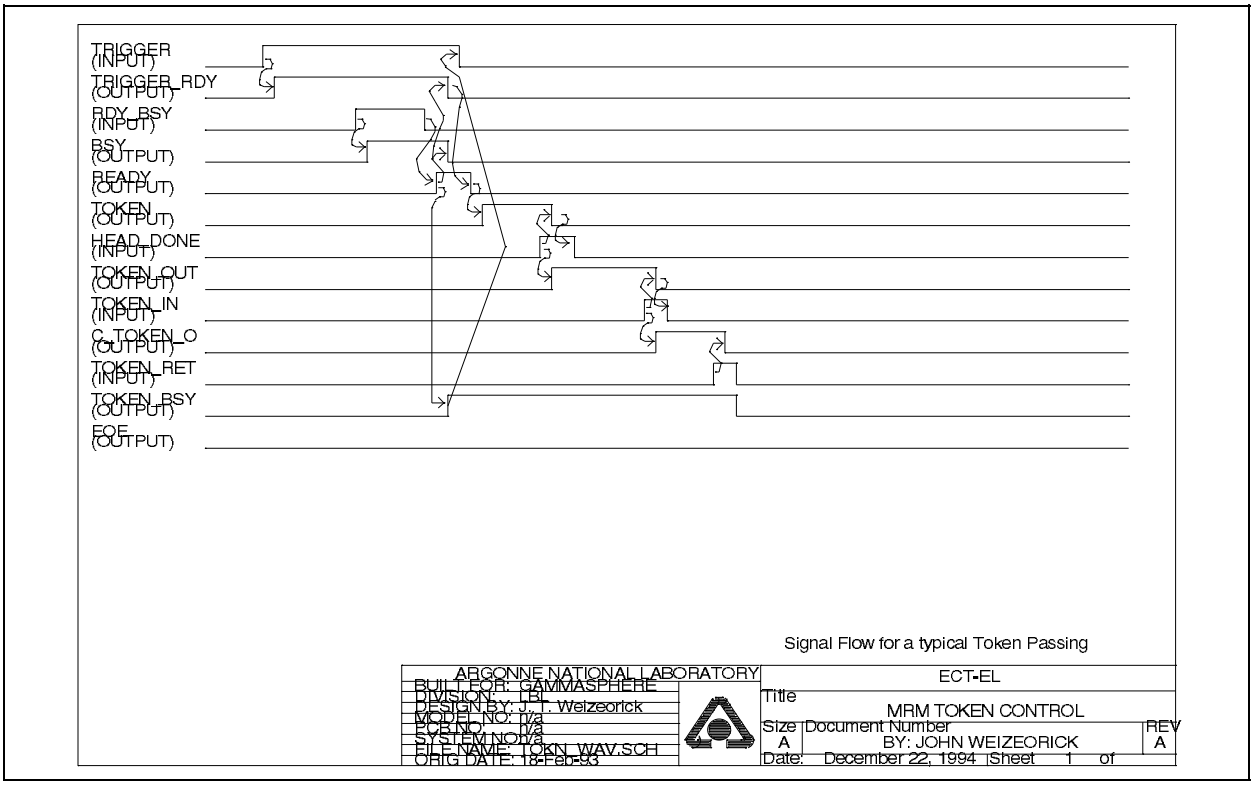
The **VXI Interface** consist of the VXI FPGA, octal buffers, octal transceivers, and the VXI connectors. This part of the design allows the VXI Resource Manager to setup and control everything on the MRM. The VXI Interface is similar to the one used for the GeBGO Modules. The VXI Interface for the Master Readout has a different memory map than the GeBGO Modules. For details on the memory map see section 1.4 in this document. For details on the FPGA Logic see the schematics of the VXI Interface for the Master Readout.

1.3.2 Token Control

The **TOKEN CONTROL** subsection consist of a 22V10 and is used to control the data flowing through the MRM. Listed below are the sequence of events which occur during a normal event. A waveform diagram can be found on the next page.

- TRIGGER from Trigger Module or a soft trigger from the MRM's VXI Interface. The Trigger Module passes the TRIGGER signal to the MRM though ECL_TRIG0 on the VXI backplane. TRIGGER is active high.
- TRIGGER_RDY gets latched high inside the 22V10. At this point the Master Readout Module is waiting for the RDY_BSY or RDY_BSYC input to go low, after being high. If they are not high at this point then the 22V10 will wait for them to go high. TRIGGER_RDY stays high until READY goes high.
- While the RDY_BSY and the RDY_BSYC inputs are high they are performing Analog to Digital conversions on their various signals.
- When the RDY_BSY and RDY_BSYC inputs go low, then the data is in the GeBGOs' pipeline registers and ready to be transferred. The READY output is pulsed high at the same time the TRIGGER_RDY latch is cleared and the BSY output goes low. The READY pulse starts the token passing scheme.

- The HEAD_TOKEN output goes high and is sent to the local FASTOUT FPGA. The data which was stored in the Dual Port Rams and transferred into the pipeline registers while RDY/BSY was low, will now be sent to the Event Builder.
- When the transfer is finished the FASTOUT FPGA on the MRM will return the token by driving the HEAD_DONE input high.
- When the HEAD_DONE input goes high, HEAD_TOKEN is removed or lowered at the same time TOKEN_OUT goes active high. When the HEAD_TOKEN signal is lowered the FASTOUT FPGA lowers its HEAD_DONE signal.
- The TOKEN_OUT output goes to the first LRM, which passes the token to it local GeBGO modules. When the LRM's last GeBGO Module returns the token then the first LRM sends the token to the next LRM. This is repeated until the last LRM returns the token to the MRM, which drives the TOKEN_IN signal high. When TOKEN_IN goes active high TOKEN_OUT is removed and C_TOKEN_O goes active high. When TOKEN_OUT goes back low this causes the TOKEN_IN to go back low. The time it takes TOKEN_IN to go low is based on the delay through all of the LRM and GeBGO Modules.
- The C_TOKEN_O output goes over the VXI Local Bus to the GeBGO Modules plugged into the MRM VXI mainframe. The last GeBGO Module or the Target Module returns the token to the MRM and drives the TOKEN_RET input high. When TOKEN_RET goes high the EOE (End of Event) signal is pulsed high for 0.1 usec.



All of the input and output signals used by the Token Control PAL are described in detail below. There are other places that some of these signals are described but this is intended to provide a single location where all of the signal descriptions can be found.

TRIGGER - This is an input form the Master Trigger Module and is used by the Token Control PAL. When the Token Control PAL receives the TRIGGER signal it and waits for the RDY_BSY and RDY_BSYC to become inactive. At this time the Token Control PAL starts the token passing scheme. The token busy signal is used by the Master Trigger Module to clear the Master Trigger Signal. In this way there can only be two events in the pipeline. One event being read out and one in the process of the event sequence.

TRIGGER_RDY - This is an internal signal used by the Token Control PAL to indicate when a trigger has been received. The trigger is latched in the Master Trigger Module and is cleared when the token busy signal becomes active.

BSY - This is an internal signal used by the Token Control PAL to indicate when a trigger has been received and the RDY_BSY signals are active. When this signal goes low it activates the READY signal.

READY - This is an internal signal used by the Token Control PAL to indicate when it is time to start passing the token around. It is only active for one clock cycle after the BSY signal goes low.

RDY_BSY - This signal is an input into the MRM Token Control PAL. It comes from the differential ribbon cable bus connected to all of the LRMs. It is the logical OR of all of the RDY/BSY signals from all of the GeBGO Modules in all of the other VXI Mainframes. The RDY_BSY signal on an individual GeBGO Module is high while that module is performing ADCs and while it is moving or waiting to move data into the pipeline registers. If there is data in a GeBGO's pipeline registers then the RDY_BSY will stay high until the previous event, stored in the pipeline register, is moved out and the new event, stored in the ADCs, is moved into the pipeline registers.

RDY_BSYC - Same as the RDY_BSY signal but this signal is driven by the Logical OR of all the RDY/BSY signals in the MRM VXI Mainframe including the Master Trigger Module.

HEAD_TOKEN - This signal comes form the Token Control PAL on the MRM and goes to the FASTOUT FPGA on the MRM. It is the first token output signal and is used control when the MRM puts the event header information into the data stream going to the Event Builder.

HEAD_DONE - This signal is the token return signal from the FASTOUT FPGA on the MRM. It lets the Token Control PAL on the MRM know when the header information is finished being sent out. When the Token Control PAL receives this signal it removes the HEAD_TOKEN signal and activates the TOKEN_OUT signal.

TOKEN_OUT - This signal goes out of the MRM through the TOKEN_OUT Lemo on the front panel of the MRM. It is the token output signal to the first LRM.

TOKEN_IN - This signal comes into the MRM through the TOKEN_IN Lemo on the front panel of the MRM. It is the token return signal from the last LRM. When the Token Control PAL in the MRM receives this signal it removes the TOKEN_OUT signal and activates the C_TOKEN_O signal.

C_TOKEN_O - This signal comes form the Token Control PAL on the MRM and goes out the VXI backplane on the MRM VXI Mainframe. This is the last token to be sent out and allows the Master Trigger Module and other devices, plugged into the Master Readout's VXI Mainframe, to place their data into the data stream.

TOKEN_RET - This is the return token form the MRM VXI Mainframe. It goes to the Token Control PAL in the MRM. When the Token Control PAL receives this return token it removes the C_TOKEN_O signal, which completes the token passing scheme.

TOKEN_BUSY - This is an output from the token control 22v10 located in the MRM. It is high while the Token is being pass around. The TOKEN_BUSY signal goes into the LRM, over the differential ribbon cable bus, where it is used to drive the TTLTRG1 signal. This TTL Trigger line goes into each FASTOUT FPGAs on all of the GeBGO Modules. The FASTOUT FPGA uses this signal to determine what to clear when it receives an EOE signal. If the Token is not busy then the FASTOUT clears both the ADCs and the Pipeline registers. If it receives an EOE while the Token is busy then the FASTOUT knows that the EOE is for a new event and not the one buffered in its pipeline registers. So the FASTOUT only clears its ADCs. The TOKEN_BUSY signal also drives a red led which will be on while the token is being passed around.

INHIBIT - The INHIBIT input comes form the Master Trigger Module and goes to the Token Control PAL located in the Master Readout Module. It is used to indicate when an event is to be ignored. When the INHIBIT becomes active the Token Control PAL generates an EOE signal which is routed to all of the FASTOUT FPGAs in the GeBGO Modules. Each FASTOUT FPGA will determine what action should be taken when it receives the EOE signal. If it has data in its pipeline registers and the token is not busy then it will clear its pipeline registers and its ADCs. If it has data in its pipeline registers, the token is busy and it has not received the token, then the FASTOUT FPGA will only clear its ADCs, and keep the data in its pipeline registers ready for transfer.

EOE - EOE is an output of the Token Control PAL located in the MRM Module. The EOE signal goes to all of the LRM over the differential ribbon cable bus. In the LRM it is used to drive TTLTRG2. This TTL Trigger signal goes to all of the FASTOUT FPGAs in all of the GeBGO Modules. The EOE signal tells the FASTOUT FPGAs to clear the data in either its ADCs or in its pipeline registers. If the Token is not busy then the FASTOUT clears the data in both the ADCs and the Pipeline registers. If it receives an EOE while the token is busy and it has, then the FASTOUT knows that the EOE is for a new event. If the FASTOUT FPGA has already received the token then it knows that the new event is in its pipeline registers and it clears both its pipeline registers and its ADCs. If it has not received the token then the FASTOUT FPGA knows that the new event is not in its pipeline registers and only clears its ADCs. By making use of the TOKEN_RESET and EOE_GEN control bits the MRM can force all of the GeBGO to dump all of their data.

TOKEN_GEN - When TOKEN_GEN is set high it forces the Token Control PAL on the Master Readout Module to start the token passing scheme. If this signal is still high when the token is returned by the last device then the Token Control PAL will pass the token around again.

EOE_GEN - When EOE_GEN is set high it forces the Token Control PAL on the Master Readout Module to send an EOE signal to all of the GeBGO Modules. This signal is routed like the SCALER_RESET, but in this case the LRM drive TTLTRG2. When the FASTOUT FPGA on the GeBGOs receives the EOE signal they either clear the ADCs or the Pipeline registers based on the TOKEN_BUSY signal. The programmer can force the EOE signal to reset the GeBGO Modules by clearing the token before generating an EOE signal.

TOKEN_RESET - This signal goes to the Token Control PAL in the Master Readout Module. When this signal goes high it resets the Token Control PAL. This signal can be used with the EOE_GEN signal to force the GeBGO Modules to clear any events stored in their pipeline registers or ADCs.

WAIT - The WAIT input comes from the Event Builder and is sent to the Master Readout Module when the CES memory cards become full. When the WAIT signal becomes active high the MRM stops sending data to the Event Builder. The WAIT signal also drives a red and green led on the front panel of the Master Readout Module as well as the Event Builder. Red indicates that the memory is full and green means that there is memory available for the Event Builder to put data into.

INT_DATA - The INT_DATA output is used to enable the internal data buffers between the VXI Local Bus and the TAXI chips. When this signal is active the data going to the Event Builder is coming from either the Master Readout Module or some other module in the MRM VXI Mainframe.

EXT_DATA - The EXT_DATA output is used to enable the data buffers between the Differential Ribbon Cable Bus and the TAXI Chip. When this signal is active the data going to the Event Builder is coming from the GeBGO Module in position of the token.

A number of the signals described above are passed over the Differential Bus to the LRM and are made available to the GeBGO Modules through the TTL Trigger Lines. There are additional description of these signals in the section describing the Differential Ribbon Cable Bus.

1.3.3 Token and RDY_BSY Time-out Logic

The Token and RDY_BSY time-out logic is designed to give the system a way of determining if the Token ever gets lost, or if the RDY_BSY signal ever gets stuck high. Since the logic is identical for both the Token and RDY_BSY timeout I will only describe the logic for the Token time-out.

The TOKEN_BUSY signal has two functions in the token time-out logic. While the TOKEN_BUSY signal is active it allows a 2MHz clock to increment an 8 bit counter. When the TOKEN_BUSY signal becomes inactive it resets the counter. If the counter output ever becomes >= to the value in the **token time-out register** then the **token time-out bit** in the **time-out status register** will become latched high. This bit will stay high until a zero is written into the **token time-out bit** in the **timeout status register**.

1.3.4 Fastout FPGA

The **FASTOUT FPGA** controls the data flow of the event header stored in the MRM's Dual Port RAM. In the GeBGO modules the FASTOUT FPGA passes sensor information over the VXI Local Bus to its LRM. In the MRM the FASTOUT FPGA passes user programmable data stored in the MRM's Dual Port RAM. The data is transmitted through local bus drivers to local bus Receivers, which interface to the fiber optic lines going to the Event Builder. The data becomes the header for each event transmitted to the Event Builder.

1.3.5 Dual Port Ram

The **DPR (Dual Port Ram)** in the MRM is only used to store the event header data, which is transmitted to the Event Builder. This means that when an event occurs the data written into the DPR will be transmitted to the Pipeline Registers and then shipped out as the event header to the Event Builder. This data in the DPR has the following format.

DPR Base Address	Bits 15-12	Bits 11-8	Bits 7-4	Bits 3-0
0x10	10xx	xxxx	xxxx	xxxx
0x12	ssss	ssss	ssss	ssss
0x14	0xxx	xxxx	xxxx	xxxx
0x16	xxxx	xxxx	xxxx	xxxx
0x18	0xxx	xxxx	xxxx	xxxx
0x1a	xxxx	xxxx	xxxx	xxxx
0x1c	0xxx	xxxx	xxxx	xxxx
0x1e	xxxx	xxxx	xxxx	xxxx

In the table above each bit positions is reperesented by a character. The first 16 bit word become the MS half of the 32 bit word stored in the memory modules, and the second 16 bit word becomes the LS half of the 32 bit word. This is repeated so that the data stored in the memory module has the format shown below.

Base Address	Bits 31-28	Bits 27-24	Bits 23-20	Bits 19-16		Bits 15-12	Bits 11-8	Bits 7-4	Bits 3-0
0x0	10xx	xxxx	xxxx	xxxx		ssss	ssss	ssss	ssss
0x4	0xxx	xxxx	xxxx	xxxx		xxxx	xxxx	xxxx	xxxx
0x8	0xxx	xxxx	xxxx	xxxx		xxxx	xxxx	xxxx	xxxx
0xc	0xxx	xxxx	xxxx	xxxx		xxxx	xxxx	xxxx	xxxx

In the formats shown above each character represents a bit. Listed below is the meaning of each character.

x = User programmable by writing to the Channel A DPR in the Master Readout Module.

s = In this location the Event Builder will write the number of 32 bit words in an event. So the value originally written into this address space in the Master Readout Module will be written over by the Event Builder.

1 = Must be 1 to indicate to the Event Builder that this is the beginning or end of an event. When the MS bit of the 32 bit words is a 1 if the next bit is a 0 then it is the beginning of an event, and if the next bit is a 1 then it is the end of an event.

0 = Must be 0 to indicate to the Event Builder that this is not the beginning or end of an event.

1.3.6 Pipeline Registers

The **Pipeline Registers** store the eight 16 bit words from channel A's DPR. The data comes from the DPR and goes into the Pipeline Registers as 16 bit words. When a token is passed to the MRM's FASTOUT FPGA the data in the Pipeline Registers is transferred to the Event Builder. The data is clocked out of the Pipeline Registers as four 32 bit words.

1.3.7 VXI Local Bus

The **VXI LOCAL BUS** consist of bus drivers and bus receivers used for transmitting and receiving data over the VXI Local Bus.

Four bus receivers are used for the 32 bits of data from the GeBGO Boards. Another bus receiver is used to receive the TOKEN_RET, RDY_BSY_BD and the STRBRD signals.

Since the MRM also sends data to the Event Builder there are four local bus drivers used to transmit this data to its own receivers. This way the MRM's data travels the same path as the GeBGO boards local to the MRM's VXI Mainframe.

TOKEN_RET — This input signal is an active high signal which comes form the last GeBGO board over the VXI Local Bus. The last GeBGO board drives the TOKEN_RET when it is returning the token to the LRM.

RDY_BSY_C — This input signal is a VXI Local Bus signal which is OR inside the Token Control PAL with the RDY/BSY signals from all of the GeBGO Modules. The Local Bus RDY_BSY_C signal is normally pulled high and goes low when any of the GeBGO Modules or the Master Trigger Module drives their RDY/BSY signals low. The VXI Local Bus receiver inverts the signal so that the RDY_BSY_C is active high. The MRM uses the RDY_BSY_C signal along with the RDY_BSY signal in the Token Control PAL to control when the token passing scheme is initiated.

STRBRD — The rising edge of this signal indicates that the data on the VXI Local bus from the GeBGO Module with the token is now valid. This signal is used by the TAXI chips to transfer the data to the Event Builder.

A bus driver is used to send the C_TOKEN_O signal to the first GeBGO module. The first GeBGO module transmits its data over the VXI Local Bus while it has the token. When it is done transmitting its data, it passes the token to the next GeBGO module. The last GeBGO module returns the token to the MRM by driving the TOKEN_RET signal. When the Token Control PAL receives the TOKEN_RET signal it removes the C_TOKEN_O signal and ends the token passing scheme.

1.3.8 Differential Bus

The **Differential Bus** is an external 32 bit differential bus which transmits data from all of the LRM into the MRM over two ribbon cables. The ribbon cables come into the MRM from the first LRM. The bus is terminated inside the MRM and at the last LRM with a termination block plugged into its differential input connector. The LRM take turns driving this bus based on who has the token. The data is clocked into the MRM on the rising edge of the STROBE signal on the differential bus.

Listed below are input and output signals which are transmitted over the Differential Bus.

RDY_BSY This signal is an input into the MRM over the differential ribbon cable bus. It is the logical OR of all of the RDY/BSY signals from all of the GeBGO Modules in all of the VXI Mainframes. The RDY_BSY signal on an individual GeBGO Module is high while that module is performing ADCs and while it is moving or waiting to move data into the pipeline registers. If there is data in a GeBGO's pipeline registers then the RDY_BSY will stay high until the previous event, stored in the pipeline register, is moved out and the new event, stored in the ADCs, is moved into the pipeline registers.

RUN_STOP This output from the MRM is controlled by a latched bit which can be programmed from the Resource Manager through the VXI INTERFACE. The input is used to drive TTLTRG0, which makes this signal available to all of the GeBGO modules.

BUSY This is an output from the token control 22v10 located in the MRM. It is high while the Token is being pass around. The BUSY signal goes into the LRM where it is used to drive the TTLTRG1 signal. This TTL Trigger line goes into the FASTOUT FPGAs on all of the GeBGO Modules. The FASTOUT FPGA uses this signal to determine what to clear when it receives an EOE signal. If the Token is not busy then the FASTOUT clears both the ADCs and the Pipeline registers. If it receives an EOE while the Token is busy then the FASTOUT knows that the EOE is for a new event and not the one buffered in its pipeline registers. So the FASTOUT only clears its ADCs.

EOE This is an output from the token control 22V10 located in the MRM. EOE is an input to the LRM where it is used to drive the TTLTRG2 signal. This makes it available to all of the GeBGO modules. When the FASTOUT FPGA on the GeBGOs receives the EOE signal they either clear the ADCs or the Pipeline registers based on the TOKEN_BUSY signal.

PULSE This is an output from the Pulser FPGA on the MRM. It is an input to the LRM where it is used to drive TTLTRG5. This makes it available to all of the GeBGO Modules. The Pulser FPGA provides a wave form whose frequency can be controlled through the VXI Interface on the MRM.

SRESET This is a latched bit which is controlled through the VXI Interface on the MRM. It is an input to the LRM where it is used to drive TTLTRG6. This makes it available to all of the GeBGO modules where it is used to resets all of the Scaler FPGAs.

GATE This is from the Pulser FPGA on the MRM. It is an input to the LRMs where it is used to drive TTLTRG7. This makes this signal available to all of the GeBGO modules where it is used to control the gate signal on all of the Scaler FPGAs. The Pulser FPGA provides a pulse whose width may be controlled through the MRM's VXI interface.

STROBE The strobe signal is an input to the MRM. Its rising edge is used to clock in the data from the differential bus. It comes through a LRM from the GeBGO module with the token.

DAT+0 -> DAT+32
DAT-0 -> DAT-32
These signals are the differential data signals on a bus shared by all of the LRMs and the MRM. The LRM which has the token drivers the bus with data from one of its GeBGO modules.

1.3.9 Fiber Optic Interface

The **FIBER OPTIC INTERFACE** takes data form the 32 bit differential bus and divides it up into four 8 bit words. Each 8 bit word gets converted by a TAXI chip into a serial bit stream which is transmitted by sperate fiber line. The four serial bit streams get converted back into a into a 32 word at the Event Builder.

1.3.10 Pulser FPGA

The **PULSER FPGA** provides a programmable frequency generator output and a programmable pulse width output. The frequency and pulse with are controlled by writing a 16 bit word to the MRM's VXI Interface. The programmable frequency signal is called PULSE and is available to every GeBGO Module as TTLTRG5. The programmable pulse signal is available to every GeBGO Module as TTLTRG6 and is used to gate all of the Scalers on all of the GeBGO Modules.

The frequency goes from 0.15Hz to 5KHz and is controlled by the 16 bit number written into the pulse latch. The output frequency is 5KHz when the pulse number is 0, and the output frequency is 0.15Hz when the pulse number is 0xffff.

The equation below gives you the frequency based on the number written to the pulse latch address.

$$Freq = \frac{1}{.2msec + (N * .1msec)}$$

The gate signal is an active high programmable pulse, which is controlled by a 16 bit number written into the gate latch. The pulse with goes from 1.58 msec to approximately 105 sec, with a step size of 1.6 msec starting at 1.58 msec. The shortest gate width of 1.58 msec, is achieved by writing 0x0 into the 16 bit gate latch address. The longest gate width of 105 sec is achieved by writing 0xffff into the gate latch.

The equation below gives the time that gate is high based on the number written into the counter latch.

Pulse Width = 1.58msec + (N * 1.6msec)

1.4 Memory Map

The Master Readout Module has 512 bytes of address, in A16 address space, per module. The absolute VXI address will be determined dynamically by the Resource Manager. The base addresses of the registers are defined as follows:

A16 ADDRESSING FORMAT
Base Address

11LL LLLH	HH00 0000

L + H = Logical address
L = Bits which are configured during Dynamic Configuration
H = Hard wired (common decode bits for 8 devices/module)

The register definition per Ge/BGO module is:

OFFSET FROM BASE	READ FUNCTION	INITIAL VALUE
00H	ID REGISTER	1111 1111 0000 0000
02H	DEVICE TYPE REGISTER	0000 0001 0000 0000
04H	STATUS REGISTER	0M00 0000 0000 1100 M = MODID bit inverted
06H	OFFSET REGISTER	0000 0000 0000 1000

OFFSET FROM BASE	WRITE FUNCTION	INITIAL VALUE
00H	LOGICAL ADDRESS REGISTER	xxxx xxxx PPPP Pxxx P = Programmable Bits
02H		
04H		
06H		

Logical Address - Used during dynamic configuration to set the Logical Address.
ID Reg - A16 only, For customized device = F00h.
Device Type Reg - 0 bytes of memory, model code = 0.
Status Reg - No A24/A32, Modid active, Ready Passed.
Control Reg - A24/A32 Disabled, Sysfail active, Reset state Off.7
Offset Reg - 8 devices (512 bytes) sharing address decoding.

Device	Relative Address	Use	Function	Dev 0 Offset Address
0	000...00f	VXI Configuration Registers	Read ID; Write Logical Address Read Device Type Read Status; Write Control Read/Write Offset Register	000h 002h 004h 006h
0	010...01f	Event Header Data	Read/Write Ram	010h
0	020...02f	Read Status Bits	0 - TOKEN_BUSY(1 = busy) 1 - RDY_BSY (1 = waiting for ADCs in lrm) 2 - RDY_BSYC (1=waiting for ADCs in mrm) 3 - TOKEN_IN (1 = token active) 4 - TOKEN_OUT (1 = token active) 5 - HEAD_TOKEN (1 = token active) 6 - HEAD_DONE (1 = token active) 7 - C_TOKEN_O (1 = token active) 8 - TOKEN_RET (1 = token active)	020h
		R/W Enable TTL Trigger signals	0 - Enable TTL trigger lines when = 1	022h
		R/W CONTROL LATCH	0 - SCALER_RESET (1 = reset all scalers) 1 - EOE_GEN (1 = generate eoe) 2 - TOKEN_GEN (1 = generate tokens) 3 - PULSE_START (1 = start pulser frq) 4 - RUN/STOP (1 = run) 5 - TEST TRIGGER (1 =generates trigger) 6 - TOKEN_RESET (1 = reset token pal)	024h
		R/W Timeout Status Latch	(WRITE 0 TO CLEAR STATUS) 0 - TOKEN_TIME_OUT (1 = timeout) 1 - RDYBSY_TIME_OUT(1 = timeout)	026h
		R/W Token Timeout value	8 Bit number compared with token timeout counter. Counter clocked every .5 usec.	028h
		R/W RDY_BSY Timeout value.	8 Bit number compared with rdy_bsy timeout counter. Counter clocked every .5 usec.	02ah
		R/W Pulser Gate from Pulser FPGA	16 Bit number to determine pulse width of GATE from Pulser FPGA to Scaler resets.	02ch
		R/W Pulser Latch to/from Pulser FPGA	16 Bit number to determine frequency of PULSE output on Pulser FPGA	02eh
0	030...03f	Scratch Memory	Read/Write Ram	030h
1	040...04f	VXI Configuration Registers	Points to same Registers as Device 0	040h
1	050...05f	Scratch Memory	Read/Write Ram	050h
1	060...06f	Read Pulser Status	Unused Input to Pulser FPGA	060h
1	070...07f	Scratch Memory	Read/Write Ram	070h

Device	Relative Address	Use	Function	Dev 0 Offset Address
2	080...08f	VXI Configuration Registers	Points to same Registers as Device 0	80h
2	090...09f	Scratch Memory	Read/Write Ram	90h
2	0a0...0af	Scratch Memory	Read/Write Ram	a0h
2	0b0...0bf	Scratch Memory	Read/Write Ram	b0h
	0c0...0cf	VXI Configuration Registers	Points to same Registers as Device 0	coh
3	0d0...0df	Scratch Memory	Read/Write Ram	d0h
3	0e0...0ef	Scratch Memory	Read/Write Ram	e0h
3	0f0...0ff	Scratch Memory	Read/Write Ram	foh

Device	Relative Address	Use	Function	Dev 0 Offset Address
4	100...1ff	VXI Configuration Registers	Points to same Registers as Device 0	100h
4	110...11f	Scratch Memory	Read/Write Ram	110h
4	120...12f	Scratch Memory	Read/Write Ram	120h
4	130...13f	Scratch Memory	Read/Write Ram	130h
5	140...14f	VXI Configuration Registers	Points to same Registers as Device 0	140h
5	150...15f	Scratch Memory	Read/Write Ram	150h
5	160...16f	Scratch Memory	Read/Write Ram	160h
5	170...17f	Scratch Memory	Read/Write Ram	170h

Device	Relative Address	Use	Function	Dev 0 Offset Address
6	180...18f	VXI Configuration Registers	Points to same Registers as Device 0	180h
6	190...19f	Scratch Memory	Read/Write Ram	190h
6	1a0...1af	Scratch Memory	Read/Write Ram	1a0h
6	1b0...1bf	Scratch Memory	Read/Write Ram	1b0h
7	1c0...1cf	VXI Configuration Registers	Points to same Registers as Device 0	1c0h
7	1d0...1df	Scratch Memory	Read/Write Ram	1d0h
7	1e0...1ef	Scratch Memory	Read/Write Ram	1e0h
7	1f0...1ff	Scratch Memory	Read/Write Ram	1f0h

2.0 TEST PROCEDURE FOR THE MASTER READOUT MODULE

The sections below is a procedures used to test out the MRM. The initial test require the following setup in the VXI Mainframe.

SLOT 0 National Instruments VXIcpu-030
SLOT 1 Master Readout Module

The National Instrument board should be running VxWorks and should be a node on a SUN's Ethernet Network. If the VXIcpu-030 board is properly configured on powerup it will load VxWorks from the network and execute a start up routine. The start up routine will execute a program called resman, and load all of the programs needed to test the GammaSphere Data Acquisition System.

In the test procedures below, commands which an operator must enter are written in **BOLD**. The computer output is a smaller font and comments are in the normal font. The commands are entered by an operator on a terminal logged into the National Instruments VXIcpu-030 node. This terminal will be running a C shell under VxWorks.

2.1 VXI Interface Test

There are two test used to verify that the VXI Interface is working on the Master Readout Module. This is the same test which was executed on the GeBGO Modules.

- Verify that resman has properly mapped both boards.
  NOTE: This first procedure may be skipped because the next step will not work if resman did not properly map the VXI Modules. It is left in the documentation as an example of how "vxitedit" works.
- Execute the VXI register test, which test the VXI registers internal to the VXI Interface Xilinx.

2.1.1 Verifying "resman" mapping

To verify that resman has properly mapped the boards, execute the following commands while in the C shell. This will verify that the VXI Interface Chip allowed the Resource Manager to map the two boards into the VXI address space.

->**vxitedit**
VXI Text Resource Editor (Ver 1.0)
(c) 1991, National Instruments

(1) resource manager display
(2) configuration editor
(3) manufacturer name editor
:
:
[VXItedit:]**1**

The two commands above put you into vxitedit resource manager display. When list is typed a list of devices plugged into the VXIbus are printed out. Since the Local Readout Module and the GeBGO Prototype Module each consist of a block of eight devices, Slot 1 should have eight devices and Slot 2 should have eight devices.

[VXItedit:ResMan:]**list**

Index	la	DevName	MfrName	ModelName	Frame	Slot
0	0	ROOT	Nat'l Insts	VXIcpu-030	0	0
1	8				0	1
2	9				0	1
:	:				:	:
8	f				0	1

The display command gives detail information about a specified device. By typing *display 0* information about the VXIcpu-030 will be displayed. By typing *display 1* information about the Local Readout Module will be displayed. By typing *display 9* information about the GeBGO Prototype Module will be displayed. Note that the only difference between display 1 and display 9 is the slot number.

```
[VXItedit:ResMan:]display 0
Fd #    Field Name          Field Value
----    ----------          -----------
Index = 0
(0)     Log Addr            0
(1)     Dev Name            ROOT
(2)     Cmdr La             ffff
(3)     Frame               0
(4)     Slot                0
(5)     ManId               ff6
(6)     Mfr Name            Nat' Insts
(7)     Model No            f7
(8)     Model Name          VXIcpu-030
(9)     Dev Class           2
(10)    Sub Class           0
(11)    Mem Space           3
(12)    Mem Base            ffffffff
(13)    Mem Size            0
(14)    Irq Level           none.
(14)    Irq Handle          7.
        Servants
            8
            :
            :
            17
[VXItedit:ResMan:] display 1
Fd #    Field Name          Field Value
----    ----------          -----------
Index = 1
(0)     Log Addr            8
(1)     Dev Name
(2)     Cmdr La
(3)     Frame               0
(4)     Slot                1
(5)     ManId               f00
(6)     Mfr Name
(7)     Model No            0
(8)     Model Name
(9)     Dev Class           3
(10)    Sub Class           0
(11)    Mem Space           3
(12)    Mem Base            ffffffff
(13)    Mem Size            0
(14)    Irq Level           none.
(14)    Irq Handle          none.
            Servants
            none ..
```

Get back to the C shell with the following commands:

```
[VXItedit:ResMan:] exit
[VXItedit:] exit
value = 0 = 0x0
->
```

2.1.2 Testing Internal VXI Registers

To test the internal VXI registers on the MRM, execute the following routine. Note that *sp mr_regtest,1* spawns a task running mr_regtest. The *,1* specifies which slot to execute the mr_regtest on. Do not worry if the task spawned id and name shown below is different from yours. The task name will be needed only if the task gets stuck in a loop. If this happens the task may be deleted by entering the command: *td "task name".*

->**ld < mr_regtest.o**
value = 0 = 0x0
->**sp mr_regtest,1**
task spawned: id = 0x3a9fd0, name = t1
value = 3842000 = 0x3a9fd0
->
This routine will test the VXI_INF registers for the board in:
Slot Number 1
PASSED: Internal VXI_INF registers.

Performing Dynamic Configuration Test.
Number of devices = 17
The LA's of the devices found are 0 8 9 10 11 12 13 14 15 16 17 18
The Number of open address Blocks for the BGO board = 28
The open addresses are:
18 20 28 30 40 48 50 58 60
68 70 78 80 88 90 a0 a8 b0
b8 c0 c8 d0 d8 e0 e8 f0
PASSED: Remapping Test
:
Resetting back to original address = 10
PASSED: Mapped back to original address

->

2.2 Dual Port Ram Test

The VXI side of the DPR (Dual Port Ram) on the MRM can be tested with the following command. In order to stop the test the operator must type *td t#* where # is the number of the task. The number of the task can be found by typing *i* which list the tasks and information about them. If an error is generated then either the Dual Port Ram or the VXI Interface is at fault.

->**sp mr_memtest,2** /* *,2* specifies the slot number */

task spawned: id = 0x3a9fd0, name = t9
value = 3842000 = 0x3a9fd0
-> VXI library already initialized

The Board's Base Address for slot 2 is c400
Memory cleared
Memory cleared verified

TESTING MEMORY .... /* Will keep testing until the task is stopped */

td t9

->

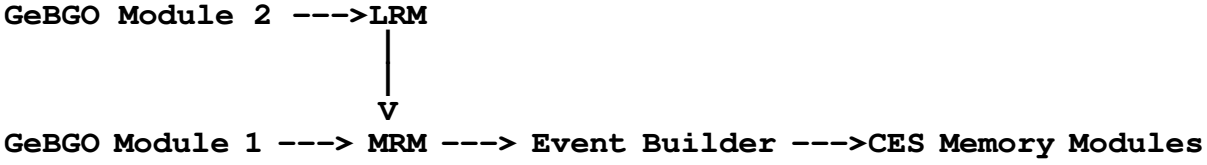
YOUR TASK NUMBER WILL PROBABLY BE DIFFERENT THAN THE ONE SHOWN HERE! TO PROPERLY DELETE THE TASK ENTER YOUR TASK NUMBER. TO VIEW ALL TASKS THAT ARE RUNNING ENTER THE COMMAND "*i*".

2.2 Data Flow Through the MRM

As described in the overview the data comes from the LRM to the MRM over the 32 bit differential ribbon cable bus. Because there is no way to read the data passing through the MRM, the only way to verify that the differential receivers and Fiber Optic transmitters are working is to verify that the data was properly passed through the MRM. The two possible locations to test the data is either in the FIFO's in the Event Builder or the data stored in the CES Memory Card.

Because the CES Memory Card is the final location of data, a System Test Program has been written, which generates test data in the GeBGO Modules and verifies that it has been transferred into the Memory Modules. This program is used to test the GammaSphere Data Acquisition System. If it passes then the MRM is properly passing data to the Event Builder.

DATA FLOW



The system test is controlled by a Sun Station on the network which is connected to the VME Rack and the VXI Mainframes. The VME and VXI nodes are running a task under VxWorks, which waits for Remote Procedure Calls (RPCs) from the network. When it receives a command it executes the task and returns the results.

In order to perform this test the operator must generate a configuration file and hook the modules up as defined in the configuration file. An example of a system test can be found in the "GammaSphere Data Acquisition Overview and System Test" documentation.

The system test was modified in June of 1994 in order to test the TOKEN Timeout and the RDYBSY Timeout functions. These features were added to the Master Readout's VXI Interface in order to assist in tracking down problems. The system test was also modified to read the status word in the Master Readout Module and report any errors that may occur.

2.3 Pulser FPGA Test

There are two types of test that can be performed on the Pulser FPGA. The first type of test is an Interactive Test where the operator types commands in by hand and makes measurements with an oscilloscope. The second type of test is an Automatic Test where the operator runs a program which uses the scaler chips to make measurements and verify that the Pulser chip is function correctly.

2.3.1 Interactive Pulser FPGA Test

The test below require the following setup:

Slot 1 - National Instruments VXI-cpu030 running VxWorks
Slot 2 - Master Readout Module with base address of 0xc200

Hook up an oscilloscope to measure the following signals:

U37 pin 1 = Pulser Frequency Signal
U37 pin 7 = Pulser Gate Signal

Interactive Pulser Frequency Test:

Give the following commands to the VXI Resource Manager running the C shell under VxWorks. The characters in bold print are typed by the user and normal print is returned by the VXI Resource Manager. The characters after the ";" are comments.

```
->ma16d16 0xc326                ; modify an A16 D16 address
0xc326:   0x???? -   0x0        ; Write zero into the Pulser Freq. latch
0xc326:   0x0000 -   .
->ma32d32 0xc224
0xc224:   0xfff0 -   0x8        ; Start the Pulser Freq Signal
0xc224:   0xfff8 -   .
```

After the above commands the operator should be able to see a 5KHz signal on U37 pin 1. The signal will be high for .1 msec and low for .1 msec. To test another frequency simply write another number into the Pulser Frequency Latch as shown below.

```
->ma16d16 0xc224
0xc224:   0x0000 -   0x100
0xc224:   0x0100 -
```

After the above command the frequency will now be 25.8Hz. The signal will be low for 25.7 msec and up for .1 msec. To test other frequencies simply write new numbers into the Pulser Frequency Latch. The equation in section 1.3.9 gives the frequency based on what number is entered into the Pulser Frequency Latch.

Interactive Pulser Gate Test:

Give the following commands to the VXI Resource Manager running the C shell under VxWorks. The characters in bold print are typed by the user and normal print is returned by the VXI Resource Manager. The characters after the ";" are comments.

```
->ma16d16 0xc226                ; modify an A16 D16 address
0xc326:   0x???? -   0x0        ; Write zero into the Gate latch
0xc326:   0x0000 -
```

Immediately after the number has been entered the Pulser FPGA will send a 1.57msec pulse out. To repeat the pulse reenter the integer. To test pulses of different widths enter different integers. See the examples below.

```
->ma16d16 0xc226
0xc326:   0x0000 -   0x1        ; generate 3.17 msec pulse
0xc326:   0x0000-    0x2        ; generate 4.78 msec pulse
0xc326:   0x0000 -   0x10       ; generate 27.1 msec pulse
0xc326:   0x0000 -   0x100      ; generate 410  msec pulse
```

2.3.2 Automatic Pulser FPGA Test

The Pulser Gate Signal goes over the Differential Ribbon Cable Bus to all of the LRMs. The LRMs route the Gate Signal to TTLTRG7. This trigger line is used by all the GeBGO Modules to control all of the Scaler's gate signal. The Pulser Frequency signal also goes over the Differential Ribbon Cable Bus to all of the LRMs. The LRMs route the Frequency signal to TTLTRG5. This trigger line goes to all of the GeBGO modules but was not used at this time. By temporally adding a jumper from TTLTRG5 to the F1 and F2 scaler test lemos the operator can make use of the following test procedure to verify the PULSER FPGA along with the Pulser's Interface to the LRMs and GeBGO Modules.

The following test requires that the Master Readout Module, Local Readout Module and GeBGO Prototype Module are in the same VXI crate. The test may be modified in the future to allow multiple VXI mainframes to be used. When using one VXI mainframe the Local bus between the MRM and any cards associated with it must be separated form the Local Readout Module and any cards associated with it. This is done by having an empty slot between the MRM group of modules and the LRM group of modules. A simple example of a possible setup is given below.

VXI Mainframe

SLOT 0 National Instruments VXIcpu-030
SLOT 1 Master Readout Module
SLOT 2 Blank
SLOT 3 Local Readout Module
SLOT 4 GeBGO Prototype Module

The program uses the Pulser FPGA to generates various frequencies with various gate pulse widths. These signals are used as inputs to the SCALER FPGAs on the GeBGO Prototype Module. The program verifies that the count stored in the Scaler FPGA matches what is expected for the given frequency and gate pulse width.

To execute the program simple enter the following command while in the C shell for the VXI mainframe.

->**sp pulser, mrm_slot, lrm_slot, ge_slot, channel_num**

where:	mrm_slot	=	Slot number of Master Readout Module
	lrm_slot	=	Slot number of Local Readout Module
	ge_slot	=	Slot number of the GeBGO Prototype Module
	channel_num	=	Channel number of the Scaler receiving the Pulser Signals. Channels 0-11 go to Scaler A Channels 12-23 go to Scaler B

After the command is executed the program will display the values being written into the PULSER's frequency and gate registers. It will also display the calculated period for the frequency and gate signals in msecs. If the number of counts in the scaler matches what is expected, the program will report that the test passed and will repeat the test for a new frequency and gate signal. This test will execute until the operator deletes the task with the *td task#* command. An example of the programs output is given below.

```
freq = 0x10,   gate = 0x5,    freq_time = 180,   gate_tm = 958    Passed!
freq = 0x10,   gate = 0x5,    freq_time = 340,   gate_tm = 958    Passed!
freq = 0x10,   gate = 0x5,    freq_time = 500,   gate_tm = 958    Passed!
freq = 0x10,   gate = 0x5,    freq_time = 660,   gate_tm = 958    Passed!
freq = 0x10,   gate = 0x5,    freq_time = 820,   gate_tm = 958    Passed!
freq = 0x10,   gate = 0x15,   freq_time = 180,   gate_tm = 3518   Passed!
freq = 0x20,   gate = 0x15,   freq_time = 340,   gate_tm = 3518   Passed!
freq = 0x30,   gate = 0x15,   freq_time = 500,   gate_tm = 3518   Passed!
freq = 0x40,   gate = 0x15,   freq_time = 660,   gate_tm = 3518   Passed!
:
:
```

APPENDIX A: ABEL PROGRAM FOR U41 (22V10) IN MRM

Pin Description of U41.

CLK (INPUT)
10Mhz Signal

TRIGGER (INPUT)
From ECLTRG0 controlled by the Trigger Module. It is an active HIGH signal.
Cannot produce trigger from the VXIcpu030 because this does not connect to ECLTRG lines.
Also controls an LED which flashes on for about 1.3 sec on the rising edge of TRIGGER.
Also goes to the SOFT_A input of the FASTOUT chip on the MRM. The strobe will cause the FASTOUT chip to load the Header Data from the MRM-DPR into the pipeline registers.

TRIGGER_RDY (OUTPUT)
Latches the TRIGGER input signal. It is used for internal logic in the 22V10. It stays latched until the READY goes High.

RDY_BSY (INPUT)
Comes into the MRM from the differential bus on the front panel ribbon cable. This signal is the OR of all RDY_BSY signals from all of the Ge_BGO Boards in the system.

A Low on this signal means that a GeBGO board is performing A/D conversions and/or transferring data into its pipeline registers. When this signal goes high then the GeBGO board is ready to transfer data out to the VXI Local Bus.

BSY (OUTPUT)
When RDY_BSY goes low BSY is latched high. It will stay latched high until READY goes high.

READY (OUTPUT)
READY goes high after a TRIGGER and a RDY_BSY have been received. When the READY signal goes high it clears TRIGGER_RDY and BSY.

TOKEN (OUTPUT)
TOKEN is latched high when READY goes high. It is connected to the HEAD_TOKEN on the MRM, which goes to the TOKEN_IN input of the FASTOUT chip on the MRM. It stays high until the HEAD_DONE input goes high.

TOKEN_IN on the MRM's FASTOUT chip sends the data in the pipeline registers out over the VXI's Local Bus. On the MRM the Local Bus is routed to the TAXI chips which send data over the 500 ft fiber optic cable.

HEAD_DONE (INPUT)

HEAD_DONE comes form the TOKEN_OUT pin of the FASTOUT chip on the MRM. When it goes high it indicates that the pipeline registers are finished clocking data out and that the next device can use the VXI's Local Bus.

When HEAD_DONE goes high it causes TOKEN to go low and TOKEN_OUT to go high.

TOKEN_OUT (OUTPUT)

TOKEN_OUT goes high when HEAD_DONE returns the token to the 22V10 by going high. TOKEN_OUT goes out of the LEMO connector to the TOKEN_IN LEMO input of the first LRM (LOCAL READOUT MODULE).

It stays high until the TOKEN is returned through the TOKEN_IN LEMO. When TOKEN_IN goes high then TOKEN_OUT goes low and C_TOKEN_O goes high.

TOKEN_IN (INPUT)

C_TOKEN_IN is returned from the last LRM. When it goes high it turns off TOKEN_OUT and C_TOKEN_O goes high.

C_TOKEN_O (OUTPUT)

C_TOKEN_OUT goes out on the MRM's VXI Local Bus to any GeBGO Boards that are present in the MRM's Mainframe. C_TOKEN_O goes high when the token is available. It stays high until TOKEN_RET goes high.

TOKEN_RET (INPUT)

Goes high when the last GeBGO Board in the MRM's Mainframe is finished putting its data out on the VXI Local Bus. The signal comes form the FASTOUT chip on the last GeBGO Board in the MRM's Mainframe.

When TOKEN_RET goes high the 22V10 lowers the C_TOKEN_O and drives the EOE (END OF EVENT) high.

EOE (OUTPUT)

EOE goes out whenever INHIBIT from the MTM becomes active. INHIBIT is used to end an event that was started with a Main Trigger, but stopped by the Master Trigger Module before the late trigger occurred. The event is to be cleared out of either the pipeline registers or the ADCs depending on location o the event to be canceled.

EOE goes out on the Differential BUS to the LRM, which transfer the EOE signal to all of the GeBGO Boards through the TTLTRG2 signal on the VXI backplane. This signal is an input to all of the FASTOUT chips on the GeBGO boards. The FASTOUT chips clear either the pipeline registers or the ADCs based on the state of the TOKEN_BUSY and TOKEN_IN signals.

TOKEN_BUSY (OUTPUT)

This signal is high whenever the TOKEN is busy. This signal is used to control an LED on the front panel of the MRM. The path of the token is outlined below.

MRM TOKEN INITIATION

- TOKEN first given to FASTOUT chip on MRM, which provides the header data for all events.
- TOKEN leaves the MRM through the LEMO Connector on its front panel and is passed to the first LRM.

LRM TOKEN PASSING ALGORITHM

- The first LRM passes the TOKEN over the VXI Local Bus, to the first GeBGO Board within its VXI mainframe
- After the first GeBGO Board has placed its data on the VXI Local Bus the token is passed to the next GeBGO board.
- The last GeBGO Board returns the Token to the LRM, which passes it out its LEMO connector to the next LRM.
- The process is repeated until the last LRM. After the last LRM returns the token back to the MRM.

MRM TOKEN PASSING ALGORITHM CONTINUED

- When the MRM receives the returned token form the last LRM it removes the token from its LEMO output and passes the token to the GeBGO Board within its VXI mainframe. This process is identical to what happens in the LRM's VXI mainframes. When the token is returned from the last GeBGO board the MRM removes the token on the VXI Local Bus and generates an EOE.

MRM_U41O.ABL

```
module MRM_U41
title 'GAMMASPHERE READOUT TOKEN CONTROL'

"Version G"
"Required that TOKEN_IN be low before an EOE is triggered"
"This will prevent TOKEN_BUSY being stuck high if there is a long delay "
"between TOKEN_OUT going low and TOKEN_IN going low.  TOKEN_OUT goes low
when"
"The token is returned to the MRM by TOKEN_IN going high.  Each device "
"reciving the token will lower its token_out as its token_in goes low."

"Version F"
"Modified TOKEN_BUSY to go high if TOKEN_IN is high."
"TOKEN_IN is high when the token is being returned or if TOKEN_IN is floating"

"Version H"
"Inverted RDY_BSY logic.  RDY_BSY is active high going into PAL"

"Version I"
"Change BSY logic so that BSY cannot become active unless TRIGGER_RDY"
"Is Latched"

"Version J"
"modify logic on EOE"
"add EOE_GEN and TOKEN_GEN

"Version K"
"Modify Logic on EOE"
"    Want EOE to be sent out when WAIT is active during a trigger"
"    Add EOE_GEN and TOKEN_GEN to all simulation"

"Version L"
"Modified Logic on TRIGGER and BSY.  While doing continuous inhibits due
"to Late Strobe failure I noticed that once and a while the Token would be
"passed arround.  This was due to Trigger not being cleared during the
"180 nsec Inhibit pulse.  This could only happen if the trigger in was active
"just before the INHIBIT occured.  Have INHIBIT really clear TRIGGER and BSY.

"Version M      11-15-94     jtw
"Modify so that the eoe that occurs during a wait only happens at the
"time that the token would have been passed arround.
"Remove TRGGER_RDY latch.  SEQ_LG1 hold TRIGGER until Token or EOE.
```

```
"Version N      12-5-95      jtw
"Where there is an INHIBIT add EOE except for EOE logic.
"Latch READY until either an EOE or the token is started being passed arround.

"Version O      12-16-95     jtw
"Forgot to add Reset to READY latch (not compiled yet)

        MRM_U41 device 'P22V10';

        CLK             pin 1;  "10MHz CLOCK
        TOKEN_RET       pin 2;  "TOKEN RETURN FROM CRATE BUS
        INHIBIT         pin 3;  "SIGNAL FROM TRIGGER MODULE TO
                                "STOP READOUT > 200 nsec
        TRIGGER         pin 4;  "START OF EVENT FROM TRIGGER MODULE
        RDY_BSY         pin 5;  "READY/BUSY FROM EXTERNAL BUS
        HEAD_DONE       pin 6;  "HEADER READOUT COMPLETE
        WAIT            pin 7;  "WAIT FEEDBACK FROM EVENT BUILDER
                                "DON'T START ANOTHER TOKEN
        TOKEN_IN        pin 8;  "TOKEN RETURN FROM EXTERNAL BUS
        RDY_BSYC        pin 9;  "READY/BUSY FROM MRM MAINFRAIM
        TOKEN_GEN       pin 10; "Sends out token regardless of RDY_BSY
        RESET           pin 11; "System Reset from VXI Backplane
        EOE_GEN         pin 13; "Generates EOE to clear GeBGO's if
                                "the token is not busy
                                "If token is busy and EOE is sent out then
                                "either the pipeline or ADCs are cleared
                                "depending on the GeBGO's Token status.

        READY           pin 14; "READY TO START NEW TOKEN
        BSY             pin 15; "RDY_BSY HAVING GONE BUSY AFTER TRIGGER
        TRIGGER_RDY     pin 16; "TRIGGER SIGNAL RECEIVED
        TOKEN_OUT       pin 17; "TOKEN OUT TO EXTERNAL BUS
        C_TOKEN_O       pin 18; "TOKEN OUT TO CRATE BUS
        TOKEN_BUSY      pin 19; "READOUT BUSY SIGNAL TO ALL CRATES
        EOE             pin 20; "END-OF-EVENT SIGNAL TO ALL CRATES
                                "How this effects data depends on the
                                "token_busy status and if the GeBGO has
                                "received the token
        HEAD_TOKEN      pin 21; "START OF HEAD_TOKEN PASSING
                                "TO HEADER READOUT CIRCUIT
        EXT_DATA        pin 22; "ENABLE SIGNAL FOR EXTERNAL BUS DATA
        INT_DATA        pin 23; "ENABLE SIGNAL FOR CRATE BUS DATA
```

```
        H,L,X,Z,C = 1,0,.X.,.Z.,.C.;

        READY, BSY, TRIGGER_RDY, TOKEN_OUT, C_TOKEN_O,
        TOKEN_BUSY, EOE, HEAD_TOKEN ISTYPE 'Reg_D';

equations

        TRIGGER_RDY :=  (TRIGGER & !INHIBIT & !EOE & RESET);
        TRIGGER_RDY.C  = CLK;
        TRIGGER_RDY.AR = 0;
        TRIGGER_RDY.SP = 0;
        TRIGGER_RDY.OE = H;

        BSY     :=      (RDY_BSYC & TRIGGER_RDY & !INHIBIT & !EOE & RESET) #
                        (RDY_BSY  & TRIGGER_RDY & !INHIBIT & !EOE & RESET) #
                        (BSY & !READY & !INHIBIT & !EOE & RESET);
        BSY.C   = CLK;
        BSY.AR  = 0;
        BSY.SP  = 0;
        BSY.OE  = H;

        "Should only stay one for one clock pulse
        READY   :=      (TRIGGER_RDY & BSY & !RDY_BSY & !RDY_BSYC
                         & !TOKEN_BUSY & !INHIBIT & !EOE & RESET) #
                        (READY & !TOKEN_BUSY & !INHIBIT & !EOE & RESET) ;
        READY.C  = CLK;
        READY.AR = 0;
        READY.SP = 0;
        READY.OE = H;

        HEAD_TOKEN   := (READY & !INHIBIT & !EOE & !WAIT & RESET) #
                        (TOKEN_GEN & !TOKEN_BUSY & !WAIT & RESET) #
                        (HEAD_TOKEN & !HEAD_DONE & RESET);
        HEAD_TOKEN.C  = CLK;
        HEAD_TOKEN.AR = 0;
        HEAD_TOKEN.SP = 0;
        HEAD_TOKEN.OE = H;

        TOKEN_OUT :=    (HEAD_DONE & HEAD_TOKEN & RESET)
                        # (TOKEN_OUT & !TOKEN_IN & RESET);
        TOKEN_OUT.C  = CLK;
        TOKEN_OUT.AR = 0;
        TOKEN_OUT.SP = 0;
        TOKEN_OUT.OE = H;
```

```
C_TOKEN_O :=    (TOKEN_IN & TOKEN_OUT & RESET)
              # (C_TOKEN_O & !TOKEN_RET & RESET);
C_TOKEN_O.C  = CLK;
C_TOKEN_O.AR = 0;
C_TOKEN_O.SP = 0;
C_TOKEN_O.OE = H;

"WILL GO BUSY IF HEAD_TOKEN BECOMES ACTIVE DUE TO TOKEN_GEN
TOKEN_BUSY :=   (READY & !INHIBIT & !EOE & !WAIT & RESET) #
              (HEAD_TOKEN) #
              (TOKEN_BUSY & !TOKEN_RET & RESET);
TOKEN_BUSY.C  = CLK;
TOKEN_BUSY.AR = 0;
TOKEN_BUSY.SP = 0;
TOKEN_BUSY.OE = H;

"EOE GOES OUT WHEN AN INHIBIT OCCURS, or THE EOE_GEN BIT IS SET
HIGH
"or THE WAIT IS ACTIVE WHEN THE TOKEN IS TO BE SENT OUT.
"THE FASTOUT FPGAs DETERMINE WHAT TO DO WITH EOE BASED ON
"TOKEN_BUSY, TOKEN_IN AND TOKEN_OUT.
"THE SEQ_LG1 PAL WILL REMOVE THE MASTER TRIGGER WHEN IT
RECEIVES AN
"EOE.
"EOE IS LATCHED UNTIL ALL OF THE RDY_BSY SIGNALS ARE LOW.  AND
THE
"TRIGGER SIGNAL IS LOW.

EOE     :=      (INHIBIT & RESET) #
              (EOE_GEN) #
              (READY & WAIT & RESET) #
              (EOE & RDY_BSY & RESET) #
              (EOE & RDY_BSYC & RESET) #
              (EOE & TRIGGER & RESET);
EOE.C  = CLK;
EOE.AR = 0;
EOE.SP = 0;
EOE.OE = H;

!EXT_DATA =     TOKEN_OUT & !C_TOKEN_O;
EXT_DATA.OE =H;

INT_DATA =      !EXT_DATA;
INT_DATA.OE = H;
```

```
"NORMAL READOUT SEQUENCE
TEST_VECTORS
([CLK,RESET,TRIGGER,RDY_BSY,RDY_BSYC,INHIBIT,HEAD_DONE,
        TOKEN_IN,TOKEN_RET,WAIT,TOKEN_GEN,EOE_GEN] ->

[TRIGGER_RDY,BSY,READY,HEAD_TOKEN,TOKEN_OUT,C_TOKEN_O,EOE,TOKE
N_BUSY,
        INT_DATA,EXT_DATA])
[C,L,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,L,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,L,L,L,L,L,L,L,H];
[C,H,H,H,L,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,H,L,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,H,H,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,L,H,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,L,H,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,H,H,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,H,H,L,L,L,H,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,L,H,L,L,L,H,L,H];
[C,H,L,L,L,L,H,L,L,L,L,L]  ->      [L,L,L,L,H,L,L,H,H,L];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,H,L,L,H,H,L];
[C,H,L,L,L,L,L,H,L,L,L,L]  ->      [L,L,L,L,L,H,L,H,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,H,L,H,L,H];
[C,H,L,L,L,L,L,L,H,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
"19
```

```
"READOUT WITH INHIBIT WITHIN RDY/BSY TIME OF SINGLE EVENT
"  EOE goes low after RDY_BSY & RDY_BSYC become inactive.
"  The MTM holds RDY_BSYC low until after late trigger so INHIBIT will
"  always occur while RDY_BSYC is active.
"  To properly clear trigger inhibit must be > 200 nsec
TEST_VECTORS
([CLK,RESET,TRIGGER,RDY_BSY,RDY_BSYC,INHIBIT,HEAD_DONE,
        TOKEN_IN,TOKEN_RET,WAIT,TOKEN_GEN,EOE_GEN] ->

[TRIGGER_RDY,BSY,READY,HEAD_TOKEN,TOKEN_OUT,C_TOKEN_O,EOE,TOKE
N_BUSY,
        INT_DATA,EXT_DATA])
[C,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L]  ->      [H,L,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L]  ->      [H,L,L,L,L,L,L,L,L,H];
[C,H,H,H,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,H,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,H,L,H,L,L,L,L,L]  ->      [L,L,L,L,L,L,H,L,L,H];
[C,H,L,H,L,H,L,L,L,L,L]  ->      [L,L,L,L,L,L,H,L,L,H];
[C,H,L,H,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,H,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
"30
```

```
"READOUT WITH INHIBIT WHILE EARILER EVENT IS BEING READ OUT
TEST_VECTORS
([CLK,RESET,TRIGGER,RDY_BSY,RDY_BSYC,INHIBIT,HEAD_DONE,
        TOKEN_IN,TOKEN_RET,WAIT,TOKEN_GEN,EOE_GEN] ->

[TRIGGER_RDY,BSY,READY,HEAD_TOKEN,TOKEN_OUT,C_TOKEN_O,EOE,TOKE
N_BUSY,
        INT_DATA,EXT_DATA])
[C,L,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,L,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,L,L,L,L,L,L,L,H];
[C,H,H,H,L,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,H,L,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,H,H,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,H,H,L,L,L,H,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,L,H,L,L,L,H,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,H,L,L,L,H,L,H];
[C,H,H,H,L,L,L,L,L,L,L,L]  ->      [H,L,L,H,L,L,L,H,L,H];
[C,H,H,H,L,L,L,L,L,L,L,L]  ->      [H,H,L,H,L,L,L,H,L,H];
[C,H,H,H,L,H,H,L,L,L,L,L]  ->      [L,L,L,L,H,L,H,H,H,L];
[C,H,L,H,L,H,L,L,L,L,L,L]  ->      [L,L,L,L,H,L,H,H,H,L];
[C,H,L,H,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,H,L,H,H,H,L];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,H,L,L,H,H,L];
[C,H,L,L,L,L,L,H,L,L,L,L]  ->      [L,L,L,L,L,H,L,H,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,H,L,H,L,H];
[C,H,L,L,L,L,L,L,H,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
"50
```

```
"READOUT WITH EVENT OCCURING DURING TOKEN PASSING TIME
TEST_VECTORS
([CLK,RESET,TRIGGER,RDY_BSY,RDY_BSYC,INHIBIT,HEAD_DONE,
        TOKEN_IN,TOKEN_RET,WAIT,TOKEN_GEN,EOE_GEN] ->

[TRIGGER_RDY,BSY,READY,HEAD_TOKEN,TOKEN_OUT,C_TOKEN_O,EOE,TOKE
N_BUSY,
        INT_DATA,EXT_DATA])
[C,L,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,L,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,L,L,L,L,L,L,L,H];
[C,H,H,H,L,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,H,L,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,H,H,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,H,H,L,L,L,H,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,L,H,L,L,L,H,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,H,L,L,L,H,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,L,H,L,L,L,H,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,L,H,L,L,L,H,L,H];
[C,H,H,H,L,L,H,L,L,L,L,L]  ->      [H,H,L,L,H,L,L,H,H,L];
[C,H,H,H,H,L,L,L,L,L,L,L]  ->      [H,H,L,L,H,L,L,H,H,L];
[C,H,H,L,H,L,L,L,L,L,L,L]  ->      [H,H,L,L,H,L,L,H,H,L];
[C,H,H,L,H,L,L,H,L,L,L,L]  ->      [H,H,L,L,L,H,L,H,L,H];
[C,H,H,L,H,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,H,L,H,L,H];
[C,H,H,L,H,L,L,L,H,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,H,H,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,H,H,L,L,L,H,L,H];
[C,H,H,L,L,L,L,L,L,L,L,L]  ->      [H,L,L,H,L,L,L,H,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,H,L,L,L,H,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,H,L,L,L,H,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,H,L,L,L,H,L,H];
[C,H,L,L,L,L,H,L,L,L,L,L]  ->      [L,L,L,L,H,L,L,H,H,L];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,H,L,L,H,H,L];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,H,L,L,H,H,L];
[C,H,L,L,L,L,L,H,L,L,L,L]  ->      [L,L,L,L,L,H,L,H,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,H,L,H,L,H];
[C,H,L,L,L,L,L,L,H,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
"83
```

```
"READOUT WITH EVENT OCCURING DURING TOKEN PASSING TIME AND
"WAIT OCCURS AFTER SECOND EVENT TRIGGERED
TEST_VECTORS
([CLK,RESET,TRIGGER,RDY_BSY,RDY_BSYC,INHIBIT,HEAD_DONE,
        TOKEN_IN,TOKEN_RET,WAIT,TOKEN_GEN,EOE_GEN] ->

[TRIGGER_RDY,BSY,READY,HEAD_TOKEN,TOKEN_OUT,C_TOKEN_O,EOE,TOKE
N_BUSY,
        INT_DATA,EXT_DATA])
[C,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L]  ->      [H,L,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L]  ->      [H,L,L,L,L,L,L,L,L,H];
[C,H,H,H,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,H,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L]  ->      [H,H,H,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L]  ->      [H,L,H,H,L,L,L,H,L,H];
[C,H,H,L,L,L,L,L,L,L,L]  ->      [H,L,L,H,L,L,L,H,L,H];
[C,H,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,H,L,L,L,H,L,H];
[C,H,H,L,L,L,L,L,L,L,L]  ->      [H,L,L,H,L,L,L,H,L,H];
[C,H,H,H,L,L,H,L,L,L,L]  ->      [H,H,L,L,H,L,L,H,H,L];
[C,H,H,H,H,L,L,L,L,L,L]  ->      [H,H,L,L,H,L,L,H,H,L];
[C,H,H,H,H,L,L,L,L,H,L]  ->      [H,H,L,L,H,L,L,H,H,L];
[C,H,H,L,H,L,L,H,L,H,L]  ->      [H,H,L,L,L,H,L,H,L,H];
[C,H,H,L,H,L,L,L,L,H,L]  ->      [H,H,L,L,L,H,L,H,L,H];
[C,H,H,L,H,L,L,L,H,H,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,H,L]  ->      [H,H,H,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,H,L]  ->      [H,L,H,L,L,L,H,L,L,H];
[C,H,H,L,L,L,L,L,L,H,L]  ->      [L,L,L,L,L,L,H,L,L,H];
[C,H,L,L,L,L,L,L,L,H,L]  ->      [L,L,L,L,L,L,L,L,L,H];
"106
```

```
"READOUT WITH EVENT OCCURING DURING TOKEN PASSING TIME AND
"WAIT BEFORE SECOND EVENT TRIGGERED
TEST_VECTORS
([CLK,RESET,TRIGGER,RDY_BSY,RDY_BSYC,INHIBIT,HEAD_DONE,
        TOKEN_IN,TOKEN_RET,WAIT,TOKEN_GEN,EOE_GEN] ->

[TRIGGER_RDY,BSY,READY,HEAD_TOKEN,TOKEN_OUT,C_TOKEN_O,EOE,TOKE
N_BUSY,
        INT_DATA,EXT_DATA])
[C,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,L,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L]  ->      [H,L,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L]  ->      [H,L,L,L,L,L,L,L,L,H];
[C,H,H,H,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,H,L,L,L,L,L,L,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,L,L]  ->      [H,H,H,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L]  ->      [L,L,H,H,L,L,L,H,L,H];
[C,H,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,H,L,L,L,H,L,H];
[C,H,H,L,L,L,L,L,L,H,L]  ->      [H,L,L,H,L,L,L,H,L,H];
[C,H,H,L,L,L,L,L,L,H,L]  ->      [H,L,L,H,L,L,L,H,L,H];
[C,H,H,H,L,L,H,L,L,H,L]  ->      [H,H,L,L,H,L,L,H,H,L];
[C,H,H,H,L,L,L,L,L,H,L]  ->      [H,H,L,L,H,L,L,H,H,L];
[C,H,H,L,H,L,L,L,L,H,L]  ->      [H,H,L,L,H,L,L,H,H,L];
[C,H,H,L,H,L,L,H,L,H,L]  ->      [H,H,L,L,L,H,L,H,L,H];
[C,H,H,L,H,L,L,L,L,H,L]  ->      [H,H,L,L,L,H,L,H,L,H];
[C,H,H,L,H,L,L,L,H,H,L]  ->      [H,H,L,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,H,L]  ->      [H,H,H,L,L,L,L,L,L,H];
[C,H,H,L,L,L,L,L,L,H,L]  ->      [H,L,H,L,L,L,H,L,L,H];
[C,H,H,L,L,L,L,L,L,H,L]  ->      [L,L,L,L,L,L,H,L,L,H];
[C,H,L,L,L,L,L,L,L,H,L]  ->      [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L]  ->      [L,L,L,L,L,L,L,L,L,H];
"122
```

```
"GENERATE TOKEN
TEST_VECTORS
([CLK,RESET,TRIGGER,RDY_BSY,RDY_BSYC,INHIBIT,HEAD_DONE,
        TOKEN_IN,TOKEN_RET,WAIT,TOKEN_GEN,EOE_GEN] ->

[TRIGGER_RDY,BSY,READY,HEAD_TOKEN,TOKEN_OUT,C_TOKEN_O,EOE,TOKE
N_BUSY,
        INT_DATA,EXT_DATA])
[C,L,L,L,L,L,L,L,L,L,L,L]  ->    [L,L,L,L,L,L,L,L,L,H];
[C,L,L,L,L,L,L,L,L,L,L,L]  ->    [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->    [L,L,L,L,L,L,L,L,L,H];
[C,H,X,X,X,L,L,L,L,L,H,L]  ->    [L,L,L,H,L,L,L,L,L,H];
[C,H,X,X,X,L,L,L,L,L,H,L]  ->    [L,L,L,H,L,L,L,H,L,H];
[C,H,X,X,X,L,H,L,L,L,H,L]  ->    [L,L,L,L,H,L,L,H,H,L];
[C,H,X,X,X,L,L,L,L,L,H,L]  ->    [L,L,L,L,H,L,L,H,H,L];
[C,H,X,X,X,L,L,H,L,L,H,L]  ->    [L,L,L,L,L,H,L,H,L,H];
[C,H,X,X,X,L,L,L,L,L,H,L]  ->    [L,L,L,L,L,H,L,H,L,H];
[C,H,X,X,X,L,L,L,H,L,H,L]  ->    [L,L,L,L,L,L,L,L,L,H];
[C,H,X,X,X,L,L,L,L,L,H,L]  ->    [L,L,L,H,L,L,L,L,L,H];
[C,H,X,X,X,L,L,L,L,L,H,L]  ->    [L,L,L,H,L,L,L,H,L,H];
[C,H,X,X,X,L,H,L,L,L,H,L]  ->    [L,L,L,L,H,L,L,H,H,L];
[C,H,X,X,X,L,L,L,L,L,H,L]  ->    [L,L,L,L,H,L,L,H,H,L];
[C,H,X,X,X,L,L,H,L,L,L,L]  ->    [L,L,L,L,L,H,L,H,L,H];
[C,H,X,X,X,L,L,L,L,L,L,L]  ->    [L,L,L,L,L,H,L,H,L,H];
[C,H,X,X,X,L,L,L,H,L,L,L]  ->    [L,L,L,L,L,L,L,L,L,H];
[C,H,X,X,X,L,L,L,L,L,L,L]  ->    [L,L,L,L,L,L,L,L,L,H];

"GENERATE EOE
TEST_VECTORS
([CLK,RESET,TRIGGER,RDY_BSY,RDY_BSYC,INHIBIT,HEAD_DONE,
        TOKEN_IN,TOKEN_RET,WAIT,TOKEN_GEN,EOE_GEN] ->

[TRIGGER_RDY,BSY,READY,HEAD_TOKEN,TOKEN_OUT,C_TOKEN_O,EOE,TOKE
N_BUSY,
        INT_DATA,EXT_DATA])
[C,L,L,L,L,L,L,L,L,L,L,L]  ->    [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->    [L,L,L,L,L,L,L,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L,H]  ->    [L,L,L,L,L,L,H,L,L,H];
[C,H,L,L,L,L,L,L,L,L,L,L]  ->    [L,L,L,L,L,L,L,H,L,H];

end MRM_U41
```

APPENDIX B: TEST SOFTWARE

This appendix contains all of the source code used to test the MRM. It also includes the start up script which is executed when the VXI crate is powered up. Listed below are all of the files included in this appendix.

File Name	Description
vxi_error.c	Global subroutines used by many of the following routines.
mr_regtest.c	sp mr_regtest, slot_number slot_number = The slot number of the LRM Board being tested. Routine test registers internal to the VXI_INF chip.
mr_memtest.c	sp mr_memtest, slot_number slot_number = The slot number of the GeBGO Board being tested. Routine test the VXI side of the Dual Port Ram on a Ge_BGO Board.
pulser.c	sp pulser, mrm_slot, lrm_slot, ge_slot, channel_num mrm_slot = Slot number of Master Readout Module lrm_slot = Slot number of Local Readout Module ge_slot = Slot number of the GeBGO Prototype Module channel_num = Channel number of the Scaler receiving the Pulser Signals. Channels 0-11 go to Scaler A Channels 12-23 go to Scaler B Routine uses scaler FPGA's in GeBGO prototype module to test PULSER FPGA.

UTILITYS

SCRIPTS:

c - Used to compile c programs to be executed on the VxWORKS Nodes.

BOOT UP FILES:

startup0.cmd - VXI Boot Up File

ROUTINES:

ma16d16.c ma16d16 (hex_address)
hex_address = 16 Bit hex address to read and possibly write.

Routine provides a way to read and write A16, D16 address on the VXI Bus. While running, this routine accepts the following commands.

<enter>	=	Read the next address.
hex_num<enter>	=	Modify the data at the current address and reads it back.
^<enter>	=	Read the previous address.
.<enter>	=	Exit the program.

wca16d16.c wca16d16 (hex_address, hex_data)
hex_address = 16 Bit hex address to write continuously to.
hex_data = 16 Bit hex data to write continuously.
Routine continuously writes the given data to the given address.

rca16d16.c rca16d16 (hex_address)
hex_address = 16 Bit hex address to read continuously to.
Routine continuously reads the given address. It reports the value read the first time and will report if the data ever changes.

APPENDIX C: PARTS LIST AND DATA SHEETS

MASTER READOUT MODULE Revised: May 3, 1993
L8825-9220-DD-00 Revision: A
Bill Of Materials May 3, 1993 15:12:20 Page 1

Item	Quantity	Reference	Part	
1	66	C001,C3,C4,C6,C11,C12, C17,C18,C20,C25,C26,C31, C32,C34,C39,C40,C45,C46, C48,C53,C54,C201,C202, C203,C204,C205,C206,C207, C208,C209,C210,C211,C212, C213,C214,C215,C216,C217, C218,C219,C220,C221,C222, C223,C224,C225,C226,C227, C228,C229,C230,C231,C232, C233,C234,C235,C236,C237, C238,C239,C240,C241,C242, C247,C248,C500	0.1	
2	4	C1,C15,C29,C43	1	
3	12	C2,C5,C7,C16,C19,C21,C30, C33,C35,C44,C47,C49	0.047	
4	8	C8,C9,C22,C23,C36,C37, C50,C51	0.01	
5	4	C10,C24,C38,C52	4.7	
6	16	C13,C14,R26,R27,C27,R28, C28,R29,R30,R31,R32,R33, C41,C42,C55,C56	0	
7	2	C57,C58	220	
8	2	C59,C60	0.22	
9	2	C300,C301	33/10V	
10	1	C302	33/10	
11	7	D1,D2,D3,D4,D5,D6,D7	LED	
12	3	F1,F2,F3	FUSE	Fuse Socket
13	4	ISO1,ISO2,ISO3,ISO4	259002-1	
14	1	ISO5	HFD3213	
15	1	JM1	JMP2	
16	2	J1T,J1B	CON40A	
17	3	J3,J4,J5	LEMO	
18	16	L1,L2,L3,L4,L5,L6,L7,L8, L9,L10,L11,L12,L13,L14, L15,L16	2.7uH	
19	9	P1C,P1B,P1A,P2C,P2B,P2A, P3C,P3B,P3A	DIN96	3 Din96 Connectors

MASTER READOUT MODULE Revised: May 3, 1993
L8825-9220-DD-00 Revision: A
Bill Of Materials May 3, 1993 15:12:20 Page 2

Item	Quantity	Reference	Part	
20	14	RP1,RP2,RP3,RP4,RP5,RP6, RP7,RP8,RP9,RP10,RP11, RP12,RP13,RP19	SIP10	
21	5	RP14,RP15,RP16,RP17,RP18	DIP8	
22	8	R1,R2,R5,R6,R9,R10,R13, R14	273	
23	8	R3,R4,R7,R8,R11,R12,R15, R16	75	
24	2	R17,R34	51	
25	2	R18,R19	330	
26	1	R20	22K	
27	1	R21	5.1K	
28	2	R22,R23	220K	
29	6	R24,R25,R35,R36,R37,R38	1K	
30	1	R39	820	
31	1	R40	5K	
32	1	U1	VXI-INF	
33	3	U2,U8,U13	XC1736A	8-pin Dip Socket
34	1	U3	74HC244	
35	2	U4,U5	74LS645	
36	1	U6	74LS08	
37	1	U7	PULSER	
38	1	U9	IDT7133	
39	2	U11,U10	IDT73201	
40	1	U12	FASTOUT	
41	5	U14,U15,U16,U17,U22	DS3892	
42	2	U18,U42	74LS123	
43	1	U19	74LS240	
44	1	U20	74S38	
45	1	U21	10125	
46	5	U23,U24,U25,U26,U27	DS3890	
47	2	U28,U48	74LS00	
48	9	U29,U30,U31,U32,U33,U34, U35,U36,U39	96F173	
49	2	U37,U38	96F172	
50	1	U40	74S140	
51	1	U41	22V10	Socket
52	4	U43,U44,U45,U46	AMD7968	
53	1	U47	74S32	
54	1	X1	XTAL-OSC	