# LOCAL READOUT MODULE

September 27, 1993

**1.0 OVERVIEW OF LOCAL READOUT MODULE** . . . . . 2
**1.1 Description of the LRM** . . . . . 2
**1.2 Block Diagram of the LRM** . . . . . 2
**1.3 Subsections of the LRM** . . . . . 4
**1.4 Memory Map** . . . . . 7

**2.0 TEST PROCEDURE FOR THE LOCAL READOUT MODULE** . . . . . 10
**2.1 VXI Interface Test** . . . . . 10
**2.1.1 Verifying "resman" mapping** . . . . . 11
**2.1.2 Testing Internal VXI Registers** . . . . . 14
**2.2 Data Flow Into LRM** . . . . . 15
**2.3 Data Flow out of LRM to Event Builder** . . . . . 16

**APPENDIX A: ABEL PROGRAM FOR 22V10s (U13 AND U27) IN LRM** . . . . . 17

**APPENDIX B: TEST SOFTWARE** . . . . . 21

**APPENDIX C: PARTS LIST AND DATA SHEETS** . . . . . 22

## 1.0 OVERVIEW OF LOCAL READOUT MODULE

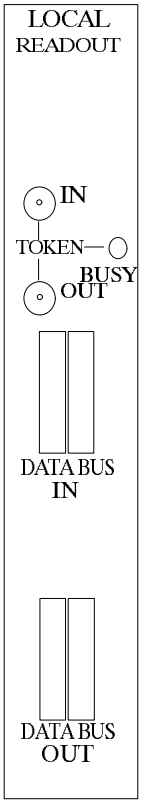
This section gives a brief overview of the LRM (Local Readout Module) which was designed and developed by Argonne National Laboratory's ECT Department (Electronics and Computing Technology). It is to be used by LBL (Lawrence Berkeley Laboratory) for data acquisition on the GammaSphere Project.

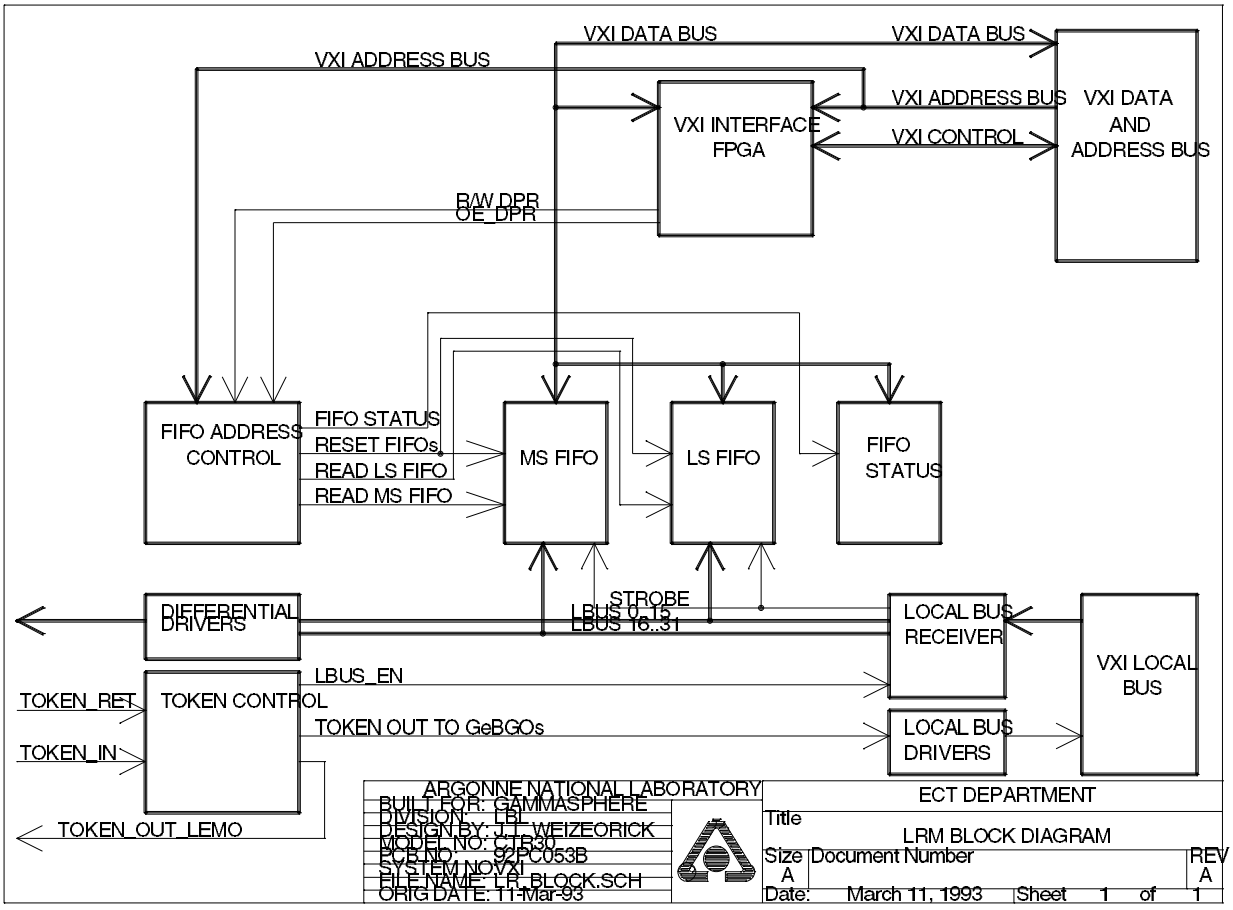
### 1.1 Description of the LRM

The main function of the LRM is to collect data from the GeBGO Modules local to its VXI mainframe and to transmit it to the MRM (Master Readout Module). The data is collected from the GeBGO Modules over the VXI Local Bus and is transmitted to the MRM through a differential ribbon cable bus. The differential ribbon cable bus is shared by all of the LRMs and the MRM. A token passing scheme is used to control which LRM may drive the differential bus. The MRM converts the 32 bit differential bus into four serial bit streams which are transmitted to the Event Builder over four fiber optic lines. The Event Builder then converts the 4 optical signals into a 32 bit wide differential ECL bus which clocks the data into a number of VME Memory Cards.

### 1.2 Block Diagram of the LRM

On the following page is a block diagram of the LRM. Section 1.3 describes the function of each section shown in the block diagram.




VXI DATA BUS
VXI DATA BUS
VXI ADDRESS BUS
VXI ADDRESS BUS
VXI DATA AND ADDRESS BUS
VXI INTERFACE FPGA
VXI CONTROL
R/W DPR
OE_DPR
FIFO ADDRESS CONTROL
FIFO STATUS
RESET FIFOs
READ LS FIFO
READ MS FIFO
MS FIFO
LS FIFO
FIFO STATUS
STROBE
LBUS 0..15
LBUS 16..31
DIFFERENTIAL DRIVERS
LOCAL BUS RECEIVER
VXI LOCAL BUS
LBUS_EN
TOKEN_RET
TOKEN CONTROL
TOKEN OUT TO GeBGOs
LOCAL BUS DRIVERS
TOKEN_IN
TOKEN_OUT_LEMO
ARGONNE NATIONAL LABORATORY
BUILT FOR: GAMMASPHERE
DIVISION: HEP
DESIGN BY: J.T.P. WEIZEORICK
MODEL NO: GTB30
PC NO: 92PC053B
SYSTEM NOVXI
FILE NAME: LR_BLOCK.SCH
ORIG DATE: 11-Mar-93
ECT DEPARTMENT
Title
LRM BLOCK DIAGRAM
Size A
Document Number
REV A
Date: March 11, 1993
Sheet 1 of 1

## 1.3 Subsections of the LRM

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

- VXI INTERFACE
- FIFOs
- TOKEN CONTROL
- VXI LOCAL BUS
- DIFFERENTIAL BUS

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 LRM. The VXI Interface is identical to the one used for the GeBGO Modules. For details see the specifications for the VXI INTERFACE FPGA which is in the documentation for the GeBGO VXI Interface Module.

The **FIFO** subsection consist of a 22V10 used to interface the VXI bus with the FIFOs, four 8K x 9 FIFOs, and a status register. The 22V10 maps the FIFO's and there status information into the VXI Memory Map. The FIFOs allow the VXI Resource Manager to look at the data coming form the GeBGO Modules over the VXI Local Bus. At the same time this data is being stored in the FIFOs it is being transmitted to the MRM over the 32 bit Differential Ribbon Cable Bus. The incoming data will always be passed out to the Differential Ribbon Cable Bus regardless of the status of the FIFOs. Because of this the operator may choose to ignore the LRM FIFOs and let them overflow. The main function of the LRM FIFO's is to allow the local Resource Manager look at the data being sent out. This is useful for diagnostic purposes and for sampling event data.

The **TOKEN CONTROL** subsection consist of a 22V10, which controls the token passing scheme, two lemo connectors for token in and out, and a token driver and receiver. The token starts as an active high signal which comes into the LRM from the Token In Lemo connector. If the token input is enabled the signal goes into the 22V10 and will perform the following if the token is high.

- Enables the LRM's differential drivers for the Differential Ribbon Cable Bus
- Turns on the TOKEN LED
- Passes the TOKEN to the first GeBGO board

When each GeBGO Module receives the token if it has data it will transmit it to the LRM over the VXI Local Bus. When the GeBGO is done clocking out its data or if it has no data then it will pass the TOKEN to the next GeBGO Module. The last GeBGO module returns the TOKEN to the 22V10 on the LRM which removes the TOKEN form its VXI Local Bus and passes it to the next LRM via its Token Out Lemo connector. If it is the last LRM then the Token Out Lemo is connected to the MRM and the LRM returns the token to the MRM.

The **VXI LOCAL BUS** consist of bus drivers and bus receivers used for transmitting 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 TOKE_RET, RDY_BSY_BD and the STRBRD signals.

TOKE_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 TOKE_RET when it is returning the token to the LRM.

RDY_BSY_BD — This input signal is a VXI Local Bus signal which is the wired OR of the RDY/BSY signals from all of the GeBGO Modules. The Local Bus RDY/BSY signal is normally pulled high and goes low when any of the GeBGO Modules drive their RDY/BSY signals low. The VXI Local Bus receiver inverts the signal so that the RDY_BSY_BD is active high. The LRM OR's the RDY_BSY_BD signal with a differential RDY/BSY input and sends it out to the next LRM or MRM.

STRBRD — This input signal is used to latch the data form the GeBGO boards. The data on the 32 bit bus is valid on the rising edge of STRBRD.

A bus driver is used drive the TOKEN_O signal which is used to control which GeBGO board may put data on the VXI Local Bus.

The **Differential Bus** is an external 32 bit differential bus which is transmitted over ribbon cables. This bus has a separate input and output connector on the front panel of the LRMs. Most of the differential signals are feed through from the input connector to the output connector. The only exception is the RDY/BSY signal which is described below. The last LRM terminates the differential bus by having a termination block plugged into its differential input connector. The data on this differential bus flows from all of the LRMs into the MRM. 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 signals which enter the LRM from the Differential Bus.

RDY\BSY — The input is from the previous LRM. The input is ORed with the local RDY\BSY signals and is sent out to the next LRM or the MRM. The function of this signal is to provide the MRM with a logical OR of all the RDY\BSY signals from all the GeBGO Modules in all of the VXI Mainframes.

RUN — From a VXI control bit on the MRM. The input is used to drive TTLTRG0, which makes this signal available to all of the GeBGO modules.

BUSY — From the 22V10 on MRM. The input is used to drive TTLTRG1, which makes it available to all of the modules in the LRM's VXI mainframe.

EOE — From the 22V10 on the MRM. The input is used to drive TTLTRG2. This signal is used by all of the GeBGO Modules to abort any events that are waiting to be collected.

PULSE — From the Pulser FPGA on the MRM. The input is used to drive TTLTRG5, which is 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 — A VXI control bit on the MRM. The input is used to drive TTLTRG6 and resets all of the Scalers on every GeBGO Modules.

GATE — From the Pulser FPGA on the MRM. The input is used to drive TTLTRG7. The Pulser FPGA provides a pulse whose width may be controlled through the MRM's VXI interface. The signal is used to enable all of the scaler inputs on every GeBGO modules.

## 1.4 Memory Map

The Local 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 0000 0000 0000 |
| 04H | STATUS REGISTER | 0M00 0000 0000 1100<br>M = MODID BIT |
| 06H | OFFSET REGISTER | 0000 0000 0000 1000 |

| OFFSET FROM BASE | WRITE FUNCTION | INITIAL VALUE |
|---|---|---|
| 00H | LOGICAL ADDRESS REGISTER | xxxx xxxx PPPP Pxxx<br>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.
Offset Reg - 8 devices (512 bytes) sharing address decoding.

| Device | Quad | Relative Address | Use | Function | Dev 0 Offset Address | Dev Offset Address |
|---|---|---|---|---|---|---|
| 0 | 0 | 000...00f | VXI Configuration Registers | Read ID; Write Logical Address<br>Read Device Type<br>Read Status; Write Control<br>Read/Write Offset Register | 000h<br>002h<br>004h<br>006h | 00h<br>02h<br>04h<br>06h |
| 0 | 1 | 010...01f | Not Used | | 010h | 10h |
| 0 | 2 | 020...02f | Latched Bits | Read/Write latched bits<br>Not Used<br>TRIGGER ENABLE bit<br>4 Bit Latch<br>0x1 - TOKEN ENABLE active high<br>0x2 -> 0x8 - Not Used<br>Not Used | <br>020h<br>022h<br>024h<br><br><br>026h | <br>20h<br>22h<br>24h<br><br><br>26h |
| 0 | 3 | 030...03f | FIFO CONTROL | READ LS FIFO WORD<br>READ MS FIFO WORD<br>READ FIFO STATUS<br>WRITE RESETS FIFOs | 030h<br>032h<br>034h<br>036h | 30h<br>32h<br>34h<br>36h |
| 1 | 0 | 040...04f | VXI Configuration Registers | Points to same Registers as Device 0 | | 00h |
| 1 | 1 | 050...05f | Not Used | | 050h | 10h |
| 1 | 2 | 060...06f | Not Used | | 060h | 20h |
| 1 | 3 | 070...07f | Not Used | | | |

| Device | Quad | Relative Address | Use | Function | Dev 0 Offset Address | Dev Offset Address |
|---|---|---|---|---|---|---|
| 2 | 0 | 080...08f | VXI Configuration Registers | Points to same Registers as Device 0 | | 00h |
| 2 | 1 | 090...09f | Not Used | | | |
| 2 | 2 | 0a0...0af | Not Used | | | |
| 2 | 3 | 0b0...0bf | Not Used | | | |
| 3 | 0 | 0c0...0cf | VXI Configuration Registers | Points to same Registers as Device 0 | | 00h |
| 3 | 1 | 0d0...0df | Not Used | | | |
| 3 | 2 | 0e0...0ef | Not Used | | | |
| 3 | 3 | 0f0...0ff | Not Used | | | |

| Device | Quad | Relative Address | Use | Function | Dev 0 Offset Address | Dev Offset Address |
|---|---|---|---|---|---|---|
| 4 | 0 | 100...1ff | VXI Configuration Registers | Points to same Registers as Device 0 | | 00h |
| 4 | 1 | 110...11f | Not Used | | | |
| 4 | 2 | 120...12f | Not Used | | | |
| 4 | 3 | 130...13f | Not Used | | | |
| 5 | 0 | 140...14f | VXI Configuration Registers | Points to same Registers as Device 0 | | 00h |
| 5 | 1 | 150...15f | Not Used | | | |
| 5 | 2 | 160...16f | Not Used | | | |
| 5 | 3 | 170...17f | Not Used | | | |

| Device | Quad | Relative Address | Use | Function | Dev 0 Offset Address | Dev Offset Address |
|---|---|---|---|---|---|---|
| 6 | 0 | 180...18f | VXI Configuration Registers | Points to same Registers as Device 0 | | 00h |
| 6 | 1 | 190...19f | Not Used | | | |
| 6 | 2 | 1a0...1af | Not Used | | | |
| 6 | 3 | 1b0...1bf | Not Used | | | |
| 7 | 0 | 1c0...1cf | VXI Configuration Registers | Points to same Registers as Device 0 | | 00h |
| 7 | 1 | 1d0...1df | Not Used | | | |
| 7 | 2 | 1e0...1ef | Not Used | | | |
| 7 | 3 | 1f0...1ff | Not Used | | | |

## 2.0 TEST PROCEDURE FOR THE LOCAL READOUT MODULE

The sections below give a step by step procedures used to test out the LRM. All of the test procedures require the following setup in the VXI Mainframe.

SLOT 0 National Instruments VXIcpu-030

SLOT 1 Local Readout Module
SLOT 2 GeBGO Prototype 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 GeBGO Board.

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 Local 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 |
| 9 | 10 | | | | 0 | 2 |
| 10 | 11 | | | | 0 | 2 |
| : | : | | | | : | : |
| 16 | 17 | | | | 0 | 2 |

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 | |
| | 9 | |
| | : | |
| | : | |
| | 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 .. | |

```
[VXItedit:ResMan:] display 9
Fd #    Field Name          Field Value
----  ----------            -----------
Index = 9
(0)     Log Addr            10
(1)     Dev Name
(2)     Cmdr La
(3)     Frame               0
(4)     Slot                2
(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 LRM, execute the following routine. Note that *sp regtest,1* spawns a task running regtest. The *,1* specifies which slot to execute the regtest on. Do not worry if the task spawned id and name shown below is different from yours. It is good to remember the task name because this may be needed to deleted the task. This is done by entering the command: *td "task name"*

```
->ld < regtest.o
value = 0 = 0x0
->sp 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 Data Flow Into LRM

The program ge_lrm.c generates soft triggers, and verifies that data flows properly for a number of different trigger methods.

DATA FLOW:

| | | |
|---|---|---|
| GeBGO Prom | -> | GeBGO's DPR, and Local Readout's FIFOs' |
| GeBGO DPR | -> | Local Readout's FIFOs' |

Each of the transfer methods listed above are triggered Three different ways.

Soft_A by itself
Soft_B by itself
Soft_A and Soft_B together

The ge_lrm.c generates its own TOKEN_IN provided the user pulls up the TOKEN_IN input on the Local Readout Module.

Before the test is run make sure the following conditions are met:

- Proper Boards plugged into the proper slots as described in the beginning of this section.
- JM28 and JM29 are set so that the U31 and U32 PROMS are enabled by the U30 22v10.
- The TOKEN_IN lemo connector on the Local Readout Module is pulled to +5 Volts.

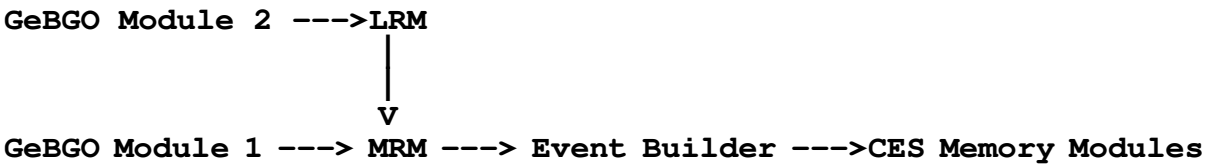
```
->sp ge_lrm,1,2,0xfff      /* 1      = Slot number of LRM board */
                           /* 2      = Slot number of GeBGO board */
                           /*    0xfff = number of passes on test */
task spawned: id = 0x3a9fd0, name = t4
value = 384200 = 0x3a9fd0
->
This program transfers data in the following modes.
    GeBGO Prom and ID Latch  ->   GeBGO DPR and LRM FIFO.
    GeBGO Dual Port Ram           ->   LRM FIFO
The data transfers are triggered in three ways:
    Soft A by itself
    Soft B by itself
    Soft A and Soft B together before token_in.

National Instruments RM:   0
Local Readout:             1
GeBGO Test Module:         2

PULL TOKEN_IN HIGH ON THE LOCAL READ OUT MODULE HIGH!!!

 Testing .....................
.....................

PASSED: ge_lrm has finished     511 passes
```

## 2.3 Data Flow out of LRM to Event Builder

As described in the overview the data coming into the LRM is stored in the FIFO's at the same time it is transmitted to the MRM over the 32 bit differential ribbon cable bus. Because there is no way to read the data coming into the MRM the only way to verify that the differential drivers are working on the LRM is to verify the data somewhere down stream. 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 form the GeBGOs we have written a System Test Program 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 differential drivers out of the LRM. If the test fails the error may be in the LRM, the MRM, the Event Builder, or the Memory Module.

**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.

## APPENDIX A: ABEL PROGRAM FOR 22V10s (U13 AND U27) IN LRM

```
File LRM_U13.ABL
module RDFIFO
title   'INTERFACE BETWEEN XILINX AND FIFOS ON LOCAL READOUT MODULE'

        RDFIFO device 'P22V10';

        AD1             PIN 1;  "ENABLE A1 INPUT
        AD2             PIN 2;  "ENABLE A2 INPUT
        AD3             PIN 3;  "ENABLE A3 INPUT
        AD4             PIN 4;  "ENABLE A4 INPUT
        AD5             PIN 5;  "ENABLE A5 INPUT
        AD6             PIN 6;  "ENABLE A6 INPUT
        AD7             PIN 7;  "ENABLE A7 INPUT
        AD8             PIN 8;  "ENABLE A8 INPUT
        I9              PIN 9;  "NOT USED
        I10             PIN 10; "NOT USED
        RW_DPR      PIN 11; "ENABLE FIFO RESET OUTPUT
        OE_DPR      PIN 13; "ENABLE FIFO LEAST, MOST, STATUS OUTPUTS
        O10             PIN 14; "NOT USED
        O9              PIN 15; "NOT USED
        O8              PIN 16; "NOT USED
        LEAST           PIN 17; "ENABLE LS BYTE ACTIVE LOW
        MOST            PIN 18; "ENABLE MS BYTE ACTIVE LOW
        STATUS          PIN 19; "ENABLE STATUS BUFFERS
        RESET_FIFO      PIN 20; "RESET ALL FIFO
        O3              PIN 21; "NOT USED
        O2              PIN 22; "NOT USED
        O1              PIN 23; "NOT USED

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

        ADDRS = [AD8, AD7, AD6, AD5, AD4, AD3, AD2, AD1, .X.];

equations

  "LEAST = LOW WHEN ADDRESS 30 AND OE_DPR LOW"
  LEAST  = !(!AD8 * !AD7 * !AD6 * AD5 * AD4 * !AD3 * !AD2 * !AD1 * !OE_DPR);
  LEAST.OE = H;

  "MOST = LOW WHEN ADDRESS 32 AND OE_DPR LOW"
  MOST  = !(!AD8 * !AD7 * !AD6 * AD5 * AD4 * !AD3 * !AD2 * AD1 * !OE_DPR);
  LEAST.OE = H;

  "STATUS = LOW WHEN ADDRESS 34 AND OE_DPR LOW"
  STATUS  = !(!AD8 * !AD7 * !AD6 * AD5 * AD4 * !AD3 * AD2 * !AD1 * !OE_DPR);
  STATUS.OE = H;
```

```
  "RESET_FIFO = HIGH WHEN ADDRESS 36 AND RW_DPR LOW"
    RESET_FIFO  = !(!AD8 * !AD7 * !AD6 * AD5 * AD4 * !AD3 * AD2 * AD1 * !RW_DPR);
  RESET_FIFO.OE = H;

test_vectors

([OE_DPR,RW_DPR,ADDRS]  -> [LEAST,MOST,STATUS,RESET_FIFO])
[1,      1,    .X.   ]  -> [H,    H,   H,     H];

[0,      1,   ^h0    ]  -> [H,    H,   H,     H];
[0,      1,   ^h2    ]  -> [H,    H,   H,     H];
[0,      1,   ^h4    ]  -> [H,    H,   H,     H];
[0,      1,   ^h6    ]  -> [H,    H,   H,     H];
[0,      1,   ^h8    ]  -> [H,    H,   H,     H];
[0,      1,   ^ha    ]  -> [H,    H,   H,     H];
[0,      1,   ^hc    ]  -> [H,    H,   H,     H];
[0,      1,   ^he    ]  -> [H,    H,   H,     H];
[0,      1,   ^h10   ]  -> [H,    H,   H,     H];
[0,      1,   ^h12   ]  -> [H,    H,   H,     H];
[0,      1,   ^h14   ]  -> [H,    H,   H,     H];
[0,      1,   ^h16   ]  -> [H,    H,   H,     H];
[0,      1,   ^h18   ]  -> [H,    H,   H,     H];
[0,      1,   ^h1a   ]  -> [H,    H,   H,     H];
[0,      1,   ^h1c   ]  -> [H,    H,   H,     H];
[0,      1,   ^h1e   ]  -> [H,    H,   H,     H];

[0,      1,   ^h30   ]  -> [L,    H,   H,     H];
[0,      1,   ^h32   ]  -> [H,    L,   H,     H];
[0,      1,   ^h34   ]  -> [H,    H,   L,     H];
[0,      1,   ^h36   ]  -> [H,    H,   H,     H];
[1,      0,   ^h36   ]  -> [H,    H,   H,     L];

end RDFIFO
```

```
File: LRM_U27.abl
module LRM_U27
title   'TOKEN CONTROL FOR LOCAL READOUT MODULE'

"DATE       WHO     WHAT"
"6-23-93    JTW     MODIFIED TOKEN_LED"

        LRM_U27 device 'P22V10';

        I1                  PIN 1;  "NOT USED
        TRIG_EN             PIN 2;  "Input to inverter
        I3                  PIN 3;  "NOT USED
        I4                  PIN 4;  "NOT USED
        I5                  PIN 5;  "NOT USED
        TOKEN_RET           PIN 6;  "TOKEN RETURN FROM LOCAL BUS-ACTIVE LOW
        I7                  PIN 7;  "NOT USED
        TOKEN_IN            PIN 8;  "TOKEN INTO THIS CRATE-ACTIVE LOW
        I9                  PIN 9;  "NOT USED
        I10                 PIN 10; "NOT USED
        RDY_BSY_BD          PIN 11; "first Input to OR gate
        RDY_BSY_DIF         PIN 13; "second Input to OR gate
        RDY_BSY_O           PIN 14; "Output of OR gate
        O9                  PIN 15; "NOT USED
        O8                  PIN 16; "NOT USED
        TOKEN_OUT           PIN 17; "SENDS TOKEN OUT OF THE CRATE-ACTIVE LOW
        DATA_EN             PIN 18; "ENABLE DIFFERENTAL OUTPUTS WHEN LOW
        TOKEN_O             PIN 19; "SENDS TOKEN OUT OVER LOCAL BUS
        TOK_LED             PIN 20; "TURNS ON THE TOKEN LED WHEN LOW
        O3                  PIN 21; "NOT USED
        TRIG_EN_INV         PIN 22; "Output of inverter
        RETL                PIN 23; "NOT USED

        H,L,X,Z,C,D,U = 1,0,.X.,.Z.,.C.,.D.,.U.;

equations

  "SEND TOKEN OUT OF CRATE WHEN TOKEN_RET HAS COME IN AND
  "TOKEN_I IS LOW"
  TOKEN_OUT  =  RETL & TOKEN_IN;
  TOKEN_OUT.OE = H;

  "SEND THE TOKEN OVER THE LOCAL BUS WHEN TOKEN_IN GOES IS LOW"
  TOKEN_O      = !RETL & TOKEN_IN;
  TOKEN_O.OE   = H;
```

```
  "ENABLE DATA WHILE TOKEN_O IS LOW"
  DATA_EN = !TOKEN_O;
  DATA_EN.OE = H;

  "TURN ON THE LED WHILE TOKEN_IN IS ACTIVE HIGH"
  "NOTE: THE LEMO INPUT IS INVERTED BEFORE IT GOES TO TOKEN_IN"
  "WITH THIS CHANGE IF THE TOKEN IS ACTIVE INTO THE LRM THE LED IS ON"
  "THIS MEANS THAT THE LED WILL BE ON WHEN THE TOKEN_IN IS FLOATING"
  TOK_LED = !TOKEN_IN;
  TOK_LED.OE = H;

  RETL = ((RETL & TOKEN_IN) # (TOKEN_RET & TOKEN_IN));
  RETL.OE = H;

  TRIG_EN_INV = !TRIG_EN;
  TRIG_EN_INV.OE = H;

  RDY_BSY_O = RDY_BSY_BD # RDY_BSY_DIF;
  RDY_BSY_O.OE = H;

test_vectors

([TOKEN_RET,TOKEN_IN]  -> [TOKEN_OUT,DATA_EN,TOKEN_O,TOK_LED,RETL])
[        0,    0   ]  -> [       0,      1,      0,      1,    0];
[        0,    U   ]  -> [       0,      0,      1,      D,    0];
[        0,    1   ]  -> [       0,      0,      1,      0,    0];
[        U,    1   ]  -> [       1,      1,      0,      0,    1];
[        1,    1   ]  -> [       1,      1,      0,      0,    1];
[        D,    1   ]  -> [       1,      1,      0,      0,    1];
[        0,    1   ]  -> [       1,      1,      0,      0,    1];
[        0,    1   ]  -> [       1,      1,      0,      0,    1];
[        0,    D   ]  -> [       0,      1,      0,      U,    0];
[        0,    0   ]  -> [       0,      1,      0,      1,    0];
[        0,    0   ]  -> [       0,      1,      0,      1,    0];

test_vectors
([TRIG_EN   ]  ->  [TRIG_EN_INV])
[     1     ]  ->  [    0      ];
[     0     ]  ->  [    1      ];

test_vectors
([RDY_BSY_BD, RDY_BSY_DIF] -> [RDY_BSY_O])
[     X     ,    1       ] -> [   1     ];
[     1     ,    X       ] -> [   1     ];
[     0     ,    0       ] -> [   0     ];
end LRM_U27
```

## APPENDIX B: TEST SOFTWARE

This appendix contains all of the source code used to test the LRM. 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. |
| regtest.c | regtest (slot_number)<br>slot_number = The slot number of the LRM Board being tested.<br>Routine test registers internal to the VXI_INF chip. |
| ge_lrm.c | ge_lrm (slot_num_lrm, slot_num_ge, num_passes)<br>slot_num_lrm = The slot number of the Local Readout Module being tested.<br>slot_num_ge = The slot number of the Ge_BGO Board being tested.<br>num_passes = The number of passes to make on the test. |

**UTILITYS**

SCRIPTS:

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

ROUTINES:

| | |
|---|---|
| ma16d16.c | ma16d16 (hex_address)<br>hex_address = 16 Bit hex address to read and possibly write.<br><br>Routine provides a way to read and write A16, D16 address on the VXI Bus. While running, this routine accepts the following commands.<br><enter> = Read the next address.<br>hex_num<enter> = Modify the data at the current address and reads it back.<br>^<enter> = Read the previous address.<br>.<enter> = Exit the program. |
| wca16d16.c | wca16d16 (hex_address, hex_data)<br>hex_address = 16 Bit hex address to write continuously to.<br>hex_data = 16 Bit hex data to write continuously.<br>Routine continuously writes the given data to the given address. |
| rca16d16.c | rca16d16 (hex_address)<br>hex_address = 16 Bit hex address to read continuously to.<br>Routine continuously reads the given address. It reports the value read the first time and will report if the data ever changes. |
| dump_fifo.c | dump_fifo (slot_number)<br>slot_number = The slot number of the Local Readout Module with the FIFO's to be Read.<br>Routine provides a way to read the FIFO's on the Local Readout Module. |

## APPENDIX C: PARTS LIST AND DATA SHEETS

LOCAL READOUT MODULE Revised: March 11, 1993
L8825-9210-DD-00 Revision: B
Bill Of Materials March 13, 1993 8:45:03 Page 1

| Item | Quantity | Reference | Part |
|---|---|---|---|
| 1 | 36 | C201,C203,C204,C205,C206, C207,C208,C209,C210,C211, C212,C213,C214,C215,C216, C217,C218,C219,C220,C222, C223,C224,C225,C226,C227, C228,C229,C230,C231,C232, C233,C234,C235,C236,C237, C321 | 0.1 |
| 2 | 2 | C300,C302 | 33/10V |
| 3 | 3 | D1,D2,D3 | LED D1-STOP = RED<br>D2-RUN = GREEN<br>D3-TOKEN = RED |
| 4 | 2 | F1,F2 | FUSE two Fuse Socket |
| 5 | 4 | J1T,J1B,J2T,J2B | CON40A two 80 pin condo connectors |
| 6 | 2 | J3,J4 | LEMO |
| 7 | 9 | P1C,P1B,P1A,P2C,P2B,P2A P3C,P3B,P3A | DIN96 three DIN96 Connectors |
| 8a | 5 | RP1,RP3,RP5,RP7,RP9 | SIP10 120 ohms |
| 8b | 5 | RP2,RP4,RP6,RP8,RP10 | SIP10 82 ohms |
| 8c | 3 | RP11,RP12,RP13 | SIP10 4.7 kohms |
| 9 | 3 | R1,R7,R9 | 1K |
| 10 | 2 | R2,R3 | 0 |
| 11 | 3 | R4,R5,R6 | 330 |
| 12 | 1 | R8 | 100 |
| 13 | 1 | R13 | 51 |
| 14 | 1 | R14 | 820 |
| 15 | 1 | U1 | VXI-INF 84 pin PLCC Socket |
| 16 | 1 | U2 | XC1736A 8 pin Dip Socket |
| 17 | 1 | U3 | 74HC244 |
| 18 | 2 | U4,U5 | 74LS645 |
| 19 | 1 | U6 | 74LS08 |
| 20 | 4 | U7,U8,U9,U10 | IDT7205 |
| 21 | 2 | U12,U11 | 74LS244 |
| 22 | 2 | U27,U13 | 22V10 two 24 pin Dip Sockets |
| 23 | 5 | U14,U15,U16,U17,U22 | DS3892 |
| 24 | 10 | U18,U19,U30,U31,U32,U33, U34,U35,U36,U37 | 96F172 |
| 25 | 1 | U20 | 74S38 |
| 26 | 1 | U23 | DS3890 |
| 27 | 2 | U25,U24 | 96F173 |
| 28 | 1 | U26 | 74LS240 |
| 29 | 1 | U28 | 74LS00 |
| 30 | 1 | U29 | 74S140 |