FERA to VXI Interface Module

April 27, 1995

1.0 OVERVIEW OF FERA to VXI Interface Module 3
1.1 Description of the FERA to VXI Interface Module 3
1.2 Block Diagram of the FERA to VXI Interface Module 7

2.0 Subsections of the FERA to VXI Interface Module 8
2.1 FERA to VXI Interface FPGA 10
2.2 Control Signals used for FERA DATA Read In 13
2.2.1 Six signals between the FVI and the 4301 FERA Driver . . . 13
2.2.2 Pre-Trigger and Gate Lemo Inputs 14
2.2.3 Master-Trigger Input, from ECLTRIG0 on the VXI Bus 14
2.2.4 RDY_BSY to LBUSC34 on the VXI Bus. 14
2.2.5 Four Position DIP Switch 14

3.0 Data Format from FERA Devices 16

4.0 FERA to VXI Interface Board Schematic 17

5.0 FVI's FPGA Schematic 18

Appendix A - Manual to CBD 8210 CAMAC Branch Driver 19

Appendix B - Manual to Model 70A type A-1 CAMAC Crate Controller 20

Appendix C - Software Listing for Test Code 21

Appendix D - Parts List and Data Sheets 33

HISTORY

Date	Who	What
4-20-95	John Weizeorick	Original

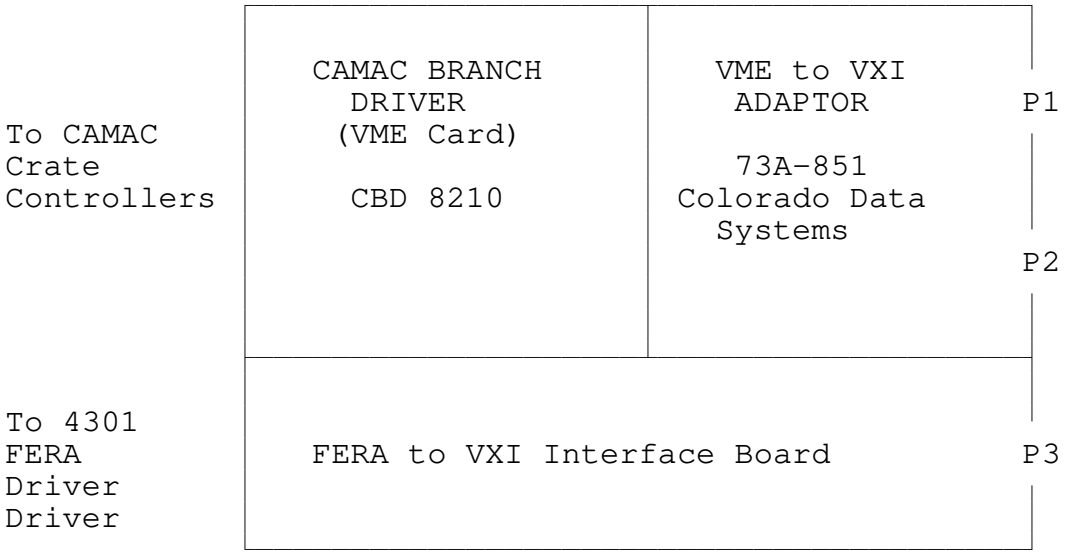
1.0 OVERVIEW OF FERA to VXI Interface Module

This section gives a brief overview of the FERA to VXI Interface Module which was designed and developed by Argonne National Laboratory's Electronics Department, to meet LBL's specifications for the GammaSphere Project.

1.1 Description of the FERA to VXI Interface Module

The basic function of the FVI (FERA to VXI Interface Module) is to take data from a 4301 FERA Driver and insert the data into the GammaSphere data stream. The FVI is a D-size VXI Module that consist of three boards.

FERA to VXI Interface Module:



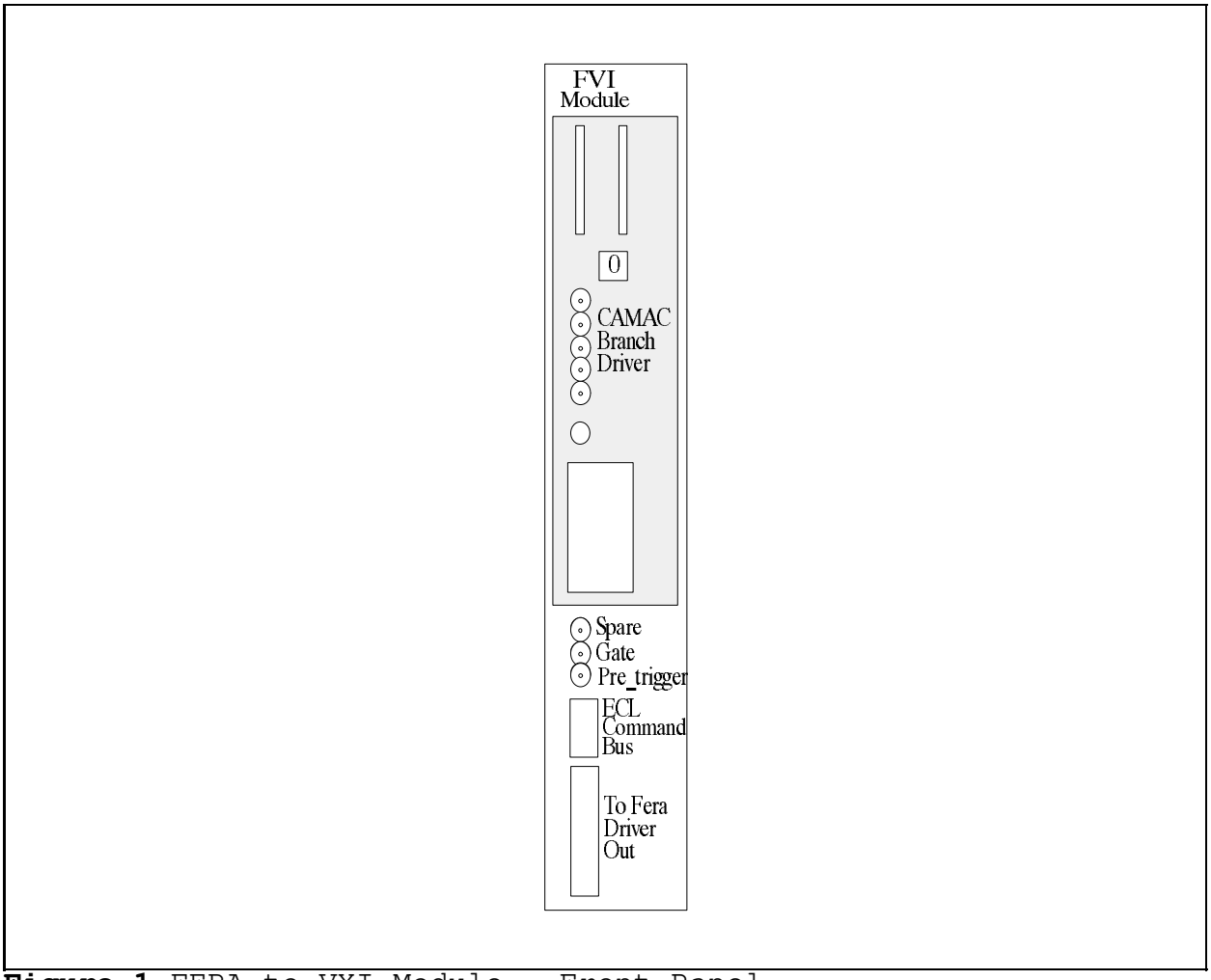


Figure 1 FERA to VXI Module - Front Panel

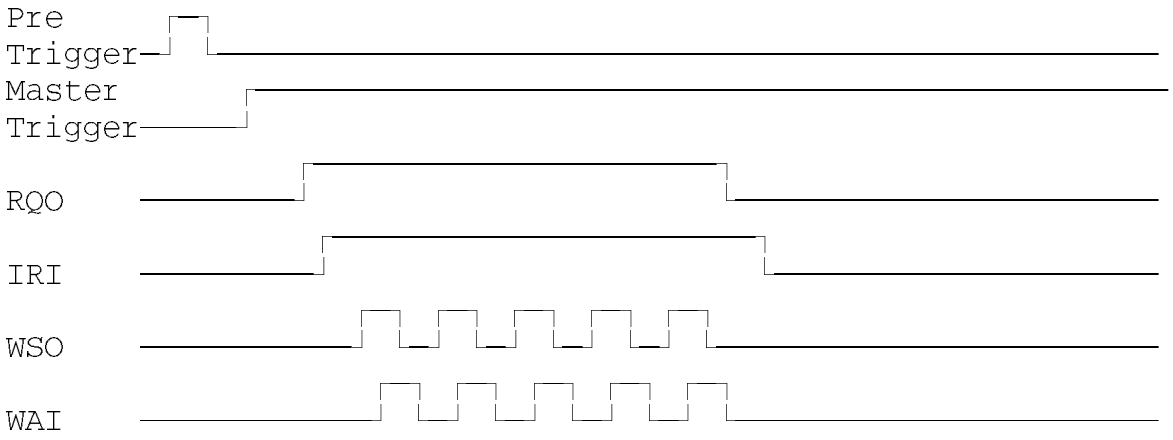
The CAMAC Branch driver is bought from Creative Electronics System, and the VME to VXI Adaptor is bought from Tektronix. The FERA to VXI Interface Board is a custom board designed by ANL. All of these boards are fitted into a single VXI D-Size Module.

Since the two commercial boards have their own manuals, this document will mainly describe the function and operations of the FERA to VXI Interface Board (FVI).

The normal sequence of events that the FVI (FERA to VXI Interface Module) follows when transferring data from the FERA devices into the GammaSphere data stream is given below.

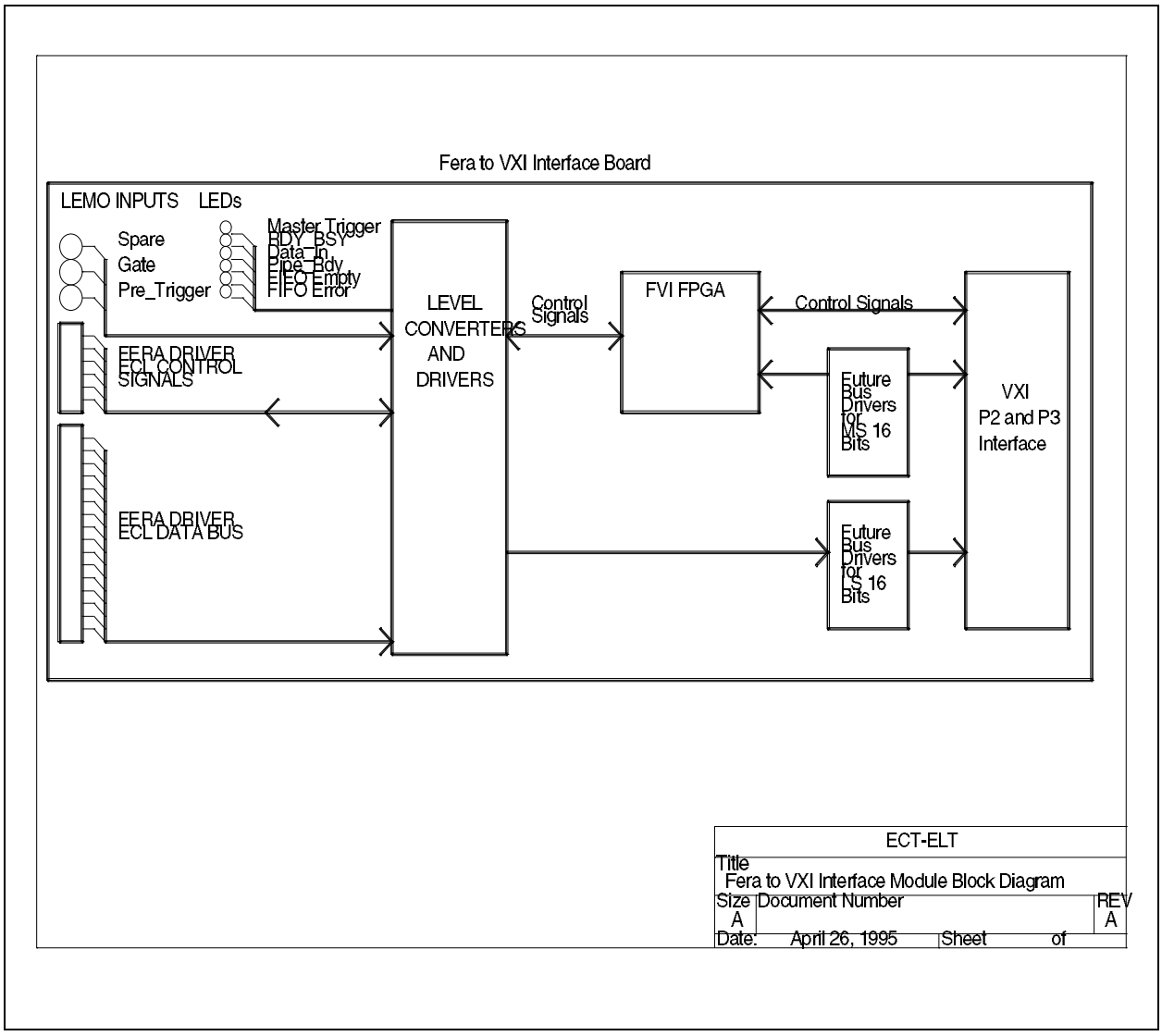
1. Waits for a Pre-Trigger pulse from the front panel lemo connector. The Pre-Trigger is active low with NIM logic levels. When it goes low it indicates that a Master-Trigger may happen. If no Master-Trigger occurs then the FERA to VXI Interface generates a CLI which resets the FERA devices.
2. A Master-Trigger pulse occurs before the Master Trigger timeout. The Master-Trigger signal is active high, and enters the FVI over the VXI Bus from the ECLTRIG0 signal. When the FERA to VXI Interface Module receivers the Master-Trigger it drives the RDY/BSY signal active, which lets the Master Trigger Module know that the system is busy performing a conversion. The Master Trigger timeout in the FVI is a fixed 1 usec timer. This time can be changed by modifying the FVI's FPGA.
3. RQO becomes active before the RQO timeout occurs. The RQO signal is an active high, differential ECL signal, form the 4301 FERA Driver. When it becomes active it indicates that there is data to be readout from the FERA devices. The RQO timeout is controlled by a four position dip switch. The next section shows the time vs dip switch positions.
4. The FERA to VXI Interface drives the IRI signal low. This starts the FERA Token Passing. The IRI signal is an active low, differential ECL signal, which goes to the 4301 FERA Driver.
5. When a FERA receives the token, it either passes the token on, or if it has data it puts its data on the bus and drives its WST signal high. The FERA Driver receives the WST and drives its WSO signal high. The WSO signal goes to the FVI (FERA to VXI Interface Module). The WSO is used to clock the data into the FVI's FIFO and increment a size counter in the FVI. WST is an active high, differential ECL signal.
6. After the data has been clocked into the FVI FIFO's it drives the WAI signal high. The FERA Driver takes the WAI and routes it to the FERA Devices as the WAK signal. This lets the FERA Device know that the FVI is done with the data and it can proceed with the next word. The FERA Device removes its WST, which cause the FVI to remove its WAI. The WAI signal is an active high, differential ECL signal.
7. Repeat steps 5 and 6 until all FERA data is read.
8. When the last word of FERA data is read the RQO signal goes back low. When the FVI detects a low RQO it removes the IRI signal, and its RDY/BSY signal. The IRI signal removes the token form the FERA Devices, and the RDY/BSY lets the Master Trigger Module know that the system is ready to be read out.
9. The FVI now waits for the GammaSphere readout system to send it a token. While waiting it could start processing another event if the pipeline is enabled.

10. When the FVI receives the GammaSphere token it starts clocking its data out over the VXI Local Bus. This is done the same way that the GeBGO Mother Boards clock out their data.



1.2 Block Diagram of the FERA to VXI Interface Module

The following is a block diagram of the FERA to VXI Interface Module. Section 1.3 describes the function of each section shown in the block diagram.



2.0 Subsections of the FERA to VXI Interface Module

The FERA to VXI Interface Module (FVI) consist of the following subsections listed below.

- FERA to VXI Interface FPGA
- Control Signals used for FERA Data Read In
- Data Bus Interface from the FERA Driver
- Control Signals to VXI Local Bus
- Data Bus Interface to VXI Local Bus
- Miscellaneous Circuits

The **FERA to VXI Interface FPGA** controls all aspects of the data flow between the FERAs and the VXI Local Bus. All of the control signals into and out of the FERA to VXI Interface board are controlled by the FERA to VXI Interface FPGA. Details of the FPGA is given in section 1.4.

There are 14 **Control Signals used for FERA Data Read In**. During FERA Data Read In the data flows from the FERA Devices into the FVI's Pipeline FIFO. Listed below are the 14 signals. For more details see section 1.5.

Six control signals between the FVI and the 4301 FERA Driver.
Pre-trigger and Gate, form two lemo connecters on the front panel of the FVI.
Master-trigger, from ECLTRIG0 on the VXI Bus
RDY_BSY to LBUSC34 on the VXI Bus.
Four inputs form a four position dip switch. These inputs are used to control how long after a Master-Trigger the FVI will wait for an RQO, comes from a

The **Data Bus Interface from the FERA Drivers** consist of 16 differential ECL data lines. These signals contain the data from the FERA Devices and the reading of this data is controlled by the Control Signals listed above.

There are 4 **Control Signals to VXI Local Bus** which are associated with the transferring of data from the FVI FIFO into the GammaSphere Data stream. They are:

Token_In	-	This input notifies the FVI that it may put its data out on the VXI Local Bus
Token_Out	-	This output is driven by the FVI when it is done putting its data out on the VXI Local Bus.
STROBE	-	This output is driven when the FVI owns the VXI Local Bus, and it has applied its data on the VXI Local Bus.
RDY_BSY	-	This output is used to notify the Master Trigger Module that the FVI is done collecting the FERA data into its pipeline FIFO.

The **Data Bus Interface to VXI Local Bus** consist of 32 bits. The lower 16 bits contain the FERA Data, and the upper 16 bits contain information from the FVI. The data is clocked out of the FVI's pipeline FIFO and out of the FVI's FPGA. The format of a data word is given below.

D31	D30 D29	D26 D25	D16 D15	D0
0	Error Bit	Dip Switch 0-3	Words Left	FERA Data

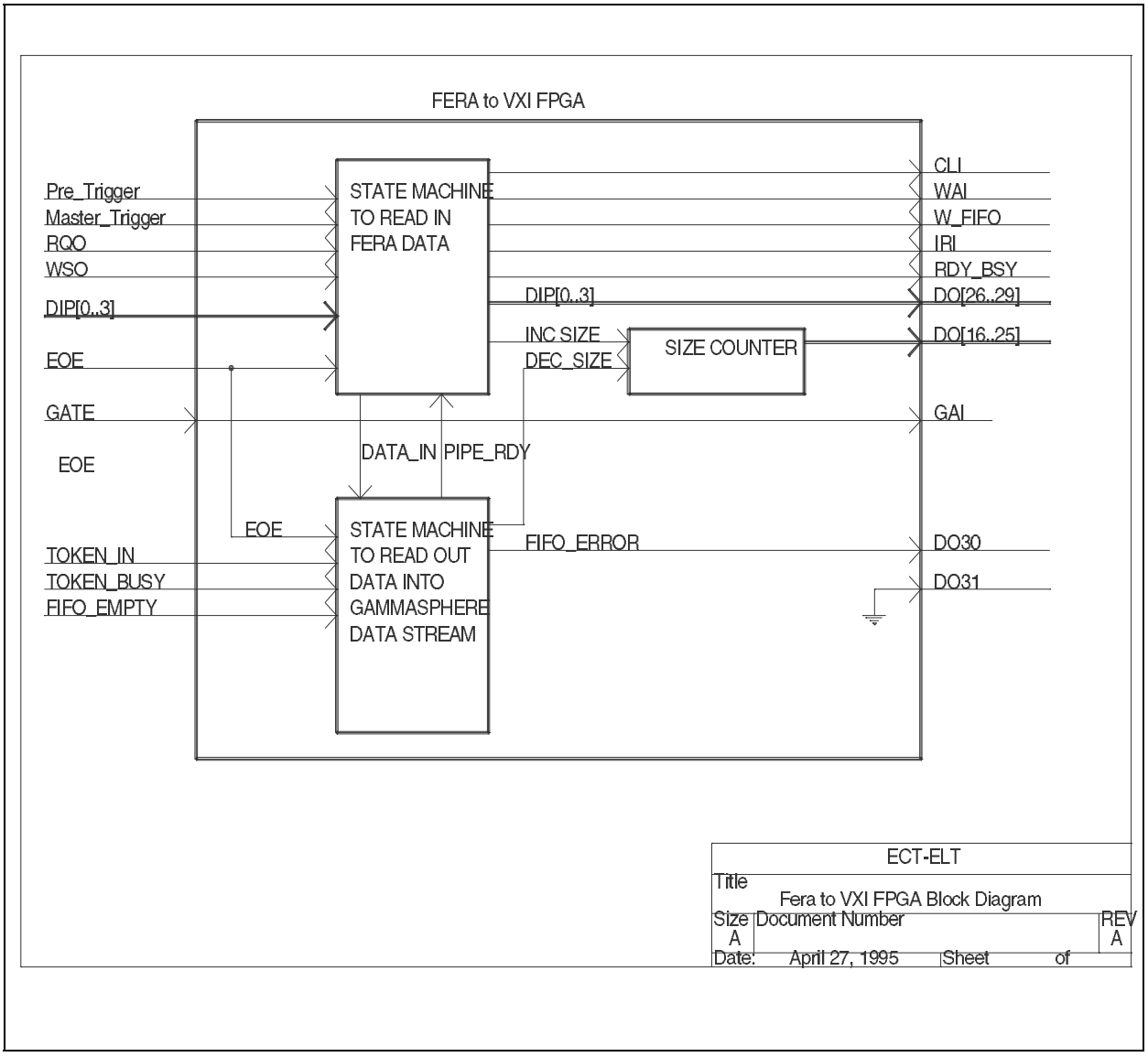
The remaining logic is grouped together in a category called **Miscellaneous Circuits.** It consist of the following devices.

- 6 Status Leds
- 2 Push Buttons
- 1 Spare Lemo

The following sections will give more details on those subsections of the FERA to VXI Interface module, which require more explanation.

2.1 FERA to VXI Interface FPGA

The FERA to VXI Interface FPGA contains all of the logic in the FVI board. For the most accurate and complete description of what is inside the FVI FPGA one should look at the schematic, which is used as an input to the Xilinx Development System. For those not interested in that level of detail I have included a block diagram of the features included in the FERA FPGA. The FVI FPGA consist of an input State Machine and an Output State Machine. A diagram of the State Machines can be seen on the following page.



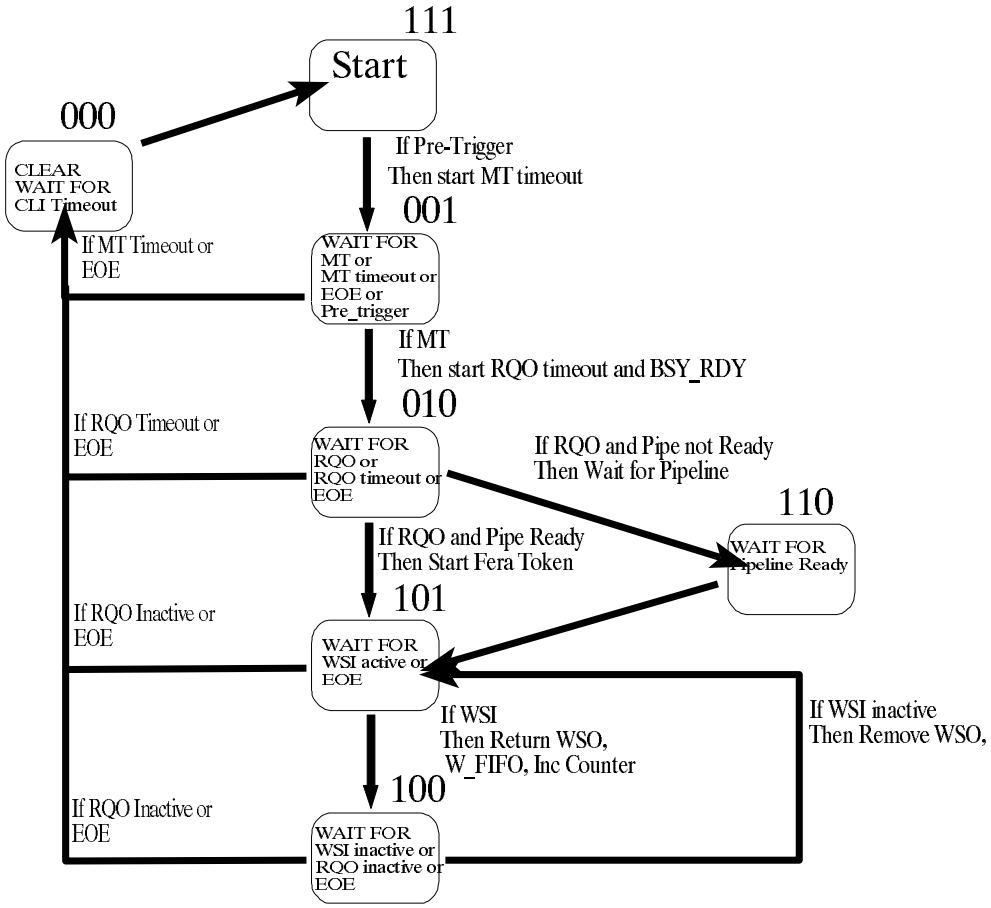


Figure 4 State Diagram for FERA Read In

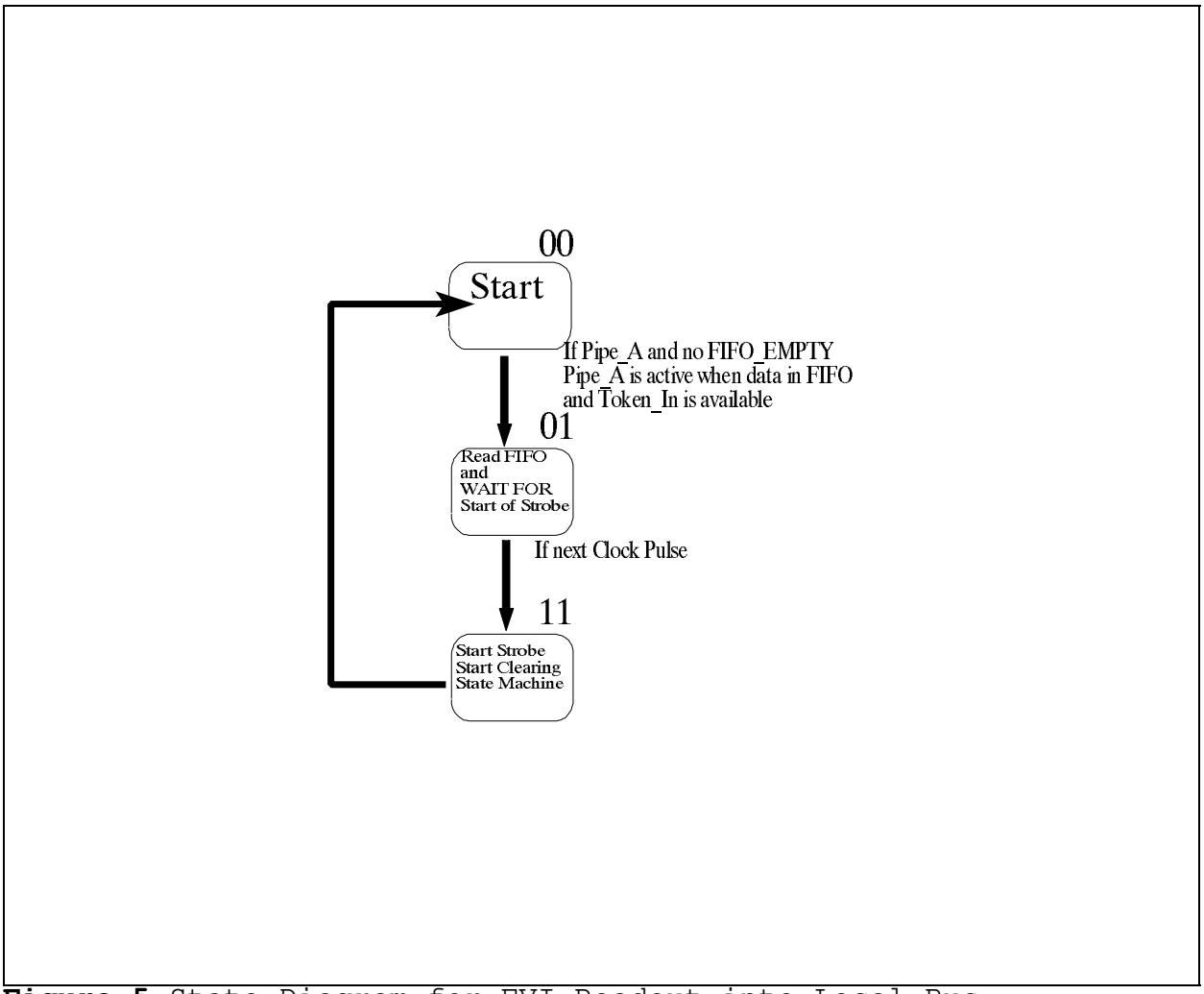


Figure 5 State Diagram for FVI Readout into Local Bus

2.2 Control Signals used for FERA DATA Read In

There are 14 **Control Signals used for FERA Data Read In**. During FERA Data Read In the data flows from the FERA Devices into the FVI's Pipeline FIFO. The 14 signals are:

Six signals between the FVI and the 4301 FERA Driver.
Pre-Trigger and Gate, form two lemo connecters on the front panel of the FVI.
Master-Trigger input, from ECLTRIG0 on the VXI Bus
RDY_BSY to LBUSC34 on the VXI Bus.
Four inputs form a four position dip switch

2.2.1 Six signals between the FVI and the 4301 FERA Driver

The six signals that go between the FERA driver and the FVI are listed below. Note that four of the signals are inputs to the FVI and two of the signals are outputs from the FVI.

OUTPUTS:

CLI - Clear signal to the FERA Devices. This is an active high, differential ECL signal, that goes to the FERA Driver.

GAI - Gate signal which tells the FERA Devices to perform an A/D conversion. This is an active high, differential ECL signal, that goes to the FERA Driver. The FERA Driver logically ORs this gate signal with its gate input on its front panel. Currently the GAI output is controlled by the Gate Lemo on the front panel of the FERA to VXI Interface Module.

IRI - The IRI signal allows the FERA Driver to start the FERA token. It is an active low, differential ECL signal, that goes to the FERA driver.

WAI - The WAI signal is the write acknowledge signal back to the FERA Driver. This is an active high, differential ECL signal, that goes to the FERA Driver.

INPUTS:

WSO - The WSO signal is the write strobe output from the FERA driver. It indicates that there is valid data present on the FERA Data Bus. The rising edge of this signal is used to clock data into the FVI's pipeline FIFOs. This is an active high, differential ECL signal, that comes form the FERA Driver.

RQO - The RQO signal is the Request Output signal from the FERA driver. When it goes high it indicates that at least one of the FERA Devices now has data, which can be read in. This is an active high, differential ECL signal, that comes form the FERA Driver.

2.2.2 Pre-Trigger and Gate Lemo Inputs

The Pre-Trigger input is the bottom lemo connecter and the Gate input is the lemo connector above the Pre-Trigger. They are both NIM logic level active low signals.

The Pre-Trigger input signal is required for correct operation of the FVI. If can come from the Pre-Trigger output from the Master Trigger Module. The Gate input signal is an optional input, which, if used, drives the ECL gate signal to the FERA Driver. If the user wants to gate the FERAs every time there is a Pre-Trigger, then this input may be connected to the Pre-Trigger input.

2.2.3 Master-Trigger Input, from ECLTRIG0 on the VXI Bus

The Master-Trigger Input is converted to TTL, and is used in the FVI's FPGA to control the flow of its input State machine. Because the Master-Trigger input is required for normal FVI operation, the FVI Module must be located in a VXI mainframe, with either a Local Trigger Module or the Master Trigger Module.

2.2.4 RDY_BSY to LBUSC34 on the VXI Bus.

The RDY_BSY output comes form the FVI's FPGA. It becomes active high on the rising edge of Master_Trigger, and becomes inactive after the FERA data has been transferred into the pipeline FIFO. Because the RDY_BSY output is used by the Master Trigger Module's readout logic, the FVI Module must be located in a VXI mainframe, with either a Local Readout Module or the Master Readout Module.

2.2.5 Four Position DIP Switch

The four inputs from the DIP switch are used to control how long after a Master-Trigger the FVI will wait for an RQO. If the FVI has not received a RQO after waiting the selected time, the FVI will generate a CLI, which clears the FERA devices. The table below shows the DIP switch setting vs the RQO timeout. The top DIP switch is the least significant bit, an and open is a one.

Dip Switch Setting	RQO Timeout
0000	200 nsec
0001	3.25 usec
0010	6.50 usec
0011	9.75 usec
0100	13.00 usec
0101	16.25 usec
0110	19.50 usec
0111	22.75 usec
1000	26.00 usec
1001	29.25 usec
1010	32.50 usec
1011	35.75 usec
1100	39.00 usec
1101	42.25 usec
1110	45.50 usec
1111	48.75 usec

3.0 Data Format from FERA Devices

The format of the data from the FERA Devices is a stream of 16 bit data. This information should be identical to what can be found in the FERA 4301, or FERA 4300B manuals. As an example I have copied the entry from the FERA 4301 manual.

With Zero or Zero-and-Overflow Suppression:

R16			R11			R1
1	WC = 0 to 15	0	0	0	VSN	
0	First SA	DATA				
0	SA	DATA				
0	:	:				
0	Last SA	DATA				

VSN: Virtual Station Number loaded in the Status Register.
WC: Number of data words following the header word, 0 indicates 16 data words.
DATA: Up to 11 bits of resolution.
SA: Channel Subaddress

The data from the FERA devices enter the GammaSphere data stream by being placed in the LS half of the 32 bit words. The upper half of the 32 bit word has data from the FVI (FERA to VXI Interface Module). The format of this data is shown below.

D31	D30 D29	D26 D25	D16 D15	D0
0	Error Bit	Dip Switch 0-3	10 bit Count down = Words Left in FIFO	FERA Data
0	Error Bit	Dip Switch 0-3	10 bit Count down = Words Left in FIFO	FERA Data

Error Bit: Set to One when the FERA to VXI Interface detects a size error. To reset send EOE.
Dip Switch 0-3: These 4 bits echoes the setting of the DIP Switches on the FVI. The DIP Switches control how long the FVI waits after a Master-Trigger for a RQO from the FERA Driver.
Words Left: These 10 bits give the number of words left in the FVI FIFOs.
FERA Data: The FERA Data in the format that it came out of the FERAs.

4.0 FERA to VXI Interface Board Schematic

5.0 FVI's FPGA Schematic

Appendix A - Manual to CBD 8210 CAMAC Branch Driver

Appendix B - Manual to Model 70A type A-1 CAMAC Crate Controller

Appendix C - Software Listing for Test Code

```
/*********************************************************************
fera_tst.c consist of a collection of subroutines used to test the
fera to vxi interface.  The test execpects the following configuration.

System Configuration Expected.
VXI Mainfraim
Slot 0  National Instrument's VXIcpu030
Slot 1  Local Readout Module
Slot 2  Jumper Board
Slot 3  Jumper Board
Slot 4  Jumper Board
Slot 5  Fera to VXI Interface Board
Slot 6  Token Return Jumper

Camac Crate
Slot 5  4301 Fera Driver
Slot 6  4300B Fera A/D Board
Slot 7  4300B Fera A/D Board
Slot 8  4300B Fera A/D Board

Pulse Generator to generate the following signals.

    Pre_Trigger    - Goes into a lemo on Fera to VXI Module.
                     NIM Voltage level.
                     Active low.
    Gate           - Goes into a lemo on Fera to VXI Module.
                     NIM Voltage level.
                     Active low.
    Master-Trigger - Goes into the ECLTRIG0 pin on VXI Backplane.
                     ECL Voltage level.
                     Active High.

For this test the operator may tie the Pre_Trigger and Gate signals together.
Be sure that the pulse generator can supply enough current to drive 25 ohms
to 0 and -1 volt.  The Pre-Trigger and Gate sould be about 200 nsec wide, and the
Master Trigger should occur within 1 usec of the Pre-Trigger.

                 __________      ____________________________________________
Pre-Trigger                 |_____|

                 __________      ____________________________________________
Gate                        |_____|
                                                  _______
Master-Trigger   ______________________________|       |____________________


The Main routine performs the following steps:

Procedures
1. init local readout module
2. init the camac branch driver
3. init the camac crate
4. init the fera modules
5. read back the fera registers and verify their contents
   if there is an error stop test and report the error.
6. Init the camac crate again
7. Send one token to clear first event
8. Wait
9. Send token
10. Read data out of LRM FIFO and verify Data.  If error report.
11. Go to 8
```

```
*****************************************************************/

static char    *copyright = "Copyright 1995, Argonne National Laboratory ";

/*
 * modification history -------------------- 01a, 4-10-95, jtw  written
*/

/***********************************************/
/* Include Files                       */
/***********************************************/
#include "nivxi.h"
#include "vxWorks.h"

/***********************************************/
/* vxi_in_error and vxi_out_error are already   */
/* loaded in the c shell.                       */
/* The source code is located in init_gamma.c   */
/***********************************************/
void  vxi_in_error(int16 ret, uint32 address);
void  vxi_out_error(int16 ret, uint32 address);

/***********************************************/
/* DEFINE CONSTANTS                          */
/***********************************************/
/* Addresses */
#define FIFO_LSWORD      0x30
#define FIFO_MSWORD      0x32
#define FIFO_STATUS      0x34
#define FIFO_RESET       0x36
#define LATCH_A          0x24
#define LATCH_B          0x124

/* Constants */
#define FIFO_EMPTY       0x0fff0
#define FIFO_NOT_EMPTY   0x0f         /* fff0 = empty */
#define FIFO_HF          0x0f0        /* ff0f = half full */
#define FIFO_FULL  0x0f00             /* f0ff = full */

#define TOKEN_EN   0x1
#define TOKEN_APLY       0x0

#define BRANCH     0x0
#define CRATE      0x1

#define CAMBAS     0x800000
#define CAMF       2     /* Shift count to F field */
#define CAMA       7     /* Shift count to A field */
#define CAMN       11   /* Shift count to N field */
#define CAMC       16   /* Shift count to C field */
#define CAMB       19   /* Shift count to Base Address of CAMAC Branches */
#define CAM16      2     /* Size of CAMAC Word to read or write */

#define MNOX       0x800   /* Mask BERR no no x.  */
#define MTO        0x20    /* Mask Branch timeout */
#define MLAM       0x10    /* Mask LAM interrupts */
#define MIT2       0x8     /* Mask interrupt 2    */
#define MIT4       0x4     /* Mask interrupt 4    */
#define QMASK      0x8000  /* Q bit in branch CSR */
#define XMASK      0x4000  /* X bit in branch CSR */

#define EVENT_SIZE 101
#define DIP_SETTING 0x4
```

```
/*********************************************/
/* DEFINE GLOBALS                            */
/*********************************************/
uint16 ped1[] = {0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,
                 0x18,0x19,0x1a,0x1b,0x1c,0x1d,0x1e,0x1f};

uint16 ped2[] = {0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,
                 0x28,0x29,0x2a,0x2b,0x2c,0x2d,0x2e,0x2f};

uint16 ped3[] = {0x30,0x31,0x32,0x33,0x34,0x35,0x36,0x37,
                 0x38,0x39,0x3a,0x3b,0x3c,0x3d,0x3e,0x3f};

uint16 test_data[] =
{
 0x32, 0x8011, 0x31,   0x38, 0x30,  0x83e, 0x2f, 0x103a,
 0x2e, 0x1832, 0x2d, 0x2034, 0x2c, 0x2838, 0x2b, 0x303c,
 0x2a, 0x3835, 0x29, 0x4038, 0x28, 0x4833, 0x27, 0x5034,
 0x26, 0x5830, 0x25, 0x6035, 0x24, 0x682d, 0x23, 0x7033,
 0x22, 0x782a,

 0x21, 0x8022, 0x20,   0x25, 0x1f,  0x824, 0x1e, 0x102a,
 0x1d, 0x1823, 0x1c, 0x2024, 0x1b, 0x2823, 0x1a, 0x301d,
 0x19, 0x381b, 0x18, 0x4023, 0x17, 0x481b, 0x16, 0x5021,
 0x15, 0x5818, 0x14, 0x601f, 0x13, 0x6820, 0x12, 0x7018,
 0x11, 0x7816,

 0x10, 0x8033, 0x0f,   0x16, 0x0e,  0x81c,  0x0d, 0x1017,
 0x0c, 0x1817, 0x0b, 0x201d, 0x0a, 0x280a,  0x09, 0x3014,
 0x08, 0x3811, 0x07, 0x4017, 0x06, 0x480b,  0x05, 0x5010,
 0x04, 0x580c, 0x03, 0x6010, 0x02, 0x680f,  0x01, 0x7008,
 0x00, 0x7808,
};

/* when diag != 1 then camrd16 and camwt16 print diagnostic message */
uint16 diag = 0;

int16 fera_tst ()
{
    uint16 cmd[10];

    uint16  data_out;
    uint16  data_out1;
    uint16      data_out2;
    uint16  data_in;
    uint16      fifo_stat;
    uint16      data_in_fifo;
    uint16      tst_data;
    uint16      b,c,n,a,f;
    uint32      i;
    int16   ret;
    int16       newline;

    char    cin;

    int8    *namepat;
    int16   manid;
    int16   modelcode;
    int16   devclass;
    int16   mainframe;
    int16   cmdrla;
    int16   la;
    int16   slot_num = 1;       /* slot number of lrm */
    uint16  board_base_lrm;
    uint32      address;
```

```
uint32      token_address;
uint32      fifo_ms_adr;
uint32      fifo_ls_adr;
uint16   accessparms;


cmd[6] = 0xff11;
cmd[7] = 0xff22;
cmd[8] = 0xff33;

/* Set up variables to get Logical Addresses and Caculate Base Addresses */
namepat = "";
manid = 0xf00;              /* 0xf00 */
modelcode = 0x100;          /* 0x100 for VXI-EE and up -1 = not checked */
devclass = 0x3;             /* 0x3 = reg based */
mainframe = -1;
cmdrla = -1;
accessparms =  AccessP_Space(A16_SPACE) |
            AccessP_Priv(NonPriv_DATA) |
            AccessP_BO(MOTOROLA_ORDER);


/* init test data based on Dip switch setting */
/* for now make user define dip switch as a constant */
for (i=0; i<= EVENT_SIZE; i=i+2)
{
  test_data[i] = test_data[i] + (DIP_SETTING * 0x400);
}

printf("\n");

ret = InitVXIlibrary();
if (ret != 0)
{
  /* if (ret == 1)  printf("VXI library already initialized\n"); */
  if (ret == -1) printf("RM tabel memory allocation failed\n");
}

/*Get Logical Address of LRMG from the given Slot */
ret = FindDevLA(namepat, manid, modelcode, devclass,
                slot_num, mainframe, cmdrla, &la);
if (ret != 0)
{
  printf ("No VXI_INF Module was found in the given Slot!!\n");
  return (0);
}
board_base_lrm = (la << 6) + 0xc000;

/*******************************/
/* init LRM                    */
/*******************************/
/* don't enable TTL Trigger lines */
/* let the VXIcpu-030 control the TTL Trigger lines */

/* SET UP LRM */
/* Enable TOKEN_IN on LRM */
address = board_base_lrm + LATCH_A;
data_out = TOKEN_EN;
ret = VXIout (accessparms, address, WORD_WIDTH, data_out);
vxi_out_error (ret, address);

/* Verify FIFOs are Empty before test */
address = board_base_lrm + FIFO_STATUS;
ret = VXIin (accessparms, address, WORD_WIDTH, &fifo_stat);
```

```
vxi_in_error(ret, address);

/* If not empty then reset */
if (fifo_stat != FIFO_EMPTY)
{
  data_out = 0;
  printf ("FIFOs being Reset\n");
  address = board_base_lrm + FIFO_RESET;
  ret = VXIout (accessparms, address, WORD_WIDTH, data_out);
  vxi_out_error (ret, address);
  ret = VXIout (accessparms, address, WORD_WIDTH, data_out);
  vxi_out_error (ret, address);

  address = board_base_lrm + FIFO_STATUS;
  ret = VXIin (accessparms, address, WORD_WIDTH, &fifo_stat);
  vxi_in_error(ret, address);
  if (fifo_stat != FIFO_EMPTY)
  {
        printf("FIFOs not Reset after Reset Pulse!!\n");
        return(-1);
  }
}

/*******************************/
/* Init Branch and Camac Crate */
/*******************************/
b = 0;
c = 1;
branchinit (b);
crateinit(b,c);

/*******************************/
/* Init FERAs                  */
/*******************************/

/* for no suppression put in 7d11 */
/* for suppression put in ff11    */
initfera(b,c,6,cmd[6], &ped1);
initfera(b,c,7,cmd[7], &ped2);
initfera(b,c,8,cmd[8], &ped3);

verifyfera(b,c,6,cmd[6], &ped1);
verifyfera(b,c,7,cmd[7], &ped2);
verifyfera(b,c,8,cmd[8], &ped3);

/* clear i again */
crateinit(0,1);

/* setup for token generation */
token_address = board_base_lrm + LATCH_A;
data_out1 = TOKEN_APLY;
data_out2 = TOKEN_EN;

fifo_ms_adr = board_base_lrm + FIFO_MSWORD;
fifo_ls_adr = board_base_lrm + FIFO_LSWORD;

/* Generate First Token to Flush the Fera's */
/* Token generated in LRM */
ret = VXIout (accessparms, token_address, WORD_WIDTH, data_out1);
vxi_out_error(ret, address);
taskDelay (5);
ret = VXIout (accessparms, token_address, WORD_WIDTH, data_out2);
vxi_out_error(ret, address);
taskDelay (5);
```

```
/* Reset FIFO in LRM to clear data */
printf ("LRM's FIFO's being Reset to flush first event \n");
address = board_base_lrm + FIFO_RESET;
data_out = 0;
ret = VXIout (accessparms, address, WORD_WIDTH, data_out);
vxi_out_error (ret, address);
/* Verify Fifo's Reset */
address = board_base_lrm + FIFO_STATUS;
ret = VXIin (accessparms, address, WORD_WIDTH, &fifo_stat);
vxi_in_error (ret, address);

if (fifo_stat == FIFO_EMPTY) printf("LRM's FIFO were properly cleared\n");
else
{
  printf("ERROR: Did not Clear the LRM FIFOs with Reset!!!!\n");
  exit(-1);
}

/* make sure TTLtrigger line is high at start */
SrcTTLtrig (-1, 0, 0, 0);

taskDelay (5);

newline = 0;
while (1)
{
  if (newline++ == 70){ printf ("\n"); newline = 0; }
  printf(".");

  /* Pulse TTLTRIG0 low,  This will temporaly lower the RUN bit */
  /* Use the falling edge of TTLTRG0 to trigger the Pulse Generator */
  /* This will cause a Pre_Trigger, Gate, and a Master Trigger */
    SrcTTLtrig (-1, 0, 1, 0);
  /* taskDelay(1); */
    SrcTTLtrig (-1, 0, 0, 0);

  /* every time newline == 10 generate an EOE and verify that */
  /* no data is sent out. */
  /* EOE = TTLTRIG2 */
  if (newline == 10)
  {
      SrcTTLtrig (-1,2,1,0);
      taskDelay (1);
      SrcTTLtrig (-1,2,0,0);
  }

  /* Keep Token active Until there is data in the FIFOs */
  data_in_fifo = 1;
  while (data_in_fifo)
  {
        /* Generate a TOKEN - TOKEN_OUT will last until data received */
        ret = VXIout (accessparms, token_address, WORD_WIDTH, data_out1);
        vxi_out_error(ret, address);
      /* printf ("ts"); */

        /* read fifo status */
        address = board_base_lrm + FIFO_STATUS;
        ret = VXIin (accessparms, address, WORD_WIDTH, &fifo_stat);
        vxi_in_error (ret, address);

        if (fifo_stat != FIFO_EMPTY)  data_in_fifo = 0;
      if (newline == 10) data_in_fifo = 0;
  }
  /* wait until the entire event is in fifo */
```

```
        taskDelay (1);

        /* remove the toke and test the data in the lrm fifo */
          ret = VXIout (accessparms, token_address, WORD_WIDTH, data_out2);
          vxi_out_error(ret, address);

          /* test contents of fifos */
        if (newline != 10)
        {
          for (i=0; i <= EVENT_SIZE; i=i+2)
          {
            /* Read fifo MS */
              ret = VXIin (accessparms, fifo_ms_adr, WORD_WIDTH, &data_in);

            /* Verify fifo MS */
              if (data_in != test_data[i]) error(test_data[i], data_in);

              /* Read fifo LS */
              ret = VXIin (accessparms, fifo_ls_adr, WORD_WIDTH, &data_in);

              /* Mask out data bits for all but the 3 header words */
              if ( ((i+1) != 1) || ((i+1) != 35) || ((i+1) != 39) )
              {
               data_in = data_in & 0xff80;
               tst_data = test_data[i+1] & 0xff80;
              }

            /* Verify fifo LS */
              if (data_in != tst_data) error(tst_data, data_in);
          }
        }
        /* Test status of the FIFOs */
        /* they should now be empty */

          address = board_base_lrm + FIFO_STATUS;
          ret = VXIin (accessparms, address, WORD_WIDTH, &fifo_stat);
          vxi_in_error (ret, address);
          if (fifo_stat != FIFO_EMPTY)
          {
            printf("ERROR: Fifo's Not empty after reading out one event\n");
              exit (-1);
          }
    }
}


/*********************************************************************
error(data_expt, data_read)
prints error message and exits program
*********************************************************************/
error(data_expt, data_read)
    uint16 data_expt;
    uint16 data_read;
{
printf("ERROR: data expected = 0x%x,  Data Read = 0x%x\n",data_expt,data_read);
dump_fifo(1);
exit(-1);
}


/*********************************************************************/
run_tst ()
{
    while (1)
```

```
    {
      SrcTTLtrig(-1,0,1,0);
      taskDelay(5);
      SrcTTLtrig(-1,0,0,0);
      taskDelay(5);
    }

}



/*********************************************************************
camac_tst.c
Used to test CAMAC system with branch driver.

In VXI Mainframe:
Slot 0 = VXIcpu030
Slot ? = Fera to VXI interface Module with Branch Number = 0

In Camac Crate Controller
Slot 8       = 3841 FIFO Memory Module
Slot 10      = 3291 Datq Way Display
Slot ?  & ?  = Branch HWY Terminator
Slot 24 & 25 = Camac Crate Controller

*********************************************************************/

int camac_tst ()
{
    uint32   address;
    uint16   data_out;
    uint16   data_in;
    uint32      ldata_in;

    uint16      b,c,n,a,f;
    uint32      i;
    int16    ret;

    char     cin;

    printf("\n");

    ret = InitVXIlibrary();
    if (ret != 0)
    {
      /* if (ret == 1)  printf("VXI library already initialized\n"); */
      if (ret == -1) printf("RM tabel memory allocation failed\n");
    }


    /* Init branch driver and Camac Crate */
    b = 0;
    c = 1;
    branchinit (b);
    crateinit(b,c);

    diag = 1;
    /* Test Bit patter to FIFO */
    printf ("Going to write a bit pattern to FIFO Card and read it back\n");
    a = 0;
    f = 16;
    n = 21;
    data_out == 1;
```

```
    for (i= 0; i <= 15; i++)
    {
      data_out = 0x1 << i;
      camwt16 (b,c,n,a,f,data_out);
    }


    f = 0;
    for (i= 0; i <= 15; i++)
    {
      data_in = camrd16 (b,c,n,a,f);
    }
    diag = 0;
    return(0);
}



/*********************************************************************
initfera.c
Used to initialize the CAMAC system with FERAs

*********************************************************************/

int initfera (b,c,n,cmd,ped)
    uint16 b;
    uint16 c;
    uint16 n;
    uint16 cmd;
    uint16 *ped;

{
    int16     sub;
    uint16    data_in;

    do
    {
        /* Clear fera */
        data_in = camrd16 (b,c,n,0,9);

        /* Write Status Word */
        camwt16 (b,c,n,0,16,cmd);

        if ( !(qtst(b)) ) busy(b,c,n);
    } while (!qtst(b));


   /* after clearing all cards init pedestal from table */
   for (sub = 0; sub <=15; sub++)
     do
     {
      /* Clear Fera */
      data_in = camrd16 (b,c,n,0,9);
      /* Write Ped */
      camwt16 (b,c,n,sub,17, *(ped + sub) );
      if (!qtst(b)) busy(b,c,n);
     } while  (!qtst(b));
}


int verifyfera (b,c,n,cmd,ped)
    uint16 b;
    uint16 c;
    uint16 n;
```

```
    uint16 cmd;
    uint16 *ped;

{
    int16     a;
    uint16    data_in;


    /* Readback Pedestal and Verify*/
    /*   diag = 1;  */
    /* printf("\nRead Status and peds for Card %i\n",n); */
    data_in = camrd16(b,c,n,0,0);
    if (data_in != cmd) error(cmd, data_in);

    for (a = 0; a <=15; a++)
    {
      data_in = camrd16(b,c,n,a,1);
      if (data_in != *(ped + a) )  error(*(ped + a), data_in);
    }
    /* diag = 0; */
}

int readfera (b,c,n,a)
    uint16 b;
    uint16 c;
    uint16 n;
    uint16 a;
{
    uint16 data_in;

    data_in= camrd16 (b,c,n,a,2);
    return (data_in);
}


/*********************************************************************/
int branchinit (b)
     uint16 b;
{
    /* Write to CSR Register */
    camwt16 (b,0,29,0,0, (MNOX | MTO | MLAM | MIT2 | MIT4) );

    /* Write to Interupt Flag Register */
    camwt16 (b,0,29,0,4, 0);

    /* Genrate Z */
    camwt16 (b,0,29,0,9, 0);
}


/*********************************************************************/
int crateinit (b,c)
     uint16 b;
     uint16 c;
{
    uint16 data_in;

    while ( (camrd16(b,0,29,0,9) & (1 << c)) == 0)
    {
      printf ("Branch %d Crate %d is offline\n",b,c);
        taskDelay (120);
    }

     /* Generate C from Crate Controller */
```

```
    data_in = camrd16 (b,c,28,9,26);

     /* Generate Z from Crate Controller */
    data_in = camrd16 (b,c,28,8,26);

    /* Remove Dataway I */
    data_in = camrd16 (b,c,30,9,24);
}



/************************************************************************/
int busy(br, cr, sl)
    uint16 br;
    uint16 cr;
    uint16 sl;

{
    printf("Module busy during programming: \n");
    printf("Br = %d Cr = %d Slot = %d \n", (br), (cr), (sl));
    taskDelay(120);
}

/************************************************************************/
int qtst(b)
    uint16 b;
{
   uint16 data_in;

   data_in = camrd16(b,0,29,0,0);
   if ( (data_in & 0x8000) == 0) return (0);
   else return (1);
}

/************************************************************************
camrd16 and camwt16 are used to read and write to the CAMAC System
through the CBD 8210.

The CBD 8210 maps into VME address space as below.
      VME Memory Map of CAMAC Branch Driver:
      23      |       |       |       |       |       0
      1 0  B B B  C C C  N N N N N  A A A A  F F F F F  L  0

************************************************************************/

int camrd16 (b,c,n,a,f)
      uint16  b;
      uint16  c;
        uint16  n;
        uint16  a;
        uint16  f;
{
    uint32 address;
    uint16 data_in;
    uint16  accessparams;
    int16   ret;

    accessparams = AccessP_Space(A24_SPACE) |
                AccessP_Priv(NonPriv_DATA) |
                AccessP_BO(MOTOROLA_ORDER);

    address  = CAMBAS  + (b << CAMB) + (c << CAMC) + (n << CAMN) + (a << CAMA) +
                (f << CAMF) + CAM16; (n * 0x800) + (a * 0x80) + (f * 0x4);
    if (diag != 0) printf("read adr  = 0x%x, b=%i, c=%i, n=%i, a=%i, f=%i, ",
```

```
                        address,b,c,n,a,f);
    ret = VXIin (accessparams, address, WORD_WIDTH, &data_in);
    vxi_in_error(ret, address);
    if (diag != 0) printf(" data = 0x%x\n",data_in);
    return(data_in);
}

int camwt16 (b,c,n,a,f,data_out)
      uint16  b;
      uint16  c;
        uint16  n;
        uint16  a;
        uint16  f;
      uint16  data_out;
{
    uint32 address;
    uint16   accessparams;
    int16    ret;

    accessparams = AccessP_Space(A24_SPACE) |
                AccessP_Priv(NonPriv_DATA) |
                AccessP_BO(MOTOROLA_ORDER);

    address  = CAMBAS  + (b << CAMB) + (c << CAMC) + (n << CAMN) + (a << CAMA) +
                (f << CAMF) + CAM16; (n * 0x800) + (a * 0x80) + (f * 0x4);
    if (diag != 0)
    printf
    ("write adr = 0x%x, b=%i, c=%i, n=%i, a=%i, f=%i, data = 0x%x\n",
            address,b,c,n,a,f,data_out);

    ret = VXIout (accessparams, address, WORD_WIDTH, data_out);
    vxi_out_error(ret, address);

   return(ret);
}
```

Appendix D - Parts List and Data Sheets

FERA TO VXI INTERFACE MODULE Revised: February 17, 1995
L8825-9560-DB-00 Revision: A
Bill Of Materials February 17, 1995 14:28:41
Page 1

Item	Quantity	Reference	Part	
1	39	C1,C2,C3,C7,C8,C9,C10, C11,C12,C13,C14,C15,C16, C17,C18,C19,C20,C21,C22, C23,C24,C25,C26,C27,C28, C29,C30,C31,C32,C33,C34, C35,C36,C37,C38,C39,C40, C41,C42	0.1	
2	3	C4,C5,C6	33/10V	
3	6	D1,D2,D3,D4,D5,D6	LED	Front Panel
4	3	F1,F2,F3	FUSE 5A	
5	1	JM1	JMP2	
6	1	J1	CON12 2x6	
7	1	J2	CON34A	Front Panel
8	3	J3,J4,J5	LEMO	Front Panel
9	2	J47,J66	CON40A	St. Up
10	3	P3C,P3B,P3A	DIN96	
11	5	RP1,RP2,RP3,RP4,RP5	SIP10-56	
12	6	RP6,RP7,RP8,RP9,RP10, RP11	SIP10-5K	
13	1	R1	120	
14	1	R2	82	
15	3	R9,R10,R11	0	
16	3	R12,R13,R14	56	
17	3	R15,R17,R18	5K	
18	6	R19,R20,R21,R22,R23,R24	330	
19	2	S1,S2	SW SPST	Push Buttons to ft panel
20	1	S3	SW DIP-4	8 pin dip
21	1	U1	XC3142A	SOCKET 84 pin PLCC
22	1	U2	XC1736A	SOCKET 8 pin dip
23	6	U3,U4,U5,U6,U11,U21	10125	16 pin dip

FERA TO VXI INTERFACE MODULE Revised: February 17, 1995
L8825-9560-DB-00 Revision: A
Bill Of Materials February 17, 1995 14:28:41
Page 2

Item	Quantity	Reference	Part	
24	1	U9	MC10H124	16 pin dip
25	1	U10	MC10190	16 pin dip
26	2	U13,U14	IDT7204	28 pin dip 600 mil
27	1	U22	DS3892	20 pin dip
28	5	U23,U24,U25,U26,U27	DS3890	20 pin dip
29	1	U28	74LS240	20 pin dip