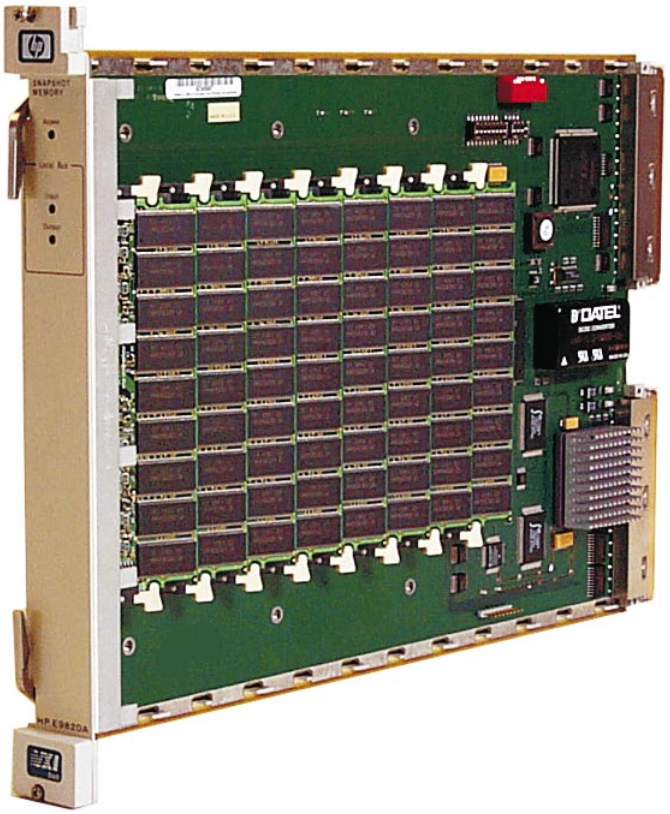
VXI bus

E9820A
Snapshot Memory
User's Guide



Agilent Technologies
Innovating the HP Way

Part Number: E9820-90000

Printed in U.S.A.
Print Date: January 2000

8600 Soper Hill Road Everett, Washington 98205-1209 U.S.A.

NOTICE

The information contained in this document is subject to change without notice.

AGILENT TECHNOLOGIES MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MANUAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Agilent Technologies shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty

A copy of the specific warranty terms applicable to your Agilent Technologies product and replacement parts can be obtained from your local Sales and Service Office.

Restricted Rights Legend

Use, duplication, or disclosure is subject to Agilent Technologies standard commercial license terms or to the following restrictions, whichever is applicable:

- For non-DoD Departments and Agencies of the U.S. Government, as set forth in FAR 52.227-19(c)(1-2)(Jun 1987);
- For the DoD and its Agencies, as set forth in DFARS 252.227-7013 (c) (1) (ii) (Oct 1988), or DFARS 252.221-7015(c) (May 1991), whichever is applicable.

Agilent Technologies, Inc.
395 Page Mill Road
Palo Alto, CA 94303-0870, USA

Table of Contents

Installation

Introduction

This chapter discusses how to install the E9820A Snapshot Memory in a VXI system, including the programming libraries.

Static Handling Precautions

Improper handling may damage or reduce the reliability of this equipment. Anyone handling this module should follow static-sensitive guidelines.

Caution

We recommend that any work that requires removing or replacing this hardware from/in a VXI chassis be performed only at a static-protected work station.

Hardware Installation

1. Turn off the power on the VXI chassis in which you wish to install this module.

Caution

Inserting or removing a VXI module with the power on can damage the module or the chassis. Before installing the E9820A, be sure to switch the chassis power switch to *Standby* and/or remove the power from the chassis.

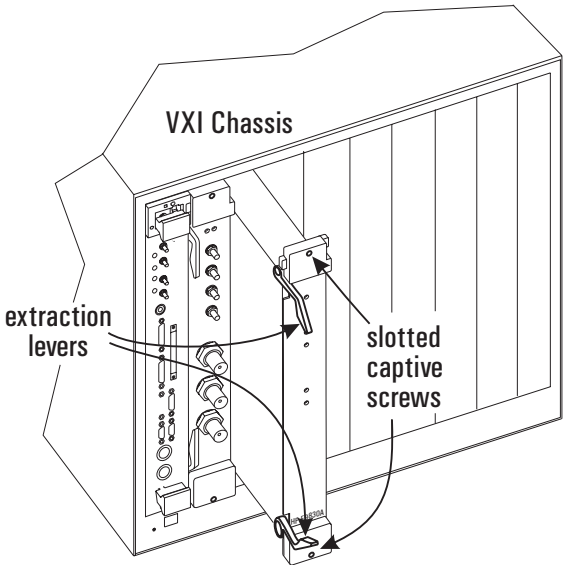
2. Make sure the chassis is ready to receive a new module. When used with the E1432, E1430, or E1437, RFI boots must be installed around the backplane connectors. See the VXI chassis installation documentation for more information.

Caution

The E9820A may be damaged by incompatible interface circuitry in other modules installed on either side of it. Be sure to install only ECL-compatible modules next to the E9820A. The local bus (LBUS) specification for the E9820A Snapshot Memory module is ECL on both the left and right sides.

3. Select a slot in the VXI chassis for the E9820A. When using it with another VXI module such that data is passed to it on the local bus, it must be installed immediately to the right of the module from which it receives data. Data on the local bus flows from left to right and must be passed to the right by a VXI module; it cannot pass through an empty slot.
4. Set the configuration switches as shown on the next page. The logical address may be changed without removing a cover.
5. Pull the extraction levers out and place the module's card edges (top and bottom) into the module slot guides. See the figure below.
6. Slide the module into the slot until the extraction levers engage the top and bottom rails at the front of the chassis*. Push both levers in toward the front panel to seat the module. The front panel should be touching or very close to the stops at the top and bottom of the module when it is completely installed.

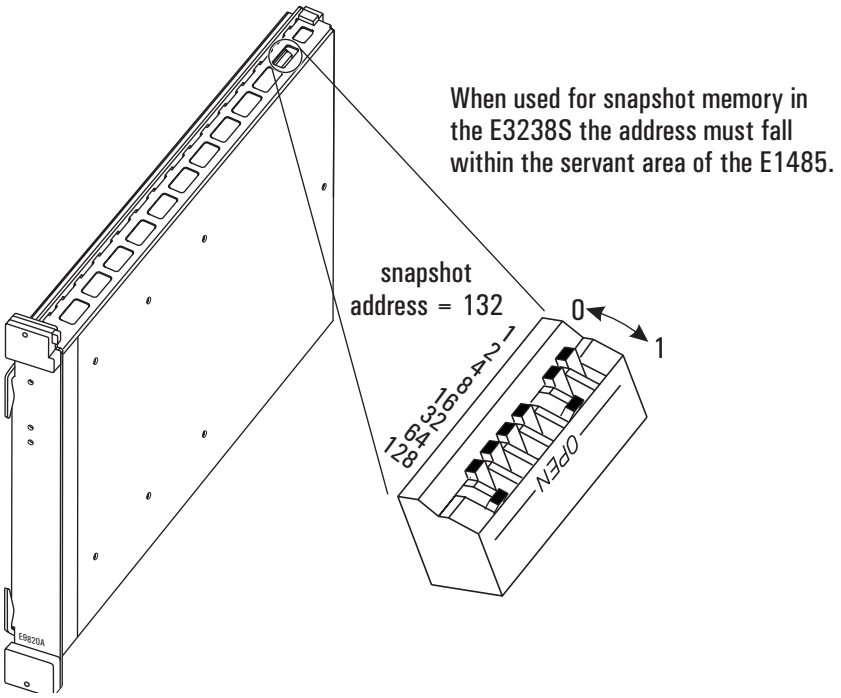
   *Note: some older VXI chassis may not support the use of modules with levers. With these you must push harder to insert the module.
7. Tighten the two slotted captive screws to secure the module in the chassis.



Logical Address

The logical address is the mechanism whereby the VXI system controller communicates with other modules via the VXI bus. Each module must have a unique address set on switches.

1. Locate the configuration switches for logical address as shown in the figure below.
2. Change the switch settings as appropriate for how the module will be used.



Note

When the address switch setting is 255 (all switches OPEN), the logical address is determined dynamically by the system resource manager during initialzation.

For information about the installed memory, see page 11.

Trouble Shooting

Introduction

This chapter discusses how to isolate and diagnose failures.

Diagnostics

Front-Panel Indicators

The LED indicators on the front panel blink to indicate the following memory configuration errors:

- **Input** LED blinks: indicates the number of DIMMs installed does not conform to the allowed number: either 1, 2, 4, or 8. The module will function properly but you may not have the data capacity needed.
- **Output** LED blinks: indicates that the DIMMs installed are of mixed sizes.

Neither condition constitutes an unusable memory configuration.

Trouble Isolation

When power is applied, the module runs a rudimentary self test of the installed memory and displays the results on the front panel LEDs as discussed above.

Self Test Program

If problems beyond memory configuration exist, another level of testing may be performed with the Self Test routine which is part of the VXI*plug&play* library. On Win32 windows platforms, run the soft front panel program:

```
<vxipnp home dir>/age9830/age9830_32.exe
```

This program is the graphic user interface used to exercise and demonstrate the functionality of the E9820A. The first tab has a **Self Test** button which runs the routine. The source code for the self-test function is included in the file:

```
<vxipnp home dir>/age9830/src/selftest.c
```

The self test routine performs a fairly thorough test of the installed memory and the Input and Output FIFOs discussed in the Theory of Operation on page 20. It does *not* test the entire Local Bus Interface, however.

Defective DIMMs

If the problem appears to be in the memory and is repeatable, try removing half of the DIMMs. Remember to populate the sockets in back first. Depending on what is wrong with the part, the failure may result in a configuration error which is indicated by the front-panel LEDs as described above or it may require the self test program to exercise the memory. Whichever is appropriate is referred to as "test."

If the test passes, then replace the DIMMs in the sockets with those you removed to see if one or more of the DIMMs are defective. In general you want to reduce the number of DIMMs installed until you find a set that works. There may be more than one defective DIMM. The most thorough approach is to test them one at a time in the rear socket.

The problem may be a defective DIMM socket. If all DIMMs pass the test individually and the test fails when a particular socket is used, then the E9820A must be returned for repair.

Local Bus Testing

Problems with the Local Bus Interface can be tricky to diagnose. If you are developing a new application and can't get the Local Bus to transfer data correctly, please consider your software as the source of the trouble. If you have more than one E9820A and both respond the same, the problem is less likely to be in the hardware. See the discussion on Software Debugging which follows.

Diagnosis

The following notes cover some debugging methods.

LED Analysis

When power is first turned on, the three LEDs on the front panel should light up for about one second, then they should all go dark. The "Access" LED may flash when the resource manager configures the system.

If the "Input" or "Output" LEDs flash regularly, there is a memory configuration error which may not be serious enough to prevent proper operation.

- The "Input" LED flashes to indicate a problem with the number of DIMMs installed; there should be either 1, 2, 4, or 8 DIMMs installed. If one of these combinations is installed (in the proper sockets) and the LED continues to flash, try installing just one DIMM in socket 1. If that works properly, then the problem is likely to be a defective DIMM; continue the isolation process as described earlier. If it continues to flash, the problem is serious enough to warrant returning the E9820A for repair.

- The "Output" LED flashes to indicate that there is more than one type (size) of DIMM installed. While this does not constitute a "failure," it is not efficient use of memory. If the installed DIMMs *are* identical and the LED flashes, then either one of the DIMMs is defective or the E9820A is recognizing it incorrectly and should be returned for repair. Isolate the problem as described above.

If the LEDs don't work as described, then there is a serious problem and the E9820A should be returned for service.

Self-Test Details

If the LEDs are acting normally, you can use the self-test program to delve deeper.

When the E9820A powers on it checks the contents of two bytes in the serial EEPROM on each installed DIMM. From this it determines the memory capacity and whether the DIMM is single or double-density. This is the minimum information needed to complete the initial configuration.

The self test program has three main stages:

1. The first stage of the self test goes beyond the minimal inquiry of the EEPROMs, checking for things such as ECC compatibility, and is more likely to catch problems like incompatible DIMMs. The test stops if there is a problem.
2. The next stage of the self test is a quick check of memory. It writes a data pattern to the first megabyte of memory and then to the first 512 bytes of each subsequent megabyte. This memory is then checked for correct values. Any failure of this test is probably a hard failure. The address printed for each failure is the byte address which can be used to find the associated socket location.

   For example, if 256 MB DIMMs are installed, each DIMM has 0x10000000 bytes of memory. For an error reported at address 0x23451234, the calculation is:

   $$0x23451234 \div 0x10000000 = 2; \quad 2 + 1 = \text{socket } 3$$

3. The third stage uses a loop mode to exercise almost all the memory space. The loop runs from the Main Memory, through the Output FIFO, then the loopback puts the data into the Input FIFO, then back into Main Memory. There are five different patterns, each of which is cycled through the loop for all memory locations, so for a 2 GB configuration, 10 GB of data are written, read, and checked. But each byte is written and read many times before it is checked so a failure indicates a problem but doesn't offer much information as to where.

None of this testing has validated the functionality of the Local Bus Interface.

Software Development Basics

This discussion provides hints to help solve programming road blocks.

If you are developing a new application using the E9820A and something isn't working correctly, first make sure the module is functional by performing the exercises discussed in the first part of this chapter. If the self-test passes but your program isn't working, you may modify the program to isolate the problem.

About the Local Bus

Sometimes local bus data doesn't want to move. Several things may be examined to see what's going on. This discussion assumes that you are familiar with the circular FIFO concept used for the Main Memory as discussed in the Theory of Operation.

- Check the Fill and Empty pointers. These are inferred by the **FIFO Size** and **Empty** registers. When local bus data is not reaching the E9820A, the FIFO size will be zero and the Empty pointer will also be zero. If the Empty pointer is non-zero, it is more likely that data has been read in and back out already.
- Also examine the **FINE** (FIFO input not empty) and **FONE** (FIFO output not empty) flags in the **Memory** register. If the **FINE** flag is set then there is data in the Input FIFO that won't flow into Main Memory until it accumulates 512 bytes. Unfortunately, there is no way to read how much is there.

Input Basics

The requirements for data input to the Local Bus are as follows:

- The **LBUS mode** must allow input, like consume or transform (page 33).
- You must set (1) all three reset bits in the **Local Bus** register (page 34).
- The **In Lbus** bit must be set in the **Mode** register (page 30).
- The **Out Lbus** bit should not be set when the **In Lbus** bit it set.
- Memory must not be full.
- The module positioned immediately to the left of the E9820A must supply data.

Output Basics

The requirements for data output to the Local Bus are as follows:

- The **LBUS mode** must allow output, like generate or transform (page 33).
- You must set (1) all three reset bits in the **Local Bus** register (page 34).
- The **Out Lbus** bit must be set in the **Mode** register (page 29).
- The **In Lbus** bit should not be set when the **Out Lbus** bit it set.
- The setting in the **Mlevel 0** register may inhibit output if it is not 0 (page 38).

Corrupted Data

If data is getting corrupted on the local bus, the most likely problem is with the reset sequence. For example, if you reset the local bus interface while the module immediately right of the E9820A is accepting data, the data may (or may not) contain a spurious byte. If this byte isn't cleared out it will cause subsequent data to be shifted by one byte, which usually causes major problems with the data. Note that at the end of a block, the receiving interface will insert garbage data to move the end of the block to a four-byte boundary. So data after that point will be shifted by *four* bytes. If this is the problem, it may be intermittent and may occur with one module but not another. This doesn't necessarily mean the module is defective.

There are two general methods that can be used to ensure the local bus resets operate properly. One or the other of these procedures should always be used whenever any module in the local bus chain is reset.

- Reset the local bus interface of all modules in the local bus chain in any order and then place all modules into operation in any order.
- From left to right, place the local bus interface of each module into reset and then restore it to normal operation. This method requires that no data is allowed to flow until all modules are reset and restored to operation.

Changing Memory Modules

Introduction

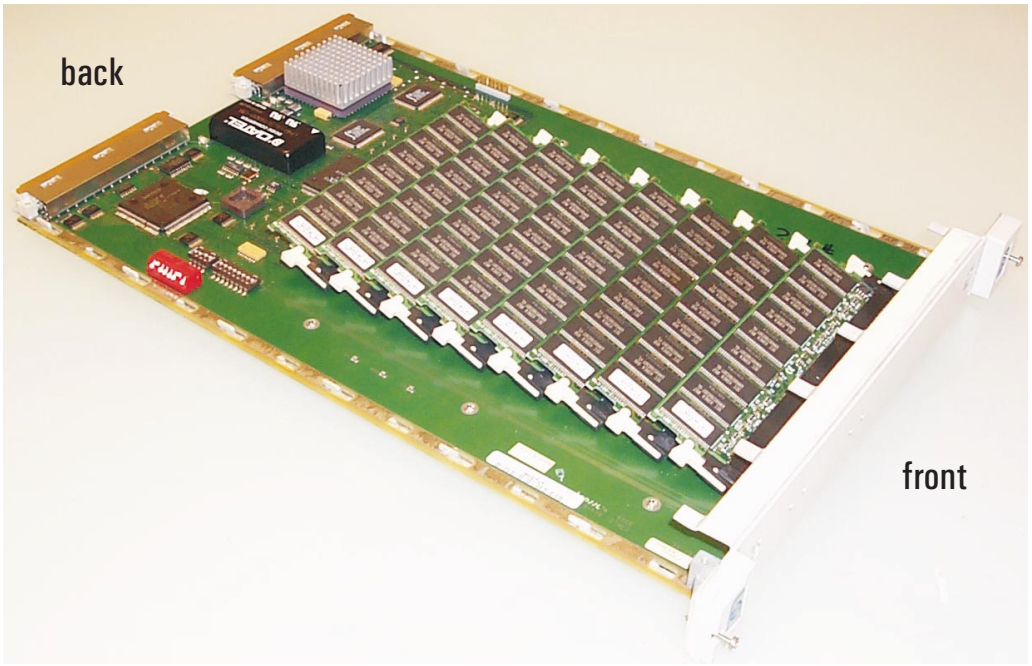
The E9820A Snapshot Memory module has 8 DIMM (dual, in-line memory module) sockets. The memory capacity ranges from 64 MB to 4 GB.

Memory configuration has the following constraints:

- The DIMM sizes supported are 64 MB, 128 MB, 256 MB, and 512 MB. As of this printing, the 512 MB DIMMs are not available. These are PC-100 unbuffered ECC DIMMs.
- The number of DIMMs installed may be 1, 2, 4, or 8.
- When the number of DIMMs installed is less than 8, fill the back first; all empty sockets must be at the front of the module. See Memory Configuration on page 11.
- The types of DIMMs supported are listed in the table below. You can install a mixed combination of DIMM types but the size of the smallest DIMM determines how much is used in all sockets.

Note

If you install four 128 MB and four 256 MB DIMMs, the useful capacity would be 1 GB because only half of the 256 MB DIMM would be used. Also, the Output LED would blink to indicate this "error" condition.



Removing the Top Cover

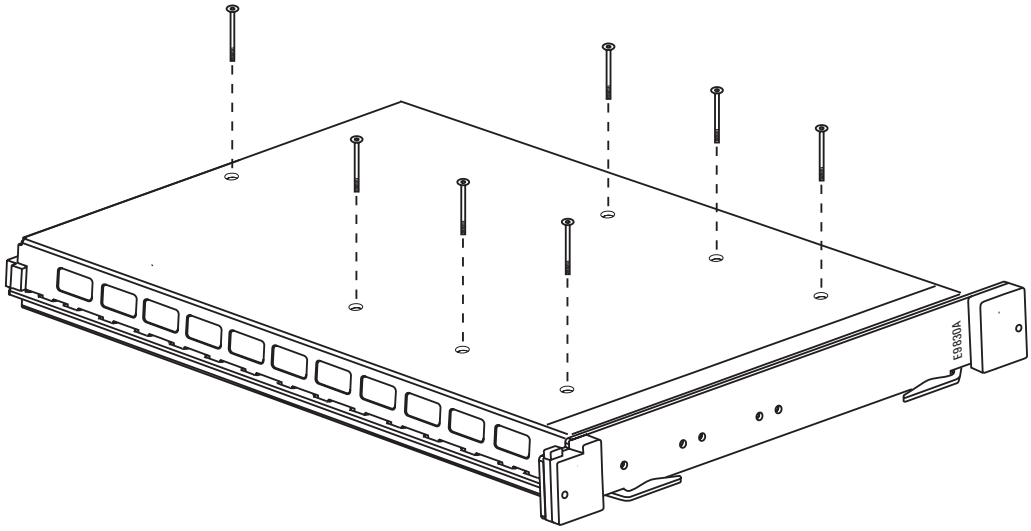
Caution

Before removing or installing the E9820A, be sure to switch the chassis power switch to *Standby* and/or remove the power from the chassis. Inserting or removing a VXI module with the power on can damage the module or the chassis.

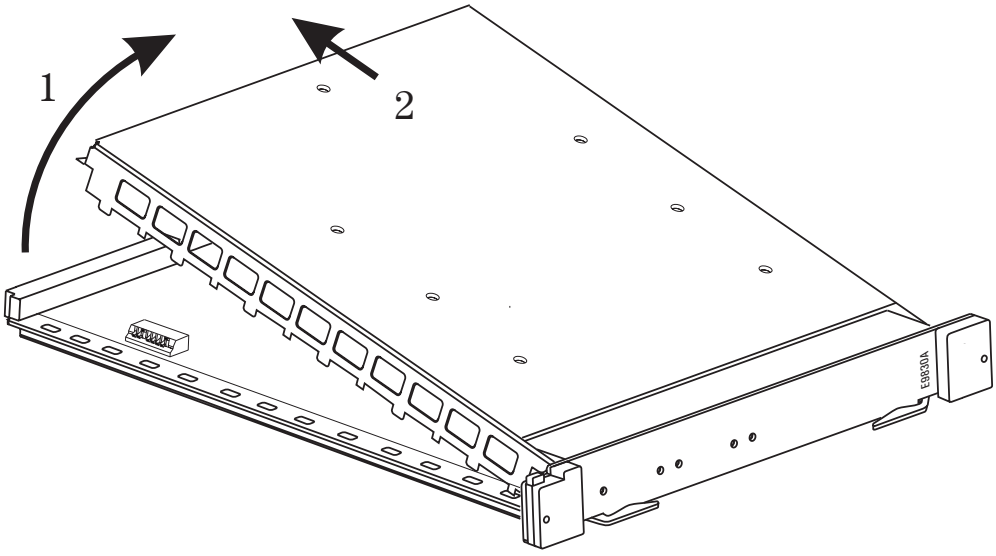
We recommend that any work that requires removing or replacing this hardware be performed only at a static-protected work station. All work done on the unit with the cover removed should be performed at a static-protected work station.

The top cover is on the right-hand side of the module as you view the front panel.

1. Using a T-10 torx driver, remove the seven cover screws.



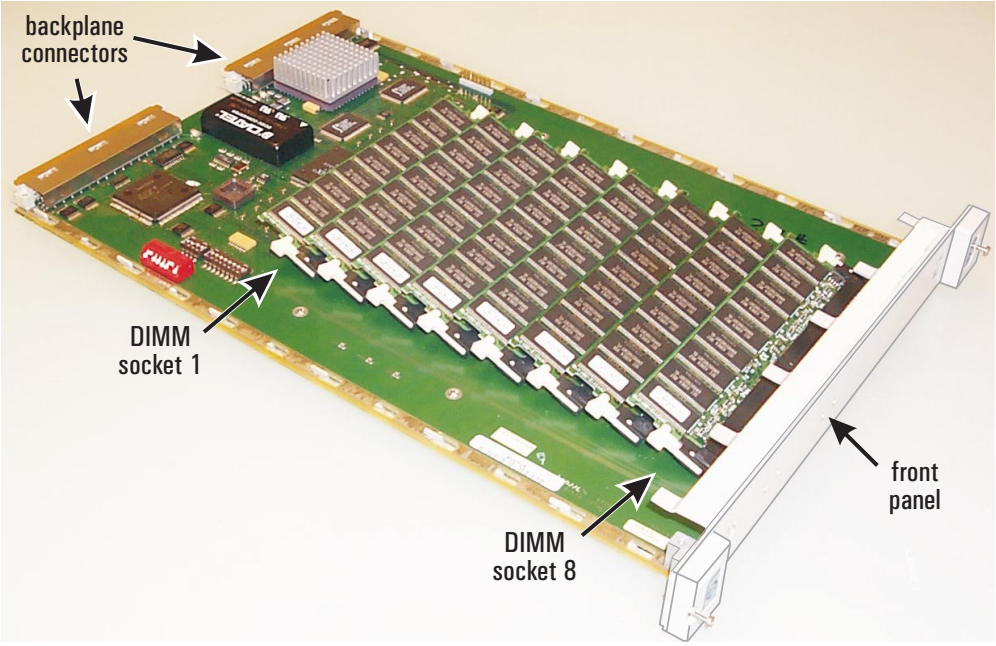
2. Remove the cover by lifting the back and then pulling it away from the front as shown below.



Memory Configuration

This discussion covers the supported DIMM (dual, in-line memory module) types, how many to install, and where to install them.

Changing DIMMs is discussed on the next page.



Number of DIMMs Installed

The number of DIMM installed may be 1, 2, 4, or 8. When less than 8 are installed, they must be in the sockets nearest the backplane. See figure above.

The number of DIMMs and socket locations are as follows:

- 1: in socket 1
- 2: in sockets 1 and 2
- 4: in sockets 1 through 4
- 8: in all sockets

DIMM Types

You may install a mix of types (i.e. type 1 and type 2) but avoid mixing capacities because it is inefficient. For example, when 4 type-2 DIMMs and 4 type-3 DIMMs are installed, only half of the memory of the type-3 DIMMs is used. See also page 6.

The following table lists the supported memory types.

Type	DIMM size	DIMM org.	SDRAM org	Max. Capacity	Part Number
0	64 MB	8M × 72	1 × 8M × 8	512 MB	
1	128 MB	16M × 72	2 × 8M × 8	1 GB	1818-7901
2	128 MB	16M × 72	1 × 16M × 8	1 GB	1818-7901
3	256 MB	32M × 72	2 × 16M × 8	2 GB	1818-7881
4	256 MB	32M × 72	1 × 32M × 8	2 GB	1818-7881*
5	512 MB	64M × 72	2 × 32M × 8	4 GB	NA*

*not available as of the print date of this document

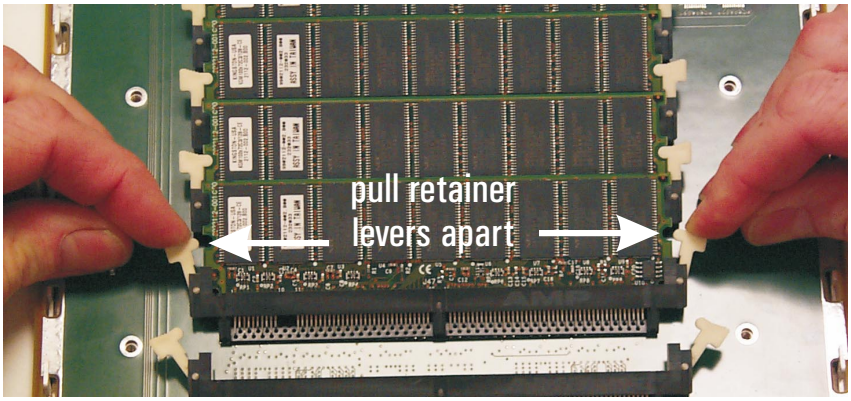
Changing DIMMs

Caution

Be sure to perform this work at a static-protected workstation to avoid damaging the DIMM parts.

Removal

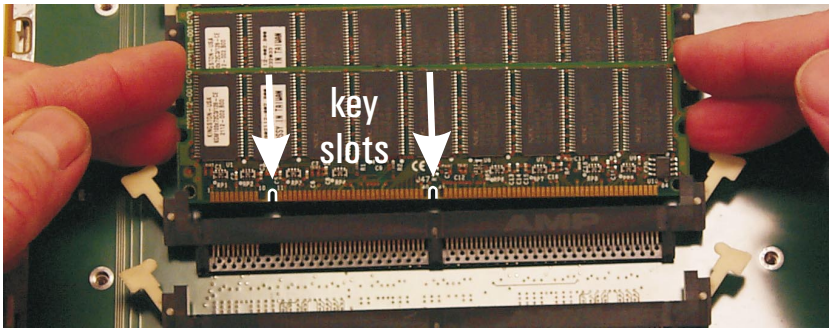
To remove a DIMM from a socket, pull the retainer levers out as shown below.

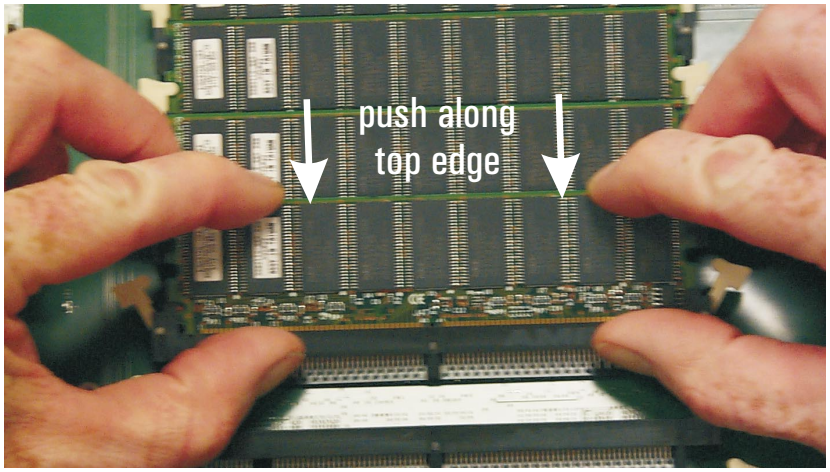


Installation

To install a DIMM in a socket:

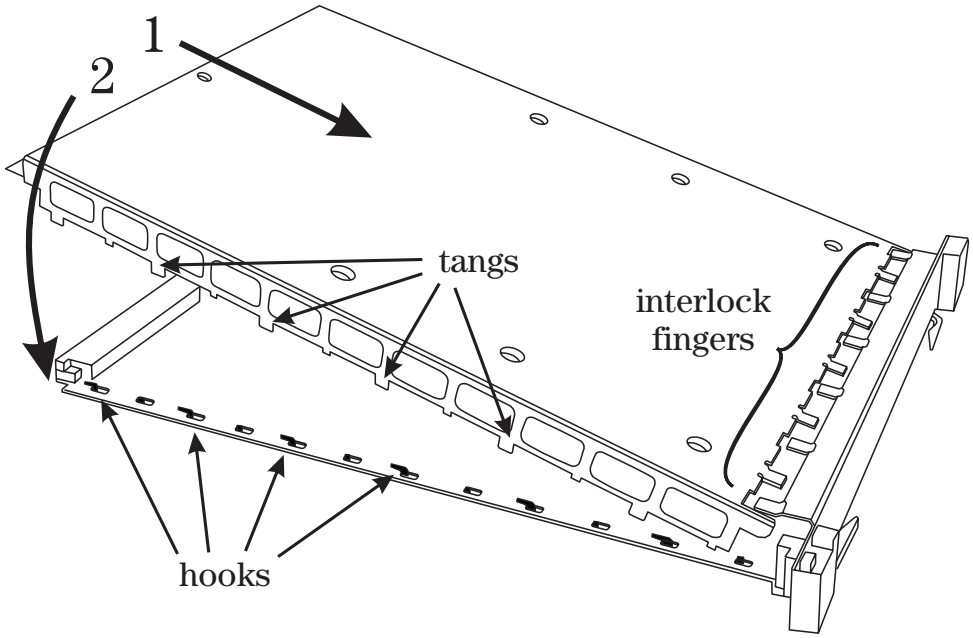
1. Position the memory strip so the two slots in the edge connector line up with the corresponding keys in the socket.
2. Apply uniform pressure along the top of the card, pushing it into the socket until the retainer levers snap closed on the card.



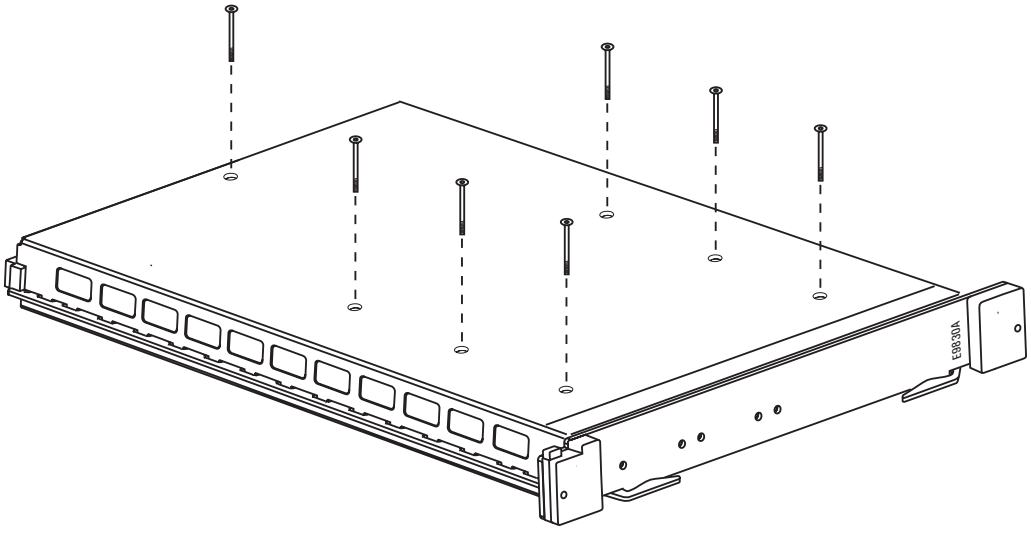


Installing the Top Cover

1. Position the top cover as shown below and slide it toward the front panel so that each of the two sets of interlocking fingers go under the opposing panel.
2. Lower the back of the cover toward the rear of the module but don't force it. You may need to press the sides of the top cover in so the tangs don't hang up on the bottom cover hooks. See drawing.



3. Using a T-10 torx driver, install the seven top cover screws.



Module Description

This chapter discusses the design and operation of the E9820A.

Introduction

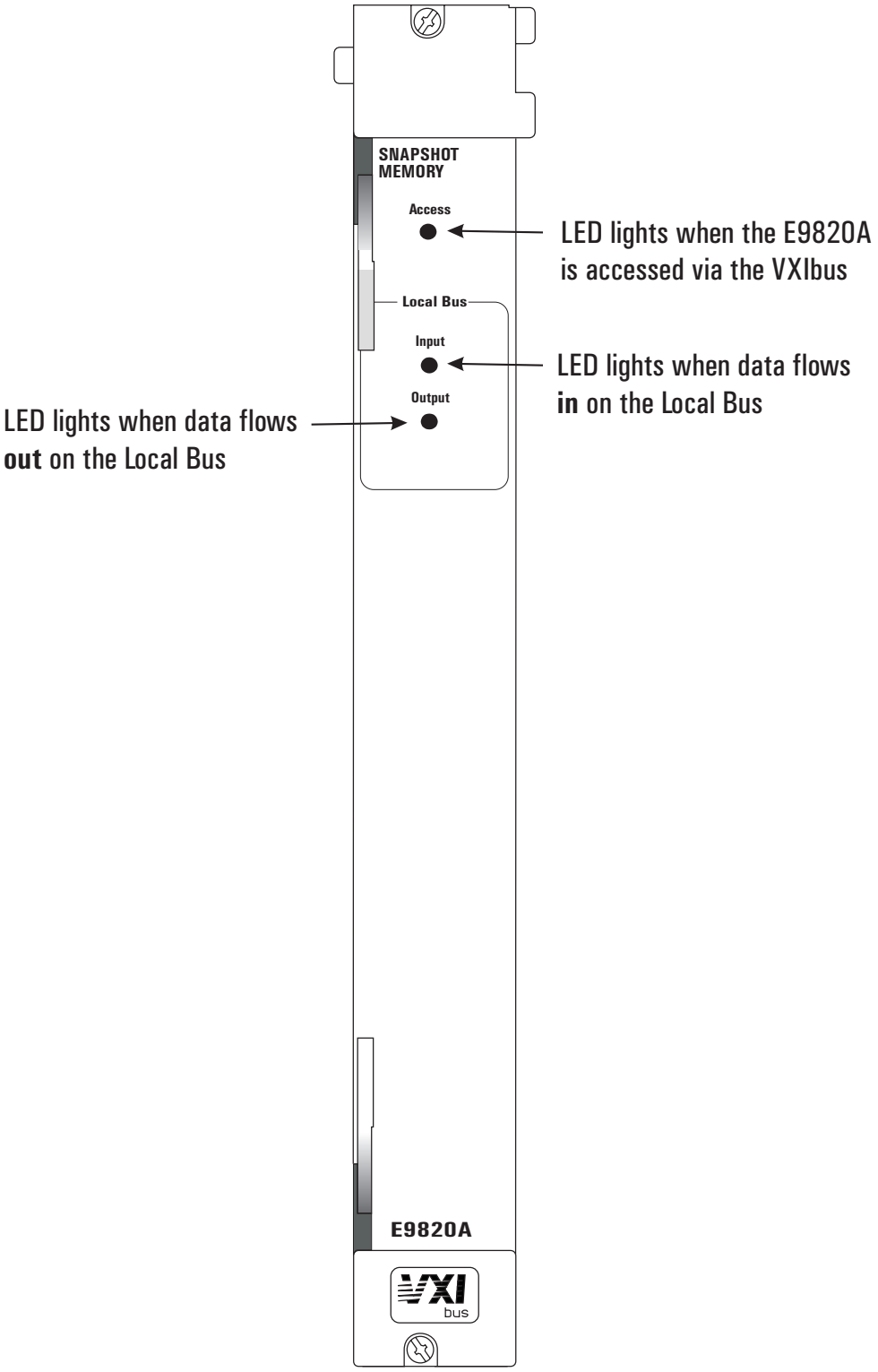
The E9820A Snapshot Memory module provides up to 4 GB of snapshot memory for data on the Local Bus. The maximum local bus data transfer rate is 53 MB/s. Data cannot be transferred in and out simultaneously on this module.

Note

As of this print date, 512 MB DIMMs are not available so the maximum capacity is limited to 2 GB with 256 MB DIMMs.

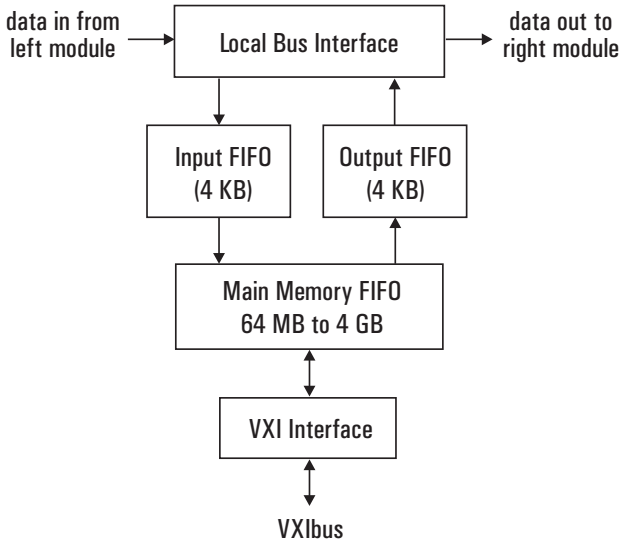
Front Panel Description

The front panel contains only LED indicators defined as follows:



Block Diagram

The following figure illustrates the E9820A block diagram.



The E9820A consists of an input local bus interface, which feeds a 4 KB input FIFO (first-in, first-out shift register). This feeds the main memory, which is managed as a FIFO. The main memory feeds a 4 KB output FIFO, which feeds the output local bus interface.

The E9820A is controlled via the VXI Interface[1] which may also be used to read or write data in the main memory. The Local Bus Interface provides a high-performance data path.

Data Flow

Typically, data originates in a module like an ADC to the left of the E9820A and the data flows in the Local Bus Interface and into Main Memory.

To perform data snapshot (capture), a specified amount of data is accumulated and then (typically) the input stops. This data may then be read out of main memory either on the Local Bus or on the VXI Interface. It is possible to keep the input going and read it out the VXI Interface but if the input data rate is greater than the VXI Interface data rate, the main memory will eventually fill.

Memory access (read/write) via the VXI Interface has the following restrictions:

- to read memory, the local bus output must be inactive (see page 29)
- to write memory, the local bus input must be inactive (see page 30)

Data Rates

The maximum Local Bus data rate is 53 MB/s. The E9820A cannot perform both input and output simultaneously, however.

The VXI Interface data rate depends on many things. One test with a FireWire slot-0 interface yielded 8 MB/s (using viMoveIn32 with source increment = 0.).

[1] See the chapter on Register Programming, page 23.

Local Bus Interface

The local bus is used to move data in and out at a high rate. Data comes in from the module just left of the E9820A and goes out to the module on its right. Data transfers on the Local Bus (both in and out) depend on the other modules to handshake the data for the transfer to function.

The local bus interface has the following modes of operation:

- **Pipe**: data from the left (input) is passed immediately to the right (output) and no data is moved into memory from the local bus.

  In this mode the snapshot function cannot be used.

  This mode has no effect on reading or writing memory via the VXIbus, so data in memory may be read while the local bus is in *pipe* mode.

- **Consume**: data from the left is read into memory; nothing is passed to the right.

  This mode may be used to capture snapshot data. To read out the snapshot data you could either use the VXIbus or change the local bus mode to generate to pass it out to the right.

- **Eavesdrop**: data from the left is passed immediately to the right and is also read into memory.

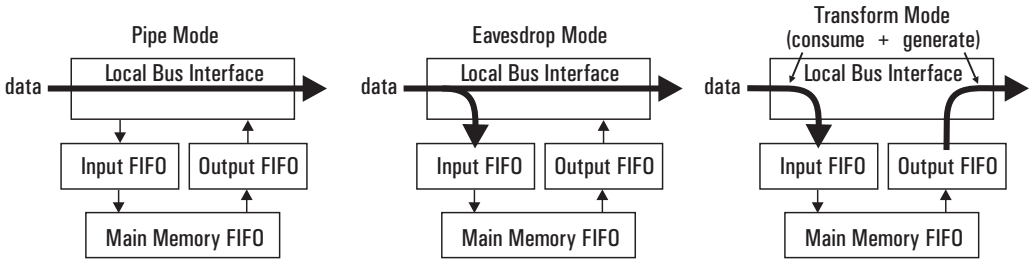
  This mode may be used to capture snapshot data. To read out the snapshot data you could either use the VXIbus or change the local bus mode to generate to pass it out to the right.

- **Generate**: no data is transferred into the local bus interface from the left; data in memory may be passed to the output.

  This mode may be used to pass whatever data is in memory to the right.

- **Transform**: data from the left is "modified" and passed out the right. This is the combination of the *consume* and *generate* modes.

  This is the preferred mode to use for the snapshot function of the E9820A. Data cannot be transferred in and out at the same time, however.



Note

To change the local bus mode requires resetting the local bus interface. Doing so will very likely cause a loss of data if it occurs during a transfer. It is best to pick a mode that supports all the data activity you anticipate using and control the flow with the programmable mechanisms provided.

Local Bus Blocks and Frames

The Local Bus has, associated with it, information which may be used to synchronize the data between the generating module and other modules. These are called the *frame* marker and the *end-of-block* (EOB) marker.

The marker "bits" are extra lines on the VXI backplane connectors. The device that generates the data sets these at whatever byte spacing is appropriate. Since the VXIbus has no marker mechanism, the E9820A supports using markers with several bits in the Mode Register (page 28) but the block size must be a multiple of 4 bytes[2].

When data flows in and out on the Local Bus this information is passed along "invisibly." When the I/O is a mix of Local Bus and VXIbus, the VXI access can read and write this "extra" information as necessary to maintain the functionality should an application require it (e.g., for synchronization).

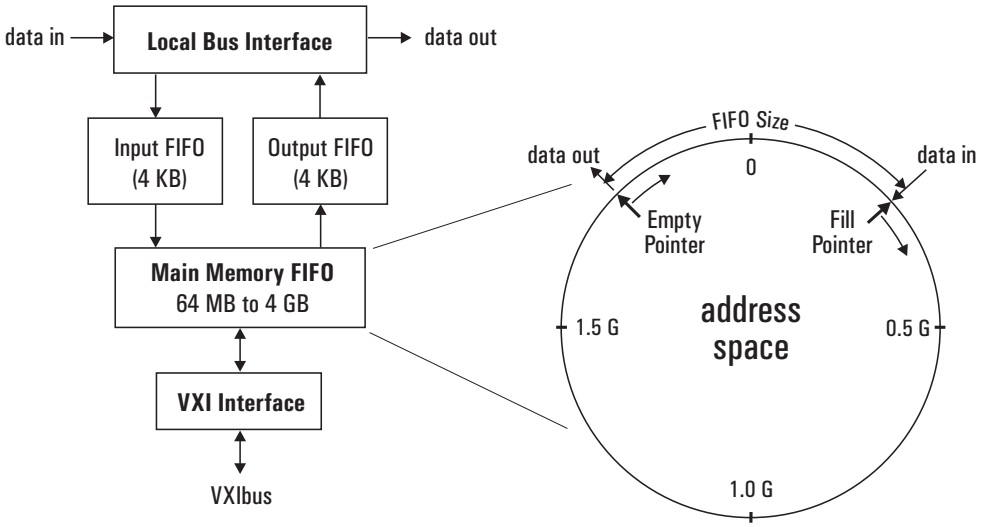
Note

Do not confuse the term "block size" used here with data output associated with the **Transfer** register. The minimum "amount" of data transferred is 512 bytes because of the way the output FIFO buffer handles data.

[2] The VXI*plug&play* library supports block sizes that are multiples of 8 bytes when using the VXI interface.

Theory of Operation

The Main Memory FIFO (first-in, first-out) buffer is implemented using circular memory addressing. A 2 GB example is shown in the following figure.



Input

Input data is placed in memory at the location identified by the **Fill Pointer**. Local Bus data comes through the Local Bus Input FIFO and is added to the main memory in 512-byte blocks. VXIbus data is added via the **Data** register. After each write, the Fill Pointer is incremented to indicate the next write location and the cycle repeats.

Output

Data is read out from the address identified by the **Empty Pointer**[3]. The data may be read out with either the **Transfer** register (via Local Bus, see page 40) or the **Data** register (via VXIbus, see page 42). Either method causes the **Empty Pointer** to move as data is read. Local Bus data moves into the Output FIFO and passes to the Local Bus interface in 512-byte blocks as shown in the block diagram above.

The 512-byte resolution is part of the FIFO buffering in the Local Bus path. The VXI Interface does not have this limitation.

Maximum Capacity

When memory is "full," the amount of data stored is actually 512 bytes less than the capacity of the installed memory. Since the Fill Pointer indexes the *next* location in which data may be written, writing data to the last available block would cause the Fill Pointer to advance to the same address indexed by the Empty Pointer, which normally indicates that memory is *empty*. Writing the last available block isn't allowed with Local Bus input. See the following caution.

When data is input from the Local Bus and the **In Cont** bit is "0" (page 30), the transfer stops when the Main Memory is "full". Besides what is in Main Memory, data may also be in the Input and Output FIFO buffers and Local Bus Interface registers.

Caution

Transfers through the VXI interface are *not* controlled to avoid memory overwrites. If you completely fill the Main Memory, the **FIFO Size** register (page 44) will read empty. Further writes will replace existing data.

[3] This address may be read from the **Empty** register (page 43) and defined with the **Output** register (page 45).

The E9820A provides the ability to capture data with the snapshot feature.

The examples in the following discussions use register programming information discussed in the next section. This may also be accomplished with the same general approach using the VXI*plug&play* library available on the installation disk.

Snapshot Operation

The E9820A can capture Local Bus data in 512-byte increments. The snapshot data can then be transferred out of the module on either the Local Bus or the VXIbus.

Example

Configuration and intended usage:

- Data input stops before output begins (simplest case)
- Data is read out on the VXIbus

Initial settings:

- **In Lbu**s = 1 (see page 30)
- **In Cont** = 0 (see page 30)
- **LBUS mode** = 0x2 (consume mode; see page 33)

You can use the **MIevel 1** register (page 39) to define a specific snapshot size; use the MDO bit (page 35) to create an interrupt that indicates when it occurs.

When the snapshot is complete, stop further input by changing the **In Lbu**s bit to 0.

To read data out over the VXIbus from any given location in memory, use the **Output** register (page 45) to define the address to start reading and the **Data** register (page 42) to read the data.

Variations

You can read data out while data input occurs but special considerations apply. See the following note.

To keep the snapshot running continuously (beyond the "full" condition), change the initial setting of the **In Cont** bit to 1. See the following note.

You can use as many as two E9820A snapshot modules to capture very large snapshots. See the example program `..\age9830\examples\age9820_multi.c`.

To read data out the Local Bus, change the startup **LBUS mode** to 0x5 (transform). Then you can control the output by any of the following methods:

- Controlling the Local Bus input of the next module downstream
- Using the **Out Lbus** bit (page 29)
- Use the **Transfer** register (page 40)
- If no data is flowing into the module, the **MIevel 0** register (page 38) may be used to halt the output transfer. For example, if x is written to the **Output** register and n is written to the **MIevel 0** register, then $x - n$ bytes of data will be output to the Local Bus (where x and n are multiples of 512).

Notes

The Empty Pointer moves when you write to the Output register or read data out. When the Empty Pointer moves toward the Fill Pointer, memory is "freed" such that new data may be written to Main Memory; e.g., over data just read out. This also affects the **DMF** status such that, if memory was full before the read, it's not full afterward and the **In Cont** setting won't stop more data input. To read "randomly" and preserve data in Main Memory for future retrieval, stop data input before redefining the Empty Pointer.

It is best to pick a Local Bus mode that will work for all phases of the activities. Changing the Local Bus mode requires resetting the Local Bus Interface which deletes any data in the Interface. See the discussion on **LBUS reset** on page 34.

Register-Based Programming

Introduction

This section describes how the E9820A module may be operated with register-based programming.

The E9820A registers are accessed via the VXI Interface. The addresses of the registers are given as an offset from the base address determined by the logical address setting which is set by switches as discussed on page 3.

The E9820A is also supported by a VXI*plug&play* library which is documented with an online help file. Check the installation disk or the web at:

`www.agilent.com/find/inst_drivers`

Register Listing

The following table lists the registers in A16 space.

Offset is the location of the register relative to the module's base address in bytes.

The E9820A is an A16-only device; there are no A24 or A32 address spaces on this module. All multi-word registers must be read and written from high-order to low-order (in order of increasing address) to function properly.

Offset	Read Register	Write Register	Description	Page
0x00	ID		Identify device class, address space, manufacturer.	25
0x02	Device Type		Identify model number.	25
0x04	Status	Control	Identify rev. & status/ specify state, reset device.	26/27
0x06				
0x08	Mode	Mode	Read/define data transfer configuration.	28
0x0A	Memory		Memory (DIMM: number & type) & buffer status.	31
0x0C	Local Bus	Local Bus	Read/define Local Bus configuration, incl. reset.	33
0x0E	IRQ Status	IRQ Config	Read/define interrupt conditions.	35/37
0x10	Mlevel 0 [31:16]	Mlevel 0 [31:16]	Read/define the FIFO size that triggers the minimum-data-available interrupt (MDA).	38
0x12	Mlevel 0 [15:0]	Mlevel 0 [15:0]		
0x14	Mlevel 1 [31:16]	Mlevel 1 [31:16]	Read/define the FIFO size that triggers the maximum-data-overflow interrupt (MDO).	39
0x16	Mlevel 1 [15:0]	Mlevel 1 [15:0]		
0x18	Transfer [31:16]	Transfer [31:16]	Read/define the amount of data to transfer out on the Local Bus.	40
0x1A	Transfer [15:0]	Transfer [15:0]		
0x1C	Block Size [31:16]	Block Size [31:16]	Read/define block size for Local Bus frames & end-of-block markers	41
0x1E	Block Size [15:0]	Block Size [15:0]		
0x20	Data [*]	Data [*]	Read/write data in memory via VXI interface	42
0x22				
0x24	Empty [31:16]		Read address of pointer from which data is read	43
0x26	Empty [15:0]			
0x28	FIFO Size [31:16]	Output [31:16]	FIFO Size: Read amount of data in memory Output: define empty pointer address	44 45
0x2A	FIFO Size [15:0]	Output [15:0]		
0x2C	Address [31:16]	Fill [31:16]	Address: read last value entered for Output or Fill Fill: define address of fill pointer	46 47
0x2E	Address [15:0]	Fill [15:0]		

*The number of bits transferred may be 16 or 32.

Register Descriptions

All registers are accessed on the VXIbus.

ID Register (read)

This register defines the basic methods used to access and control the VXI module.

Offset 0x00

Bits:	15..14	13..12	11..0
Contents:	Device Class	Address Space	Manufacturer ID
Value:	0x3	0x3	0xFFF

Device Class: A value of 0x3 indicates the module is register-based.

Address Space: A value of 0x3 indicates the module is A16 only.

Manufacturer ID: 0xFFF indicates the module is made by Agilent Technologies.

Device Type Register (read)

This register defines the module's model code.

Offset 0x02

Bits:	15..0
Contents:	Model Code
Value:	0x02B1

Model Code: A value of 0x02B1 identifies the module as an E9820A.

Status Register (read)

This register indicates the general status of the E9820A as described below.

Offset 0x04

Bits:	15	14	13..9	8..4	3	2	1	0
Contents:	A24/A32 Active	Modid*	Rsv	Revision	Ready	Passed	Sysfail Inhibit	Reset
Value:	0	MODID	0	REV	RDY	PASS	SFINH	RST

A24/A32 Active: This bit reflects the state of the Control register's A24/A32 enable bit. It is used only for A16/A24 and A16/A32 devices. Since the E9820A is an A16-only device, this bit should always be 0.

Modid*: This bit reflects the inverted state of the MODID pin on the VXIbus backplane connector. It is used during system initialization to identify the slot in which the module is installed.

Rsv: Reserved for future use.

Revision: Module hardware revision, 0-31. This document describes revision 2.

Ready: This bit indicates to the system controller that the module is ready to begin normal operation. This bit is cleared during reset and power-up conditions while the module performs reset functions. The Ready bit is set when the reset functions are successfully completed which typically takes less than 2 ms.

Passed: This bit indicates the success or failure of the module's self test. It always reads the same as the Ready bit.

Sysfail Inhibit: Indicates the state of the **Sysfail Inhibit** bit of the Control register. See page 27.

Reset: This bit indicates the state of the Reset bit of the Control register.

Control Register (write)

This register is used to inhibit the Sysfail line and reset the module.

Offset: 0x04

Bits:	15	14..2	1	0
Contents:	A24/A32 active	Rsv	Sysfail Inhibit	Reset

A24/A32 Active: This bit is used only for A16/A24 and A16/A32 devices. Even though it is ignored by the E9820A, it should always be written as "0".

Sysfail Inhibit: A "1" in this field disables the module from driving the SYSFAIL* line.

Reset: A "1" in this field forces the module into a soft reset state. This resets all registers on the E9820A to their power-on state. After releasing the reset by clearing this bit, it takes the E9820A about 2 milliseconds to return to operating condition. The user should verify that the E9820A is ready for operation by checking the Ready bit in the Status register.

Mode Register (read/write)

This register is used to control the transfer of data between the main memory and the local bus. See page 17.

Offset: 0x08

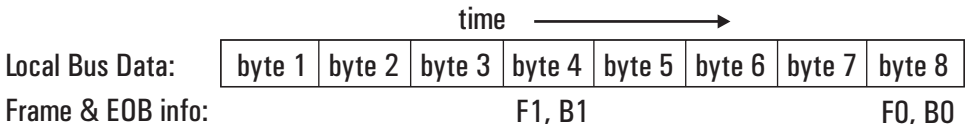
Bits:	15	14	13	12	11	10	9	8
Contents:	F1	B1	F0	B0	Out Xfer	Out Reblock	Rsv	Out Lbus
Initial Value:	0	0	0	0	0	0	0	0
Bits:	7	6	5	4	3	2	1	0
Contents:	Rsv	In Cont	Rsv	In Lbus	Loopback	Sdat	Sclk	Reset
Initial Value:	0	0	0	0	0	0	0	0

Frame & Block Markers

See the discussion of Local Bus frames and block on page 19.

In the following discussion, the read and write perspective is that of the VXIbus via the Data register. The figure below illustrates either of two scenarios:

- The data has come in on the Local Bus Input, was put in main memory, and will be read out on the VXI Interface.
- The data is being written into main memory via the VXIbus such that it can be moved out the local Bus Output with the proper frame and EOB markers.



To read markers:

1. Read the data block from the Data register
2. Then read these bits in the Mode register.

To write the markers:

1. With F1, B1, F0, B0 all clear (0), write all but the last 8 bytes of a block of data.
2. Write the marker bits in the Mode register as appropriate (for a frame or EOB).
3. Write the last 8 bytes of data into the Data register.
4. Clear F1, B1, F0, B0 (0) in preparation for the next block.

F1 (write): Writing a 1 to this bit causes a frame marker to be saved in main memory with the next write to the Data register such that when byte 4 is sent out the local bus output, its frame marker is set.

F1 (read): After reading the 8-byte word shown in the figure above, the value read from this bit is the frame marker that accompanied byte 4 when it was read in the local bus input.

B1 (write): Writing a 1 to this bit causes a block marker to be saved in main memory with the next write to the Data register such that when byte 4 is sent out the local bus output, its block marker is set.

B1 (read): After reading the 8-byte word shown in the figure above, the value read from this bit is the block marker that accompanied byte 4 when it was read in the local bus input.

F0 (write): Writing a 1 to this bit causes a frame marker to be saved in main memory with the next write to the Data register such that when byte 8 is sent out the local bus output, its frame marker is set.

F0 (read): After reading the 8-byte word shown in the figure above, the value read from this bit is the frame marker that accompanied byte 8 when it was read in the local bus input.

B0 (write): Writing a 1 to this bit causes a block marker to be saved in main memory with the next write to the Data register such that when byte 8 is sent out the local bus output, its block marker is set.

B0 (read): After reading the 8-byte word shown in the figure above, the value read from this bit is the block marker that accompanied byte 8 when it was read in the local bus input.

For block lengths that are a multiple of 8 bytes, F1 and B1 should always be 0.

Local Bus Settings

Out Xfer: This bit is used to control how data is moved out the Local Bus.

- Set this bit to "0" for delay operation; data is moved to the Output FIFO when:
  - **Out Lbus** is set "1" and
  - Fill_Pointer – Empty_Pointer > Mlevel 0
- Set this bit to "1" for snapshot operation; data is moved to the Output FIFO when **Out Lbus** is set "1" and data flow is stopped when the amount defined in the Transfer register (page 40) has been passed.

It is best to control data flow with the **Out Lbus** bit as as shown in the step-by-step procedure given on page 40.

Out Reblock: This bit controls whether existing frame and block markers are used.

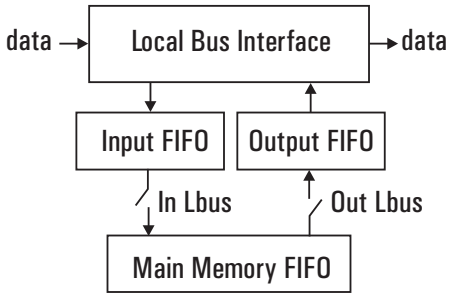
- When this bit is set to "0":
  - The frame and block markers that come in the Local Bus Input go out the Local Bus Output unchanged.
  - The frame and block markers for data entered into memory via VXI Interface go out as defined by the F1, F0, B1, B0 bits in the Mode Register.
- When this bit is set to "1", the Local Bus Output is reblocked according to the Block Size register such that the EOB and frame markers are both set at the end of each block of data. See page 41.

Out Lbus: This bit controls the data flow from the Main Memory to the Output FIFO.

- When set to "0", no data is transferred to the Output FIFO.
- When set to "1", data is transferred from Main Memory to the Output FIFO.

Data flow between the Output FIFO and the Local Bus Interface is controlled with the Local Bus register as discussed on page 33.

To read data from Main Memory through the VXIbus, it is best to turn off the feed to the Local Bus Output FIFO.



Caution

Data is lost when the **Output FIFO Reset** bit in the Local Bus register is clear while the **Out Lbus** bit is set. In this condition, data flows from Main Memory to the Output FIFO and is immediately lost. See also, Local Bus register on page 33.

In Cont: This bit controls whether local bus data input stops when memory is full (when clear, 0), or runs continuously, overwriting the oldest data (when set, 1).

- When this bit is "0", data stops being written into Main Memory when the Fill Pointer is the same as the Empty Pointer (minus 512 bytes).
- When this bit is set "1", the memory-full condition is ignored and data from the Input FIFO is written to memory overwriting the oldest data. See page 20.

Rsv: Reserved for future use.

In Lbus: This bit controls the data flow between the Input FIFO and Main Memory. See the figure on the previous page.

- When this bit is "0", no data is written into Main Memory.
- When set to "1", data is transferred from the Input FIFO to Main Memory.

Data flow from the Local Bus Interface to the Input FIFO is controlled with the Local Bus register as discussed on page 33. When the **In Lbus** bit is "0" as much as 4 KB of data may flow from the Local Bus Interface to the Input FIFO before it is full.

To write data into Main Memory from the VXIbus, it is best to turn off the feed from the Local Bus Input FIFO.

When the **LBUS mode** is Pipe, no data is routed to the Input FIFO and this bit has no effect. See page 33.

Note

Stopping data flow may halt the flow of data to downstream modules. For example, if the **LBUS mode** is eavesdrop, and data through the Input FIFO is stopped, none will flow out the Local Bus output, either.

In the E9820A, data cannot flow from the Input FIFO to the Main Memory and from Main Memory to the Output FIFO at the same time. When both **Out Lbus** and **In Lbus** are set "1", only output occurs.

Loopback: This bit may be used for testing purposes. When set "1", data in the Output FIFO is transferred to the Input FIFO. When the Loopback, Out Lbus, and In Lbus bits are all set and the amount of data in Main Memory exceeds the Mlevel 0 setting, data is read from memory and then written back into memory at a very high rate. This can be used to exercise all of memory and much of the data path. It does not exercise the Local Bus Interface. See page 17.

Sdat: This bit is used to communicate with the serial identification ROMs on the installed DIMMs. See the file `selftest.c` for an example of how this is used.

Sclk: This bit is used to communicate with the serial identification ROMs on the installed DIMMs. See the file `selftest.c` for an example of how this is used.

Note

selftest.c is part of the VXI Plug&Play library source code on the install disk. You can also find the latest drivers for instruments at the following URL:

`www.agilent.com/find/inst_drivers`

Reset: This bit is used in conjunction with the FIFO/LBUS reset bit to prepare the module for a new measurement. See page 33.

When set "1", the Fill and Empty pointer are cleared. This effectively clears the data in the main memory. It must be set "0" before any useful operation can occur.

Setting this bit does *not* change the data in main memory; it can still be retrieved by manipulating the other registers.

Memory Register (read)

This register contains status information for the memory components, including the 4 KB input and output FIFOs as well as main memory configuration.

Offset: 0x0A

Bits:	15	14	13	12	11..10	9..8	7..0
Contents:	Config Err	Rsv	FONE	FINE	Memory #	Memory Size	Rows
Initial Value:	0	0	0	0	MEMNUM	MEMCAP	MEMROWS

Config Err: This bit is set (1) when the E9820A senses an error in the memory configuration. One or more front-panel LEDs will flash when this condition exists. This bit is clear (0) when there is no error.

A memory configuration error is one of the following:

- an occupied socket is not being used (e.g., 3, 5, 6, or 7 DIMMs are installed)
- the installed DIMMs are of more than one size/capacity
- one of the DIMMs is a type not supported; see page 11.

This condition indicates that the installed hardware should be checked for proper configuration. You may not have as much capacity as you expect but it does not necessarily mean the module won't function.

Rsv: Reserved for future use.

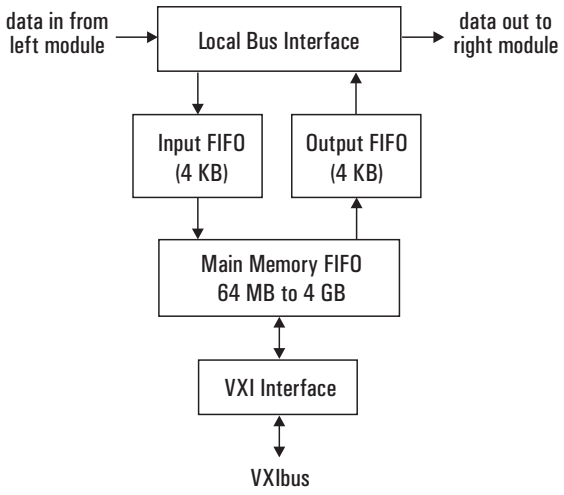
FONE: This bit is set (1) when the 4 KB FIFO *output* buffer is *not empty*. It may contain from 4 bytes to 4 KB of data. When this bit is clear (0) the FIFO is empty. Be aware that, when the receiving local bus device is not accepting data, one or more bytes may still be in the local bus interface chip.

This information is significant only when accessing data via the local bus. See figure below.

FINE: This bit is set when the 4 KB FIFO *input* buffer is *not empty*. It may contain from 4 bytes to 4 KB of data. When this bit is clear (0) the FIFO is empty.

Note

A transfer from the Input FIFO to the Main Memory FIFO cannot occur until the Input FIFO contains at least 512 bytes of data. See figure below.



Memory #: These 2 bits indicate the number of DIMMs installed as shown in the following table. Note that only 1, 2, 4, or 8, DIMMs are supported and they must be installed in the sockets such that all empty sockets are at the front of the module.

When these conditions are not met the Config Err bit is set, an LED on the front panel flashes, and the module will function with 1, 2, or 4 modules if the necessary sockets are occupied.

These DIMMs are non-buffered ECC SDRAM, either PC66 or PC100.

MEMNUM	Number of DIMMs
0	1
1	2
2	4
3	8

Memory Size: These 2 bits indicate the size of the individual DIMMs installed as shown in the following table. All DIMMs installed should be of the same size.

When more than one size is installed, the Config Err bit is set, an LED on the front panel flashes, and the module will function as if all DIMMs are the size of the smallest installed DIMM.

MEMCAP	DIMM size
0	64 MB
1	128 MB
2	256 MB
3	512 MB

Rows: These bits indicate the memory density of each occupied socket.

- Double-sided DIMMs (low-density SDRAM) are indicated with a bit value of 1.
- Single-sided DIMMs (high-density SDRAM) are indicated with bit values of 0.

This information is set automatically at power-on or when the E9820A is reset.

Row bit numbers	0	1	2	3	4	5	6	7
Socket numbers	1	2	3	4	5	6	7	8

Socket numbering is shown in the figure on page 11.

Local Bus Register (read/write)

This register is used to monitor and control the operation of the Local Bus Interface shown in the block diagram on page 17.

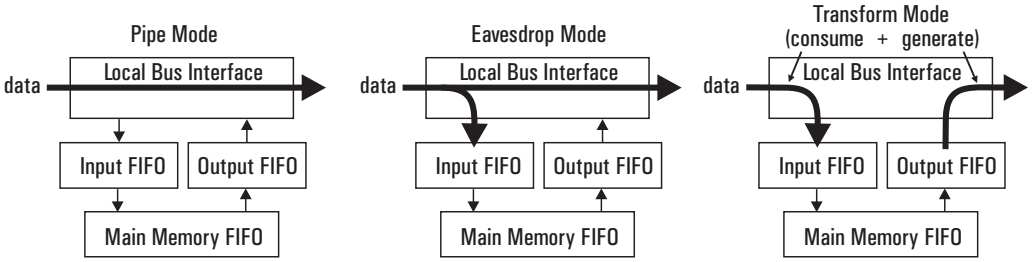
Offset 0x0C

Bits:	15..8	7..4	3	2	1	0
Contents:	Rsv	LBUS mode	Rsv	Output FIFO reset*	Input FIFO reset*	LBUS reset*
Initial Value:	0	0x1	0	0	0	0

LBUS mode: Defines how the Local Bus Interface moves data within or through the module. To make use of many of the modes requires a good understanding of the Local Bus Interface and the EOB and Frame markers. For most situations, the pipe P(0), eavesdrop E(3), or transform T(5) mode should be used (see table below).

- The *pipe* mode transfers data through the Local Bus Interface without loading anything into memory.
- The *eavesdrop* mode moves data in the Local Bus Input and out both the Local Bus Output as well as into the Input FIFO. This mode may be used for snapshot.
- The *transform* mode moves data in the Local Bus Input, through the FIFOs, and out the Local Bus Output. This mode may be used for snapshot or delay.

The **LBUS mode** value is latched when the **LBUS reset*** bit is set (1). To change the local bus mode the local bus must be reset by clearing the **LBUS reset*** bit. See note.



All possible modes are listed in the following table.

Mode	Description	Mode	Description
P(0x0)	Pipe	PC(0x6)	Pipe then Consume
P(0x1)	Pipe	PE(0x7)	Pipe then Eavesdrop
C(0x2)	Consume (input)	PG(0x8)	Pipe then Generate
E(0x3)	Eavesdrop	CP(0xC)	Consume then Pipe
G(0x4)	Generate (output)	EP(0xD)	Eavesdrop then Pipe
T(0x5)	Transform	GP(0xE)	Generate then Pipe

Notes

There is no difference between P(0) and P(1) for the E9820A.

It is best to pick a Local Bus mode that will work for all phases of the activities. Changing the Local Bus mode requires resetting the Local Bus Interface which deletes any data in the Interface. See the following discussion of the **LBUS reset*** bit.

Rsv: Reserved for future use.

Output FIFO reset: This bit is used to reset the Output FIFO shown in the previous figure. A "0" resets the FIFO, clearing the data in it and resetting the FIFO pointers. It must be set back to "1" for normal operation. Normally the FIFO reset bits are toggled at the same time as the **LBUS reset** bit.

I**nput FIFO reset**: This bit is used to reset the Input FIFO shown in the previous figure. A "0" resets the FIFO, clearing the data in it and resetting the FIFO pointers. It must be set back to "1" for normal operation. Normally the FIFO reset bits are toggled at the same time as the **LBUS reset** bit.

LBUS reset: This bit is used to reset the Local Bus Interface. A "0" resets the interface and a "1" returns it to normal operation. This returns the interface to a known state and is required to change the Local Bus mode.

Example: To start with or change to the transform mode, write the Local Bus register with the following two values:

```
0x0050       Picks transform mode (5) and resets the FIFOs & LBUS
0x0057       "Un-resets" the FIFOs & LBUS with transform mode selected
```

To end all activity, issue the reset state and stop. The mode selected at the same time doesn't matter unless you want to start again later in another mode. In that case, you should issue another reset command including the desired mode and then un-reset it with the same mode value included, as shown in the example above.

IRQ Status Register (read)

This register provides information about four conditions of the E9820A.

1. memory full
2. transfer complete
3. memory level 0 flag "up"
4. memory level 1 flag "up"

This register can be read at any time.

The E9820A response during an interrupt acknowledge cycle includes bits 15..8 of the **IRQ Status** register returned as data, and bits 7..0 which are the logical address of the interrupting device.

Offset 0x0E

Bits:	15	14	13	12	11	10	9	8
Contents:	DMFE	TCZE	MDOE	MDAE	DMFL	TCZL	MDOL	MDAL
Initial Value:	0	0	0	0	0	0	0	0
Bits:	7	6	5	4	3	2..0		
Contents:	DMF	TCZ	MDO	MDA	IEN	PRIO		
Initial Value:	0	0	0	0	0	0		

Interrupt Flags

Four flag indicators **DMF**, **TCZ**, **MDO**, and **MDA** track memory conditions as described. Each has a latched version which is provided during an interrupt acknowledge cycle to indicate which module pulled the interrupt and why. The flag bits can't be used as interrupt information because, by the time the interrupt is serviced, the condition causing it may have gone away and the flag would no longer be set.

DMF: Data Memory Full indicates that Main Memory is full. Whether input stops to avoid overwriting data is controlled by the **In Cont** bit in the **Mode** register (page 30).

The amount of data in the E9820A is a combination of the Input FIFO capacity and the Main Memory capacity. The Input FIFO may hold as much as 4 KB but it may also be empty; the **DMF** bit flags the "full" condition of Main Memory, only. The capacity of Main Memory is 512 bytes less than what is installed. See page 20.

This flag can generate an interrupt with the priority given in the **PRIOrity** field. The interrupt occurs when the **DMFL** bit is "1," the **DMFE** bit is "1," and the **IEN** bit is "1".

TCZ: Transfer Count Zero. This bit indicates that the amount of data requested in the Transfer register (page 40) has been output to the Output FIFO. As much as 4 KB of data may still be in the FIFO, however.

This flag can generate an interrupt with the priority given in the **IRQ priority** bit. The interrupt occurs when the **TCZL** bit is "1," the **TCZE** bit is "1," and the **IEN** bit is "1".

MDO: Maximum Data Overflow indicates that the amount of data equal to or greater than the value in the Mlevel 1 register is currently in Main Memory. This value may be used to indicate that the amount of delay has increased to a level that needs attention. See page 39.

This flag can generate an interrupt with the priority given in the **IRQ priority** bit. The interrupt occurs when the **MDOL** bit is "1," the **MDOE** bit is "1," and the **IEN** bit is "1".

MDA: Minimum Data Available indicates that the amount of data equal to or greater than the value in the Mlevel 0 register is currently in Main Memory. See page 38.

This flag can generate an interrupt with the priority given in the **IRQ priority** bit. The interrupt occurs when the **MDAL** bit is "1," the **MDAE** bit is "1," and the **IEN** bit is "1".

PRIO: IRQ priority: This field reflects the IRQ priority set in the IRQ Config register. See page 37.

Latched Bits

DMFL, TCZL, MDOL, MDAL: These bits are the latched version of the similarly-named interrupt flag bits. These bits transition high when their counterpart does so and they remain set until they are explicitly cleared. See the discussion on clearing interrupt flags on page 37.

Whether any of these bits cause an interrupt is determined by the settings in the IRQ Config register.

IRQ Config Register (write)

This register is used to define which interrupt conditions are allowed to generate an interrupt.

Offset 0x0E

Bits:	15	14	13	12	11	10	9	8
Contents:	DMFE	TCZE	MDOE	MDAE	DMFC	TCZC	MDOC	MDAC
Initial Value:	0	0	0	0	0	0	0	0
Bits:	7..4	3	2..0					
Contents:	Rsv	IEN	PRIO					
Initial Value:	0	0	0					

DMFE: Data Memory Full Enable. This bit turns on/off the DMF interrupt.
TCZE: Transfer Count Zero Enable. This bit turns on/off the TCZ interrupt.
MDOE: Max. Data Overflow Enable. This bit turns on/off the MDO interrupt.
MDAE: Min. Data Available Enable. This bit turns on/off the MDA interrupt.

- Setting the bit to "0" disables the interrupt.
- Setting the bit to "1" enables the interrupt.

Note To enable interrupts for a module you must also enable its IEN bit.

DMFC: Data Memory Full Clear. Clears the DMFL bit in the IRQ Status register.
TCZC: Transfer Count Zero Clear. Clears the TCZL bit in the IRQ Status register.
MDOC: Max. Data Overflow Clear. Clears the MDOL bit in the IRQ Status register.
MDAC: Min. Data Available Clear. Clears the MDAL bit in the IRQ Status register.

When an interrupt is serviced, use the appropriate clear bit to reset the latched bit that caused the interrupt. For example, an interrupt service routine (ISR) for the MDA interrupt should write a "1" to the MDAC bit to clear the MDAL flag.

Be sure to clear the flags *before* you re-enable IEN. See the IEN discussion below.

Note If the corresponding interrupt flag is set, the latched version of it will not clear when the clear bit is set. For example:

1. MDA "happens" (and interrupts are enabled for it so an interrupt occurs)
2. MDAL gets set to indicate what caused the interrupt
3. service routine "handles" it and sets MDAC while MDA is set (still or again),
4. So MDAL stays set.

IEN: Interrupt Enable. This bit sets the interrupt status for the module as a whole. When an interrupt occurs, this bit is set to "0" to indicate, internally, whether an interrupt has occurred that has not yet been serviced (no further interrupts are issued until the current one is serviced), so the service routine should re-enable it.

PRIO: IRQ priority: This field sets the VXIbus interrupt priority level (1 of 7) for all interrupts on the module and is used by the slot-0 module to arbitrate system interrupts. Setting this field to "0" effectively disables all interrupts.

Mlevel 0 Register (read/write)

This register's main purpose supports the delay feature found in the E9830A, but it has some functionality for the E9820A. In general it controls local bus output based on the amount of data in memory such that no local bus output occurs until the value specified by the value in this register is accumulated in memory.

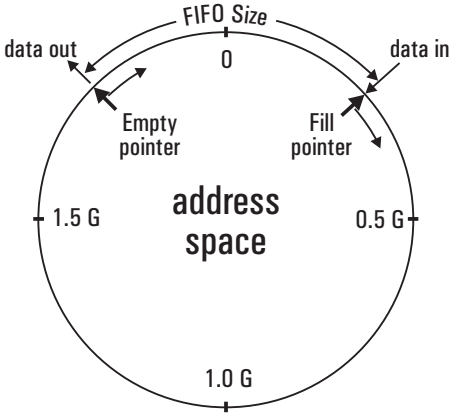
Offset 0x10

Bits:	15..0
Contents:	Mlevel 0[31:16]
Initial Value:	0

Offset 0x12

Bits:	15..9	8..0
Contents:	Mlevel 0[15:9]	Mlevel 0[8:0]
Initial Value:	0	always 0

Local Bus output must first be enabled by setting the **Out Lbus** bit of the **Mode** register is to "1", (see page 29). When the amount of data in Main Memory exceeds the setting in the **Mlevel 0** register by 512 bytes, a block of 512 bytes is transferred from Main Memory to the Output FIFO. If the Local Bus Interface mode is one that performs output (transform or generate) and the next module to the right of the E9820A is taking data, data will flow out of the E9820A such that the amount of data in the main memory corresponds to the setting in the **Mlevel 0** register.



When **FIFO Size** = **Mlevel 0**, the MDA bit (minimum data available) is set in the IRQ Status register.

When **FIFO Size** > **Mlevel 0**, data is allowed to move to the Output FIFO.

When **FIFO Size** = **Mlevel 1**, the MDO bit (maximum data overflow) is set in the IRQ Status register. In some applications, this may indicate that output can't keep up with the data input rate.

Notes

Since Main Memory's maximum capacity is *full* –512 and **FIFO Size** must exceed **Mlevel 0** by 512 bytes to trigger output, the maximum usable value for this register is *full* –1024, where *full* is the amount of memory installed on the board.

Setting this register to *full* is the same as setting it to 0; doing so means output is not constrained by this register. Setting it to *full* –512 creates a condition in which the **FIFO size** can never exceed it and no output is allowed. See the discussion of maximum capacity on page 20.

Since the contents of this register is a 32-bit number and the E9820A is a D16 device, two register accesses are required to read or write it. Always access multi-word registers in order of increasing address (most-significant bits first).

The "resolution" of the addressing is 512 bytes. When reading this register bits 8..0 will always be "0". When writing this register, the values in bits 8..0 are ignored and always treated as "0".

Mlevel 1 Register (read/write)

This register is another memory-level (accumulation indicator) flag like Mlevel 0 which may be used to warn of a pending data overflow.

Offset 0x14

Bits:	15..0
Contents:	Mlevel 1[31:16]
Initial Value:	0

Offset 0x16

Bits:	15..9	8..0
Contents:	Mlevel 1[15:9]	Mlevel 1[8:0]
Initial Value:	0	0x200

This register is a capacity value that is compared to the **FIFO Size**. When the **FIFO Size** reaches this value, the **MDO** flag is set in the interrupt register. See page 35.

Since the contents of this register is a 32-bit number and the E9820A is a D16 device, two register accesses are required to read or write it. Always access multi-word registers in order of increasing address (most-significant bits first).

The "resolution" of the addressing is 512 bytes. When reading this register, bits 8..0 will always be "0". When writing this register, the values in bits 8..0 are ignored and always treated as "0".

Transfer Register (read/write)

This register is used to specify the amount of data to move from memory to the Local Bus output, starting at the location indexed by the Empty register. It has nothing to do with the VXIbus. See Data register (page 42) for VXIbus data I/O.

Offset 0x18

Bits:	15..0
Contents:	Transfer[31:16]
Initial Value:	0

Offset 0x1A

Bits:	15..9	8..0
Contents:	Transfer[15:9]	Transfer[8:0]
Initial Value:	0x400	

When the **Out Xfer** bit in the Mode register (page 29) is set "1" and the **Out Lbus** bit is enabled, data flows out the Local Bus and stops when the number of bytes set in the Transfer register have been passed. This assumes that the active **LBUS mode** allows output on the Local Bus. See page 33.

Since the contents of this register is a 32-bit number and the E9820A is a D16 device, two register accesses are required to read or write it. Always access multi-word registers in order of increasing address (most-significant bits first).

The E9820A handles the data as 64-bit words and transfers data in 64-word blocks (512 bytes), so the "resolution" of the addressing is 512 bytes, which corresponds to the least significant 9 bits of the address. When reading this register bits 8..0 will always be "0". When writing this register, the values in bits 8..0 are ignored and always treated as "0".

To transfer data from Main Memory:

1. Specify how much to transfer in the Transfer register.
2. If you want to start reading at an index other than the Empty pointer, specify where to start with the Output register. This is a value prior to the newest data in memory (before the Fill pointer). See page 45.
3. Set the **Out Xfer** bit. See page 29. This clears the **TCZ** bit (transfer count zero).
4. Set the **Out Lbus** bit. See page 29. This causes the data transfer to occur.
5. Monitor the **TCZ** bit (either polling or with an interrupt). It becomes set when the transfer is complete. When this occurs, clear the **Out Lbus** bit in the Mode register.
6. Clear the **Out Xfer** bit.
7. To transfer another batch of data of the same size, return to step 3.
   To transfer a batch of data with a different size, return to step 1.

Caution

Avoid setting the **Out Lbus** bit when the **Out Xfer** bit is clear. This scenario is used for delay and, if the **FIFO Size** is larger than the value in **MIevel 0**, causes data to flow out the Local Bus. Setting the **MIevel 0** register to 0xFFFFFE00 avoids this and would allow you to transfer data without having to clear and reset the **Out Lbus** bit as described in the above instructions.

Note

The E9820A cannot transfer data from the Input FIFO to Main Memory at the same time as data is transferred from Main Memory to the Output FIFO; when both **In Lbus** and **Out Lbus** are set "1", only output occurs. So you can't capture new snapshot data while transferring data out the Local Bus. You can move data out the VXIbus while capturing new data. See the Data register on page 42.

Block Size Register (read/write)

This register is used to specify the size of the Local Bus blocks and frames as described in the discussion on page 19.

Offset 0x1C

Bits:	15..8	7..0
Contents:	Reserved	Block Size[23:16]
Initial Value:	0	0

Offset 0x1A

Bits:	15..3	2..0
Contents:	Block Size[15:3]	Block Size[2:0]
Initial Value:	0x400	

When the **Out Reblock** bit (page 29) is set, this register specifies the amount of data in a Local Bus frame or block (the amount of data between frame and EOB markers).

The **Block Size** register should be set to the number of bytes to be output per block and the value must be a multiple of eight. At the end of each block, both the EOB and frame bits will be set.

The maximum block size is 0x00fffffC (16,777,212) bytes.

The reset value of this register is 0x400 or 1024 bytes.

Since the contents of this register is a 24-bit number and the E9820A is a D16 device, two register accesses are required to read or write it. Always access multi-word registers in order of increasing address (most-significant bits first).

The "resolution" of the setting is 8 bytes. When reading this register bits 2..0 will always be "0". When writing this register, the values in bits 2..0 are ignored and always treated as "0".

Note

This register does *not* control the amount of data transferred. It controls the EOB and FRAME bits in the Local Bus data stream.

Data Register (read/write)

This register is used to read and write Main Memory through the VXI Interface.

Offset 0x20 (16-bit access)

Bits:	15..0
Contents:	Data[15:0]

Offset 0x20 (32-bit access)

Bits:	31..0
Contents:	Data[31:0]

The **Data** register allows either 32-bit or 16-bit access. It is the *only* register that allows 32-bit data access.

The Data register acts like a FIFO in that you read from it or write to it repeatedly to access sequential data.

Example

Reading captured data. When capture is complete, the Empty Pointer indexes the first data read in (oldest) and the Fill Pointer indexes the last data (most recent).

- To read all of this data, simply begin reading from the **Data** register and stop when the Empty Pointer is equal to the Fill Pointer.
- To read data starting at a point between the pointers, use the **Output** register to indicate where to start reading relative to the Fill Pointer. This moves the Empty Pointer to the desired start point. See page 45.

Mixing Transfer Methods

If you want to alternate between using the VXI Interface and using the Local Bus, there are special considerations to observe. Unlike the VXI Interface which can read or write data two bytes or four bytes at a time, the Local Bus moves data in 512-byte blocks. The issues to consider depend on the transfer direction and transition:

- Read, transition from VXI to Local Bus: If the Empty Pointer is not on a 512-byte increment when the Local Bus transfer begins, the data will read out in the wrong order. To avoid this, use one of the following methods before changing to Local Bus transfers:
  - Make sure the VXI reads a multiple of 512-bytes worth of data.
  - Set the Empty Pointer index with the **Output** register.
- Write, transition from VXI to Local Bus: If the Fill Pointer is not on a 512-byte increment when the Local Bus transfer begins, the data will be written into memory in the wrong order. To avoid this, use one of the following methods before changing to Local Bus transfers:
  - Make sure the VXI writes a multiple of 512-bytes worth of data.
  - Set the Fill Pointer index with the **Fill** register.
- Read, transition from Local Bus to VXI: Since data is buffered in the Output FIFO, be sure that the module consuming data reads it all out before switching to VXI. To stop data flow to the Local Bus Output, use the **Out Lbus** bit; see page 29.
- Write, transition from Local Bus to VXI: Since data is transferred in 512-byte blocks between the Local Bus FIFOs and Main Memory, any amount of data less than this is left in the Input FIFO until it accumulates a size of 512 bytes. So be sure to write a multiple of 512 bytes over the Local Bus before stopping the transfer and switching to VXI writing, or some data will be left in the Input FIFO.

Empty Register (read)

This register indicates the index of the Empty Pointer, the address in Main Memory from which data is being read, for either Local Bus or VXI access. See page 20.

Offset 0x24

Bits:	15..0
Contents:	Empty[31:16]
Initial Value:	0

Offset 0x26

Bits:	15..9	8..0
Contents:	Empty[15:9]	Empty[8:0]
Initial Value:	0	always 0

Unlike other address registers, the upper bits of the **Empty** register are not masked based on the amount of memory installed on the module. Thus, the value read from this register does not roll until a full 4 GB (2^{32}) of data has been output. Also, the value indicates the amount of data read since the Main Memory was last reset[1].

Notes

Writing to the **Output** register redefines the contents of the **Empty** register to be a fixed offset from the Fill Pointer.

There is no guarantee as to what values the unused upper bits of the Empty register will be after a write to the **Output** register.

Since the contents of this register is a 32-bit number and the E9820A is a D16 device, two register accesses are required to read it. Always access multi-word registers in order of increasing address (most-significant bits first).

The "resolution" of the addressing is 512 bytes. When reading this register bits 8..0 will always be "0".

[1] Main Memory may be reset with the Control Register's **Reset** bit. See page 30.

FIFO Size Register (read)

This register reflects how much data is currently in main memory. See page 20.

Offset 0x28

Bits:	15..0
Contents:	FIFO Size[31:16]
Initial Value:	0

Offset 0x2A

Bits:	15..9	8..0
Contents:	FIFO Size[15:9]	FIFO Size[8:0]
Initial Value:	0	always 0

Since bits 8 through 0 are always 0, the resolution of this register value is 512 bytes.

FIFO Size is the difference between the Fill Pointer and the Empty Pointer. Since the **Output** register changes the address of the Empty Pointer, it also changes the FIFO size. This does not change the data currently in main memory, but further input may overwrite existing data in memory.

See the discussion on Maximum Capacity on page 20.

Note

Besides the data in main memory, there may be data in the Local Bus FIFO buffers and in the local bus interface. See the discussion on page 17.

Since the contents of this register is a 32-bit number and the E9820A is a D16 device, two register accesses are required to read it. Always access multi-word registers in order of increasing address (most-significant bits first).

The "resolution" of the addressing is 512 bytes. When reading this register bits 8..0 will always be "0".

Output Register (write)

This register is used to define the address of the Empty Pointer relative to the Fill Pointer to provide flexible read access to data in main memory. See page 20.

Offset 0x28

Bits:	15..0
Contents:	Output[31:16]
Initial Value:	0

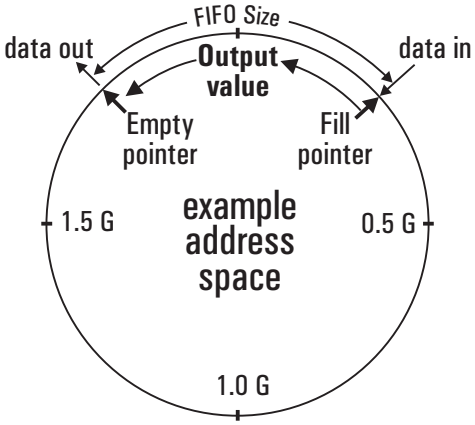
Offset 0x2A

Bits:	15..9	8..0
Contents:	Output[15:9]	Output[8:0]
Initial Value:	0	always 0

As shown in the figure below, the value written to the **Output** register changes the address of the Empty Pointer as:

```
Empty_Pointer = Fill_Pointer - Output_Register
```

This also changes the **FIFO Size**; e.g., setting the **Output** register to 0 effectively empties or "clears" memory[2]. Subsequent reads from memory start at this address and the act of reading out data also moves the Empty Pointer toward the Fill Pointer.



Since the contents of this register is a 32-bit number and the E9820A is a D16 device, two register accesses are required to write it. Always access multi-word registers in order of increasing address (most-significant bits first).

The "resolution" of the addressing is 512 bytes. When writing this register, the values in bits 8..0 are ignored and always treated as "0".

Caution

Do not write to the **Output** register while moving data out on the Local Bus. To assure that it is safe to write to the **Output** register, first clear the **Out Lbus** bit in the **Mode** register and wait at least 2 µs. See page 29.

[2] This doesn't mean that any data is deleted from memory; just that the pointers are positioned such that new data can overwrite any/all of it.

Address Register (read)

This register reflects the last value written to either the Output or Fill register.

Offset 0x2C

Bits:	15..0
Contents:	Address[31:16]
Initial Value:	0

Offset 0x2E

Bits:	15..9	8..0
Contents:	Address[15:9]	Address[8:0]
Initial Value:	0	always 0

Since the contents of this register is a 32-bit number and the E9820A is a D16 device, two register accesses are required to read it. Always access multi-word registers in order of increasing address (most-significant bits first).

The "resolution" of the addressing is 512 bytes. When reading this register bits 8..0 will always be "0".

Fill Register (write)

This register sets the address of the Main Memory Fill Pointer. See page 20.

Offset 0x2C

Bits:	15..0
Contents:	Fill[31:16]
Initial Value:	0

Offset 0x2E

Bits:	15..9	8..0
Contents:	Fill[15:9]	Fill[8:0]
Initial Value:	0	always 0

The **Fill** register is mainly used for diagnostic purposes.

Since the contents of this register is a 32-bit number and the E9820A is a D16 device, two register accesses are required to write it. Always access multi-word registers in order of increasing address (most-significant bits first).

The "resolution" of the addressing is 512 bytes. When writing this register, the values in bits 8..0 are ignored and always treated as "0".

Caution

Do not write to the **Fill** register while moving data in on the Local Bus. To assure that it is safe to write to the **Fill** register, first clear the **In Lbus** bit in the **Mode** register and wait at least 2 µs. See page 30.

Index

F

G

H

I

L

M

N

O

P

R

S

T

V