Ziatech ZT 8825

Expanded / Extended Memory Systems

ZT 8825 and ZT 88CT25
Expanded\Extended Memory Systems

HARDWARE MANUAL

For ZT 8825
and ZT 88CT25 REVISION A.4

Reorder Part Number ZT M8825
April 29, 1993

ZIATECH
CORPORATION
1050 Southwood Drive
San Luis Obispo, CA 93401 USA
FAX (805) 541-5088
Telephone (805) 541-0488

ZIATECH WARRANTY

Ziatech Hardware: Within two years of shipping date, Ziatech will repair or replace products which prove to be defective in materials and/or workmanship, provided they are promptly returned to Ziatech at customer's expense and have not been repaired, altered, or damaged by non-Ziatech personnel. Service after warranty is available at a predesignated service charge. Batteries are not covered by this warranty. No other warranty is expressed or implied.

Ziatech Software: Within 90 days of shipping date, Ziatech will replace software (PROM or diskette) should it prove defective.

Products not manufactured by Ziatech: Limited to the warranty provided by the original manufacturer.

Notice: Contact Ziatech for a Return Materials Authorization (RMA) number before returning any product to Ziatech for repair.

Life Support Policy: Ziatech products are not authorized for use as critical components in life support devices or systems without the express written approval of the president of Ziatech Corporation. As used herein:

1. Life support devices or systems are devices or systems that support or sustain life and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

CUSTOMER SUPPORT

If you have a technical question, please call Ziatech's Customer Support Service at the following number:

Corporate Headquarters: (805) 541-0488
(805) 541-5088 (FAX)

You can also use a modem to leave a message on the 24 hour Ziatech Bulletin Board Service (BBS) by calling (805) 541-8218. The BBS will provide you with current Ziatech product revision and upgrade information.

If you have a sales question, please contact your local Ziatech Sales Representative.

PREFACE

This manual discusses hardware and systems level topics for the ZT 8825 and ZT 88CT25 extended byte-wide memory systems. All references in this manual to the ZT 8825 include the ZT 88CT25 unless otherwise indicated. The ZT 8825 Software Support Package manual addresses software support issues.

The following organizational outline describes the focus of each chapter in this manual. Section headings enclosed in boxes indicate the location of a labeled tab for quick access to the appropriate information.

I. INTRODUCTION

Chapter 1, "Introduction," contains a brief overview of the ZT 8825. It includes a product definition and a listing of product features.

I. GETTING STARTED

Chapter 2, "Getting Started," summarizes the information essential to getting your ZT 8825 up and running. You can refer to subsequent chapters for further explanation of the material covered in "Getting Started."

III. USER’S REFERENCE

Chapter 3, "Theory Of Operation," presents a more detailed look at the way the ZT 8825 functions. Some topics covered in this chapter include I/O addressing, memory addressing, memory mapping, and extended memory mode.

Chapter 4, "Application Examples," includes specific examples of the ZT 8825 in operation, including code to implement these applications. Examples include initializing the ZT 8825 as continuous memory and as extended memory.

Chapter 5, "Configurable Options," details the various jumper-selectable options on the ZT 8825.

IV. APPENDICES

Appendix A, "Jumper Configurations," lists the ZT 8825 jumpers in numerical order and describes the function of each. This appendix also indicates the factory default configuration and includes a figure that shows the location of each jumper.

Appendix B, "Specifications," includes electrical, mechanical, and environmental specifications for the ZT 8825.

Appendix C, "Hardware Quick Reference," supplies a centralized location for frequently referenced tables and illustrations.

Appendix D, "Customer Support," offers a product revision history, the Ziatech warranty, and the necessary information should you need to return your ZT 8825 for repair.

Appendix E, "Glossary," lists definitions for acronyms and important terms used in the manual.

CONTENTS

I. INTRODUCTION

Chapter 1. INTRODUCTION 1-1

OVERVIEW 1-1
PRODUCT DESCRIPTION 1-2
FEATURES OF THE ZT 8825 1-4

II. GETTING STARTED

Chapter 2. GETTING STARTED 2-1

OVERVIEW 2-1
UNPACKING 2-2
WHAT'S IN THE BOX 2-2
SYSTEM REQUIREMENTS 2-3
Backplane Recommendations 2-4
Processor Compatibility 2-4
SOCKET ASSIGNMENTS 2-5
WRITE PROTECT 2-5
MEMORY SPEED REQUIREMENTS 2-6
BOARD CONFIGURATIONS 2-7
CHIP SIZE SELECTION 2-10
CHIP TYPE SELECTION 2-14
I/O ADDRESSING 2-16

III. USER'S REFERENCE

Chapter 3. THEORY OF OPERATION 3-1

OVERVIEW 3-2
MEMORY CHIP TYPE 3-4
MEMORY CHIP SIZE 3-4
WAIT STATES 3-6
WRITE PROTECTION 3-6
BATTERY BACKUP 3-7
I/O ADDRESSING 3-10
MEMORY ADDRESSING 3-13
20-bit Memory Addressing 3-13
24-bit Memory Addressing 3-13
MEMORY MAPPING 3-14
Paging 3-14
Map Registers 3-15
Configuration Register 3-17
Setting the Map Register 3-18
Absolute Address Mode 3-19
EXTENDED MEMORY MODE 3-20
Setting the 24-bit Map Register 3-20

Chapter 4. APPLICATION EXAMPLES 4-1

OVERVIEW 4-1
INITIALIZING ZT 8825 AS CONTINUOUS MEMORY 4-1
Assembly Language Setup 4-2
Microsoft C Setup 4-4
EXAMPLE PROGRAMS USING EMS 4-7
Example 1 4-7
Example 2 4-12
Example 3 4-21
Example 4 4-23

Chapter 5. CONFIGURABLE OPTIONS 5-1

OVERVIEW 5-2
MEMORY CHIP TYPE SELECTION 5-3
MEMORY CHIP SIZE SELECTION 5-6
WAIT STATE SELECTION 5-8
BATTERY BACKUP SELECTION 5-9

I/O BASE ADDRESS SELECTION 5-10
MEMORY ADDRESS SELECTION (20- VS. 24-BIT) 5-12

IV. APPENDICES

Appendix A. JUMPER CONFIGURATIONS A-1

OVERVIEW A-1
JUMPER DESCRIPTIONS A-2
USER JUMPER CONFIGURATION A-5
SWITCH CONFIGURATION A-5

Appendix B. SPECIFICATIONS B-1

ELECTRICAL SPECIFICATIONS B-1
ZT 8825 Power Requirements B-1
ZT 88CT25 Power Requirements B-2
STD Load Characteristics B-2
ENVIRONMENTAL SPECIFICATIONS B-4
Operating Temperature B-4
Storage Temperature B-4
MECHANICAL SPECIFICATIONS B-4
Card Dimensions & Weight B-4

Appendix C. HARDWARE QUICK REFERENCE C-1

JUMPER DESCRIPTIONS C-2
JUMPER REFERENCE TABLES C-5

Appendix D. CUSTOMER SUPPORT D-1

ZT 8825 REVISION HISTORY D-1
Revision 0.1 . . . . D-1
Revision A - June 7, 1988 . . . . D-1
Revision A.1 - January 1, 1989 . . . . D-2
Revision A.2 - July 6, 1989 . . . . D-2
Revision A.3 - January 30, 1991 . . . . D-2
Revision A.4 - September 23, 1991 . . . . D-2
TECHNICAL ASSISTANCE D-4
RELIABILITY D-4
WARRANTY D-5
RETURNING FOR SERVICE D-6

Contents

Appendix E. GLOSSARY E-1

V. INDEX

Index i

TABLES

Table 2–1 ZT 8825 Basic Access Time Requirements. 2-6
Table 2–2 Chip Size Selection. 2-12
Table 2–3 Valid Chip Combinations. 2-13
Table 2–4 Chip Type Jumper Configurations. 2-15
Table 2–5 ZT 8825 I/O Registers. 2-16
Table 2–6 Default I/O Address Assignments. 2-17
Table 3–1 Chip Size Combinations. 3-5
Table 3–2 Est. Battery Life @ Max. Standby Current. 3-7
Table 3–3 Default I/O Address Assignments. 3-10
Table 3–4 ZT 8825 I/O Registers. 3-11
Table 5–1 Signal Choices on Each Socket. 5-3
Table 5–2 Jumper Selection of Socket Signals. 5-4
Table 5–3 Chip Type Jumper Configurations. 5-5
Table 5–4 Switch Settings for All Possible Chip Sizes. 5-7
Table 5–5 ZT 8825 Basic Access Time Requirements. 5-8
Table 5–6 Jumper Settings for Battery Backup. 5-9
Table 5–7 Jumpers for I/O Base Address Decoding. 5-11
Table A–1 ZT 8825 Jumper Descriptions. A-2
Table A–2 Switch Settings for All Possible Chip Sizes. A-5
Table B–1 ZT 8825 STD Bus Loading. B-3
Table B–2 ZT 8825 Rev. A.4 Material List. B-6
Table C–1 ZT 8825 Jumper Descriptions. C-2
Table C–2 Jumpers for I/O Base Address Decoding. C-5
Table C–3 Chip Type Jumper Configurations. C-6
Table C–4 Switch Settings for All Possible Chip Sizes. C-7

ILLUSTRATIONS

Figure 2–1 Factory Default Jumper Configuration. 2-8
Figure 2–2 Example Setting for 24-bit Addressing. 2-9
Figure 2–3 Memory Chip Orientation. 2-11
Figure 3–1 Functional Block Diagram. 3-2
Figure 3–2 Logical to Physical Address Mapping. 3-15
Figure 3–3 Map Register. 3-16
Figure 3–4 Configuration Register. 3-17
Figure A–1 Customer Jumper Configuration. A-6
Figure B–1 ZT 8825 Card Dimensions. B-5

Chapter 1

INTRODUCTION

Contents **Page**

OVERVIEW 1-1
PRODUCT DESCRIPTION 1-2
FEATURES OF THE ZT 8825 1-4

OVERVIEW

This chapter briefly introduces some key features and some possible applications for the ZT 8825. Unless specifically called out, all references to ZT 8825 also refer to the CMOS extended temperature version ZT 88CT25.

PRODUCT DESCRIPTION

The ZT 8825, a member of Ziatech's growing STD-80 Series product line, is a state-of-the-art memory board that accommodates 28- and 32-pin byte-wide memory chips. The board is divided into two rows of four memory chip sockets. Device type can be configured for each row of four sockets individually. Address decoding can be configured separately for each 16 Kbytes of memory. This provides an extremely flexible board that can accommodate the memory requirements of nearly any system.

The ZT 8825 can be used with several different processors in many modes of operation:

1. Normal Memory: The ZT 8825 memory can be mapped in 16 Kbyte blocks to anywhere in the 1 Mbyte address space of the *STD-80 Series Bus Specification.*
2. Extended Memory: For processors like the Intel 80286 (protected mode) and 80386 or the Motorola 680XX family, the ZT 8825 provides memory that can be mapped in 16 Kbyte blocks at any address, including those above the 1 Mbyte limit. The board supports 24-bit addressing if the CPU meets the *STD-80 Series Bus Specification.*
3. Expanded Memory: The ZT 8825 hardware supports the *Lotus-Intel-Microsoft Expanded Memory Specification* (EMS), Versions 3.2 & 4.0 and the *AST Enhanced Expanded Memory Specification* (EEMS), Version 3.2. These specifications allow access to additional memory outside of the normal 1 Mbyte address space of the processor or DOS. The Ziatech ZT 8825 Software Support Package provides many of the DOS subroutines required for the above specifications. See the ZT 8825 Software Support Package manual for more information.
4. RAM or PROM disk: The ZT 8825 can be organized as a disk using the software provided with the product. DOS can work with these pseudo-disks in the same manner as with regular disks.

Devices that can be used in the ZT 8825 include: EPROMs (for example, 27128, 27256, 27512, 27010, 27020, 27040, up to 1 Mbyte chips), 5V-only EEPROMs (for example, 28C256, 28C010), and static RAMs (for example, HM62256, HM628128). The maximum amount of memory on a ZT 8825 is limited to 2 Mbytes. There are also some limitations to the combinations of chip sizes that can be used (see page 3-5).

The STD-80 Series 20-bit memory address protocol is used for memory addressing. The board will also accept 24-bit addressing. The on-board registers use 8- or 16-bit I/O addresses. The ZT 8825 configured with 120 ns memories operates without wait states when used with an 8 MHz processor board such as the ZT 8809. The ZT 8825 configured with 250 ns memories operates without wait states when used with a 5 MHz CPU board. An optional wait state is user-selectable, allowing the use of slower memory devices (see page 3-6).

An example of the versatility of the ZT 8825 is that EPROM can be used on one half of the ZT 8825 and RAM on the other half. This arrangement will provide the total memory requirements for many STD systems. The ZT 8825 is effectively two byte-wide boards in one. See Chapter 4 for details on additional application examples.

FEATURES OF THE ZT 8825

- Direct 20-bit or 24-bit addressing (compatible with the *STD-80 Series Bus Specification*)
- Each 16 Kbyte block is independently addressable
- Two sets (four sockets each) of 32-pin JEDEC sockets
- Various combinations of RAM (32K-2M) and EPROM (16K-2M)
- Optional wait state for slow device types
- Selectable hardware and software write protect to aid debugging
- No wait states needed at 5 or 8 MHz with fast memory chips
- Supports EMS versions 3.2 and 4.0 as well as EEMS version 3.2
- Supports 5V-only Flash EPROMS
- 2 Mbyte maximum on-board storage
- Optional 1 Amp-hour battery backup
- Burned in and tested
- TTL-compatible
- The CMOS version ZT 88CT25 operates at substantially lower power (0.5 W typical as compared to 2.5 W typical for the ZT 8825)
- The ZT 88CT25 operates over a temperature range of -40˚ to +85˚C

Chapter 2

GETTING STARTED

Contents **Page**

OVERVIEW 2-1
UNPACKING 2-2
WHAT'S IN THE BOX 2-2
SYSTEM REQUIREMENTS 2-3
Backplane Recommendations 2-4
Processor Compatibility 2-4
SOCKET ASSIGNMENTS 2-5
WRITE PROTECT 2-5
MEMORY SPEED REQUIREMENTS 2-6
BOARD CONFIGURATIONS 2-7
CHIP SIZE SELECTION 2-10
CHIP TYPE SELECTION 2-14
I/O ADDRESSING 2-16

OVERVIEW

This chapter is intended for users who require only a brief summary of information essential to getting their systems up and running. Other users should refer back to this chapter after familiarizing themselves with the more detailed information found in subsequent chapters. Unless explicitly noted, all references to ZT 8825 in this manual also refer to the CMOS version ZT 88CT25.

UNPACKING

Upon receipt of the shipping carton, check for any damage to its contents. Normally, the contents will survive considerable abuse since the packing is designed to protect against typical shipping and handling stresses. However, if the board is damaged or fails to function, notify the carrier and Ziatech to arrange for insurance settlement. Retain the shipping carton and packing material for the carrier's inspection. Under no circumstances should a board or cable be returned to Ziatech without a Return Material Authorization (RMA) Number (see page D-6).

WHAT'S IN THE BOX

When the package is opened, you should find:

1. The ZT 8825 Operator's Manual and the ZT 8825 Software Support Package manual (in a binder).
2. One ZT 8825 or ZT 88CT25 PC Board.
3. Anti-static packing material.
4. One software driver diskette.

Attach the sticker found with the manual onto the spine of the binder for easy identification. Be sure to save the anti-static packing material for storing or shipping the ZT 8825.

WARNING!

Like all equipment utilizing CMOS devices, the ZT 8825 must be protected from static discharge. Never remove or install any of the socketed parts except at a static-free work station.

SYSTEM REQUIREMENTS

The ZT 8825 is designed to be physically compatible with card cages manufactured to meet the STD-80 Series Specification and should normally be mounted in one.

The board requires +5 V DC ±5% @ 0.67 A maximum, not including any memory chips. A typical fully-populated EPROM board requires approximately 1.17 A maximum. Maximum continuous power dissipation in this configuration is 6.0 W. A typical fully-populated SRAM board requires approximately 0.77 A maximum.

The ambient temperature must be maintained between 0˚ and +65˚ Celsius to avoid improper operation and possible damage to the ZT 8825. The ZT 88CT25 operates in an ambient temperature range of -40˚ to +80˚C. The relative humidity should be less than 95% at 40˚ C, non-condensing.

Vertical mounting is recommended in convective cooling systems not equipped with a fan. However, most systems require a fan for reliable operation. Horizontal mounting is not recommended unless forced air cooling of at least 30 cubic ft./min. is provided.

If a standard STD bus card cage is not used, refer to the figure on page B-5 for dimensions. Be sure to allow enough clearance for good convective or forced cooling.

Backplane Recommendations

Ziatech recommends the use of a multilayer backplane for good power distribution and noise suppression. The ZT 32 series of card cages and backplanes work well in STD-80 and STD 32 systems.

Processor Compatibility

The ZT 8825 is designed for STD-80 Series systems. It can be used with other microprocessors only if they meet the STD-80 Series multiplexed memory addressing and timing specifications.

SOCKET ASSIGNMENTS

Memory chip socket Row A includes sockets 4A, 5A, 6A, and 7A. Memory chip socket Row B includes sockets 4B, 5B, 6B, and 7B. Socket 4A/B is always in the lowest address range of the row when using Ziatech initialization software, and socket 7A/B is always the highest.

28-pin memory chips must be inserted with socket pins 1, 2, 31 and 32 empty. When looking at the ZT 8825 circuit board with the component side facing you and the edge connector at the bottom, 28-pin chips should be installed in the bottom-most section of the socket.

WRITE PROTECT

When debugging programs in RAM, there is a tendency for certain programming errors to cause undesired writes to RAM. The ZT 8825 has a write-protect switch (SW2) that can be used to prevent writes to any of the memory chips installed on the board. This switch is located near the outside edge of the board for easy access. The switch is positioned up away from the board for write enable, down toward the board for write protect.

The RAM may also be software write-protected by setting bit 3 of the Configuration Register to logic 0. Bit 3 is logic 0 at power-on or reset time. Both SW2 and bit 3 must be enabled to allow writing to the RAM. If either is in protect mode, the RAM will be protected.

MEMORY SPEED REQUIREMENTS

Table 2-1 shows the basic chip access time requirements for static RAMs (SRAMs), EPROMs or EEPROMs. See page 3-6 for more information about wait states.

Table 2-1
ZT 8825 Basic Access Time Requirements.

	0 WAIT STATES				1 WAIT STATE			
TIMING	ZT 8825		ZT 88CT25		ZT 8825		ZT 88CT25	
	5 MHz	8 MHz	5 MHz	8 MHz	5 MHz	8 MHz	5 MHz	8 MHz
Tce	290	120	255	80	490	240	455	205
Taa	315	140	295	130	515	265	495	255
Toe	190	80	190	80	390	205	390	205
Twp	160	95	135	70	360	220	335	195
Tdw	250	130	250	130	550	255	450	255

BOARD CONFIGURATIONS

The ZT 8825 incorporates several jumpers that can be set to select different board configurations. The default configuration includes 20-bit memory addressing, 16-bit I/O port addressing, 0 wait states on all memory accesses, no battery backup, high IOEXP for access to registers, and is configured for 128K SRAM chips in both rows. If the default configuration (shown in Figure 2-1 on page 2-8) meets your needs, you do not need to change any jumpers. An example configuration that substitutes 24-bit addressing but is otherwise set for the default options appears in Figure 2-2 on page 2-9.

Each configurable option is covered in detail in Chapter 5, "Configurable Options," to assist you to correctly set your board for non-default operation. Appendix A, "Jumper Configurations," outlines the function of each jumper.

Note: Even if you have purchased RAMs, EPROMs, a battery, or other memory devices from Ziatech, your ZT 8825 will still have the default jumper settings and must be reconfigured as appropriate for the type and speed of the memory chips you install.

We recommend that you document your configuration on the blank layout provided in Appendix A on page A-6. This will allow you to easily restore the configuration if it is changed for any reason.

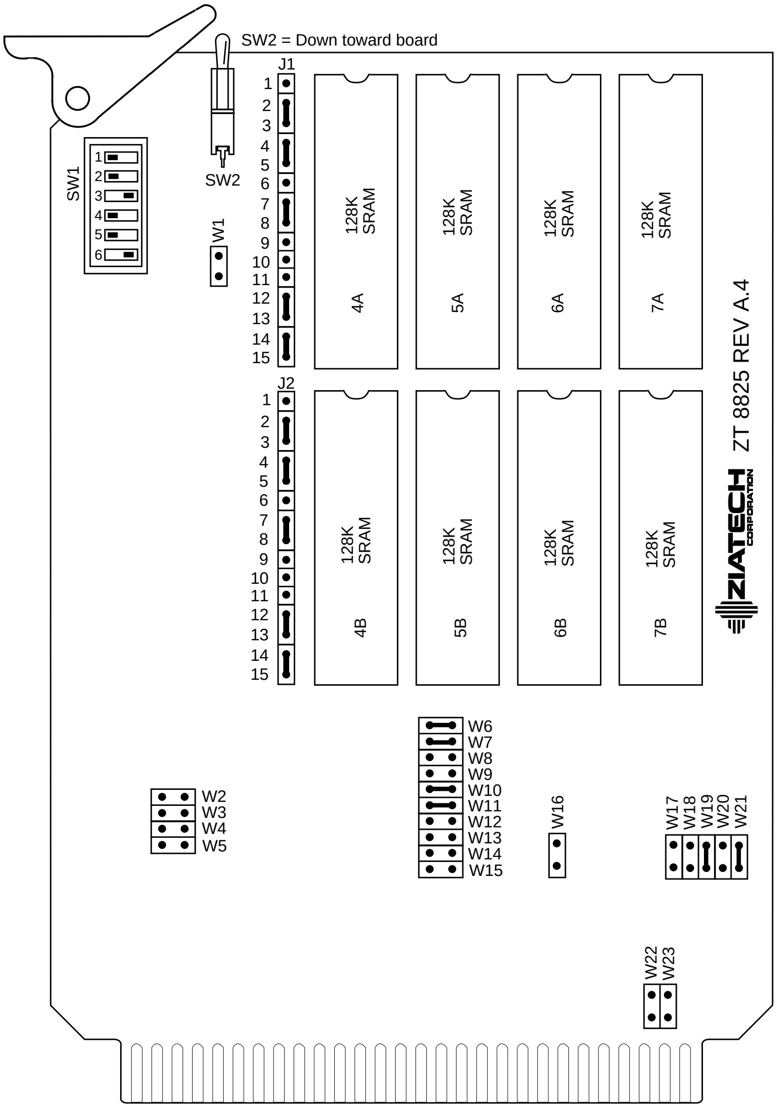


Figure 2–1. Factory Default Jumper Configuration.

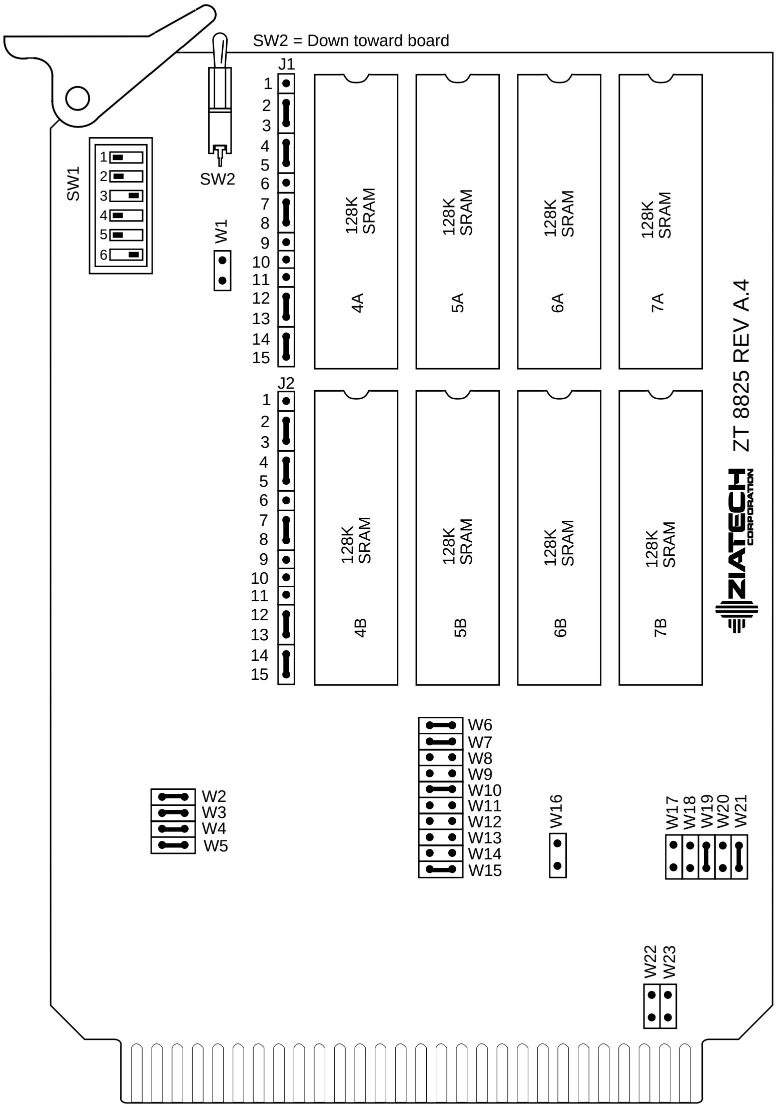


Figure 2–2. Example Setting for 24-bit Addressing.

CHIP SIZE SELECTION

Six-segment switch SW1 must be set according to the size of memory chips installed. Segments 6, 5, and 4 correspond to Row B chip size while segments 3, 2, and 1 correspond to Row A. See Table 2-2 on page 2-12 and Table 2-3 on page 2-13 for allowable settings.

WARNING!

Memory chips **must** *be properly oriented to avoid damage. Pin 1 must be installed in the upper left hand corner of the socket, as indicated in Figure 2-3. Note that some hybrid chips contain labeling oriented differently from what you might expect. Some chips have half moons on both ends. Because of this, extra care should be taken to identify pin 1.*

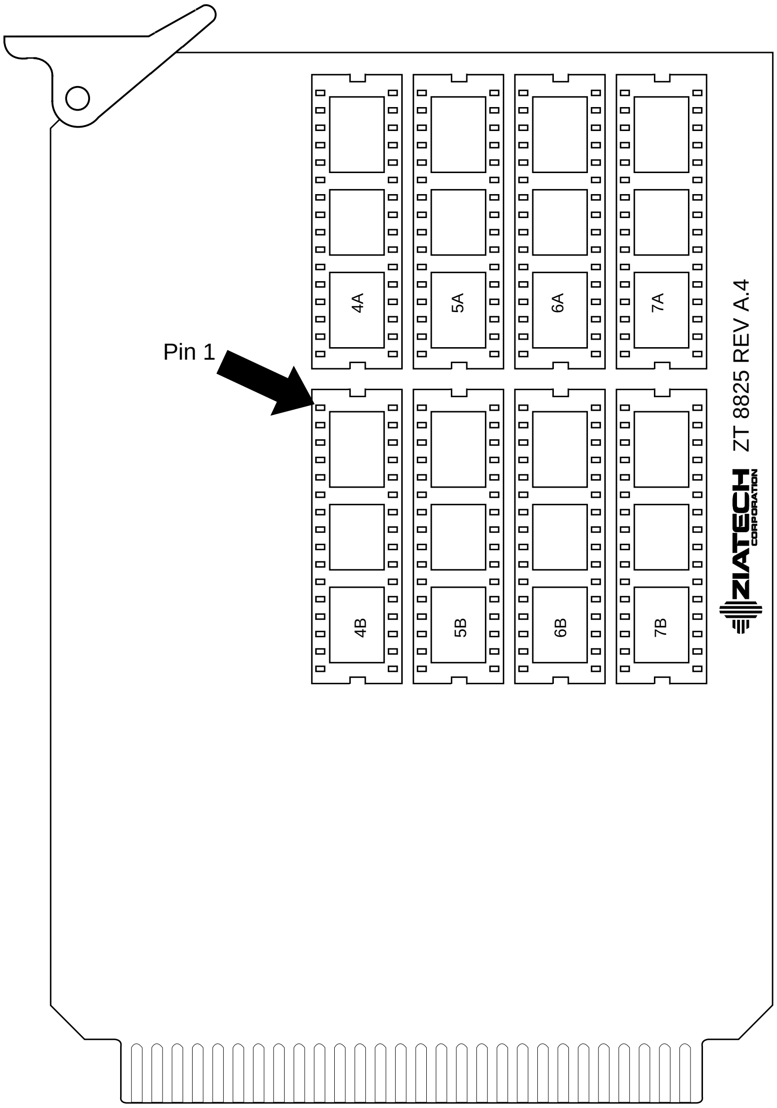


Figure 2–3. Memory Chip Orientation.

Table 2-2
Chip Size Selection.

Size	Switch section† 321 (or 654)	
16K	000	
32K	001	
64K	010	
128K	011	
256K	100	
512K	101	
1M	110	
512K	111	special - two chips in row A

† Switch section setting of 0 = ON, 1 = OFF.

Table 2-3
Valid Chip Combinations.

ROW A		ROW B	
Size	SW 321	Size	SW 654
16K †	000	16K †	000
32K †	001	32K †	001
64K †	010	32K †	001
64K †	010	64K †	010
128K †	011	32K †	001
128K †	011	128K †	011
256K †	100	128K †	011
256K †	100	256K †	100
512K †	101	—	xxx
1M **	110	—	xxx
512K *	111	256K †	100
512K *	111	512K *	101

* 2 chips (sockets 4 and 5) max; e.g., use Row A for RAM and Row B for PROM.

** 2 chips (sockets 4 and 5) only, due to 2 Mbyte board limit.

† 1, 2, 3, or 4 chips per row are acceptable.

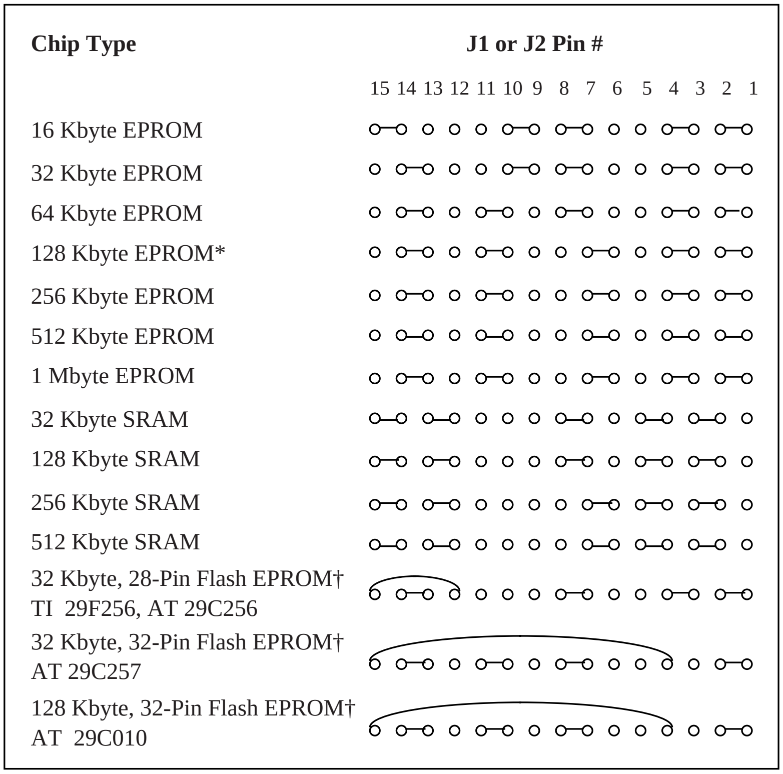
Notes: Any empty sockets must be accounted for by the software when it accesses the physical pages represented by the empty sockets.

Switch setting of 0 = ON, 1 = OFF. XXX = Don’t care; switch bits not used.

CHIP TYPE SELECTION

All chips in each row must be of the same size and type. Jumper sets J1 and J2 define the configuration for memory chip Rows A and B, respectively. The configurations shown in Table 2-4 cover all chip types known at publication time.

Table 2-4
Chip Type Jumper Configurations.



Chip Type	J1 or J2 Pin #
	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
16 Kbyte EPROM	○—○ ○ ○ ○ ○—○ ○—○ ○ ○ ○—○ ○—○
32 Kbyte EPROM	○ ○—○ ○ ○ ○—○ ○—○ ○ ○ ○—○ ○—○
64 Kbyte EPROM	○ ○—○ ○ ○—○ ○ ○—○ ○ ○ ○—○ ○—○
128 Kbyte EPROM*	○ ○—○ ○ ○—○ ○ ○ ○—○ ○ ○—○ ○—○
256 Kbyte EPROM	○ ○—○ ○ ○—○ ○ ○ ○—○ ○ ○—○ ○—○
512 Kbyte EPROM	○ ○—○ ○ ○—○ ○ ○ ○—○ ○ ○—○ ○—○
1 Mbyte EPROM	○ ○—○ ○ ○—○ ○ ○ ○—○ ○ ○—○ ○—○
32 Kbyte SRAM	○—○ ○—○ ○ ○ ○ ○—○ ○ ○—○ ○—○ ○
128 Kbyte SRAM	○—○ ○—○ ○ ○ ○ ○—○ ○ ○—○ ○—○ ○
256 Kbyte SRAM	○—○ ○—○ ○ ○ ○ ○ ○—○ ○—○ ○—○ ○
512 Kbyte SRAM	○—○ ○—○ ○ ○ ○ ○ ○—○ ○—○ ○—○ ○
32 Kbyte, 28-Pin Flash EPROM† TI 29F256, AT 29C256	○ ○—○ ○ ○ ○ ○ ○—○ ○ ○ ○—○ ○—○
32 Kbyte, 32-Pin Flash EPROM† AT 29C257	○ ○—○ ○ ○—○ ○ ○—○ ○ ○ ○ ○ ○—○
128 Kbyte, 32-Pin Flash EPROM† AT 29C010	○ ○—○ ○ ○—○ ○ ○—○ ○ ○ ○ ○ ○—○

Note: ○—○ indicates a jumper is installed between those two posts.

* The jumper configuration shown is for Intel and AMD 128K x 8 EPROMS only. For all other I28K x 8 manufacturers, please remove the jumpers between pins 1-2 and 3-4. Wire-wrap pin J to pin 8 and pin 4 to pin 9. This applies +5V to EPROM pins 1 and 31 (Vpp and /pgm) to meet non-Intel/AMD requirements.

† Flash EPROMs require special software routines for programming. See ZT 8825 Software Support Package documentation.

I/O ADDRESSING

The ZT 8825 has four I/O registers that are used to configure the board (see Table 2-5. These registers require jumpers to set the base I/O address (see Table 2-6 on page 2-17). The default base address is 0EE68h.

Table 2-5
ZT 8825 I/O Registers.

Base+	Default	Description
000	EE68h	Low order six bits of map register address (write only)
001	EE69h	High order four bits of map register address (write only)
010	EE6Ah	Data to/from map register previously addressed (read/write)
011	EE6Bh	Configuration register (write only, reset to 0 at power-on)

Table 2-6
Default I/O Address Assignments.

Address Line	Default Condition	Default Base Address (Binary)	Default Base Address (Hexadecimal)
A15	W14 out	1	E
A14	W13 out	1	
A13	W12 out	1	
A12	W11 in	0	
A11	1 hardwired	1	E
A10	1 hardwired	1	
A9	1 hardwired	1	
A8	0 hardwired	0	
A7	W10 in	0	6
A6	W9 out	1	
A5	W8 out	1	
A4	W7 in	0	
A3	1 hardwired	1	8
A2	-	0	
A1	-	0	
A0	-	0	

III. USER'S REFERENCE

THEORY OF OPERATION 3-1

APPLICATION EXAMPLES 4-1

CONFIGURABLE OPTIONS 5-1

Chapter 3

THEORY OF OPERATION

Contents **Page**

OVERVIEW 3-2
MEMORY CHIP TYPE 3-4
MEMORY CHIP SIZE 3-4
WAIT STATES 3-6
WRITE PROTECTION 3-6
BATTERY BACKUP 3-7
I/O ADDRESSING 3-10
MEMORY ADDRESSING 3-13
20-bit Memory Addressing 3-13
24-bit Memory Addressing 3-13
MEMORY MAPPING 3-14
Paging 3-14
Map Registers 3-15
Configuration Register 3-17
Setting the Map Register 3-18
Absolute Address Mode 3-19
EXTENDED MEMORY MODE 3-20
Setting the 24-bit Map Register 3-20

OVERVIEW

This chapter presents a high-level look at the way the ZT 8825 functions. It is designed to help you become more familiar with the board. Figure 3-1 illustrates the functional relationship between the key components on the board. Refer to Figure 3-1 as you read this chapter.

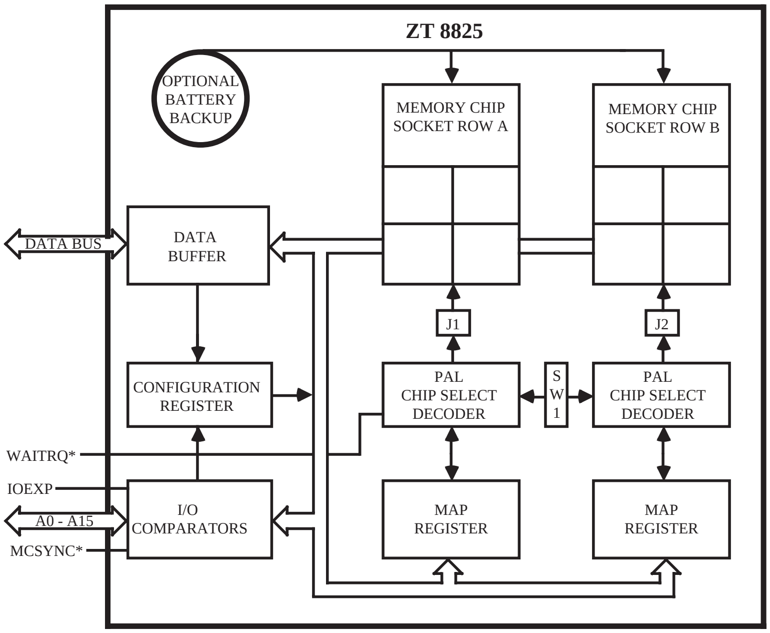


Figure 3–1. Functional Block Diagram.

The ZT 8825 is a state-of-the-art STD bus compatible memory board that accommodates most popular byte-wide memory chips. It is electrically, physically, and logically compatible with the STD bus as defined by the *STD-80 Series Bus Specification*. In addition, the CMOS version ZT 88CT25 is TTL compatible. Unless specifically noted, all references to ZT 8825 in this manual refer also to ZT 88CT25.

There are eight memory chip sockets on the ZT 8825. These sockets are divided into two rows of four. Each row can be independently configured for memory chip type (for example, EPROM, EEPROM, static RAM).

The ZT 8825 uses the STD-80 Series 20-bit memory address protocol. It can also accommodate 24-bit addressing. The user sees the board as many separate 16 Kbyte memory boards. In addition to implementing the normal 1 Mbyte STD-80 Series address space, the ZT 8825 allows access to Extended Memory address space up to 16 Mbytes. Also implemented are the *Lotus-Intel-Microsoft Expanded Memory Specification* (EMS), Versions 3.2 and 4.0, and the *AST Enhanced Expanded Memory Specification* (EEMS), Version 3.2, allowing access to up to 2 Mbytes of memory on the board. Multiple boards may be used to achieve up to 32 Mbytes of memory.

MEMORY CHIP TYPE

The ZT 8825 accepts JEDEC-compatible byte-wide memory devices of either 28 or 32 pins. All chips in each row of four must be of the same type (for example, all static RAM in Row A, all EPROM in Row B) and the same size (all 64K in Row A, all 32K in Row B).

Each row can be selected to be ROM, EPROM, EEPROM, or Flash or static RAM. Jumper set J1 selects the chip type for Row A and jumper set J2 selects the chip type for Row B. See page 5-3 for more information on configuring the board for memory chip type.

MEMORY CHIP SIZE

Switch 1 is a six-segment switch that provides a signal to a pre-programmed PAL indicating the chip size used in each row. This allows the PAL to decode the appropriate address bits for the chips used. Switch 1 segments 6, 5, and 4 select the chip size for Row B while switch 1 segments 3, 2, and 1 select the chip size for Row A. Table 3-1 shows the allowable chip size combinations for Row A and Row B along with the corresponding settings for switch 1. Refer to page 5-6 for more information on configuring your board for memory chip size.

Table 3-1
Chip Size Combinations.

Row A		Row B		Switch 1 654321
16K	†	16K	†	000000
32K	†	32K	†	001001
64K	†	32K	†	001010
64K	†	64K	†	010010
128K	†	32K	†	001011
128K	†	128K	†	011011
256K	†	128K	†	011100
256K	†	256K	†	100100
512K	†	—		xxx101
1M	**	—		xxx110
512K	*	256K	†	100111
512K	*	512K	*	101111

* 2 chips (sockets 4 and 5) max; e.g., use Row A for RAM and Row B for PROM.

** 2 chips (sockets 4 and 5) only, due to 2 Mbyte board limit.

† 1, 2, 3, or 4 chips per row are acceptable.

Notes: Any empty sockets must be accounted for by the software when it accesses the physical pages represented by the empty sockets.

Switch setting of 0 = ON, 1 = OFF. XXX = Don’t care; switch bits not used.

WAIT STATES

You may select either zero wait states or one wait state for the board. If one wait state is selected, the ZT 8825 requests a wait state only for 16 Kbyte blocks of memory that you have programmed as enabled. All chips on the board must use the same number of wait states; the slowest chip, therefore, determines the speed of the board.

Generally speaking, devices with a maximum access time of 120 ns will work on the ZT 8825 with no wait states when used with an 8 MHz STD-80 Series CPU board, such as the Ziatech ZT 8809. For 5 MHz STD-80 Series CPUs (for example, with a ZT 8808), access times of 290 ns or less will need no wait states.

For no wait states, the ZT 88CT25 requires devices of < 250 ns for 5 MHz STD-80 Series CPUs and ≤ 80 ns for 8 MHz STD-80 Series CPUs. Note that when a ZT 88CT25 is used with a ZT 8817 or any other CPU that requires WAITRQ* to be asserted by mid T2, the CPU *must* insert the wait state. The ZT 88CT25 wait state generator will not be fast enough at 8 MHz to meet the ZT 8817 requirement. Access time requirements are listed in the table on page 5-8. Consult your CPU board manual to answer any questions.

WRITE PROTECTION

The ZT 8825 has a write-protect switch (SW2) to assist in program debugging. When this toggle switch is positioned down (default) toward the board, the write signal is inhibited, preventing loss of data should a program run wild. This switch is located at the top of the board for easy access.

Write protection can also be controlled through software by manipulating bit 3 of the Configuration Register. Setting bit 3 to a logical 0 will inhibit writing to the memory (refer to Figure 3-4 on page 3-17). This bit is reset to zero at power-on time and by a system reset. SW2 must be enabled *and* Configuration Register bit 3 must be a logical 1 to write to the memory.

BATTERY BACKUP

An optional battery backup is available as an on-board 1 Amp-hour lithium battery. Either or both of the rows of four sockets may be battery backed. The lithium battery has a long shelf life and will provide years of backup for low power static RAMs (SRAMs) such as the Hitachi HM62256LP-15SL. For even longer back-up, use all-CMOS SRAMs such as Toshiba TC55256 PL-12, NEC µPD44256-12L or Sony CXK58255P-12L which have a 1 µA data retention current. See Table 3-2 for the estimated life of batteries used with the ZT 8825 with various SRAM chips installed.

Table 3-2
Est. Battery Life @ Max. Standby Current.

CHIPS INSTALLED	CURRENT DEMAND Max. (µA)	ESTIMATED BATTERY LIFE (HOURS) 0˚C to +65˚C
Eight 200 µA SRAMs	1640	610
Eight 50 µA SRAMs	440	2,273
Eight 10 µA SRAMs	120	8,333
Eight 1 µA SRAMs	48	20,833

Note: It is highly unlikely that the worst case given in Table 3-2 will ever occur. Typical ZT 8825s with 512K of SRAM draw less than 5 µA at +25˚ C, which corresponds to more than 10 years (shelf life limited).

Convert the number of hours given in Table 3-2 to days, months, or years based on the number of hours that the main power supply is off (the battery is the source of power) per day, month, or year.

For example, a board with eight 10 µA RAMs installed is used (powered-on) 8 hours per day, 5 days per week at +25˚C. In this example, the battery acts as the power supply for the board for different amounts of time on weekdays than on weekends (16 hrs/day on weekdays, 24 hrs/day on weekends). Therefore, an average daily rate must first be calculated. Each week has:

$$24 \text{ hrs/day} \times 7 \text{ days/week} = 168 \text{ hrs/week}$$

Since the battery is not in use 40 hrs/week, the weekly rate of battery use is:

$$(168\text{-}40) \text{ hrs/week} = 128 \text{ hrs/week}$$

which gives a daily rate of:

$$\frac{128 \text{ hrs/week}}{7 \text{ days/week}} = 18.29 \text{ hrs/day}$$

This rate can be used to calculate the estimated battery life in days, weeks, months, or years, as illustrated below.

The estimated battery life in days is:

$$\frac{8333 \text{ hrs}}{18.29 \text{ hrs/day}} = 455.6 \text{ days}$$

in weeks:

$$\frac{8333 \text{ hrs}}{18.29 \text{ hrs/day} \times 7 \text{ days/week}} = 65.1 \text{ weeks}$$

in months:

$$\frac{\text{8333 hrs}}{\text{18.29 hrs/day x 30 days/month}} \quad = \quad \text{15.2 months}$$

in years:

$$\frac{\text{8333 hrs}}{\text{18.29 hrs/day x 365 days/year}} \quad = \quad \text{1.25 years}$$

Again, the actual battery life is likely to be much greater than this worst case estimate.

The ZT 8825 supplies a DCLOW signal for use with the on-board battery backup. DCLOW is asserted when the 5 V supply is below 4.50 V. The ZT 8825 uses this signal to disable any glitching on the write line or chip selects to the memory chips. This signal may optionally assert DCLOW* (STD bus pin 6, Jumper W22 IN) and/or PBRESET* (STD bus pin 48, Jumper W23 IN) on the backplane.

Refer to page 5-9 for more information on configuring your board for battery backup.

I/O ADDRESSING

An I/O address is chosen for the ZT 8825 to distinguish it from any other I/O board that exists on the STD bus. If more than one ZT 8825 is used in a system, each must have a unique address.

In addition, the two least significant bits (LSBs) of the I/O address define the location of the Map Registers and Configuration Register used on the ZT 8825 during implementation of EMS. See page 3-14 for more information about EMS and the ZT 8825 Software Support Package manual for EMS programming considerations.

Table 3-3
Default I/O Address Assignments.

Address Line	Default Condition	Default Base Address	Default (HEX)
A15	W14 out	1	E
A14	W13 out	1	
A13	W12 out	1	
A12	W11 in	0	
A11	1 hardwired	1	E
A10	1 hardwired	1	
A9	1 hardwired	1	
A8	0 hardwired	0	
A7	W10 in	0	6
A6	W9 out	1	
A5	W8 out	1	
A4	W7 in	0	
A3	1 hardwired	1	8
A2	-	0	
A1	-	0	
A0	-	0	

The I/O base address for the board may be placed at one of 256 locations in the 64 Kbyte I/O address space of the STD-80 Series bus. The base address can be defined by either 8 or 16 bits (removing Jumper W6 restricts it to 8 bits).

Address lines A15-A0 supply the 16 bits of I/O address to the ZT 8825 (see Table 3-3). A15-A12 are jumper configurable. A11-A8 are hardwired to 1110b. A7-A4 are jumper configurable. A3 is hardwired to a logical 1. A2 is not decoded. A1-A0 define the four unique addresses (000b, 001b, 010b, and 011b) used on the ZT 8825 (see Table 3-4).

Note: Since A2 is not decoded, four addresses (100b, 101b, 110b, and 111b) are redundantly mapped and should not be used.

Table 3-4
ZT 8825 I/O Registers.

Base+	Default	Description
000	EE68h	Low order six bits of Map Register address (write only)
001	EE69h	High order four bits of Map Register address (write only)
010	EE6Ah	Data to/from Map Register previously addressed (read/write)
011	EE6Bh	Configuration Register (write only, reset to 0 at power-on)

Address lines A15-A12 and A7-A4 are tied to Jumpers W14-W11 and W10-W7, respectively. Since the hardware restricts address lines A11-A8 to the value 1110b (Eh) and A7-A4 to the value 1000b (8h), this limits the choice of base addresses to XEY8h, where X and Y may be chosen by Jumpers W14-W11 and W10-W7, respectively. The default base address is 1110111001101XXXb or 0EE68h. See page 5-10 for more information on configuring I/O addresses.

The base address is decoded by two 8-bit comparators. IOEXP (Input/Output Expansion signal) is included in the decode of the I/O address. Its polarity is determined by W15 (default is high). The output of the base address comparator enables a preprogrammed PAL that provides a chip select for each read or write register on the board.

MEMORY ADDRESSING

The ZT 8825 is designed to operate with the STD-80 Series multiplexed memory address scheme. Therefore, while address bits A0-A15 have their own dedicated backplane pins, A16-A23 are multiplexed onto the data bus pins during T1 of a memory access. The MCSYNC* (machine cycle sync) signal is used to latch the addresses on the ZT 8825. The MEMEX (memory expansion) signal is not used by the ZT 8825.

20-bit Memory Addressing

The default memory addressing configuration (Jumpers W2-W5 out) is for 20-bit addressing. On-board pull-down resistors on the internal addressing lines A20-A23 make the Map Registers believe that these upper four address bits are zeros. That is, the 1 Mbyte address space is at the bottom of the 16 Mbyte Extended Memory address space. 20-bit addressing allows for 64 Map Registers (see page 3-15 for more information on Map Registers).

24-bit Memory Addressing

Even though the ZT 8825 was designed to work explicitly with the multiplexed 20-bit memory address scheme of the *STD-80 Series Bus Specification*, it will work with a 24-bit address STD bus (for example, 80286 or similar processors). 24-bit addressing allows for 1,024 Map Registers or 16 Mbytes of address space. Memory between 1 Mbyte and 16 Mbytes is known as Extended Memory. Install Jumpers W2-5 to configure the ZT 8825 for 24 bit addressing.

For more information on configuring your board for memory addressing, refer to page 5-12.

MEMORY MAPPING

This section describes memory mapping for EMS and non-EMS applications. The ZT 8825 supports the *Lotus-Intel-Microsoft Expanded Memory Specification* (EMS), Versions 3.2 and 4.0. It also satisfies the *AST Enhanced Expanded Memory Specification* (EEMS), Version 3.2. The ZT 8825 software drivers implement a subset of those specifications. The EMS Version 3.2 is a subset of the EEMS capabilities. The ZT 8825 can also be set in an "absolute address mode" for easy startup in small systems. True 24-bit address memory anywhere in the 16 Mbyte Extended Memory space is also supported.

Paging

The memory resident on the ZT 8825 can be located outside the normal memory map of the PC when the ZT 8825 is used as an expanded memory board. Using a technique called "paging", the CPU can access 16 Kbyte blocks of physical memory called "pages" through 16 Kbyte blocks of logical memory called "page frames" (see Figure 3-2). Each page frame's logical address points to a page's physical address in expanded memory. That page of memory is then mapped in and out of the page frame space. This mapping may be changed dynamically by your software as required.

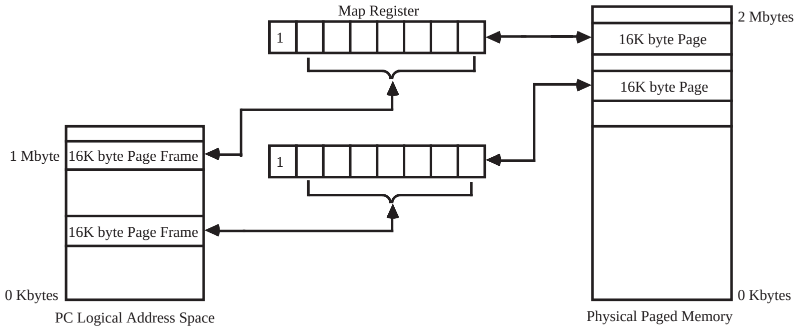


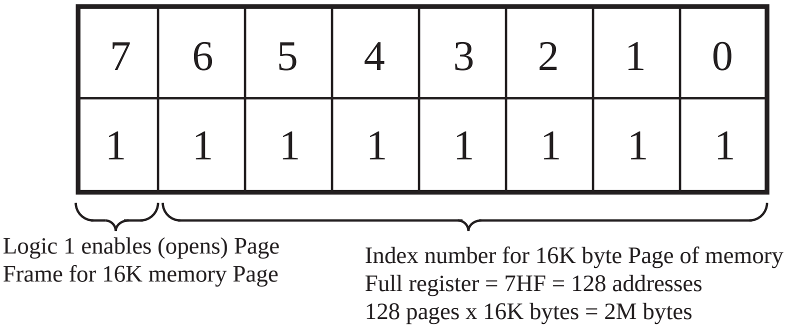
Figure 3–2. Logical to Physical Address Mapping.

Map Registers

Each page frame in the logical address space has an associated Map Register on the ZT 8825. 64 Map Registers are used in 20-bit address systems. 1,024 Map Registers are available in 24-bit address systems. In addition, a complete alternate set of Map Registers can be loaded at the same time as the primary set and switched in under software control (see "Setting the Map Register" on page 3-18). The physical registers are composed of two 4K x 4 SRAMs with 35 ns access times. Memory address bits A14-A19 (A14-A23 for 24-bit systems) form the index number of the Map Register for each 16K block.

Each 8-bit wide Map Register points to a 16 Kbyte physical page of memory that is to occupy the page frame currently associated with the Map Register. The most significant bit (MSB) of each register is used as an enable bit for the page frame (see Figure 3-3). If the enable bit is a logical 0, the board does not supply physical memory for that particular 16 Kbyte page frame. If the enable bit is a logical 1, the remaining 7 bits become the index number of that 16 Kbyte page. Once this index number is assigned to the page, the page frame is free to associate with another page.

In this manner, more than 1 Mbyte of physical memory can be mapped into the 1 Mbyte logical memory space. Since 7 bits can point to 128 pages of 16 Kbytes each, the maximum on-board memory is limited to 2 Mbytes (16K x 128). Other ZT 8825s may be used to provide additional pages of memory if the chip size limits the memory to less than 2 Mbytes on one board, or if more than 2 Mbytes is required.



7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1

Logic 1 enables (opens) Page
Frame for 16K memory Page

Index number for 16K byte Page of memory
Full register = 7HF = 128 addresses
128 pages x 16K bytes = 2M bytes

Figure 3–3. Map Register.

Configuration Register

The Configuration Register allows software control of several board parameters associated with memory mapping. Figure 3-4 illustrates the Configuration Register bits that affect these parameters. The following is a description of the purpose of each Configuration Register bit:

Bit 0 Map Enable bit. Setting bit 0 to logical 1 turns on the board’s mapping functions. Bit 0 must be set to logical 0 when changing the Map Registers.

Bit 1 Absolute Address Disable bit. Use Bit 1 to choose between an "absolute address mode" (see page 3-19) and addressing modes using the mapping function. Set bit 1 to logical 0 to select absolute address mode.

Bit 2 Map Write Enable bit. Set bit 2 to logical 0 to protect the Map Register data from being overwritten. Bit 2 must be set to logical 1 to load the Map Registers.

Bit 3 RAM Write Enable bit. Use bit 3 to control write protection of RAM (see page 3-6) in software. Set bit 3 to logical 0 to protect RAM from being altered.

Bits 4 - 6 Undefined.

Bit 7 Alternate Map Register Enable bit. Set bit 7 to logical 1 to enable access to a complete alternate set of Map Registers.

7	6	5	4	3	2	1	0
Alternate Map Register Enable bit				RAM Write Enable bit	Map Write Enable bit	Absolute Address Disable bit	Map Enable bit

Figure 3–4. Configuration Register.

Setting the Map Register

Follow these steps to load the Map Register:

1. Disable the board by setting the Configuration Register Map Enable bit (bit 0) to logical 0.
2. Set the Configuration Register Map Write Enable bit (bit 2) to logical 1.
3. Set the Configuration Register Alternate Map bit (bit 7) to:

    a) logical 0 to load the primary Map Register set.

    b) logical 1 to load the Alternate Map Register set.
4. Put the upper four bits of the Map Register index number into the Map High Address Register. This register must be set to 0000 for 20-bit address systems and need not be changed once initially loaded (see page 3-20 for 24-bit addressing).
5. Put the lower six bits of the Map Register index number into the Map Low Address Register. Typically, the lowest index number will be entered and will then be incremented by software for each successive Map Register number.
6. Write the data to the Map Register (bit 8 = logical 1 to enable the Map Register, bits 7-0 = page number).
7. When all 64 (or 1,024) Map Registers have been loaded, reset the Configuration Register Map Write Enable bit (bit 2) to logical 0 to protect the registers from further changes.
8. Disable Absolute Addressing by setting Configuration Register bit 1 to logical 1.
9. Set the Configuration Register Alternate Map Register bit (bit 7) to:

    a) logical 0 to access the Primary Map Register set.

    b) logical 1 to access the Alternate Map Register set.

10. Normally, the Configuration Register RAM Write Enable bit (bit 3) will be set to logical 0 at this time.
11. Enable the board by setting the Configuration Register Map Enable bit (bit 0) to logical 1.

Absolute Address Mode

For ease of start-up in simple systems, the ZT 8825 can be configured to come up at power-on or reset time in an "absolute address mode." To enable absolute address mode:

1. Install Jumper W1.
2. Set Configuration Register bit 1 to logical 0.
3. Set Configuration Register bit 3 to logical 1.

The board behaves in this mode as though its memory is a linear block starting at address 0 and continuing to the upper limit of memory on the board.

This is useful for small systems where the ZT 8825 is the only RAM board or in systems where it is used for RAM disks and for lower memory. Only one ZT 8825 in a system can use absolute address mode.

The ZT 8825 cannot be used in absolute address mode in DOS systems because the ZT 8825 powers up with its RAM write protected.

Non-DOS systems with boot ROMs can disable ZT 8825 write protection while executing custom code. To do this, the ROM code must output 08h to I/O location base+3 (default EE68h).

Absolute address mode can be turned off in software by setting the Configuration Register bit 1 to logical 1 after programming the Map Registers as described above. It may be permanently disabled by removing Jumper W1.

EXTENDED MEMORY MODE

The ZT 8825 can be used as normal memory anywhere in the 16 Mbyte address space by using 24-bit addressing. Memory is mapped in 16 Kbyte blocks, allowing it to be distributed over 16 Mbytes of address space.

Setting the 24-bit Map Register

To set the Map Register for 24-bit addressing, after installing Jumpers W2-W5:

1. Disable the board by setting the Configuration Register Map Enable bit (bit 0) to logical 0.
2. Set the Configuration Register Map Write Enable bit (bit 2) to logical 1.
3. Set the Configuration Register Alternate Map bit (bit 7) to:

   a) logical 0 to load the primary Map Register set.

   b) logical 1 to load the Alternate Map Register set.
4. Set the Map High Register to 0000.
5. Set the Map Low Register to 000000.
6. Allow software to increment the Map Low Register index number by one for each successive Map Register number until the Map Low Register has been filled (64 increments).
7. Allow software to increment the Map High Register by one and then reset the Map Low Register to 000000.
8. Repeat steps 6 and 7 until both low and high registers have filled (16 times).
9. When all 1,024 Map Registers have been loaded, reset the Configuration Register Map Write Enable bit (bit 2) to logical 0 to protect the map data.

10. Set the Configuration Register Alternate Map Register bit (bit 7) to:

    a) logical 0 to access the Primary Map Register set.

    b) logical 1 to access the Alternate Map Register set.

11. Enable the board by setting Configuration Register bit 0 to logical 1.

Chapter 4

APPLICATION EXAMPLES

Contents **Page**

OVERVIEW 4-1
INITIALIZING ZT 8825 AS CONTINUOUS MEMORY 4-1
Assembly Language Setup 4-2
Microsoft C Setup 4-4
EXAMPLE PROGRAMS USING EMS 4-7
Example 1 4-7
Example 2 4-12
Example 3 4-21
Example 4 4-23

OVERVIEW

This chapter includes specific examples of the ZT 8825 in operation.

INITIALIZING ZT 8825 AS CONTINUOUS MEMORY

The examples in this section demonstrate initialization of the ZT 8825 as a linear memory board.

Assembly Language Setup

The following example is written in Assembly language. The ZT 8825 is loaded with 8 x 32K RAM. The 20-bit memory address starts at 8000:000 (segment 8000 or 512K).

```
;MEMORY EQUATES
 START_MAP   EQU   20H         ;=SEGMENT 8000H SHIFTED RIGHT 10 TIMES
 SIZE        EQU   16          ;16 X 16K = 256K
;8825 I/O ADDRESSES
 LO_ADDR     EQU   0EE68H
 HI_ADDR     EQU   0EE69H
 DATA        EQU   0EE6AH
 CONFIG      EQU   0EE6BH
;CONFIG REGISTER EQUATES
 ALT_REG     EQU   80H
 VALID       EQU   80H         ;VALID PAGE
 EN_MAP_WR   EQU   0EH
 EN_BOARD    EQU   0BH

 MOV    DX, CONFIG
 MOV    AL, EN_MAP_WR
 OUT    DX, AL                 ;ENABLE WRITES TO MAP REGISTER
 MOV    DX, HI_ADDR
 MOV    AL, 0
 OUT    DX, AL                 ;SELECT LOWER 64 MAP REGISTERS
 MOV    AL, 0                  ;START AT REGISTER 0
 MOV    CX, 64                 ;CLEAR ALL 64 MAP REGISTERS
 MOV    AH, 0                  ;NOT VALID PAGE

 CLEAR_LOOP:
 MOV    DX, LO_ADDR
 OUT    DX, AL                 ;SELECT NEXT MAP REGISTER
 XCHG   AH, AL
 MOV    DX, DATA
 OUT    DX, AL                 ;CLEAR IT TO 0
 XCHG   AH, AL
 INC    AL
 LOOP   CLEAR_LOOP

 MOV    CX, SIZE
 MOV    AL, START_MAP
 MOV    AH, VALID
 INIT_LOOP:
 MOV    DX, LO_ADDR
 OUT    DX, AL                 ;SELECT WINDOW
 MOV    DX, DATA
 XCHG   AH, AL
 OUT    DX, AL                 ;ENABLE PAGE
 XCHG   AH, AL
 INC    AL                     ;INCREMENT WINDOW NUMBER
 INC     AH                      ;INCREMENT PAGE NUMBER
 LOOP    INIT_LOOP               ;SELECT NEXT PAGE

 MOV     DX, CONFIG
 MOV     AL, EN_BOARD
 OUT     DX, AL                  ;TURN ON BOARD
```

Microsoft C Setup

```
/********************************************************************
 *                                                                  *
 *               Ziatech  8825  Setup  Utility                      *
 *               Copyright Ziatech Corp.  1988                      *
 *                                                                  *
 * Setup is a simple configuration utility for the ZT 8825. Its     *
 * only purpose is to allow you to map the ZT 8825 into a single    *
 * contiguous memory space in an STD DOS system. It is implemen-    *
 * ted using Microsoft C compiler Version 4.0. Any questions or     *
 * comments should be directed to Ziatech Technical Support at      *
 * (805) 541-0488.                                                  *
 ********************************************************************/

#include <stdio.h>

#define VERSION                 "BETA v0.1"

#define DEFAULT_IO_ADDR         0x0ee68
#define DEFAULT_BASE_SEG        0x08000
#define DEFAULT_PAGES           16

#define REG_LO_ADDR             0
#define REG_HI_ADDR             1
#define REG_DATA                2
#define REG_CONFIG              3

#define CONFIG_ENABLE_MAPPING   0x01
#define CONFIG_NO_ABS_ADDR      0x02
#define CONFIG_MAP_WRITE_ENABLE 0x04
#define CONFIG_RAM_WRITE_ENABLE 0x08
#define CONFIG_SELECT_ALT_REGS  0x80

#define MAP_PAGE_ENABLE         0x80

char  Copyright[50] =
"Copyright  Ziatech Corp. 1988 All Rights Reserved\0";

/******************************************************************/
main(argc,argv)
int argc;
char *argv[];
{
   unsigned int ioaddr, baseaddr, pages;

   printf("\n  Ziatech  ZT8825  Setup  Utility   %s\n%s\n\n",
   VERSION,Copyright);
   if (argc != 4) {
      getargs(&ioaddr, &baseaddr, &pages);
   } else {
      sscanf(argv[1],"%x",&ioaddr);
      sscanf(argv[2],"%x",&baseaddr);
      sscanf(argv[3],"%d",&pages);
      }
   printf("\nMapping ZT8825 at %x to %x:0 for %d pages (%dK).\n",
      ioaddr,baseaddr,pages,(pages*16));

   clear(ioaddr);
   setup(ioaddr,baseaddr,pages);
}
/*******************************************************************/
/*
   clear: zeros all mapping registers on the specified ZT8825.

 */
clear(ioaddr)
unsigned int ioaddr;
{
   int i;

   outp( (ioaddr + REG_CONFIG), CONFIG_MAP_WRITE_ENABLE);
   outp( (ioaddr+REG_HI_ADDR), 0);
   for (i=0; i<64; i++) {
      outp( (ioaddr+REG_LO_ADDR), i);
      outp( (ioaddr+REG_DATA), 0);
      }
}
/*******************************************************************/
/*
   setup: maps the ZT8825 as per the requested configuration.

 */
setup(ioaddr,baseaddr,pages)
unsigned int ioaddr,baseaddr,pages;
{
   int basemap,i;

   basemap = (baseaddr >> 10);
   outp( (ioaddr + REG_CONFIG), CONFIG_MAP_WRITE_ENABLE );
   outp( (ioaddr + REG_HI_ADDR), 0);

   for (i = 0; i < pages; i++) {
      outp( (ioaddr + REG_LO_ADDR), (basemap + i) );
      outp( (ioaddr + REG_DATA), (MAP_PAGE_ENABLE + i) );
   }
   outp( (ioaddr + REG_CONFIG), (CONFIG_ENABLE_MAPPING |
   CONFIG_NO_ABS_ADDR | CONFIG_RAM_WRITE_ENABLE) );
}
/*******************************************************************/
getargs(ioaddr, baseaddr, pages)
unsigned int *ioaddr, *baseaddr, *pages;
{
   char ch,Input[30];

   printf("\nI/O address of 8825 (in hex) [%x] : ",DEFAULT_IO_ADDR);
   gets(Input);
   if (strlen(Input) == 0) {
      *ioaddr = DEFAULT_IO_ADDR;
   } else {
      sscanf(Input,"%x",ioaddr);
      }
   printf("Start at memory segment (in hex) [%x] : ",DEFAULT_BASE_SEG);
   gets(Input);
   if (strlen(Input) == 0) {
      *baseaddr = DEFAULT_BASE_SEG;
   } else {
      sscanf(Input,"%x",baseaddr);
      }

   printf("Number of pages to initialize (in decimal) [%d] : ",
   DEFAULT_PAGES);
   gets(Input);
   if (strlen(Input) == 0) {
      *pages = DEFAULT_PAGES;
   } else {
      sscanf(Input,"%d",pages);
      }
}
/********************************************************************/
```

EXAMPLE PROGRAMS USING EMS

These example programs have been included in this manual as a convenient quick reference for EMS users. Further information on EMS can be found in the ZT 8825 Software Support Package manual.

Example 1

The following program was written using the Microsoft C compiler Version 3.0. EMM function calls are made with the int86 function found in the dos.h library. Use the following command line to create an executable program:

msc program,,program;

```
#include <dos.h>
#include <stdio.h>
#define EMM_INT                   0x67  /* EMM interrupt number */
#define GET_PAGE_FRAME            0x41  /* EMM get page frame */
                                        /* function number */
#define GET_UNALLOC_PAGE_COUNT    0x42  /* EMM get unallocated */
                                        /* page count */
                                        /* function number */
#define ALLOCATE_PAGES            0x43  /* EMM allocate pages */
                                        /* function number */
#define MAP_PAGES                 0x44  /* EMM map pages
                                        /* function number */
#define DEALLOCATE_PAGES          0x45  /* EMM deallocate pages */
                                        /* function number */
#define DEVICE_NAME_LENGTH        8     /* Length of a device
                                        /* name string */
#define TRUE                      1
#define FALSE                     0

union REGS input_regs, output_regs;
struct SREGS segment_regs;
int pf_addr;

/* ------------------------------------------------------------------ */
/* Routine to convert a segment:offset pair to a far ptr.             */
/* ------------------------------------------------------------------ */
char *build_ptr(segment,offset)

unsigned int               segment;
unsigned int               offset;
```

```
{
        char  *ptr;

        ptr = (char *) (((unsigned long)segment << 16) + offset);
        return (ptr);
}
/* ------------------------------------------------------------------ */
/* Function which determines whether EMM device driver is installed.  */
/* ------------------------------------------------------------------ */
        char     emm_installed()

{
        char *EMM_device_name = "EMMXXXX0";
        char *int_67_device_name_ptr;

        /* -------------------------------------------------------- */
        /* AH = DOS get interrupt vector function.                  */
        /* -------------------------------------------------------- */
        input_regs.h.ah = 0x35;

        /* -------------------------------------------------------- */
        /* AL = EMM interrupt vector number.                        */
        /* -------------------------------------------------------- */
        input_regs.h.al = EMM_INT;

        intdosx (&input_regs, &output_regs, &segment_regs);

        /* -------------------------------------------------------- */
        /* Upon return ES:0Ah points to location where device name  */
        /* should be.                                               */
        /* -------------------------------------------------------- */
        int_67_device_name_ptr = build_ptr(segment_regs.es,0xA);

        /* -------------------------------------------------------- */
        /* Compare memory with EMM device name.                     */
        /* -------------------------------------------------------- */
        if (memcmp(EMM_device_name, int_67_device_name_ptr,
                                       DEVICE_NAME_LENGTH) == 0)
                  return (TRUE);
        else
                  return (FALSE);
}
/* ------------------------------------------------------------------ */
/* Function which determines if there are enough unallocated expanded */
/* memory pages for the application.                                  */
/* ------------------------------------------------------------------ */
char enough_unallocated_pages (pages_needed)

        int  pages_needed;
{
        input_regs.h.ah = GET_UNALLOCATED_PAGE_COUNT;
        int86 (EMM_INT, &input_regs, &output_regs);
        if (output_regs.h.ah! = 0 || pages_needed > output_regs.x.bx)
                  return (FALSE);
        else
                  return (TRUE);
}
```

```
/* ------------------------------------------------------------------ */
/* Routine which allocates expanded memory pages and passes           */
/* back to the main EMM handle.                                       */
/* ------------------------------------------------------------------ */
char allocate_expanded_memory_pages (pages_needed, emm_handle_ptr)

        int         pages_needed;
        unsigned int *emm_handle_ptr;
{
        input_regs.h.ah = ALLOCATE_PAGES;
        input_regs.x.bx = pages_needed;
        int86 (EMM_INT, &input_regs, &output_regs);
        if (output_regs.h.ah ==0) {
                    *emm_handle_ptr = output_regs.x.dx;
                  return (TRUE);
        else
                  return (FALSE);
}

/* ------------------------------------------------------------------ */
/* Routine to map a logical page to a physical page.                  */
/* ------------------------------------------------------------------ */
char map_expanded_memory_pages (emm_handle, physical_page, logical_page)
        unsigned int emm_handle;
        int          physical_page;
        int          logical_page;
{
        input_regs.h.ah = MAP_PAGES;
        input_regs.h.al = physical_page;
        input_regs.x.bx = logical_page;
        input_regs.x.dx = emm_handle;
        int86 (EMM_INT, &input_regs, &output_regs);
        if (output_regs.h.ah == 0)
                     return (TRUE);
        else
                     return (FALSE);
}

/* ------------------------------------------------------------------ */
/* Routine which gets the page frame base address from EMM.           */
/* ------------------------------------------------------------------ */
char   get_page_frame_address (pf_ptr)

        char **pf_ptr;
{
        input_regs.h.ah = GET_PAGE_FRAME;
        int86 (EMM_INT, &input_regs, &output_regs);
        if (output_regs.h.ah ! = 0)       /* check EMM status */
                  return (FALSE);
        else
               *pf_ptr = build_ptr(output_regs.x.bx,0);
        return (TRUE);
}
```

```
/* ------------------------------------------------------------------ */
/* Routine to release all expanded memory pages allocated by an EMM   */
/* handle.                                                            */
/* ------------------------------------------------------------------ */
char deallocate_expanded_memory_pages (emm_handle)
        unsigned int                    emm_handle;
{
        input_regs.h.ah = DEALLOCATE_PAGES;
        input_regs.x.dx = emm_handle;
        int86 (EMM_INT, &input_regs, &output_regs);
        if (output_regs.h.ah == 0)
                  return (TRUE);
        else
                  return (FALSE);
}

main() {
        unsigned int emm_handle;
        char *pf_addr;
        int pages_needed;
        int physical_page;
        int logical_page;
        int           index;

        /* -------------------------------------------------------- */
        /* Determine if EMM is installed.                           */
        /* -------------------------------------------------------- */
        if (!emm_installed())
                        exit (1);

        /* -------------------------------------------------------- */
        /* Determine if enough expanded memory pages exist for      */
        /* application.                                             */
        /* -------------------------------------------------------- */
        pages_needed = 1;
        if (!enough_unallocated_pages (pages_needed))
                        exit (1);

        /* -------------------------------------------------------- */
        /* Allocate expanded memory pages.                          */
        /* -------------------------------------------------------- */
        if (!allocate_expanded_memory_pages (pages_needed, &emm_handle))
                        exit (1);

        /* -------------------------------------------------------- */
        /* Map in the required pages.                               */
        /* -------------------------------------------------------- */
        physical_page = 0;
        logical_page = 0;
        if (!map_expanded_memory_pages (emm_handle, physical_page,
                                           logical_page))
                        exit (1);
        /* -------------------------------------------------------- */
        /* Get expanded memory page frame address.                  */
        /* -------------------------------------------------------- */
        if (!get_page_frame_address (&pf_addr))
                        exit (1);
        /* -------------------------------------------------------- */
        /* Write to expanded memory.                                */
        /* -------------------------------------------------------- */
        for (index = 0; index < 0x3fff; index++)
                        pf_addr[index] = index;

        /* -------------------------------------------------------- */
        /* Return expanded memory pages before exiting.             */
        /* -------------------------------------------------------- */
        if (!deallocate_expanded_memory_pages (emm_handle))
                        exit (1);
}
```

Example 2

The following program is an example of how to use the basic EMM functions with Turbo Pascal. The program performs the following steps:

1. Makes sure that the EMM has been installed
2. Displays the EMM version number
3. Determines whether there are enough pages of memory for the test program
4. Displays the total number of pages present in the system and the number of pages available for use
5. Requests the desired number of pages from the EMM
6. Maps a logical page into one of the logical memory page frames
7. Displays the base address of this EMM memory page frame
8. Performs a simple read/write test on the EMM memory
9. Returns the EMM memory back to the EMM
10. Exits

All the calls are structured for the result or error code of the EMM function performed to be returned as an integer. If the error code is not zero, an error has occurred. A simple error procedure is then called and the program terminates.

```
Type
  ST3  = string[3];
  ST80 = string[80];
  ST5  = string[5];

  Registers = record
    case integer of
      1: (AX,BX,CX,DX,BP,SI,DI,DS,ES,FLAGS: Integer);
      2: (AL,AH,BL,BH,CL,DL,DH            : Byte);
    end;

Const
  EMM_INT                   = $67;
  DOS_Int                   = $21;
  GET_PAGE_FRAME            = $41;
  GET_UNALLOCATED_PAGE_COUNT= $42;
  ALLOCATE_PAGES            = $43;
  MAP_PAGES                 = $44;
  DEALLOCATE_PAGES          = $45;
  GET_VERSION               = $46;
  STATUS_OK                 = 0;

  {-----------------------------------------------------------}
  { Assume the application needs one EMM page.                }
  {-----------------------------------------------------------}
  APPLICATION_PAGE_COUNT    = 1;

Var
  Regs: Registers;

  Emm_Handle,
  Page_Frame_Base_Address,
  Pages_Needed,
  Physical_Page,
  Logical_Page,
  Offset,
  Error_Code,
  Pages_EMM_Available,
  Total_EMM_Pages,
  Available_EMM_Pages: Integer;

  Version_Number,
  Pages_Number_String: ST3;
  Verify: Boolean;

  {-----------------------------------------------------------}
  { The function Hex_String converts an integer into a four   }
  { character hexadecimal number (string) with leading zeros. }
  {-----------------------------------------------------------}
  Function Hex_String (Number: Integer): ST5;
    Function Hex_Char (Number: Integer): Char;
      Begin
        If Number < 10 then
          Hex_Char := Char (Number+48)
        else
          Hex_Char := Char (Number+55);
      end; { Function Hex_Char}
  Var
    S: ST5;

  Begin
    S := ’’;
    S := Hex_Char ((Number shr 1) div 2048);
    Number:=(((Number shr 1) mod 2048) shl 1)+(Number and 1);
    S := S+Hex_Char (Number div 256);
    Number := Number mod 256;
    S := S+Hex_Char (Number div 16);
    Number := Number mod 16;
    S := S+Hex_Char (Number);
    Hex_String := S+’h’;
end; { Function Hex_String}
```

```
{------------------------------------------------------------}
{ The function Emm_Installed checks to see if the            }
{ EMM is loaded in memory.  It does this by looking          }
{ for the string 'EMMXXXX0', which should be located         }
{ at 10 bytes from the beginning of the code segment         }
{ the EMM interrupt, 67h, points to.                         }
{------------------------------------------------------------}
Function Emm_Installed: Boolean;
  Var
    Emm_Device_Name       : string[8];
    Int_67_Device_Name    : string[8];
    Position              : integer;
    Regs                  : registers;
  Begin
    Int_67_Device_Name := '';
    Emm_Device_Name    := 'EMMXXXX0';
    with regs do
      Begin
        {----------------------------------------------------}
        { Get the code segment interrupt 67h points to       }
        { the EMM interrupt by using DOS function 35h.       }
        { (get interrupt vector)                             }
        {----------------------------------------------------}
        AH := $35;
        AL := EMM_INT;
        Intr (DOS_int,Regs);
        {----------------------------------------------------}
        { The ES pseudo-register contains the segment        }
        { address pointed to by interrupt 67h.  Create an    }
        { eight character string from the eight successive  }
        { bytes at address ES:$000A (10 bytes from ES).      }
        {----------------------------------------------------}
        For Position := 0 to 7 do
          Int_67_Device_Name :=
             Int_67_Device_Name+Chr (mem[ES:Position+$0A]);
        Emm_Installed := True;
        {----------------------------------------------------}
        { If the string is the EMM manager signature,        }
        { 'EMMXXXX0', then EMM is installed and ready        }
        { for use.  If not, then EMM is not present.         }
        {----------------------------------------------------}
        If Int_67_Device_Name <> Emm_Device_Name
          then Emm_Installed := False;
      end; { with Regs do}
  end;  { Function Emm_Installed }

{------------------------------------------------------------}
{ This function returns the total number of EMM pages        }
{ present in the system, and the number of EMM pages         }
{ that are available.                                        }
{------------------------------------------------------------}
Function EMM_Pages_Available
  (Var Total_EMM_Pages, Pages_Available: Integer): Integer;
  Var
    Regs: Registers;
  Begin
    with Regs do
      Begin
```

```
        {---------------------------------------------------}
        { Get the number of currently unallocated pages and }
        { the total number of pages in the system from EMM. }
        { Load pseudo-registers prior to invoking EMM.      }
        {    AH = get unallocated page count function.      }
        {---------------------------------------------------}
        AH := Get_Unallocated_Page_Count;
        intr (EMM_INT, Regs);
        {---------------------------------------------------}
        { Unload the pseudo-registers after invoking EMM.   }
        {    BX = currently unallocated pages               }
        {    DX = total pages in the system                 }
        {    AH = status                                    }
        {---------------------------------------------------}
        Pages_Available := BX;
        Total_EMM_Pages := DX;
        EMM_Pages_Available := AH;
      end;
  end; { EMM_Pages_Available }

{-----------------------------------------------------------}
{ This function requests the specified number of pages      }
{ from the EMM.                                             }
{-----------------------------------------------------------}
Function Allocate_Expanded_Memory_Pages
  (Pages_Needed: Integer; Var Handle: Integer): Integer;
  Var
    Regs: Registers;

  Begin
    with Regs do
      Begin
        {---------------------------------------------------}
        { Allocate the specified number of pages from EMM.  }
        { Load pseudo-registers prior to invoking EMM.      }
        {    AH = allocate pages function.                  }
        {    BX = number of pages to allocate.              }
        {---------------------------------------------------}
        AH := Allocate_Pages;
        BX := Pages_Needed;
        intr (EMM_INT,Regs);
        {---------------------------------------------------}
        { Unload the pseudo-registers after invoking EMM.   }
        {    DX = EMM handle                                }
        {    AH = status                                    }
        {---------------------------------------------------}
        Handle := DX;
        Allocate_Expanded_Memory_Pages := AH;
      end;
  end; { Function Allocate_Expanded_Memory_Pages}
  {
{-----------------------------------------------------------}
{ This function maps a logical page allocated by the        }
{ Allocate_Expanded_Memory_Pages function into one of       }
{ the four physical pages.                                  }
{-----------------------------------------------------------}
Function Map_Expanded_Memory_Pages
  (Handle, Logical_Page, Physical_Page: Integer): Integer;
  Var
    Regs: Registers;
  Begin
    with Regs do
      Begin
        {---------------------------------------------------}
        { Map a logical page at physical page 0.            }
        { Load pseudo-registers prior to invoking EMM.      }
        {    AH = map page function                         }
        {    DX = handle                                    }
        {    BX = logical page number                       }
        {    AL = physical page number                      }
        {---------------------------------------------------}
        AH := Map_Pages;
        DX := Handle;
        BX := Logical_Page;
        AL := Physical_Page;
        Intr (EMM_INT, Regs);
        {---------------------------------------------------}
        { Unload the pseudo-registers after invoking EMM.   }
        {    AH = status                                    }
        {---------------------------------------------------}
        Map_Expanded_Memory_Pages := AH;
      end; { with Regs do}
  end; { Function Map_Expanded_Memory_Pages}

{-----------------------------------------------------------}
{ This function gets the physical address of the EMM page  }
{ frame we are using.  The address returned is the segment }
{ of the page frame.                                        }
{-----------------------------------------------------------}
Function Get_Page_Frame_Base_Address
  (Var Page_Frame_Address: Integer): Integer;
  Var
    Regs: Registers;
  Begin
    with Regs do
      Begin
        {---------------------------------------------------}
        { Get the page frame segment address from EMM.      }
        { Load pseudo-registers prior to invoking EMM.      }
        {    AH = get page frame segment function           }
        {---------------------------------------------------}
        AH := Get_Page_Frame;
        intr (EMM_INT,Regs);
        {---------------------------------------------------}
        { Unload the pseudo-registers after invoking EMM.   }
        {    BX = page frame segment address                }
        {    AH = status                                    }
        {---------------------------------------------------}
        Page_Frame_Address := BX;
        Get_Page_Frame_Base_Address := AH;
      end; { with Regs do }
  end; { Function Get_Page_Frame_Base_Address }
  {----------------------------------------------------------}
  { This function releases the EMM memory pages allocated to }
  { us, back to the EMM memory pool.                         }
  {----------------------------------------------------------}
  Function Deallocate_Expanded_Memory_Pages
    (Handle: Integer): Integer;
    Var
      Regs: Registers;
    Begin
      with Regs do
        Begin
          {--------------------------------------------------}
          { Deallocate the pages allocated to an EMM handle. }
          { Load pseudo-registers prior to invoking EMM.     }
          {    AH = deallocate pages function                }
          {    DX = EMM handle                               }
          {--------------------------------------------------}
          AH := DEALLOCATE_PAGES;
          DX := EMM_Handle;
          Intr (EMM_INT,Regs);
          {--------------------------------------------------}
          { Unload the pseudo-registers after invoking EMM.  }
          {    AH = status                                   }
          {--------------------------------------------------}
          Deallocate_Expanded_Memory_Pages := AH;
        end; { with Regs do}
    end;  { Function Deallocate_Expanded_Memory_Pages}

  {----------------------------------------------------------}
  { This function returns the version number of the EMM as   }
  { a three-character string.                                }
  {----------------------------------------------------------}
  Function Get_Version_Number (Var Version_String: ST3):
Integer;
    Var
      Regs: Registers;
      Integer_Part, Fractional_Part: Char;

    Begin
      with Regs do
        Begin
          {--------------------------------------------------}
          { Get the version of EMM.                          }
          { Load pseudo-registers prior to invoking EMM.     }
          {    AH = get EMM version function                 }
          {--------------------------------------------------}
          AH := GET_VERSION;
          Intr (EMM_INT,Regs);
          {--------------------------------------------------}
          { If the version number returned was OK, then      }
          { convert it to a three-character string.          }
          {--------------------------------------------------}
          If AH=STATUS_OK then
            Begin
```

```
              {-----------------------------------------------}
              { The upper four bits of AH are the integer     }
              { portion of the version number, the lower four }
              { bits are the fractional portion.  Convert the }
              { integer value to ASCII by adding 48.          }
              {-----------------------------------------------}
              Integer_Part    := Char (AL shr 4  + 48);
              Fractional_Part := Char (AL and $F + 48);
              Version_String  := Integer_Part + ’.’ +
Fractional_Part;
            end; { If AH=STATUS_OK}
          {---------------------------------------------------}
          { Unload the pseudo-registers after invoking EMM.   }
          {    AH = status                                    }
          {---------------------------------------------------}
          Get_Version_Number := AH;
        end; { with Regs do }
    end; { Function Get_Version_Number }
  {-----------------------------------------------------------}
  { This procedure prints an error message passed by the      }
  { caller, prints the error code passed by the caller in     }
  { hex, and then terminates the program with an error        }
  { level of 1.                                               }
  {-----------------------------------------------------------}
  Procedure Error (Error_Message: ST80; Error_Number: Integer);
    Begin
      Writeln (Error_Message);
      Writeln (’  Error_Number = ’,Hex_String (Error_Number));
      Writeln (’EMM test program aborting.’);
      Halt (1);
    end; { Procedure Error_Message}
{-------------------------------------------------------------}
{ This program is an example of the basic EMM functions that  }
{ you need in order to use EMM memory with Turbo Pascal.      }
{-------------------------------------------------------------}
Begin
  Clrscr;
  Window (5,2,77,22);
  {-----------------------------------------------------------}
  { Determine if the Expanded Memory Manager is installed; if }
  { not, then terminate ’main’ with an error level code of 1. }
  {-----------------------------------------------------------}
  If not (Emm_Installed) then
    Begin
      Writeln (’The LIM EMM is not installed.’);
      Halt (1);
    end
  else
    Begin
      { Get the version number and display it}
      Error_Code := Get_Version_Number (Version_Number);
      If Error_Code <> STATUS_OK then
        Error (’Error getting EMM version number ’, Error_code)
      else
        Writeln (’LIM Expanded Memory Manager, version ’,
                Version_Number,’ is ready for use.’);
    end;
  Writeln;
```

```
{-----------------------------------------------------------}
{ Determine if there are enough expanded memory pages for   }
{ this application.                                         }
{-----------------------------------------------------------}
Pages_Needed := APPLICATION_PAGE_COUNT;
Error_Code   := EMM_Pages_Available (Total_EMM_Pages,
                                     Available_EMM_Pages);
If Error_Code <> STATUS_OK then
  Error (’Error determining number of EMM pages available.’,
        Error_code);
Writeln (’There are a total of ’, Total_EMM_Pages,
         ’ expanded memory pages present in this system.’);
Writeln (’  ’, Available_EMM_Pages,
         ’ of those pages are available for use.’);
Writeln;

{-----------------------------------------------------------}
{ If there is an insufficient number of pages for the       }
{ application, then report the error and terminate the      }
{ EMM example program.                                      }
{-----------------------------------------------------------}
If Pages_Needed > Available_EMM_Pages then
  Begin
    Str (Pages_Needed, Pages_Number_String);
    Error (’We need ’ + Pages_Number_String+
           ’ EMM pages. There are not that many available.’,
           Error_Code);
  end; { Pages_Needed . Available_EMM_Pages }

{-----------------------------------------------------------}
{ Allocate expanded memory pages for our use.               }
{-----------------------------------------------------------}
Error_Code :=
  Allocate_Expanded_Memory_Pages (Pages_Needed, Emm_Handle);
Str (Pages_Needed,Pages_Number_String);
If Error_Code <> STATUS_OK then
  Error (’EMM test program failed trying to allocate ’
         + Pages_Number_String
         + ’ pages for usage.’,Error_Code);
Writeln (APPLICATION_PAGE_COUNT,
         ’ EMM page(s) allocated for the EMM test program.’);
Writeln;

{-----------------------------------------------------------}
{ Map in the required logical pages to the physical pages   }
{ given to us, in this case just one page.                  }
{-----------------------------------------------------------}
Logical_Page  := 0;
Physical_Page := 0;
Error_Code := Map_Expanded_Memory_Pages (EMM_Handle,
                                         Logical_Page,
                                         Physical_Page);
If Error_Code <> STATUS_OK then
  Error (’EMM test program failed trying to map ’
         + ’logical pages into physical pages.’,
         Error_Code);
Writeln (’Logical Page ’,
         Logical_Page,
           ' successfully mapped into Physical Page ',
           Physical_Page);
  Writeln;

  {-----------------------------------------------------------}
  { Get the expanded memory page frame address.               }
  {-----------------------------------------------------------}
  Error_Code := Get_Page_Frame_Base_Address
(Page_Frame_Base_Address);
  If Error_Code <> STATUS_OK then
    Error ('EMM test program unable to get the base page'
           + ' Frame Address.',
           Error_Code);
  Writeln ('The base address of the EMM page frame is = '
           + Hex_String (Page_Frame_Base_Address));
  Writeln;

  {-----------------------------------------------------------}
  { Write a test pattern to expanded memory.                  }
  {-----------------------------------------------------------}
  For Offset := 0 to 16382 do
    Begin
      Mem[Page_Frame_Base_Address:Offset mod 256;
    end;
  {-----------------------------------------------------------}
  { Make sure that what is in EMM memory is what was just     }
  { written.                                                  }
  {-----------------------------------------------------------}
  Writeln ('Testing EMM memory.');

  Offset := 1
  Verify := True;
  while (Offset <= 16382) and (Verify = True) do
    Begin
      If Mem[Page_Frame_Base_Address: Offset] <> Offsey mod 256
then
        Verify := False;
      Offset := Succ (Offset);
    end;  { while (Offset <= 16382) and (Verify = True) }

  {-----------------------------------------------------------}
  { If what is read does not match what was written,          }
  { an error occurred.                                        }
  {-----------------------------------------------------------}
  If not Verify then
    Error ('What was written to EMM memory was not found during
           + 'memory verification test.',
           0);
  Writeln ('EMM memory test successful.');
  Writeln;

  {-----------------------------------------------------------}
  { Return the expanded memory pages given to us back to the  }
  { EMM memory pool before terminating our test program.      }
  {-----------------------------------------------------------}
  Error_Code := Deallocate_Expanded_Memory_Pages (Emm_Handle);
  If Error_Code <> STATUS_OK then
    Error ('EMM test program was unable to deallocate '
           + 'the EMM pages in use.',
           Error_Code);
  Writeln (APPLICATION_PAGE_COUNT,
           ' page(s) deallocated.');
  Writeln;
  Writeln ('EMM test program completed.');

end.
```

Example 3

The following program is written in Microsoft's macro assembler.

```
                    CODE  SEGMENT
       ASSUME CS:CODE, DS:CODE

       MOV  AX, CS
       MOV  DS, AX
check_emm_installed:

   MOV   AH,35h                ; AH = DOS get interrupt vector function
   MOV   AL,67h                ; AL = EMM interrupt vector number
   INT   21h
   MOV   DI,0Ah                ; ES:DI points to where device name
                               ; should be
   LEA   SI,EMM_device_name    ; DS:SI points to ASCIIZ string
                               ; containing EMM device name

   MOV   CX,device_name_length ; set up loop counter for string op
   CLD                         ; set up direction flag for forward
   REPE  CMPSB                 ; Compare the strings
   JNE   exit                  ; IF strings are not equal THEN exit
                               ; ELSE

check_enough_unallocated_pages:
   MOV   AH,41h                ;      AH = EMM get unallocated page
                               ;      count function code
   INT   67h
   OR    AH,AH                 ; Check EMM status
   JNZ   emm_error_handler     ; IF emm_error THEN goto error handler
                               ; ELSE

allocate_expanded_memory_pages:
   MOV   AH,43h                ;      AH = EMM allocate pages
                               ;      function code
   MOV   BX,2                  ;      BX = number of pages needed
   INT   67h
   OR    AH,AH                 ; Check EMM status
   JNZ   emm_error_handler     ; IF emm_error THEN goto error handler
                               ; ELSE
   MOV   emm_handle,DX         ;      save EMM handle

map_expanded_memory_pages:

   MOV   AH,44h                ; AH = EMM map pages function
   MOV   DX,emm_handle         ; DX = application's handle
```

```
map_0_to_0:
   MOV   BX,0                   ; BX = logical page 0
   MOV   AL,0                   ; AL = physical page 0
   INT   67h
   OR    AH,AH                  ; Check EMM status
   JNZ   emm_error_handler      ; IF error THEN goto error handler
                                ; ELSE
get_page_frame_address:
   MOV   AH,41h                 ; AH = EMM get page frame base
                                ; address function
   INT   67h
   OR    AH,AH                  ; Check EMM status
   JNZ   emm_error_handler      ; IF error THEN goto error handler
   MOV   pf_addr,BX             ; ELSE save pf_addr

write_to_expanded_memory:       ; Write zeros to memory mapped at
                                ; physical page 0.
   MOV   AX,pf_addr
   MOV   ES,AX                  ; Set up ES to point to physical page 0
   MOV   DI,0                   ; DI used as index into physical page 0
   MOV   AL,0                   ; Initialize AL for string STOSB operation
   MOV   CX,4000h               ; Initialize loop counter to length
                                ; of expanded memory page size
   CLD                          ; Set direction forward for string op
   REP   STOSB

deallocate_pages:
   MOV   AH,45h                 ; AH = EMM deallocate pages function
   MOV   DX,emm_handle
   INT   67h                    ; return handle’s pages to EMM
   OR    AH,AH                  ; Check EMM status
   JNZ   emm_error_handler      ; IF error THEN goto error handler

exit:
   MOV   AH,4ch                 ; AH = DOS exit function
   INT   21h                    ; Return to DOS


EMM_device_name DB’EMMXXXX0"    ; ASCIIZ EMM device name string

device_name_length EQU 8
CODE     ENDS
  END
```

Example 4

The following program is an example of how to change a 256 Kbyte block of data from conventional memory to expanded memory.

```
                    CODE  SEGMENT
 ASSUME    CS:CODE,DS:CODE
xchg_packet_set_up:

;DS:SI = xchg_packet
   MOV      AX,SEG xchg_packet
   MOV      DS,AX
   MOV      SI,OFFSET xchg_packet

;Moving 256K of data from conventional memory to expanded memory
   MOV      WORD PTR [SI].region_length [0],0
   MOV      WORD PTR [SI].region_length [2],4
   MOV      [SI].src_mem_type, 0
   MOV      [SI].dest_mem_type, 1

;Starting at segment: 4000h, offset: 0
   MOV      [SI].src_init_seg_page, 4000h
   MOV      [SI].src_init_offset, 0

;Move data into expanded memory logical page 0, offset 0.
   MOV      [SI].dest_init_seg_page, 0
   MOV      [SI].dest_init_offset, 0

;Initialize for future compatibility
   MOV      [SI].src_handle, 0

;Need handle for expanded memory distination.
   MOV      DX,emm_handle
   MOV      [SI].dest_handle,DX

;AX = EMM Exchange Memory function
   MOV      AX,5F01h
   INT      67h
   OR       AH,AH
   JNZ      emm_error_handler

                                             xchg_struct STRUC
   region_length                          DD?
   src_mem_type                           DB?
   src_handle                             DW?
   src_init_offset                        DW?
   src_init_seg_page                      DW?
   dest_mem_TYPE                          DB?
   dest_handle                            DW?
   dest_init_offset                       DW?
   dest_init_seg_page                     DW?
xchg_struct                               ENDS

xchg_packet                               xchg_struct
CODE    ENDS
END
```

Chapter 5

CONFIGURABLE OPTIONS

Contents **Page**

OVERVIEW 5-2
MEMORY CHIP TYPE SELECTION 5-3
MEMORY CHIP SIZE SELECTION 5-6
WAIT STATE SELECTION 5-8
BATTERY BACKUP SELECTION 5-9
I/O BASE ADDRESS SELECTION 5-10
MEMORY ADDRESS SELECTION (20- VS. 24-BIT) 5-12

OVERVIEW

This chapter details the various jumper-selectable options on the ZT 8825. It provides specific information to allow you to correctly configure your board.

Since the ZT 8825 is a flexible board, several jumpers must be set to select the desired configuration. These jumpers can be divided into the following groups:

- Memory chip type selection
- Memory chip size selection
- Number of wait states
- Battery backup
- I/O port base address
- Memory addressing

The following sections describe these functions and the required jumper settings for their different modes of operation. Refer to Appendix A for a description of each jumper and its functions.

MEMORY CHIP TYPE SELECTION

This board typically will be used with EPROMs or SRAMs. Different jumper configurations are required for each chip size and type. All chips in each row of four must be of the same size and type. Jumpers are provided to select various signals for pins 1, 3, 29, 30, and 31 as is shown in Table 5-1.

Note: pin numbers are given for 32-pin chips/sockets; 28-pin devices are loaded with socket pins 1, 2, 30, and 31 empty.

Table 5-1
Signal Choices on Each Socket.

Pin #	Signal 1	Signal 2	Signal 3	Wire Wrap
1	A18	A19	—	Vcc
3	A14	A15	Vcc	
29	A14	$\overline{WE}$	—	
30	A17	Vcc	—	
31	A15	A18	—	Vcc or $\overline{WE}$

Jumper set J1 defines the configuration for memory chip Row A, and J2 defines the configuration for Row B. Not all jumper positions in J1 and J2 are valid (see Table 5-2). The example configurations shown in Table 5-3 cover all chip types known at publication time.

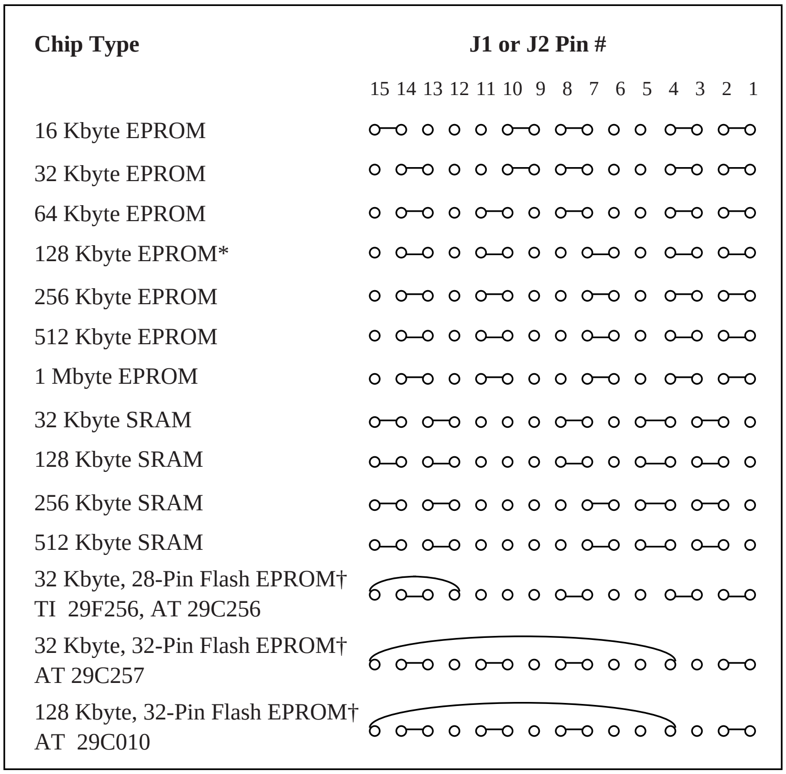
Table 5-2
Jumper Selection of Socket Signals.

J1 or J2 PIN†	Description
1 to 2	Connects A19 to EPROM pin 1
2 to 3	Connects A18 to SRAM pin 1
3 to 4	Connects A18 to EPROM pin 31
4 to 5	Connects A15 to SRAM pin 31
5 to 6	Invalid jumper position
6 to 7	Connects A17 to EPROM pin 30
7 to 8	Connects Vcc to chip pin 30
8 to 9	Invalid jumper position
9 to 10	Connects Vcc to Chip pin 3
10 to 11	Connects A15 to EPROM pin 3
11 to 12	Invalid jumper position
12 to 13	Connects A14 to SRAM pin 3
13 to 14	Connects A14 to EPROM pin 29
14 to 15	Connects $\overline{\text{WE}}$ to SRAM pin 29

† All pin numbers refer to 32-pin sockets.

Note: EEPROMS and Flash EPROMs must be +5 V only and must have latched addresses and data, as well as normal write time (see the table on page 5-8), for write operations to be performed correctly. Other chips on the ZT 8825 may be accessed while the EEPROM is carrying out its long internal write cycle. However, software must keep track, using delay loops or hardware counter/timers, of the interval required before the chip may be accessed again.

Table 5-3
Chip Type Jumper Configurations.



Chip Type	J1 or J2 Pin # 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
16 Kbyte EPROM	○─○ ○ ○ ○ ○─○ ○─○ ○ ○ ○─○ ○─○
32 Kbyte EPROM	○ ○─○ ○ ○ ○─○ ○─○ ○ ○ ○─○ ○─○
64 Kbyte EPROM	○ ○─○ ○ ○─○ ○ ○─○ ○ ○ ○─○ ○─○
128 Kbyte EPROM*	○ ○─○ ○ ○─○ ○ ○ ○─○ ○ ○─○ ○─○
256 Kbyte EPROM	○ ○─○ ○ ○─○ ○ ○ ○─○ ○ ○─○ ○─○
512 Kbyte EPROM	○ ○─○ ○ ○─○ ○ ○ ○─○ ○ ○─○ ○─○
1 Mbyte EPROM	○ ○─○ ○ ○─○ ○ ○ ○─○ ○ ○─○ ○─○
32 Kbyte SRAM	○─○ ○─○ ○ ○ ○ ○─○ ○ ○─○ ○─○ ○
128 Kbyte SRAM	○─○ ○─○ ○ ○ ○ ○─○ ○ ○─○ ○─○ ○
256 Kbyte SRAM	○─○ ○─○ ○ ○ ○ ○ ○─○ ○─○ ○─○ ○
512 Kbyte SRAM	○─○ ○─○ ○ ○ ○ ○ ○─○ ○─○ ○─○ ○
32 Kbyte, 28-Pin Flash EPROM† TI 29F256, AT 29C256	○ ○─○ ○ ○ ○ ○ ○─○ ○ ○ ○─○ ○─○
32 Kbyte, 32-Pin Flash EPROM† AT 29C257	○ ○─○ ○ ○─○ ○ ○─○ ○ ○ ○ ○ ○─○
128 Kbyte, 32-Pin Flash EPROM† AT 29C010	○ ○─○ ○ ○─○ ○ ○─○ ○ ○ ○ ○ ○─○

Note: ○─○ indicates a jumper is installed between those two posts.

* The jumper configuration shown is for Intel and AMD 128K x 8 EPROMS only. For all other I28K x 8 manufacturers, please remove the jumpers between pins 1-2 and 3-4. Wire-wrap pin J to pin 8 and pin 4 to pin 9. This applies +5V to EPROM pins 1 and 31 (Vpp and /pgm) to meet non-Intel/AMD requirements.

† Flash EPROMs require special software routines for programming. See ZT 8825 Software Support Package documentation.

MEMORY CHIP SIZE SELECTION

The previous section discussed the need to configure certain socket pins for different chip sizes and types. The chip size also determines which address lines must be used by the decoders to generate the eight socket/chip selects. Preprogrammed PALs are used on the ZT 8825 to decode the appropriate address bits based on the chip sizes used. The chip sizes are set on six-segment DIP switch 1 (SW1), which provides the inputs to the PALs. Switch 1 positions 3, 2, and 1 select the chip sizes for Row A. Switch 1 positions 6, 5, and 4 select the chip sizes for Row B. Table 5-4 provides the chip sizes that correspond to each switch setting.

Table 5-4
Switch Settings for All Possible Chip Sizes.

Switch					
Section	Sockets				Total Bytes
654321	B		A		On Board
000000	16K	†	16K	†	128K
001001	32K	†	32K	†	256K
001010	32K	†	64K	†	384K
010010	64K	†	64K	†	512K
001011	32K	†	128K	†	640K
011011	128K	†	128K	†	1M
011100	128K	†	256K	†	1.5M
100100	256K	†	256K	†	2M
xxx101	—		512K	†	2M
xxx110	—		1M	**	2M
100111	256K	†	512K	*	2M
101111	512K	*	512K	*	2M

* 2 chips (sockets 4 and 5) max; e.g., use Row A for RAM and Row B for PROM.

** 2 chips (sockets 4 and 5) only, due to 2 Mbyte board limit.

† 1, 2, 3, or 4 chips per row are acceptable.

Notes: Any empty sockets must be accounted for by the software when it accesses the physical pages represented by the empty sockets.

Switch setting of 0 = ON, 1 = OFF. XXX = Don’t care; switch bits not used.

It is not necessary to use all sockets in a row. Normally, you will begin loading sockets at the low end of the row desired (start with socket 4A for Row A or 4B for Row B).

WAIT STATE SELECTION

The memory access time required for no wait state operation depends on the speed and type of processor board as well as on the type of memory.

The ZT 8825 loaded with chips that have access times of less than 290 ns will operate with no wait states with any 5 MHz CPU that meets the *STD-80 Series Bus Specification*. The chips must have access times not greater than 130 ns for an 8 MHz CPU meeting the specifications (see Table 5-5).

Installing Jumper W16 selects one wait state for the board. This allows slower chips with longer access times to be used. You must select one wait state for the board if any chip on the board is slower than the times given. The factory default is no wait states (W16 OUT).

Table 5-5
ZT 8825 Basic Access Time Requirements.

	0 WAIT STATES				1 WAIT STATE			
TIMING	ZT 8825		ZT 88CT25		ZT 8825		ZT 88CT25	
	5 MHz	8 MHz	5 MHz	8 MHz	5 MHz	8 MHz	5 MHz	8 MHz
Tce	290	120	255	80	490	240	455	205
Taa	315	140	295	130	515	265	495	255
Toe	190	80	190	80	390	205	390	205
Twp	160	95	135	70	360	220	335	195
Tdw	250	130	250	130	550	255	450	255

BATTERY BACKUP SELECTION

A lithium battery can be loaded on the ZT 8825 PC board for use as backup voltage for CMOS RAMs. Ziatech offers a 1 Amp-hour battery as an option for the ZT 8825 (option 003). For maximum battery life and data retention time, the CMOS RAMs should be low-power types specifically designed for such applications (for example, Hitachi 62256LP-15SL).

On-board battery backup is intended **only** for use with 32K, 128K, or 512K static CMOS RAMs (such as the Hitachi 62256LP-12SL). A special Schottky diode with very low reverse leakage current (CR1) and a 100 ohm current limiting resistor are used to prevent charging the lithium battery.

Table 5-6 lists the jumper configurations for backup use for each socket row. Jumpers 18, 19, 20, and 21 select the Vss source for each row.

Table 5-6
Jumper Settings for Battery Backup.

TYPE OF BATTERY BACKUP	JUMPER NUMBER			
	18	19	20	21
No battery backup	OUT	IN	OUT	IN
On-board backup: Row A	OUT	IN	IN	OUT
On-board backup: Row B	IN	OUT	OUT	IN
On-board backup: Row A,B	IN	OUT	IN	OUT

When using on-board battery backup, the ZT 8825 uses a voltage comparator to detect when the Vcc supply is less than 4.5 V. When this is true, the ZT 8825 prevents any access to the SRAMs. This comparator can also drive the DCLOW* signal (pin 6, low true) to disable other boards to protect the RAM and/or EEPROM data. Installing Jumper W22 will assert DCLOW*. PBRESET* can be asserted by installing Jumper W23.

I/O BASE ADDRESS SELECTION

Before choosing the base addresses for the four registers, you must decide on 8-bit or 16-bit addressing. The concept is the same in both cases, except for the number of bits used to define the 16-bit base address.

You must choose a base address that does not conflict with any other I/O board that exists on the STD bus. Any overlap causes conflict on the data bus; it can produce invalid data and cause the data bus driver ICs to fail. We suggest you draw a complete I/O map of the system showing starting addresses and ranges for each board in the system before selecting the base address for the ZT 8825. This is especially true if 8-bit and 16-bit address I/O boards will exist on the same backplane. IOEXP is often used to resolve these difficulties. This signal is always used in determining the validity of an I/O access to this board. You must choose whether IOEXP must be HIGH (Jumper W15 OUT) or LOW (Jumper W15 IN) to allow access to the board.

For 8-bit I/O addresses, remove Jumper W6 to inhibit the upper 8 bits of address. For 16-bit I/O addresses, Jumper W6 must be IN. Only bits 15-12 and 7-4 may be changed in the I/O base address (see Table 5-7). Bits 11-8 are hardwired to 1110 (0Eh), and bit 3 is hardwired to logical 1. Bits 2-0 are used to decode the various registers on the ZT 8825. Jumpers W14-11 are used to set address bits 15-12, and Jumpers W10-7 are used to set address bits 7-4. If the jumper is OUT, the corresponding address bit must be a logical 1 in the base address to access the board. Similarly, if the jumper is IN, the address bit must be a logical 0.

Table 5-7
Jumpers for I/O Base Address Decoding.

I/O Address bits															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W14	W13	W12	W11	1	1	1	0	W10	W9	W8	W7	1	Register Number		

The default I/O base address on the ZT 8825 is 0EE68h (1110 1110 0110 1XXX).

MEMORY ADDRESS SELECTION (20- VS. 24-BIT)

The ZT 8825 is designed to use the multiplexed 20-bit memory addresses as specified by the STD-80 Series standard. The extra four address bits (A16 to A19) are multiplexed onto the data bus (bits 0 to 3) at the beginning of each cycle when MCSYNC* is asserted. This does not add any additional time overhead to the typical 8088 cycle. The MEMEX signal is not used on this board. For 24-bit memory addresses from 80286, 80386 & 68K processors, there are eight bits multiplexed (A16 to A23) in the same manner. Install Jumpers W2-5 for 24-bit addressing.

IV. APPENDICES

JUMPER CONFIGURATIONS A-1

SPECIFICATIONS B-1

HARDWARE QUICK REFERENCE C-1

CUSTOMER SUPPORT D-1

GLOSSARY E-1

Appendix A

JUMPER CONFIGURATIONS

Contents **Page**

OVERVIEW A-1
JUMPER DESCRIPTIONS A-2
USER JUMPER CONFIGURATION A-5
SWITCH CONFIGURATION A-5

OVERVIEW

The ZT 8825 includes several jumper options that tailor the operation of the board to the requirements of specific applications. Jumper designators are shown on the ZT 8825 where space permits and are illustrated in full in Figure A-1 on page A-6.

Table A-1 lists each jumper in numerical order and provides a functional description for each. A dagger (†) appearing in Table A-1 indicates a jumper that is installed for the factory default configuration. See page 2-8 for an illustration of the factory default configuration.

Refer to Chapter 5, "Configurable Options," for a detailed discussion of the jumper-configurable options on the ZT 8825.

JUMPER DESCRIPTIONS

Table A-1
ZT 8825 Jumper Descriptions.

JUMPER #	DESCRIPTION
W1	Enables absolute addressing mode for the board at power-on or reset time (see page 3-19).
W2-5	Enable 24-bit memory addressing mode (see page 3-13).
W6†	Enables 16-bit I/O port addressing for the on-board registers (see page 3-10).
W7†,8,9,10†	Set the value of I/O port address bits A4-A7 (see page 3-11).
W11†,12,13,14	Set the value of I/O port address bits A12-A15 (see page 3-11).
W15†	W15 OUT selects IOEXP high. W15 IN selects IOEXP low (see page 3-11).

† Installed for factory default configuration.

Table A-1
ZT 8825 Jumper Descriptions (continued).

JUMPER #	DESCRIPTION
W16††	Enables one wait state on all memory accesses (see page 3-6).
W17	Used for Ziatech test only; must not be installed.
W18	Selects battery backup mode for Row B chips (see page 3-7).
W19†	Selects non-battery backup mode for Row B chips (see page 3-7).
W20	Selects battery backup mode for Row A chips (see page 3-7).

†† May not be used on a ZT 88CT25 with a ZT 8817 CPU to insert one wait state for slower memory chips. See page 3-6.

†Installed for factory default configuration.

Table A-1
ZT 8825 Jumper Descriptions (continued).

JUMPER #	DESCRIPTION
W21†	Selects non-battery backup mode for Row A chips (see page 3-7).
W22	Enables board to drive DCLOW* when Vcc < 4.5 V (see page 3-7).
W23	Enables board to drive PBRESET* when Vcc < 4.5 V (see page 3-7).

†Installed for factory default configuration.

USER JUMPER CONFIGURATION

Figure A-1 illustrates jumper locations for the ZT 8825. You may also wish to document your board configuration here. This documentation will allow you to easily restore your configuration in case it is changed.

SWITCH CONFIGURATION

Table A-2 shows settings for six-segment DIP switch SW1 that correspond to available memory chip sizes. See page 5-6 for more information.

Table A-2
Switch Settings for All Possible Chip Sizes.

Switch Section	Sockets		Total Bytes On Board
654321	B	A	
000000	16K †	16K †	128K
001001	32K †	32K †	256K
001010	32K †	64K †	384K
010010	64K †	64K †	512K
001011	32K †	128K †	640K
011011	128K †	128K †	1M
011100	128K †	256K †	1.5M
100100	256K †	256K †	2M
xxx101	—	512K †	2M
xxx110	—	1M **	2M
100111	256K †	512K *	2M
101111	512K *	512K *	2M

* 2 chips (sockets 4 and 5) max; e.g., use Row A for RAM and Row B for PROM.

** 2 chips (sockets 4 and 5) only, due to 2 Mbyte board limit.

† 1, 2, 3, or 4 chips per row are acceptable.

Notes: Any empty sockets must be accounted for by the software when it accesses the physical pages represented by the empty sockets.

Switch setting of 0 = ON, 1 = OFF. XXX = Don’t care; switch bits not used.

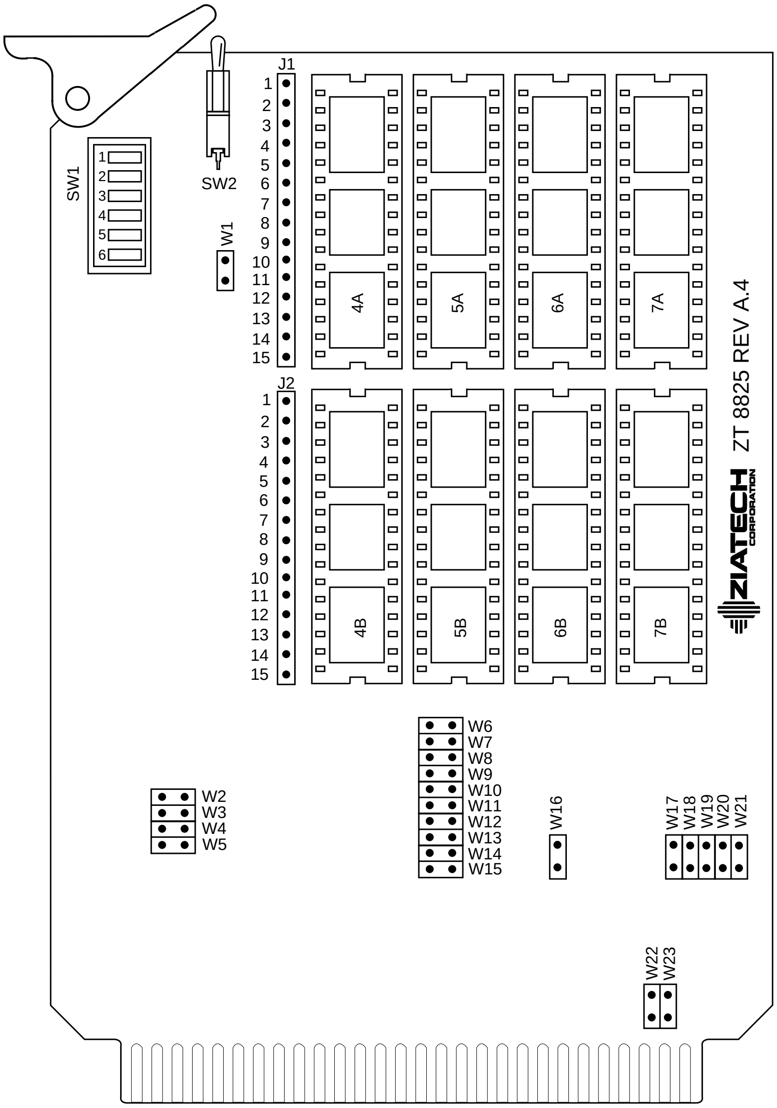


Figure A–1. Customer Jumper Configuration.

Appendix B

SPECIFICATIONS

Contents **Page**

ELECTRICAL SPECIFICATIONS B-1
ZT 8825 Power Requirements B-1
ZT 88CT25 Power Requirements B-2
STD Load Characteristics B-2
ENVIRONMENTAL SPECIFICATIONS B-4
Operating Temperature B-4
Storage Temperature B-4
MECHANICAL SPECIFICATIONS B-4
Card Dimensions & Weight B-4

ELECTRICAL SPECIFICATIONS

ZT 8825 Power Requirements

+5 VDC ±5% @ .50 A typ., 0.67 A max (not including memory chips)

ZT 96008 memory kit: +5 V ±10% @ 0.05 A max, including four 32K x 8 RAM chips

ZT 96009 memory kit: +5 V ±10% @ 0.13 A max, including one 64K x 8 EPROM chip

ZT 96010 memory kit: +5 V ±10% @ 0.04 A max, including one 128K x 8 RAM chip

ZT 96011 memory kit: +5 V ±10% @ 0.04 A max, including four 128K x 8 RAM chips

ZT 88CT25 Power Requirements

Operating: +5 VDC ±5% @ 0.10 A typ., 0.22 A max (not including memory chips)

Powered Standby: +5 VDC ±5% @ 0.06 A max (ZT 88CT08 processor HALT) (not including memory chips)

STD Load Characteristics

Both the ZT 8825 and the ZT 88CT25 are TTL compatible on the STD bus.

The unit load is a convenient method for specifying input and output drive capability of STD bus cards. In STD bus systems, one unit load is equal to one LSTTL load as follows:

- Max. high level input current: 20 µA.
- Max. low level input current: -400 µA.

The STD bus unit load reflects input current requirements at worst case conditions over recommended supply voltage and ambient temperature ranges. An output rated at 60 unit loads can drive 60 STD bus cards having input rated at one unit load.

Note: The ZT 8825 and the ZT 88CT25 use CMOS devices that have high and low input currents of ±10 µA.

Table B-1
ZT 8825 STD Bus Loading.

MNEMONIC	INPUT LOAD (CMOS)†	OUTPUT DRIVE (LSTTL)	PIN (CIRCUIT SIDE)	PIN (COMPONENT SIDE)	OUTPUT DRIVE (LSTTL)	INPUT LOAD (CMOS)†	MNEMONIC
+5 VOLTS			2	1			+5 VOLTS
GROUND			4	3			GROUND
DC LOW*	1	60	6	5			N.C.
D7 3-S (A23)	7	60	8	7	60	2	D3 (A19) 3-S
D6 3-S (A22)	7	60	10	9	60	2	D2 (A18) 3-S
D5 3-S (A21)	7	60	12	11	60	2	D1 (A17) 3-S
D4 3-S (A20)	7	60	14	13	60	2	D0 (A16) 3-S
A15	2		16	15		1	A7
A14	2		18	17		1	A6
A13	1		20	19		1	A5
A12	1		22	21		1	A4
A11	1		24	23		1	A3
A10	1		26	25		1	A2
A9	1		28	27		1	A1
A8	1		30	29		1	A0
RD*	1		32	31		2	WR*
MEMRQ*	1		34	33		1	IORQ*
MEMEX N.C.			36	35		1	IOEXP
MCSYNC*	3		38	37			INTRQ1*
STATUS 0* N.C.			40	39			STATUS 1* N.C.
BUSRQ* N.C.			42	41			BUSAK* N.C.
INTRQ* N.C.			44	43		1	INTAK*
NMIRQ* N.C.			46	45	60		WAITRQ*
PBRESET*			48	47			SYSRESET* N.C.
INTRQ2*			50	49		2	CLOCK*
PCI			52	51			PCO
AUX GND N.C.			54	53			AUX GND N.C.
AUX-V N.C.			56	55			AUX+V N.C.

† CMOS input loads ±10µA maximum

ENVIRONMENTAL SPECIFICATIONS

Operating Temperature

ZT 8825: 0˚ through +65˚ Celsius @ < 95% non-condensing, relative humidity @ 40˚ C

ZT 88CT25: -40˚ through +85˚ Celsius @ 15% to 90% non-condensing, relative humidity @ 40˚ C

Storage Temperature

-55˚ through +125˚ Celsius @ < 95% non-condensing, relative humidity @ 40˚ C, except the battery, which must be kept at < 110˚ C

MECHANICAL SPECIFICATIONS

Card Dimensions & Weight

Width:113.8 mm (4.5 inches)
Height: 165.1 mm (6.5 inches)
Weight: 5.0 oz.

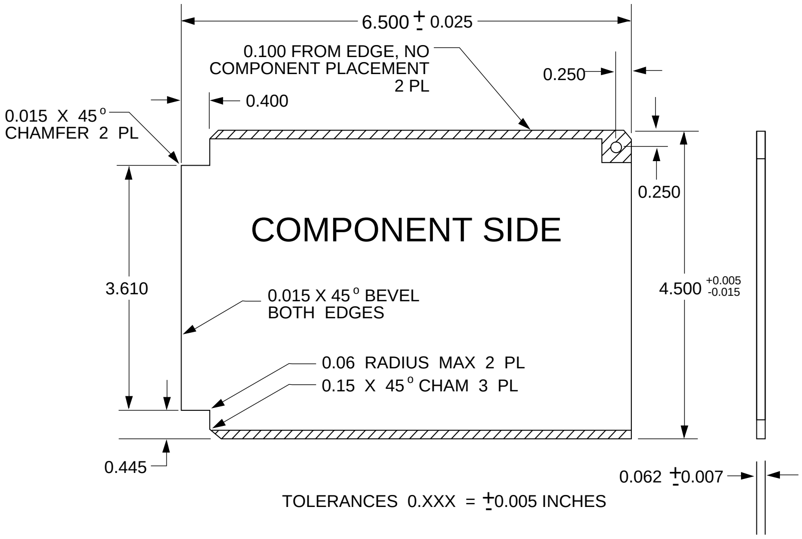


Figure B–1. ZT 8825 Card Dimensions.

Table B-2
ZT 8825 Rev. A.4 Material List.

PART DESCRIPTION	PART #	QTY
0.1 µF ceramic 0.3" DIP	CAP-00061	28
22µF, tantalum, 25 V, radial	CAP-00026	4
0.025 single row, square, wire-wrap	CON-00033	30
0.025 double row, square, wire-wrap	CON-00042	23
0.1" shorting jack	CON-00055	17
signal diode, , 1N4148	CRD-00001	2
HP Schottky HSCH 1001	CRD-0002	2
PC extractor, with pin	GEN-00003	1
0.2 V hysteresis AC/DC power monitor	ICPC-1231-20	1
octal 3 state transceiver	ICPCI-74ACTQ245	2
quad 2 input NAND	ICPCI-74ACT00	1
triple three input NAND	ICPCI-74ACT10	1
octal D flip-flop with clear	ICPCI-74ACT273	1
hex 3-state buffer	ICPCI-74ACT367	1
octal comparator without pullups	ICPCI-74ACT521	2
octal 3-state buffer	ICPCI-74ACT541	2
octal 3-state latch	ICPCI-74ACT573	4
D flip-flop with preset, clear	ICPCI-74ACT74	1
10 wide 3-state latch	ICPCI-74ACT841	1
I/O chip select decode	PAL-50057B	1
Row A memory chip select	PAL-50058A	1
Row B memory chip select	PAL-50059B	1
PC board, ZT 8825 Rev A	PCB-8825-A	1
SRAM, 90-120mA, 35nS, DIP	RAMC-4K4-35K	2
100 Ohm, 1/4 W, 1%, MF	RES-01000	1
10k Ohm, 1/4 W, 1%, MF	RES-01002	4
100k Ohm, 1/4 W, 1%, MF	RES-01003	1
511 Ohm, 1/4 W, 1%, MF	RES-05110	1
5x10k Ohm SIP	RPK-00010	1
9x100k Ohm SIP	RPK-00014	5
7X100k Ohm SIP	RPK-00019	2
3 X 10k Ohm IN SIP	RPK-00037	1

Table B-2
ZT 8825 Rev. A.4 Material List (continued).

PART DESCRIPTION	PART #	QTY
socket, cage jack	SKT-00009	3
socket, 24-pin face wipe 0.3" Ctr	SKT-00025	3
socket, 32-pin face wipe 0.6"	SKT-00034	8
dip switch, 6 position low profile	SWD-00007	1
switch, toggle SPDT tiny PC mount	SWD-00008	1
PNP 250 mW B=100 min	TRN-2N3906	1
NPN B=100 @ 150 mA	TRN-PN2222	4

Parts Differences Between ZT 8825 and ZT 88CT25

LOCATION	ZT 8825 PART #	ZT 88CT25 PART #
Pack 3B	PAL-50057B	PAL-60057C
Pack 2B	PAL-50058A	PAL-60058B
Pack 1B	PAL-50059B	PAL-60059C
Pack 1C, 2C	RAMC-4K4-35K	RAMCI-4K4-30K

Appendix C

HARDWARE QUICK REFERENCE

Contents **Page**

JUMPER DESCRIPTIONS C-2
JUMPER REFERENCE TABLES C-5

JUMPER DESCRIPTIONS

Table C-1
ZT 8825 Jumper Descriptions.

JUMPER #	DESCRIPTION
W1	Enables absolute addressing mode for the board at power-on or reset time (see page 3-19).
W2-5	Enable 24-bit memory addressing mode (see page 3-13).
W6†	Enables 16-bit I/O port addressing for the on-board registers (see page 3-10).
W7†,8,9,10†	Set the value of I/O port address bits A4-A7 (see page 3-11).
W11†,12,13,14	Set the value of I/O port address bits A12-A15 (see page 3-11).
W15†	W15 OUT selects IOEXP high. W15 IN selects IOEXP low (see page 3-11).

† Installed for factory default configuration.

Table C-1
ZT 8825 Jumper Descriptions (continued).

JUMPER #	DESCRIPTION
W16††	Enables one wait state on all memory accesses (see page 3-6).
W17	Used for Ziatech test only; must not be installed.
W18	Selects battery backup mode for Row B chips (see page 3-7).
W19†	Selects non-battery backup mode for Row B chips (see page 3-7).
W20	Selects battery backup mode for Row A chips (see page 3-7).

†† May not be used on a ZT 88CT25 with a ZT 8817 CPU to insert one wait state for slower memory chips. See page 3-6.

†Installed for factory default configuration.

Table C-1
ZT 8825 Jumper Descriptions (continued).

JUMPER #	**DESCRIPTION**
W21†	Selects non-battery backup mode for Row A chips (see page 3-7).
W22	Enables board to drive DCLOW* when Vcc < 4.5 V (see page 3-7).
W23	Enables board to drive PBRESET* when Vcc < 4.5 V (see page 3-7).

†Installed for factory default configuration.

JUMPER REFERENCE TABLES

Table C-2
Jumpers for I/O Base Address Decoding.

I/O Address bits															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
W14	W13	W12	W11	1	1	1	0	W10	W9	W8	W7	1	Register Number		

The default I/O base address on the ZT 8825 is 0EE68h (1110 1110 0110 1XXX).

Table C-3
Chip Type Jumper Configurations.

Chip Type	**J1 or J2 Pin #** 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
16 Kbyte EPROM	○—	—○	○	○	○	○—	—○	○—	—○	○	○	○—	—○	○—	—○
32 Kbyte EPROM	○	○—	—○	○	○	○—	—○	○—	—○	○	○	○—	—○	○—	—○
64 Kbyte EPROM	○	○—	—○	○	○—	—○	○	○—	—○	○	○	○—	—○	○—	—○
128 Kbyte EPROM*	○	○—	—○	○	○—	—○	○	○	○—	—○	○	○—	—○	○—	—○
256 Kbyte EPROM	○	○—	—○	○	○—	—○	○	○	○—	—○	○	○—	—○	○—	—○
512 Kbyte EPROM	○	○—	—○	○	○—	—○	○	○	○—	—○	○	○—	—○	○—	—○
1 Mbyte EPROM	○	○—	—○	○	○—	—○	○	○	○—	—○	○	○—	—○	○—	—○
32 Kbyte SRAM	○—	—○	○—	—○	○	○	○	○—	—○	○	○—	—○	○—	—○	○
128 Kbyte SRAM	○—	—○	○—	—○	○	○	○	○—	—○	○	○—	—○	○—	—○	○
256 Kbyte SRAM	○—	—○	○—	—○	○	○	○	○	○—	—○	○—	—○	○—	—○	○
512 Kbyte SRAM	○—	—○	○—	—○	○	○	○	○	○—	—○	○—	—○	○—	—○	○
32 Kbyte, 28-Pin Flash EPROM† TI 29F256, AT 29C256	○⌒	○—	—○	⌒○	○	○	○	○—	—○	○	○	○—	—○	○—	—○
32 Kbyte, 32-Pin Flash EPROM† AT 29C257	○⌒	○—	—○	○	○—	—○	○	○—	—○	○	○	⌒○	○	○—	—○
128 Kbyte, 32-Pin Flash EPROM† AT 29C010	○⌒	○—	—○	○	○—	—○	○	○—	—○	○	○	⌒○	○	○—	—○

Note: ○—○ indicates a jumper is installed between those two posts.

* The jumper configuration shown is for Intel and AMD 128K x 8 EPROMS only. For all other I28K x 8 manufacturers, please remove the jumpers between pins 1-2 and 3-4. Wire-wrap pin J to pin 8 and pin 4 to pin 9. This applies +5V to EPROM pins 1 and 31 (Vpp and /pgm) to meet non-Intel/AMD requirements.

† Flash EPROMs require special software routines for programming. See ZT 8825 Software Support Package documentation.

Table C-4
Switch Settings for All Possible Chip Sizes.

Switch Section	Sockets				Total Bytes
654321	B		A		On Board
000000	16K	†	16K	†	128K
001001	32K	†	32K	†	256K
001010	32K	†	64K	†	384K
010010	64K	†	64K	†	512K
001011	32K	†	128K	†	640K
011011	128K	†	128K	†	1M
011100	128K	†	256K	†	1.5M
100100	256K	†	256K	†	2M
xxx101	—		512K	†	2M
xxx110	—		1M	**	2M
100111	256K	†	512K	*	2M
101111	512K	*	512K	*	2M

* 2 chips (sockets 4 and 5) max; e.g., use Row A for RAM and Row B for PROM.

** 2 chips (sockets 4 and 5) only, due to 2 Mbyte board limit.

† 1, 2, 3, or 4 chips per row are acceptable.

Notes: Any empty sockets must be accounted for by the software when it accesses the physical pages represented by the empty sockets.

Switch setting of 0 = ON, 1 = OFF. XXX = Don’t care; switch bits not used.

Appendix D

CUSTOMER SUPPORT

Contents **Page**

ZT 8825 REVISION HISTORY D-1
Revision 0.1 . . . . D-1
Revision A - June 7, 1988 . . . . D-1
Revision A.1 - January 1, 1989 . . . . D-2
Revision A.2 - July 6, 1989 . . . . D-2
Revision A.3 - January 30, 1991 . . . . D-2
Revision A.4 - September 23, 1991 . . . . D-2
TECHNICAL ASSISTANCE D-4
RELIABILITY D-4
WARRANTY D-5
RETURNING FOR SERVICE D-6

ZT 8825 REVISION HISTORY

Revision 0.1

Original release of the ZT 8825.

Revision A - June 7, 1988

The ZT 8825 Revision A was created in order to implement the CMOS version ZT 88CT25 and to remove wire modifications to the ZT 8825. There are no functional differences between Revision 0.1 and Revision A ZT 8825s. The Revision 0.1 manual was expanded into two manuals for Revision A and subsequent boards: the ZT 8825

Operators Manual, which covers hardware and systems level topics, and the ZT 8825 Software Support Package manual, which addresses software support issues. Chip access times stated in Table 5-5 of the Operators Manual were changed.

Revision A.1 - January 1, 1989

This revision was implemented to reduce noise on the data lines caused by some new technology EPROMs.

Revision A.2 - July 6, 1989

This revision caused the ZT 88CT25 to support the same features as the ZT 8825.

The A0-7 address receiver was modified to not latch A0-7 with MCSYNC*. This modification was implemented to make the ZT 8825 compatible with future Ziatech products.

Revision A.3 - January 30, 1991

This revision supported additional memory part sizes: 256K, 512K, and 1 Mbyte. The manual for this revision described how to jumper for Flash EPROMs.

Revision A.4 - September 23, 1991

The following changes were made:

1. All Samsung 74AHCTXXX parts were replaced by other brands of 74ACTXXX or 74ACTQXXX. This change has no effect on the board's specifications or on the functionality.

2. The "zero-power" PALs used on the ZT 88CT25 are no longer available and have been replaced with Atmel 22V10L low-power PALs. This affects the power requirements in powered standby mode (see page B-2). The new spec is:

   +5 VDC ±5% @ .065 A max (ZT 88CT08 Processor HALT)
   (not including memory chips)

3. The battery cage jacks no longer extend beyond the STD specification (.040") on the solder side of the board. However, when the battery is installed, the component height is .500" on the topside. This is within spec for .625" center card cage spacing.

TECHNICAL ASSISTANCE

You can reach Ziatech's Customer Support Service at the following number:

Corporate Headquarters: (805) 541-0488
(805) 541-5088 (FAX)

You can also use your modem to leave a message on the 24 hour Ziatech Bulletin Board Service (BBS) by calling (805) 541-8218. The BBS will provide you with current Ziatech product revision and upgrade information.

RELIABILITY

Ziatech has taken extra care in the design of the ZT 8825 to assure reliability. Three major ways in which reliability is achieved are:

1. The product is designed in top-down fashion, utilizing the latest in hardware and software design techniques, so that unwanted side effects and unclean interactions between parts of the system are eliminated.
2. Ziatech tests each board by exercising its functions and retests it to assure that the infant mortality phase is passed before the product is shipped.
3. Ziatech maintains a lifetime data base on each ZT 8825 and on the parts used on each board. Any negative trends in reliability are spotted and Ziatech's suppliers are informed and/or changed.

WARRANTY

Ziatech Hardware: Within two years of shipping date, Ziatech will repair or replace products which prove to be defective in materials and/or workmanship, provided they are promptly returned to Ziatech at customer's expense and have not been repaired, altered, or damaged by non-Ziatech personnel. Service after warranty is available at a predesignated service charge. Batteries are not covered by this warranty. No other warranty is expressed or implied.

Ziatech Software: Within 90 days of shipping date, Ziatech will replace software (PROM or diskette) should it prove defective.

Products not manufactured by Ziatech: Limited to the warranty provided by the original manufacturer.

Batteries: Batteries are not covered by Ziatech's warranty. For the estimated life of batteries on Ziatech board-level products, see the Technical Data Book or page 3-7 of this manual.

Notice: Contact Ziatech for a Return Materials Authorization (RMA) number before returning any product to Ziatech for repair.

Life Support Policy: Ziatech products are not authorized for use as critical components in life support devices or systems without the express written approval of the president of Ziatech Corporation. As used herein:

1. Life support devices or systems are devices or systems that support or sustain life and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system or to affect its safety or effectiveness.

RETURNING FOR SERVICE

Before returning any of Ziatech's products, you must obtain a Returned Material Authorization (RMA) number (call (805) 541-0488). We will need the following information to expedite the shipment of the repaired board or a replacement to you:

1. Your company name and address for invoice
2. Shipping address and phone number
3. Product I.D. number
4. If possible, the name of a technically qualified individual at your company who is familiar with the mode of failure on the board

If the unit is out of warranty, service is available at a predesignated service charge. Contact Ziatech for pricing and please supply a purchase order number for invoicing the repair.

Pack the unit in anti-static material and ship in a sturdy cardboard box with enough packing material to adequately cushion it. Mark the RMA number clearly on the outside of the box before returning.

Appendix E

GLOSSARY

Absolute Address Mode ZT 8825 memory address mode that automatically maps memory linearly between 0 and 7FFFFh.

BRAM Battery Backed Random Access Memory.

CMOS Complementary Metal Oxide Semiconductor. Provides low power density and high noise immunity.

DCLOW* STD-80 power bus signal that indicates DC power has dropped below 4.5 V. Used on the ZT 8825 to protect memory data during power failure.

EEMS Enhanced Expanded Memory Specification.

EMS Expanded Memory Specification.

EEPROM Electrically Erasable Programmable Read Only Memory. Memory can be erased a byte at a time without removing the chip from its socket.

EPROM Erasable Programmable Read Only Memory. Memory is programmed and erased with ultra violet light.

Expanded Memory Memory outside the normal 1 Mbyte address space. The Expanded memory manager allows access up to 2 Mbytes per board, 20 Mbytes summed over all EMS boards in the system.

Extended Memory Memory outside of the normal 1 Mbyte address space up to 16 Mbytes, accessed through 24-bit memory addressing.

Flash EPROM Electrically Erasable Programmable Read Only Memory that allows pages of its memory to be erased one at a time. Memory can be erased without removing the chip from its socket.

Handle Index number for a 16 Kbyte page of memory stored in a map register.

IOEXP Input/Output Expansion. STD-80 control signal. Determines the validity of an I/O access to the board.

LSB Least Significant Bit.

LSTTL Low power Schottky Transistor Transistor Logic.

MCSYNC*
Machine Cycle Sync. STD-80 control signal. Occurs once during each machine cycle of the processor. Used on the ZT 8825 to latch multiplexed memory bits 20-23.

MEMEX
Memory Expansion. STD-80 control signal. Used on some STD boards to enable 16-bit memory addressing. Not used on the ZT 8825.

MSB
Most Significant Bit.

Page
A 16 Kbyte block of physical memory that can be mapped in and out of logical memory by the Expanded Memory Manager.

Page Frame
A 16 Kbyte block of logical memory space that points to a 16 Kbyte "page" of physical memory space in expanded memory.

PAL
Programmable Array Logic. Uses a fixed OR array fed by the output of a programmable AND array to produce "sum of products" logic expressed in a boolean transfer function. On the ZT 8825, PALs decode address bits for an array of memory chip sizes indicated to it by settings on a six segment switch.

PBRESET*
Push Button Reset. STD-80 control signal. This signal is an input line to the system reset circuit.

PROM disk
See *pseudo-disk.*

Pseudo-disk Special software allows PROM and RAM chips to be organized as a "disk" so that DOS can work with this memory in the same manner as regular disks. These chips provide high speed and compact storage.

RAM disk See *Pseudo-disk.*

Schottky TTL Provides high speed, low noise, and compatibility with standard TTL.

SRAM See *Static RAM*.

Static RAM Random Access Memory stored in flip flops. This construction allows for RAM that does not need to be refreshed.

Taa Address Access Time.

Tce Chip Enable Time.

Tdw Data valid to Write High Time.

Toe Output Enable Time.

Twp Write Pulsewidth Time.

TTL Transistor Transistor Logic.

INDEX

- *A* -

absolute address 3-14, 3-17, 3-18, 3-19, A-2, C-2, E-1
access time requirements 2-6, 5-8
addressing
    20-bit 1-3, 2-7, 3-3, 3-13, 3-15, 5-12
    24-bit 1-2, 1-3, 2-7, 2-9, 3-3, 3-13, 3-15, 3-20, 5-12, A-2, C-2
    16-bit I/O 1-2, 5-10
    8-bit I/O 1-2, 5-10
    I/O addressing 3-10, 5-10
address mapping 3-14, 3-15
alternate map register 3-15, 3-17, 3-18, 3-20
ambient temperature requirements 2-3

- *B* -

backplane recommendations 2-4
base I/O address 2-16, 3-10, 5-10
batteries D-5
battery backup 1-3, 3-7, 5-9, A-3, A-4, C-3, C-4
battery life 3-7
bill of materials B-6
blank board layout for user A-6
block diagram 3-2
BRAM E-1

- C -

card dimensions and weight B-4
chip size 2-12, 3-4, 5-6
chip size combinations 3-5
chip type 3-4, 5-3
CMOS E-1
configuration, default 2-7, 2-8
configuration register 2-5, 3-6, 3-10, 3-15, 3-17, 3-18, 3-19, 3-20

- D -

damage during shipping 2-2
DCLOW* 3-9, 5-10, A-4, C-4, E-1
debugging programs 2-5
default base address 2-16, 3-11
default configuration 2-7, 2-8
default I/O address assignments 2-17, 3-10
dimensions and weight of ZT 8825 . . . . B-4

- E -

EEMS 1-2, 1-3, 3-3, 3-14, E-1
EEPROM 1-3, 2-6, 3-3, 3-4, 5-4, 5-10, E-1
electrical specifications B-1
EMM version number 4-12
EMS 1-2, 1-3, 3-3, 3-10, 3-14, 3-15
  example programs 4-7, 4-12, 4-21, 4-23
environmental specifications B-4
EPROM 1-3, 2-6, 3-3, 3-4, 5-3, 5-4, E-2
example EMS programs 4-7, 4-12, 4-21, 4-23
expanded memory 1-2, E-2
extended memory 1-2, 3-13, 3-20, E-2

- F -

features of the ZT 8825 1-3
Flash EPROM E-2
functional block diagram 3-2

- G -

glossary E-1

- H -

handle E-2

- I -

introduction 1-1
I/O address 3-10
  16-bit 1-3, 5-10
  8-bit 1-3, 5-10
I/O addressing 2-16
I/O base address 2-16, 3-10, 5-10
IOEXP 3-12, 5-10, A-3, C-3, E-2
I/O port addressing, 16-bit A-3, C-3
I/O registers 3-11

- J -

jumper configuration 5-9
jumper descriptions A-2, C-2
jumper settings, default 2-7, 2-8

- L -

list of materials B-6
load characteristics (STD) B-2
logical to physical address mapping 3-15

LSTTL E-2

- *M* -

map register 3-10, 3-13, 3-15, 3-18, 3-20
map write enable 3-18
map write protect 3-20
materials list B-6
MCSYNC* 3-13, 5-12, E-3
mechanical specifications B-4
MEMEX 3-13, 5-12, E-3
memory access time 5-8
memory addressing 3-13
  20-bit 3-3, 3-13, 5-12
  24-bit 3-3, 3-13, 3-20, 5-12
memory chip size 3-4, 5-6
memory chips, 28-pin 2-5
memory chip type 3-4, 5-3
memory mapping 3-14
memory speed requirements 2-6, 5-8
microprocessor compatibility 2-4
Microsoft C program example 4-7
Microsoft macro example program 4-21
mounting 28-pin memory chips 2-5
mounting the ZT 8825 2-3

- *N* -

noisy environment recommendations 2-4

- *O* -

operating requirements 2-3
operating temperature B-4

- *P* -

page 3-14, 3-15, 4-12, E-3
page frame 3-14, 3-15, 4-12, E-3
PAL E-3
PBRESET* 3-9, A-4, C-4, E-3
28-pin chips 5-3
power dissipation requirements 2-3
power requirements, ZT 8825 . . . . B-1
power requirements, ZT 88CT25 . . . . B-2
processor compatibility 2-4
PROM disk 1-2, E-3
pseudo-disk E-4
pseudo-disks 1-2

- *R* -

RAM 1-3, 3-17, 3-18, 3-21, 5-9
RAM disk 1-2, 3-19, E-4
registers
- configuration 3-17, 3-18, 3-19, 3-20
- I/O 2-16, 3-11
- map 3-13, 3-15, 3-18, 3-20

reliability of the ZT 8825 . . . . D-4
requirements, system 2-3
Return Material Authorization D-6
revision history D-1
ROM 3-4

- *S* -

schottky TTL E-4
service to the ZT 8825 . . . . D-6
shipping damage 2-2
socket assignments 2-5
SRAM 1-3, 2-6, 3-3, 3-4, 3-7, 3-15, 5-3, 5-4, 5-10, E-4

STD load characteristics B-2
STD-80 Series compatibility 2-4
storage temperature B-4
supply current requirements 2-3
supply voltage requirements 2-3
switch 1 . . . . 3-4, 5-6
switch 2 3-6
system requirements 2-3

- T -

technical assistance D-4
time requirements 2-6, 5-8
TTL E-4
Turbo Pascal program example 4-12

- U -

unpacking 2-2

- V -

valid chip size combinations 2-13, 3-5
Vss 5-9

- W -

wait state 1-3, 3-6, 5-8, A-3, C-3
warranty D-5
weight and dimensions of ZT 8825 . . . . B-4
what’s in the box 2-2
write enable 3-18
write protect 1-3, 2-5, 3-6, 3-17, 3-20

- Z -

ZT 88CT25 . . . . 1-3, 2-1, 2-6, 3-6, 5-8, D-1
ZT 8825 revisions D-1