



Tattletale Model 8

Installation and Operation Manual

ANSI-C Version

Onset Computer Corporation
470 MacArthur Blvd., Bourne, MA 02532
PO Box 3450, Pocasset, MA 02559-3450

Tel: (508) 759-9500
Fax: (508) 759-9100
www.onsetcomp.com

P/N MAN-TT8C



Copyright

© 1999, Onset Computer Corporation

All rights reserved. No part of this manual may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, optical, magnetic, or otherwise, without authorization from Onset Computer Corporation.

Trademarks

Tattletale and CrossCut are trademarks of Onset Computer Corporation. Aztec C is a trademark of Manx Software Systems. MS-DOS is a trademark of Microsoft. UNIX is a trademark of AT&T Bell Laboratories. PC-DOS is a trademark of IBM.

Warranty

Within one year after delivery, Onset will repair or at its option replace, without charge, any of its products found to have a manufacturing defect. Boards damaged by customer error or negligence, or that have failed after the one year period, may be returned for evaluation.

Replacement Policy

New replacements for damaged Tattletale products will be made available as long as the product has not been discontinued. The cost is approximately 1/2 the quantity one price. The replacement is warranted for one year. A customer may request a replacement for any reason as long as the damaged board can be returned. Onset may suggest replacing an item submitted for repair if the cost of the repair will exceed the cost of the replacement.

Repair Policy

Onset will attempt to repair Tattletale products returned with an Onset RMA number. Estimates will not be given but in no case will the customer be charged more than the cost of a new replacement. After the item is repaired or replacement is recommended the customer will be notified of the price. Repairs or replacements will not be shipped until a valid purchase order is received. Electronic items which have been repaired may be more prone to future failure than new items. Onset does not guarantee the appropriateness or necessity of any repair. Repairs will usually be finished in less than 2 weeks.

ASAP Repair Policy

Repairs will be started the same day they are received if the following conditions are met:

- *The damaged item is clearly labeled "ASAP repair requested".
- *The damaged item is accompanied with an open purchase order and an Onset RMA number.
- *The customer's account is not over due.
- *The damaged item is received before noon.
- *The appropriate Onset Computer employees are present on the day of receipt.

Onset will do its best to repair the item the same day it is received, however, due to circumstances beyond our control this may not always be possible. ASAP repairs carry a higher retest and troubleshooting charge. ASAP repairs will be returned UPS red or Fed X priority 1 at the customer's expense unless another option is requested.

Disclaimer

Onset makes no warranties, either express or implied, regarding the Tattletale, its merchantability, or its fitness for any particular purpose. The exclusion of implied warranties is not permitted by some states. As such, the exclusion may not apply to you. The Tattletale and its development boards are not authorized for use as critical components in life support, or other medical devices or systems without the express written approval of the President of Onset Computer Corp.

Contents

<u>Title</u>	<u>Page</u>
Section 1 - Introduction to the Model 8	
Quick Start Information	1-1
Welcome to the Tattletale Model 8	1-2
Onset Computer Corporation.....	1-4
Conventions used in this Document	1-4
Included with the Model 8 (C Version)	1-5
Where to Start	1-5
What You Should Know	1-5
Warnings and Precautions	1-5
Additional Information Resources	1-6
Documentation	1-6
Technical Support	1-7
Aztec Compiler Updates and Manx Tech Support	1-7
Development Software Description	1-7
IBM Software	1-7
Macintosh Software	1-8
Getting Started	1-8
Tattletale Model 8 Development Kit Contents	1-8
What to do if something is Missing or Broken upon Arrival	1-9
Tools Required to Connect and Test the Tattletale.....	1-10
Safety Precautions	1-10
Section 2 - How to Connect and Setup the Model 8	
The Prototyping Board	2-1
Attaching the Tattletale Model 8 to the IO-8 Prototyping Board	2-1
Installing a Temporary Sensor onto the Prototyping Board for Testing.....	2-1
Connecting the Tattletale Model 8 to the Computer.....	2-3
Installing the Tattletale Software	2-3
Installing the Onset C Libraries and Directories onto the Hard Drive	2-4
Installing CrossCut onto the Hard Drive	2-4
Installing the Aztec C Software onto the Hard Drive.....	2-4
Modifying your Autoexec.bat File	2-5
Verifying the Software Installation	2-6
Testing the Operation of the Tattletale Model 8	2-6
Section 3 - Operating the CrossCut Program	
Introduction	3-1
What is CrossCut and how is it used?.....	3-1
Learning to Use CrossCut on the IBM PC (or Compatible)	3-3
Getting Started with CrossCut	3-3

Contents (continued)

<u>Title</u>	<u>Page</u>
Keyboard Shortcuts for Mouse Actions.....	3-3
Explanations of CrossCut Menu and Window Options	3-5
Introduction.....	3-5
Explanation of CrossCut Window Types	3-5
Terminal Window	3-5
Edit File Window Description	3-6
File Menu Option Descriptions.....	3-8
Edit Menu Option Descriptions	3-11
Search Menu Option Descriptions	3-12
Tattletale Menu Option Descriptions.....	3-14
CommPort Menu Option Descriptions	3-15
Windows Menu Option Descriptions.....	3-17
Help Menu Option Descriptions	3-18
Software Change Information for CrossCut	3-19
Program Parameters Saved in the Configuration File	3-19

Section 4 - C Programming Guide

Introduction	4-1
TOM8 - Tiny Onset Monitor	4-1
Command Summary	4-2
Memory Display	4-2
Memory Modify.....	4-2
Load S-records	4-2
Jump to address.....	4-3
Print Command summary	4-3
Learning to Use C on the Model 8	4-3
Tutorials	4-3
Overview.....	4-3
Tutorial 1 - Loading a program into the Model 8 using CrossCut.....	4-4
Launching CrossCut	4-4
Loading a Motorola S-Record	4-4
Tutorial 2 - Compiling, linking and generating hex files.....	4-5
Compiling -vs- Cross-Compiling	4-5
Compiling a Source File	4-7
Compiler Flags	4-8
Note about Static Arrays	4-8
Linking an Object File	4-8
Linker Flags	4-8
Generating S-Records	4-9
S-Record Flags	4-9
Using the Generic Makefile	4-10
Loading the Program into Flash Memory	4-11

Contents (continued)

<u>Title</u>	<u>Page</u>
Tutorial 3 - A Simple Logger Application.....	4-12
Going through the tutor3.c source... ..	4-12
Program Initialization	4-13
Dynamic Memory Allocation	4-13
Logging Data	4-13
Displaying Data	4-14
Storing Data to a File	4-15
Running tutor3... ..	4-15
Using the 8run and 8app Batch Files... ..	4-15
General Model 8 Logger Techniques	4-16
Reducing power in the Simple Logger	4-16
Lowering System Frequency	4-16
Interrupt Driven Sampling	4-16
Long Term Data Logging Techniques.....	4-16
Example Programs	4-17
Running Programs in RAM	4-17
Startup Applications	4-18
Loading your own Programs into the EEPROM	4-18
How to Erase a Program from the Flash EEPROM Memory	4-19
Tattletale Model 8 Memory (256K/256K Version)	4-19
Memory Map - Load to RAM.....	4-19
Memory Map - Load to Flash	4-20
RAM Configurations	4-20
Memory Symbols.....	4-21
Exception Handler Info	4-23

Section 5 - C Library Reference

How to Use this Section	5-1
Tattletale Model 8 Library Functions	5-1
Model 8 Header Files.....	5-1
Typedefs used	5-1
Library Object Files	5-1
Model 8 Function Descriptions	5-2
PIC	5-2
Introduction	5-2
PIC Basics	5-2
Time Format Conventions	5-3
Short Descriptions	5-3
Functions by Category (and TypeDefs)	5-7
AtoD Converter Functions.....	5-7
Assembly Shortcuts	5-7
Digital I/O macros /* #include <dio332.h> */	5-7

Contents (continued)

<u>Title</u>	<u>Page</u>
Flash EEPROM Functions	5-7
Interval Timer Functions.....	5-7
Initialization Routines	5-7
Miscellaneous Functions.....	5-7
Real-time functions /* #include <time.h> */	5-8
Serial I/O Functions	5-8
User I/O Extras /* #include <userio.h> */	5-8
System Functions	5-8
TPU Functions	5-8
User EEPROM Functions	5-9
Function Descriptions	5-9

Section 6 - Hardware and Interface Specifications

Getting Started	6-1
Hardware Do's and Don'ts	6-1
Tattletale Model 8 Connectors	6-2
Pin and Socket Connector Specifications	6-2
SquishyBus Connector Specifications	6-3
Mounting the Model 8 to the Prototyping Boards	6-3
Prototyping Board Details	6-4
Power Supply Considerations	6-5
The UART	6-6
Tattletale Model 8 Connections and Specifications	6-8
Model 8 Components	6-12
TPU Functions	6-12
Analog Input Connections and Specifications	6-14
12-Bit, 8-Channel A-D Converter	6-14
Analog Inputs	6-15
Main UART	6-15
Changing the Baud Rate	6-15
Current Drain	6-17
A-D Converter Circuit	6-19
Using the Model 8 in Bipolar Operation	6-19
RESET	6-19
Memory	6-19
RAM	6-19
Flash Memory	6-20
QSM - Queued Serial Module	6-20
SCI - Serial Communications Interface	6-20
QSPI - Queued Serial Peripheral Interface	6-20
SIM - System Integration Module	6-20
System Clock	6-20

Contents (continued)

<u>Title</u>	<u>Page</u>
Peripheral Chip Selects	6-20
BDM (Background Debug Mode)	6-21
PIC 16C64.....	6-21
Oscillator.....	6-21
Model 8 Accessories	6-22
IO-8 Prototyping Board	6-22
PR-8 Prototyping Board.....	6-23
Aztec C Compiler for DOS.....	6-23
Aztec C Source Level Remote Debugger	6-23
Tattletale 8 C Libraries for DOS.....	6-23
SquishyBus Connectors	6-24
PCMCIA Card Adapter (Obsolete, contact Onset for alternate part.)	6-24

Section 7 - Application Notes for the Model 8

Converting the MAX186 A-D Converter to Bipolar Operation	7-1
Hardware Modification	7-1
Software Modification	7-2
Convert Bipolar Input to Unipolar	7-2
Operational Amplifiers and Instrumentation Amplifiers	7-4
Digital Output Protection	7-7
Digital Input Protection	7-8
Input Protection	7-9
Using Flash Memory for Non-Volatile Data Storage	7-10
Adding Additional Flash Memory	7-18
Adding Additional RAM	7-26
Low Power Sleep Mode Examples	7-29
Watchdog Reset Test Program	7-29

Section 8 - Troubleshooting

Introduction	8-1
Common Model 8 Problems and Possible Solutions	8-1
Problems in Aztec C	8-1
Problems with the Model 8	8-1
Problems with CrossCut	8-2
Problems with the TOM8 Monitor	8-2
Troubleshooting Tattletale Problems	8-3
What to do if the Operation Test Fails	8-4
Troubleshooting Procedure #1	8-4
Troubleshooting Procedure #2.....	8-4
Troubleshooting Procedure #3.....	8-6
Troubleshooting Procedure #4.....	8-7

Contents (continued)

<u>Title</u>	<u>Page</u>
Contacting Onset Computer Product Support	8-8
Sending a FAX to Product Support	8-9
Sending E-mail to Product Support over the Internet.....	8-9
Calling Onset Computer Product Support	8-9
Using the Onset Computer BBS	8-9

Appendix A - Manufacturer Contact Numbers

Appendix B - Data Sheets

Glossary

Index

List of Illustrations

<u>Figure</u>	<u>Title</u>	<u>Page</u>
Figure 1-1	Block Diagram of the Four Major Sections of a Tattletale	1-3
Figure 2-1	IO-8 with Test Components Soldered onto it	2-2
Figure 2-2	Schematic of the Test Circuit	2-2
Figure 3-1	Flow Chart of the Development Cycle for a C Program	3-2
Figure 3-2	Terminal Window Display	3-5
Figure 3-3	Edit File Window	3-6
Figure 3-4	Typical Window after Opening a CrossCut Program File	3-7
Figure 3-5	File Menu Options	3-8
Figure 3-6	Open File Dialog Box	3-10
Figure 3-7	File Name Extension Dialog Box	3-11
Figure 3-8	Edit Menu Options	3-11
Figure 3-9	Search Menu Options	3-12
Figure 3-10	Find Option Dialog Box	3-13
Figure 3-11	Replace Option Dialog Box	3-14
Figure 3-12	Tattletale Menu Options	3-14
Figure 3-13	CommPort Menu Options	3-15
Figure 3-14	Baud Rate / Protocol Option Dialog Box	3-17
Figure 3-15	ASCII Transfer Option Dialog Box	3-17
Figure 3-16	Windows Menu Options	3-17
Figure 3-17	Help Menu Options	3-18
Figure 4-1	Model 8 Memory Map	4-22
Figure 6-1	Model 8 Dimensions	6-2
Figure 6-2	SquishyBus Dimensions	6-3
Figure 6-3	DC Power Jack w/o Connector Plugged in	6-6
Figure 6-4	DC Power Jack with Connector Plugged in	6-6
Figure 6-5	Communication Cables	6-6
Figure 6-6	Schematic of the Model 8	6-7
Figure 6-7	MC68332 Block Diagram	6-12
Figure 6-8	Diagram of A-D Regulator Circuit	6-19
Figure 6-9	Graph of Oscillator Error vs Temperature	6-22
Figure 6-10	IO-8 Prototyping Board	6-22
Figure 6-11	PR-8 Prototyping Board	6-23
Figure 6-12	Side View of IO-8, TT8 and PCMCIA Adapter Assembly	6-24
Figure 7-1	Location of the Trace for Bipolar A-D Operation	7-1
Figure 7-2	-5V Circuit for Bipolar A-D Operation	7-2
Figure 7-3	Circuit for Converting Bipolar to Unipolar	7-3
Figure 7-4	Another Circuit for Converting Bipolar to Unipolar	7-4

List of Illustrations (continued)

<u>Figure</u>	<u>Title</u>	<u>Page</u>
Figure 7-5	Operational Amplifiers	7-4
Figure 7-6	Instrumentation Amplifier Circuit	7-7
Figure 7-7	Digital Output Protection Circuit	7-7
Figure 7-8	Digital Input Protection Circuit	7-8
Figure 7-9	Input Protection with just a Resistor	7-9
Figure 7-10	Input Protection with a Resistor and Capacitor	7-9
Figure 7-11	Input Protection with a Zener Diode	7-10
Figure 7-12	Adding One Megabyte of Flash Memory	7-18
Figure 7-13	One Megabyte of Static RAM (Two 512K x 8's)	7-26
Figure 8-1	Troubleshooting Flow Chart	8-3
Figure 8-2	Communication Cable Pin Layouts	8-6
Figure 8-3	Testing the Communication Cable	8-7

List of Tables

<u>Table</u>	<u>Title</u>	<u>Page</u>
Table 1-1	Quick Start Sections for Experienced Users	1-1
Table 1-2	Suggested Section Reading Order for New Users	1-1
Table 1-3	Model 8 Specifications	1-2
Table 1-4	Tattletale Block Diagram Section Descriptions	1-3
Table 1-5	Document Conventions	1-4
Table 1-6	Model 8 C Development Kit Contents	1-8
Table 1-7	Model 8 Deluxe Development Kit Contents	1-9
Table 2-1	Model 8 Communication Settings	2-6
Table 3-1	Keyboard Shortcuts for Mouse Actions	3-4
Table 3-2	Terminal Window Feature Descriptions	3-5
Table 3-3	Edit File Window Feature Descriptions	3-7
Table 3-4	File Menu Option Descriptions	3-8
Table 3-5	Edit Menu Option Descriptions	3-11
Table 3-6	Search Menu Option Descriptions	3-13
Table 3-7	Tattletale Menu Option Descriptions	3-14
Table 3-8	CommPort Menu Option Descriptions	3-15
Table 3-9	Window Menu Option Descriptions	3-17
Table 3-10	Help Menu Option Descriptions	3-18
Table 4-1	Extension Naming Conventions	4-10
Table 5-1	C Library Function List	5-3
Table 6-1	Model 8 Specifications	6-8
Table 6-2	IO-8 Pin Functions	6-9
Table 6-3	PR-8 Pin Functions	6-10
Table 6-4	TPU Channel Functions	6-13
Table 6-5	Custom Functions for Jack #2	6-13
Table 6-6	Thirteen I/O Line Alternate Functions	6-14
Table 6-7	Analog Input Pin Specifications	6-15
Table 6-8	68332 Clock Rates for the 40000 Crystal	6-16
Table 6-9	Current Drain Due while using Low Power Modes	6-18
Table 6-10	Digital I/O Line Specifications (from Motorola)	6-19
Table 7-1	Op Amp Limitation Data	7-5
Table 8-1	Tattletale Communication Default Settings	8-3

Section 1 - Introduction to the Model 8

Quick Start Information

If you are already an experienced Tattletale user and you want to jump right in and start using the Tattletale, read the sections listed in Table 1-1 for detailed information regarding this specific model of the Tattletale line.

If you are a new user of Tattletale products, read the sections listed in Table 1-2 in the order listed for the easiest learning curve.

Table 1-1: Quick Start Sections for Experienced Users

Section Number	Description of Section Contents
2 - Installation	Step-by-step instructions for connecting the Tattletale to your computer system
6 - Hardware Specifications	Detailed design specifications for this Tattletale model. This information is needed for designing interfaces for the Tattletale
7 - Application Notes	Detailed examples showing interface techniques for many different common uses for the Tattletale

Table 1-2: Suggested Section Reading Order for New Users

Section Number	Description of Section Contents
1 - Introduction	Explains general information and safety information about the Tattletale
2 - Installation	Step-by-step instructions for connecting the Tattletale to your computer system
3 - Operating CrossCut	A step-by-step tutorial for learning CrossCut and detailed descriptions of all the menu options in CrossCut
4 - Model 8 Programming in C	Step-by-step tutorials for learning to operate the Model 8 with C programming
5 - C Library Commands	Detailed information on using C programming commands to operate the Tattletale. All possible commands are covered in this section
6 - Hardware Specifications	Detailed design specifications for this Tattletale model. This information is needed for designing interfaces for the Tattletale
7 - Application Notes	Detailed examples showing interface techniques for many different common uses for the Tattletale

Welcome to the Tattletale Model 8

Table 1-3: Model 8 Specifications

Size (inches)	2 x 3 x 0.5
Weight (oz.)	1
Processor	68332
Data capacity (RAM)	256K (or 1M)
Additional capacity	PCMCIA
Flash EEPROM	256K
A-D converter	12-bit
Analog channels	8
Max sampling rate (Hz)	100K
Digital I/O lines	up to 25
Count channels	up to 25
Minimum current	<200 μ A typical
Peak current	150mA
Main UART baud (default) at RS-232 Levels:	9600
TPU UART baud rates (others available):	The 14 TPU lines can be set to any standard rate up to 500K
Serial EEPROM (bytes)	7190
Voltage input	7 to 15V
Battery RAM backup	No
Real-time clock	Hardware
Programming languages	C, TxBASIC
Operating temperature range	-40 to +85° C
Relative humidity range	0 - 95% non-condensing

NOTE: TPU I/O pins may be configured through software as either a UART Rx or Tx. The TPU handles all timing and buffering of serial transmissions and therefore achieves a CPU independence equivalent to a hardware UART.

Congratulations on your purchase of the Tattletale Model 8! The Model 8 includes a powerful Motorola 68332 microprocessor, a 500 KBaud RS-232 interface, a tunable system clock adjustable from 160 KHz to 16 MHz and highly efficient, typically drawing 200 μ A in low power sleep mode. The Model 8 includes a PIC 16C64 microcontroller which operates as a super-programmable clock, improved external bus expansion, provision for an external clock and increased memory.

Operating Temperature Range

Model 8 components are specified to operate over a temperature range of -40°C to +85°C with the following exceptions. The switch used to enter the Background Debugging Mode (BDM) during the power up sequence has an operating range of -20°C to +70°C. The Light Emitting Diode (LED) used to indicate a low signal on IRQ3 (pin 61 on PR-8, pin A-5 on IO-8) has an operating range of -30°C to +85°C.

The Standard Logger configuration comes with 256K of flash memory and 256K of RAM. Options include an ANSI C compiler with source debugging for DOS, industrial temperature range components (-40° C to +85° C), and PCMCIA memory and I/O expansion. A one megabyte version is also available.

The compact design of the Tattletale Model 8 places the most important logger / controller components on a single 2in. x 3in. x 1/2in. printed circuit board. The Model 8 can be described as having four functional sections, each one described in Table 1-4 and shown Figure 1-1.

Table 1-4: Tattletale Block Diagram Section Descriptions

Section on Block Diagram	Description
A	Analog and digital I/O, including UARTs, individually programmable digital I/O lines, counter, square wave generator and three-wire serial interface.
B	CMOS CPU, CMOS RAM and FLASH EEPROM for non-volatile program storage.
C	Data storage (the Datafile) for storing the results of measurements.
D	Voltage regulators to control supply voltages from a battery input or 7 - 15V power supply.

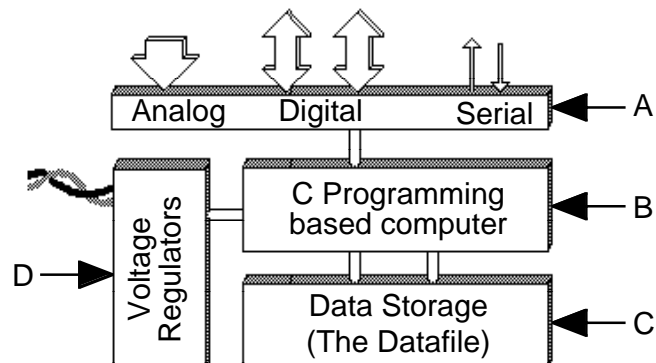


Figure 1-1: Block Diagram of the Four Major Sections of a Tattletale

The Model 8 has up to 25 digital I/O lines and an 8 channel, 12-bit A-D converter; it can be powered by any 7 to 15V power supply. The Model 8 has two on board voltage regulators: one for the digital circuits and another for the A-D converter. Both regulators are current and thermally limited, protecting them from unintentional overloads during development, and have about 50mA of excess capacity for powering external circuits (when using a power supply with 200mA of current).

The Model 8 supports ANSI C and inline assembly allowing access to the full power of the 68832. We are sure that you will be very pleased with your Tattletale purchase.

Onset Computer Corporation

Onset Computer has specialized in the design and manufacture of low power computers for data logging and control applications since 1981. Our machines fly on the Space Shuttle, monitor conditions at the bottom of the ocean and control a myriad of data gathering systems worldwide.

Onset's StowAway™, Hobo and Tattletale lines have gathered data in the world's oceans, in balloons, aircraft, parachutes, race cars, boats, trains, pipelines, animals, humans, oil fields, forests and streams.

The company's first product was the C-44 bus card set, designed specifically for battery-powered applications. Building on this concept, Onset engineers developed the Tattletale line of machine control and data logging engines. From the beginning, the Tattletales offered significant improvements, reducing physical size while facilitating program development by compressing the capabilities of a rack of C-44 bus cards onto a single board and adding built-in BASIC. Further developments included the addition of plotting software and increased processing capability. Your Tattletale Model 8 is the product of thirteen years of refinement.

Onset has also developed a line of low cost, single channel, dedicated data loggers. Hobo and StowAway data loggers are configured and launched with BoxCar or LogBook host software. When the logger's mission is complete, the software downloads the collected data and displays it graphically. BoxCar and LogBook also provide data conversion to most popular formats. The Hobo/StowAway line features non-volatile data storage, incredibly small size and exceptionally low power drain. BoxCar and LogBook software is available for both the Macintosh and IBM PC environments.

Conventions used in this Document

To help you identify commands, information and safety warnings easily, this manual uses the text formats and visual aids listed in Table 1-5.

Table 1-5: Document Conventions

Type Style	Used for
bold lowercase	Command names for the IBM PC. Any text in this format must be typed exactly as shown or the commands will not work correctly.
<i>italic</i>	Used to emphasize important information in a step or paragraph.
ALL CAPITALS	Directory names and file names on the IBM PC and acronyms. Also the RETURN key and the ENTER key will be in all capitals when you are being told to press them in a procedure.
NOTICE boxes	Indicates that equipment could be damaged if the instructions that follow the notice are not strictly observed.

NOTE: References made to figures, tables and specific pages will always show the Section number first and then the page or reference number.

Included with the Model 8 (C Version)

As delivered, your Model 8 comes pre-loaded with a mini-monitor program called TOM8 which is the entry point for programming the Tattletale. When first powered up, the 8 will automatically launch the monitor and display version and copyright information, before displaying the “TOM8>” monitor prompt. What you do next will depend on what you intend to do with the Model 8 and which programming options you have purchased.

If you're new to the Model 8, be sure you have obtained the appropriate development kit (there are three kits; one for C, another for TxBASiC and the deluxe kit that combines the C and TxBASiC), which contains the programming tools, manuals, and accessories that you need to work with a Tattletale Model 8.

Where to Start

Having purchased a development kit, you may be surprised (terrified?) by the sheer volume of manuals and documentation. Fortunately, most of this material needs only infrequent reference; and then only when accessing some of the 8's more esoteric capabilities. Everyone should read Section 2. It has the information you need to safely proceed to the next stage—connecting the Model 8 and getting down to business.

What You Should Know

We assume that you already know how to program in C. The combination of Aztec C, CrossCut, and a fairly complex piece of hardware like the Model 8 make it an inappropriate vehicle for learning to program in C. You should start with a native C compiler to get familiar with the language first before trying to cross-compile programs meant for another system.

If you are an experienced C programmer who has worked primarily on the newer integrated C development systems creating native applications, you may be disappointed at the relatively primitive tools available for cross development. We've tried to present you with a turnkey development system, but be aware that cross development is inherently slower and more complex than native development.

We suggest that you approach cross development by creating and testing non-machine specific algorithms and functions on your native C compiler, providing stubs to simulate the target specific operations. Limit the target-level testing of new code and concepts to those operations which must be performed on the Model 8.

Warnings and Precautions

If you've read ahead to Section 2, you may feel a little timorous about touching the board. If so, we've achieved our goal: the Tattletale has survived the introductory phase. Though we want you to take the warnings to heart, the reality is that the Model 8 is a remarkably robust board, capable of surviving even in a busy development environment.

Additional Information Resources

We try to keep the printed documentation and development kit diskettes up to date, but they invariably lag behind the electronically distributed files and documentation available on our bulletin board system and on internet (see Technical Support heading). If you have access to a modem or internet, periodically check for new and interesting files. In addition, your purchase puts you on our mailing list for the TattleTips newsletter in which we announce significant new offerings and provide information listing the latest software revisions. Some files are password protected with new keys applied to each new release. If you are a registered customer, call Onset at (508) 759-9500 between 9 AM and 5 PM EST with the name of the file you want to access, and we will provide you with the key.

The PC development diskettes have an EXAMPLES directory which contains additional information and examples. Some of these are of a more technical nature which generally are not required for simple data logger applications, but may be very valuable if you are creating more demanding applications. All of the documentation files are distributed as Adobe Acrobat Documents—an encapsulated document display utility that can be viewed with a Macintosh or Windows.

Documentation

The Model 8 is a complex and powerful machine with many different features. As a result, the descriptions of many components are beyond the scope of this manual. If you're not sure of where to look for information on a particular topic related to the Model 8, below is a summary of where to start:

TT8 Installation and Operation Manual	Model 8 specific information: Installation Hardware Model 8 C Libraries CrossCut Communications Software TOM8 Mini-Monitor
Aztec C68K/ROM	General information about the Aztec C compiler.
Motorola Manuals: 68332 Manual	General information about the various functions and modules of the 68332 System Integration Module.
CPU32 Manual	Specific information about CPU (Instruction Set).
TPU Manual	Specific information about the Time Processor Unit.
LTC1121 Data Sheet	Information about Linear Technologies LTC1121, Low Power Voltage Regulator.
LTC1174 Data Sheet	Information about Linear Technologies LTC1174, DC/DC Converter.
MAX186 Data Sheet	Information about the Maxim MAX186, 12 Bit A/D Converter.
MAX242 Data Sheet	Information about the Maxim MAX242, RS-232 Driver.

Technical Support

Onset should be your first line of defense for problems with Model 8 hardware and software although we may vector your cross-development questions to the appropriate vendor. Manx has excellent technical support for their compilers, but they will not be able to help you with any problems relating to the Model 8.

Since the Model 8 lends itself to the creation of complex programs, if you find what you believe to be a bug, you will have to reduce the complexity of your application to focus on just the failing portion so that we can reproduce and correct the fault. The problem is often found during this process. Also, to continue to provide virtually unlimited support for the Model 8, we have to limit help to questions relating directly to the Model 8 and its libraries and ask that you direct general C programming questions to local consultants or in-house experts.

We would prefer that you use the internet (onset@ccsnet.com) to post questions, comments and suggestions. We realize however, that some problems require more immediate attention so call if you need help.

Onset Phone:	(508) 759-9500
Onset Fax:	(508) 759-9100
Onset Web:	www.onsetcomp.com

Aztec Compiler Updates and Manx Tech Support

Manx Software Systems frequently updates their Aztec C development software. To obtain information about updates to their compiler you must register it with Manx Software. Send the registration card included in the Aztec manual to Manx Software (not to Onset Computer). We cannot process registration cards for Manx products (such as the compiler). For more information on their update policies see the Aztec C Embedded Development System manual or call:

Manx Technical Support:	(908) 780-5005
Manx Sales:	(800) 221-0440
Manx International Sales & Updates:	(908) 308-3800
Manx Fax:	(908) 308-3322
Manx BBS (2400 Baud):	(908) 780-6363

Development Software Description

IBM Software

Depending on your purchase, you will have received at least two of the following Onset diskettes. The TxBASIC diskette is only used when developing programs in TxBASIC and is discussed in another manual.

- DISK-D-8-AZC Onset C Libraries for Aztec C with TT8
- DISK-D-8-XCT CrossCut software
- DISK-D-8-TXB TxBASIC, TxTools, and BYOB for the TT8

Diskettes that come with each developer kit:

<u>Kit P/N</u>	<u>Diskette P/N</u>
TT8-C-DK-PC	DISK-D-8-AZC DISK-D-8-XCT
TT8-DLX-DK-DOS	DISK-D-8-AZC DISK-D-8-XCT DISK-D-8-TXB

If you purchased just the C libraries package, the following diskettes are included along with this manual:

TT8-C-LIB-PC	DISK-D-8-AZC DISK-D-8-XCT
--------------	------------------------------

Macintosh Software

The CrossCut program for the Model 8 (for using C) is only available for the IBM PC or compatible.

Getting Started

Tattletale Model 8 Development Kit Contents

Table 1-6 shows the kit contents for the IBM PC Tattletale development kits.

Table 1-6: Model 8 C Development Kit Contents

Part Numbers for Kit TT8-C-DK-DOS	Description
Finished Goods TT8 IO-8 PC-3.5 Cable	Model 8 Tattletale (purchased separately) 2 x 3 inch prototyping board for the Tattletale Model 8 Communications cable to connect the Tattletale to a PC
Manuals MAN-TT8C MAN-MC68332 MAN-CPU-32 MAN-TPU	Manual for the Tattletale Model 8 for use with ANSI C 3-Ring binder for the manual Tab set for dividing the sections of the manual 3.5in. looseleaf diskette holder Motorola manual for the 68332 processor Motorola manual for the CPU Motorola manual for the Time Processor Unit
Diskettes DISK-D-8-AZC DISK-D-8-XCT	Onset C libraries for using Aztec C with the Model 8 CrossCut program software

Table 1-6: Model 8 C Development Kit Contents (Continued)

Part Numbers for Kit TT8-C-DK-DOS	Description
Misc. Goody Bag	Thermistor, 10K resistor and FET for experimenting

Table 1-7: Model 8 Deluxe Development Kit Contents

Part Numbers for Kit TT8-DLX-DK-DOS	Description
Finished Goods TT8 IO-8 PR-8 PC-3.5 Cable	Model 8 Tattletale (purchased separately) 2 x 3 inch prototyping board for the Tattletale Model 8 5 x 7 inch prototyping board for the Tattletale Model 8 Communication cable to connect the Tattletale to a PC
Manuals MAN-TT8C MAN-TT8TXB MAN-MC68332 MAN-CPU-32 MAN-TPU	Manual for the Tattletale Model 8 for use with ANSI C 3-Ring binder for the manual Tab set for dividing the sections of the manual 3.5in. looseleaf diskette holder Manual sections for using the Model 8 with TxBASIC 3-Ring binder for the manual Tab set for dividing the sections of the manual 3.5in. looseleaf diskette holder Motorola manual for the 68332 processor Motorola manual for the CPU Motorola manual for the Time Processor Unit
Diskettes DISK-D-8-AZC DISK-D-8-TXB DISK-D-8-XCT	Onset C Libraries for using Aztec C with the Model 8 TxBASIC, Build Your Own Basic and TxTools software CrossCut program software
Misc. Goody Bag 1/4in. Nut Driver	Thermistor, 10K resistor and FET for experimenting Nut driver for fastening the Squishy bus to the PR-8

What to do if something is Missing or Broken upon Arrival

If you inspect the contents of your development kit and find that an item is missing or damaged, contact Onset Computer Product Support for assistance at (508) 759-9500.

Tools Required to Connect and Test the Tattletale

NOTE: The “[Testing the Operation of the Tattletale Model 8](#)” procedure on page 2-6 gives you the option of using an actual temperature sensor by soldering two components (from the goody bag) and a jumper wire to the IO-8 prototyping board. The tutorial also takes advantage of the temperature sensor for a more comprehensive tutorial. By doing so you will have actual data being read by the Tattletale. You may however need to either unsolder the components when done or purchase another IO-8 prototyping board (or make your own) before adding your own circuitry to the Tattletale. You can also leave the temperature sensor on the IO-8 and add your own circuitry to the other A-D channels of the Tattletale.

There are no tools required to connect the Tattletale to your computer; however, to test the battery power, communications cable and channel 7 of the A-D converter of the Tattletale, you will need:

Anti-static wrist strap

Soldering iron

Solder

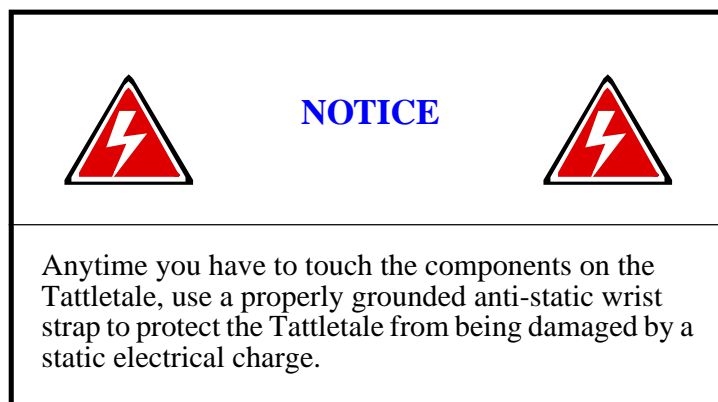
10K thermistor from the goody bag (DALE 9M1002-C3 or YSI 44006)

10K 0.1% resistor from the goody bag (a 5 or 10% resistor can be used if precise results are not needed)

Jumper wire

Safety Precautions

While installing or removing the Tattletale board wear a properly grounded anti-static strap. This will protect the board in the event that your body is holding a static electrical charge which can damage the CMOS chips on the Tattletale.



Proceed to [Section 2 -How to Connect and Setup the Model 8](#) for step-by-step instructions on installing and verifying the operation of the Tattletale.

Section 2 - How to Connect and Setup the Model 8

The Prototyping Board

The prototyping board that comes with the Model 8 development kit allows you to create your own interface for the Tattletale. A prototyping board is required to use the Tattletale Model 8. An IO-8 prototyping board is included in all the development kits. If you bought the deluxe kit you will also have a PR-8 prototyping board in the development kit (it's the 5 x 7 inch board). We will be using the IO-8 board for the tutorials. You can design your own interface using the specifications in [Section 6 - Hardware and Interface Specifications](#). The use of the PR-8 board will be covered in Section - 6.

In order to test the power supply, communications cable and channel 7 of the A-D converter of the Tattletale, it will be necessary to solder a couple of components to the supplied IO-8 board. After completing the operational testing in this section and the tutorials, you can remove the components from the prototyping board and start designing your own interface. You can also leave the temperature sensor on the IO-8 and add your own circuitry to the other A-D channels of the Tattletale.

Attaching the Tattletale Model 8 to the IO-8 Prototyping Board

Before plugging the Tattletale into the prototyping board, two components from the goody bag and a jumper wire need to be soldered to the board.

Installing a Temporary Sensor onto the Prototyping Board for Testing

Refer to Figure 2-1 and Figure 2-2 for the following steps.

NOTE: If the temperature sensor will be removed after going through the tutorials, you can use a less precise resistor. The only difference is that the temperature read will be less accurate.

1. Solder the 10K thermistor to the pin B11 etch extension and the etch extension for pin B12.

NOTE: Use a heat sink connected to the thermistors leads to protect the thermistor while soldering it to the board.

2. Solder a small jumper wire between the pin B12 etch and the etch extension.
3. Solder the 10K 0.1% resistor to the pin B20 etch and the etch extension for pin B12.

NOTE: A 5 or 10% resistor can be used if precise results are not needed.

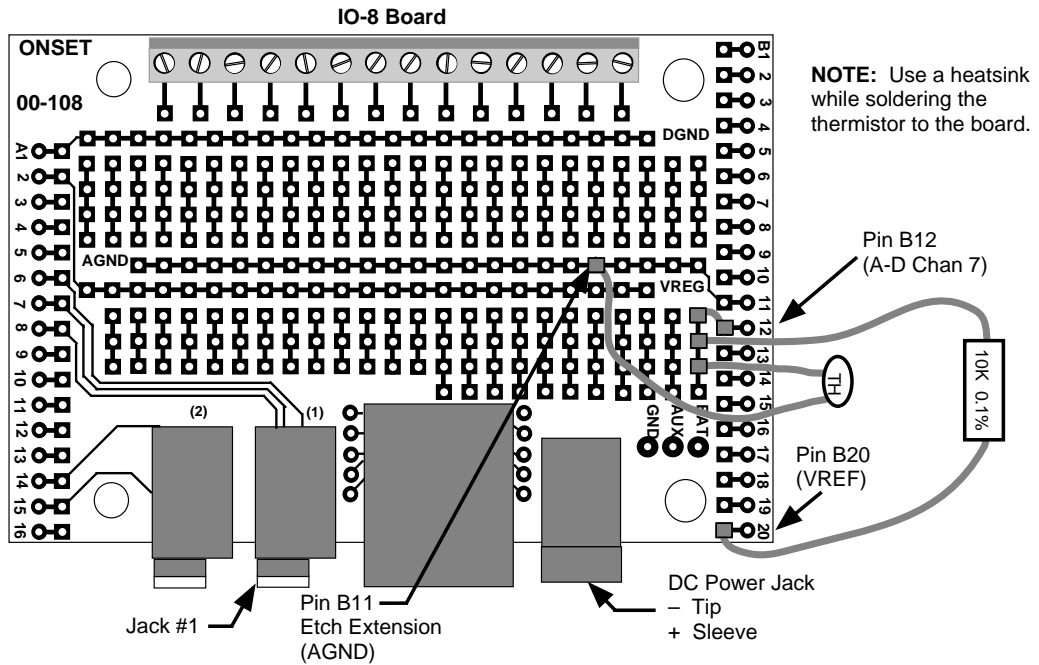


Figure 2-1: IO-8 with Test Components Soldered onto it

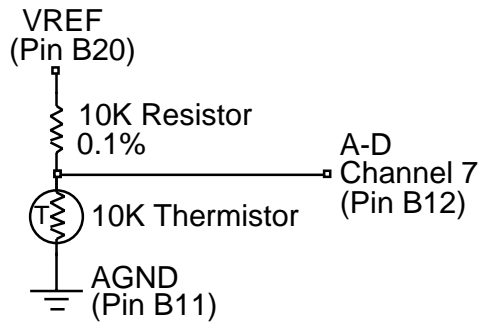
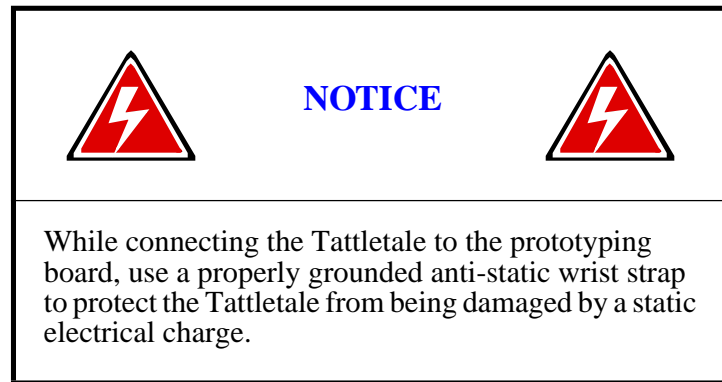


Figure 2-2: Schematic of the Test Circuit

Connecting the Tattletale Model 8 to the Computer



1. Carefully remove the Tattletale from the static protection bag. Touch only the edges of the Tattletale board.
2. Line up the pins of the Tattletale with the sockets of the IO-8 prototyping board (there is a short connector on one end) and carefully mount the Tattletale onto the prototyping board. Make sure that the Tattletale pins are inserted completely into the sockets on the board.
3. Connect the PC-3.5 communication cable to either COM Port 1 or COM Port 2 on the back of your computer. No other port should be used.

Record the serial port to which you have connected the cable. This data is required during the setup of the Tattletale.

4. Insert the other end of the communication cable completely into jack #1 on the IO-8 prototyping board (refer to Figure 2-1).

NOTE: DO NOT connect a battery or power supply to the Tattletale at this point.

Installing the Tattletale Software

This procedure describes how to setup and operate the Manx Aztec C68k/ROM development environment for Tattletale Model 8 development on an IBM PC. We have put together an installation strategy and provided several libraries and utilities to provide a much simpler startup than the one in the several hundred page Manx manual.

Much of the Manx documentation deals with complexities arising from supporting a variety of CPU and board configuration options. We hope to shield you from these complexities by providing an installation procedure which limits support to the Tattletale Model 8.

We strongly recommend that you use Onset libraries, utilities and procedures while becoming familiar with the Model 8. After that, you can contact us for help if your application has requirements which are not properly addressed by our defaults.

Installing the Onset C Libraries and Directories onto the Hard Drive

This procedure creates a new root directory named TT8.

1. While you are at the DOS C: prompt on the IBM PC (and at the root directory), insert the C Libraries disk (P/N DISK-D-8-AZC) into the floppy drive.
2. At the DOS C: prompt, enter the command **xcopy a:\ c:\ /-y /v /s** to copy the files from the floppy disk to the hard drive and automatically create any new directories needed. (The **-y** causes the program to ask you if you want to overwrite any existing files or directories with identical names. The **/v** verifies the copying process and the **/s** creates and copies any sub-directories.)

Installing CrossCut onto the Hard Drive

1. Switch to the TT8\BIN directory.
2. Insert the CrossCut disk (P/N DISK-D-8-XCT) into the floppy drive and enter the command **xcopy a:\ c:\TT8\BIN\ /-y /v /s** and press RETURN to copy the files from the floppy disk to the TT8\BIN directory on the hard drive.
3. If you are updating CrossCut from an earlier version, please delete all copies of CROSSCUT.CRS, CROSSCUT.CFG and CROSSCUT.HLP before loading this new version.

NOTE: You must place the files CROSSCUT.EXE (the CrossCut executable file), CROSSCUT.CFG (CrossCut current serial port configuration), CROSSCUT.HLP (the CrossCut help file) and CROSSCUT.CRS (a resource file containing the menus and dialog boxes used by CrossCut) in the directory shown so that the program is in your PATH. This way, no matter where you execute CrossCut from, you will always have access to the help system and the resources. Also, it keeps multiple configuration files from being created in each directory you work in.

NOTE: CrossCut will not start without access to the resource file CROSSCUT.CRS.

Installing the Aztec C Software onto the Hard Drive

NOTE: This software is only included when the Deluxe development kit or the C development kit is purchased.

We assume that you will be installing the Aztec compiler to a hard disk. We also assume that you will be installing to a root directory and that you will use the default destination directory “\az68” offered by the installation tools. You can select a different drive and destination, but your life will be greatly simplified if you accept the default. If you choose not to accept the defaults, we assume you’re a DOS expert and prepared to read and master the batch files and makefiles.

1. While you are at the DOS prompt on the IBM PC, insert the first Aztec C distribution diskette (labeled disk 1 of 3) into the disk drive.

2. Change to the drive with the disk in it and type “**install**”. You will see a screen like the one below:

Aztec C68k/ROM Installation Options:

Installation options

Source drive: a:

Destination directory: c:\az68

Install programs

Install include files

Install library source

Install optional components:

Remote Debugger *

Install these object libraries:

C libraries

IEEE math emulation libs

68881 math libraries

68040 math libraries

Library memory models:

Small Large

Library int sizes:

16 bit 32 bit

3. Use the mouse (or the tab and space keys) to adjust the settings so that they match the screen above, then click on the button labeled “**Proceed**”. You only need to install the options labeled “programs” and “include files” to run the Model 8, but you can also safely install any of the additional options. If you have the space, you may want to install everything. The installer program will prompt you to insert each diskette as needed.

* **NOTE:** The Remote Debugger program is purchased separately and is located on a fourth disk in a separate software box in the development kit. If it was not purchased, do not check its box in the Aztec C installation window.

Modifying your Autoexec.bat File

The installed batch file “\tt8\bin\aztt8dev.bat” contains commands that prepare your system for Model 8 development. We recommend adding the command string “call \tt8\bin\aztt8dev” to your autoexec.bat file to run the batch file automatically on restart, or you can type “call \tt8\bin\aztt8dev” every time you start your computer. If DOS complains about “Out of environment space”, read the additional comments and instructions in the “aztt8dev.bat” file.

Verifying the Software Installation

Restart your PC, then at the DOS prompt type “**set**” <CR> (make sure you have run “\TT8\bin\aztt8dev” first). You will see a list of DOS environment variables which should contain settings for “CCOPTS”, “CLIB68” and “INCL68”. These set up the path names and compiler options for the Aztec C compiler. This is used in place of the “\az68\bin\az68.bat” files that Manx provides. Next, type “c68” <CR>. This should invoke the compiler. If successful, you should see something like:

```
Aztec C68k/ROM 5.2b Dec 13 1994 12:25:30
Copyright 1994 by Manx Software Systems, Inc.
No input file was specified!
```

If this is not displayed, check the autoexec.bat file and make sure that you properly added the batch file “\tt8\bin\aztt8dev.bat”. You must restart the computer after making any changes to the autoexec.bat file for the changes to take effect.

Testing the Operation of the Tattletale Model 8

Before you start interfacing your own products to the Tattletale, test it with the following procedure to make sure that the battery or power supply, communications cable and channel 7 of the A-D converter is functioning correctly.

1. At the DOS prompt, change to the CROSSCUT directory.
2. Enter the command **dir** and make sure that the CROSSCUT.EXE file is in the directory. If it is not, double check that you are in the right directory.
3. Enter the command **crosscut -p 1 -b 9600** (If you connected the Tattletale to COM port 2, enter -p 2 instead of -p 1.) The CrossCut program will start and open a blank window.

NOTE: The command entered in step 3 automatically sets the com port to the number you specified and the baud rate to the second number you specified. You can enter these manually by selecting “**Port Setup**” from the “**CommPort**” menu.

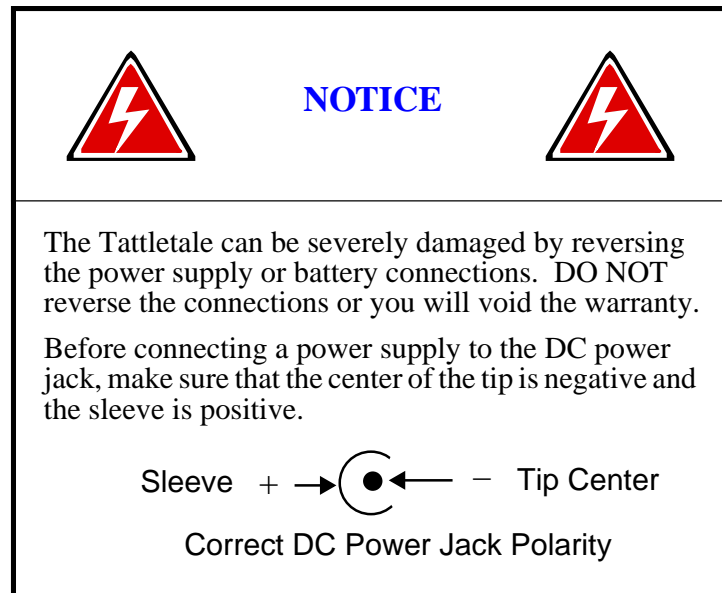
The remaining settings should be left at the default settings listed in Table 2-1.

The default settings can be changed as you design your own interface; however, during the installation leave them at the default settings.

Table 2-1: Model 8 Communication Settings

Protocol	Setting
Baud Rate	9600
Data Bits	8
Stop Bits	1
Handshake	None
Parity	None

Later when you exit the CrossCut program, the settings will be automatically saved for the next session.



Connect a battery or a 7 to 15V power supply to the prototyping board's DC power jack. Make sure that the polarity is not reversed when you connect the power to the board, or you may severely damage the board. We strongly recommend a current limited power source for the first time user and during development. The Model 8 uses up to 150mA of current.

NOTE: DO NOT use a power source that will supply more than 500mA of current or the Model 8 can be severely damaged if a short circuit should occur. We recommend using a 300mA, or less, current limited supply during development for added protection.

4. As soon as the power source is connected, the following should be displayed:

```
Tattletale Model 8
Onset Computer, Pocasset MA USA
TOM8 Vx.xx, PIC Vx.xx, Copyright 1994
TOM8>
```

5. Press the RETURN or ENTER key. The TOM8> prompt should be displayed again, if you get no response from the return or enter key or any other key, refer to the [“What to do if the Operation Test Fails”](#) procedure on page 8-4.

When the TOM8> prompt repeats, it proves that the serial interface can send as well as receive so you know that the communications cable and power source are working fine. You are now ready to use the Tattletale with an actual program.

6. Pull down the **“Tattletale”** menu and select **“Load S-record”**. The standard open file dialog box will appear. From the TEMPTTEST sub-directory of the TUTORIAL directory (located in the TT8 directory), load the file named **“TEMPTTEST.RHX”**. Once the file is loaded, a message reading **“Load Successful”** will be displayed.

7. We will be explaining the complete operation of the CrossCut program in section 3, so for now enter “g” and press the RETURN key. The program will start executing.
8. The program will ask you to enter which channel the sensor is connected to. Enter 7 and press the RETURN key.
9. Enter the number of readings you want displayed and press the RETURN key. If you soldered the suggested components onto the IO-8 board correctly, the current temperature in Fahrenheit will be displayed the number of times you entered. This verifies the proper operation of the software, the computer’s serial port, the communications cable, the Tattletale, the battery or power supply and channel 7 of the A-D converter. You can run the program as many times as you want. When you are finished, select “Quit” from the “File” menu. The TEMPTEST directory also contains the source code for the TEMPTEST program if you want to experiment with the temperature circuit.

This completes the installation and operational verification of the Tattletale. For additional information on the CrossCut software (and a tutorial for CrossCut operation) refer to [Section 3 - Operating the CrossCut Program](#).

To write your own programs in C (and to perform several tutorials using C) refer to [Section 4 - C Programming Guide](#) and [Section 5 - C Library Reference](#). These sections show the details of all the commands available to operate the Tattletale.

Refer to [Section 6 - Hardware and Interface Specifications](#) if you are ready to start designing products using the Model 8.

If you purchased the PR-8 board and want to start using it immediately (instead of the IO-8 board), refer to [Section 6 - Hardware and Interface Specifications](#) for instructions on installing the Squishy bus connectors and mounting the Model 8 to the PR-8 board.

Section 3 - Operating the CrossCut Program

Introduction

What is CrossCut and how is it used?

The user interface used by CrossCut was developed by Borland International for use in their DOS based language products. They packaged this interface and have made it available to programmers through their Pascal and C++ programming languages. This user interface, called Turbo Vision, looks and acts a little like Microsoft Windows but runs in text mode only. This interface allows using a Microsoft compatible mouse to select and manipulate items.

If you don't have a mouse, you can use special keys and key combinations to select and manipulate objects on the screen. Table 3-1 on page 3-4 lists the keyboard equivalents for mouse actions.

The methods of using the mouse, clicking on an object to select it and dragging to move an object are all similar to techniques used by Windows. At this time, only the left mouse button is recognized.

CrossCut running on a host IBM PC works in collaboration with a companion TOM8 mini-monitor program running on the Tattletale along with Aztec C on the PC to form a complete C development system.

The TOM8 mini-monitor program in the Tattletale communicates with the host computer through a serial port to accept and execute C programs, interact with a user and offload logged data for final analysis.

The development process for creating and debugging a C program is shown in Figure 3-1. Aztec C is an important part of the development cycle since it allows you to write and debug C programs with ease. You will notice that the tutorial for CrossCut is very short. CrossCut was designed for ease of use and the majority of this section will be used for reference only.

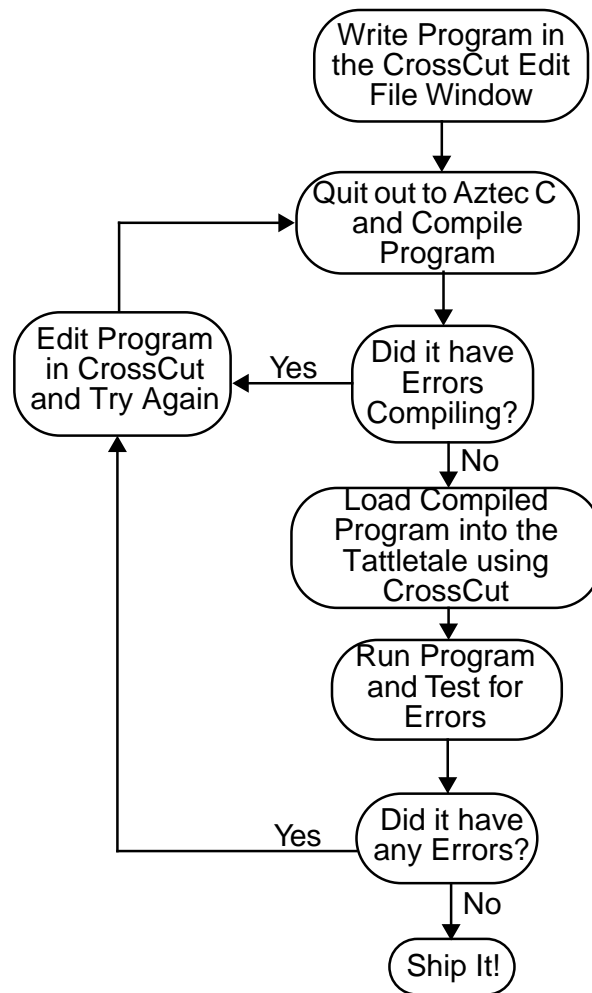


Figure 3-1: Flow Chart of the Development Cycle for a C Program

Learning to Use CrossCut on the IBM PC (or Compatible)

Getting Started with CrossCut

NOTE: CrossCut is only available for DOS, there is not a Windows version but it can run from Windows as a DOS application.

The following procedure shows you step-by-step how to start the CrossCut program, load a completed C program and start that program.

1. From the DOS prompt, go to the directory that has the CrossCut executable file in it and enter the command **crosscut**. The program will launch and display a blank window (which is called the “Terminal Window” see Figure 3-2 on page 3-5). Refer to the [“Explanations of CrossCut Menu and Window Options”](#) on page 3-5 for detailed descriptions of each of the CrossCut commands as you perform this procedure.
2. With the communication cable already connected to the Tattletale and to the computer, connect the power supply or battery. The Tattletale startup message will be displayed and the TOM8> prompt will be displayed.
3. Press the ENTER key. The TOM8> prompt should be displayed again. This verifies that the serial interface is operating correctly.
4. Pull down the Tattletale menu and select **“Load S-record”**.
5. Locate the file named **“HELLO.RHX”** in the TUTORIAL directory and click the **OK** button. Once the “Load Successful” is displayed enter **“g”** and press the **RETURN** key. The program will be executed and will display “Hello World” on the screen.

You now know how to load programs written in C. The tutorial in Section 4 will show you how to convert a C program from an editing program to a compiled C program. The .RHX files are loaded into and executed from RAM and the .AHX files are loaded into flash.

This completes the CrossCut tutorial. The other CrossCut commands and options are explained in detail in the [“Explanations of CrossCut Menu and Window Options”](#) on page 3-5. Please proceed to [Section 4 - C Programming Guide](#) for a tutorial on using C and refer to [Section 5 - C Library Reference](#) for detailed explanations of each C library command.

Keyboard Shortcuts for Mouse Actions

If your IBM PC computer does not have a mouse it will be necessary to use these keyboard commands. Table 3-1 shows all the keyboard commands that are usually operated by using the mouse.

Table 3-1: Keyboard Shortcuts for Mouse Actions

Menu Option	Command Key	Menu Option	Command Key
Main Menu Selections			
File	Alt-F	CommPort	Alt-C
Edit	Alt-E	Windows	Alt-W
Search	Alt-S	Help	Alt-H
Tattletale	Alt-T		
Sub-Menu Selections			
Open	F3	Load S-rec	Alt-L
Save	F2	Snd XMODEM	Ctrl-S
Quit	Alt-Q	Rcv XMODEM	Ctrl-R
Cut	Shift-Del	Hex Display	Alt-X
Copy	Ctrl-Ins	Capture to File	Alt-Z
Paste	Shift-Ins	Port Setup	Alt-P
Clear	Ctrl-Del	Next	F6
Paste Date/time	Alt-D	Previous	Shift-F6
Find Again	Ctrl-L		
Action	Key to Press	Action	Key to Press
Dialog Box			
Cancel	Escape	Toggle check box	Space
OK	Enter	Toggle radio button	Space
Next group	Tab	Next item in group	Up/Down arrow
Editing Controls			
Move cursor	Arrow keys	Move cursor a page	PgUp, PgDn
Move cursor word left	Ctrl-Left arrow	Move cursor word right	Ctrl-Right arrow
Move to line start	Home	Move to line end	End
Delete line	Ctrl-Y	Delete to word end	Ctrl-T
Delete character to left	Backspace	Delete character	Del
Toggle insert mode	Ins	Marking blocks	Shift-Arrow keys
NOTE: If a menu item is "grayed out" it is unavailable in the current mode.			

Explanations of CrossCut Menu and Window Options

Introduction

This part of the manual describes the window environment and the various areas of the window you need to understand CrossCut. A description of each menu and sub-menu option is also included.

Explanation of CrossCut Window Types

Terminal Window

When you first start CrossCut, you will immediately see one window that fills the screen (see Figure 3-2 and Table 3-2). This window displays any characters that are received by the serial port (which is anything sent out from the Tattletale). Keyboard characters go out to the Tattletale while the Terminal window displays any replies being sent from the Tattletale.

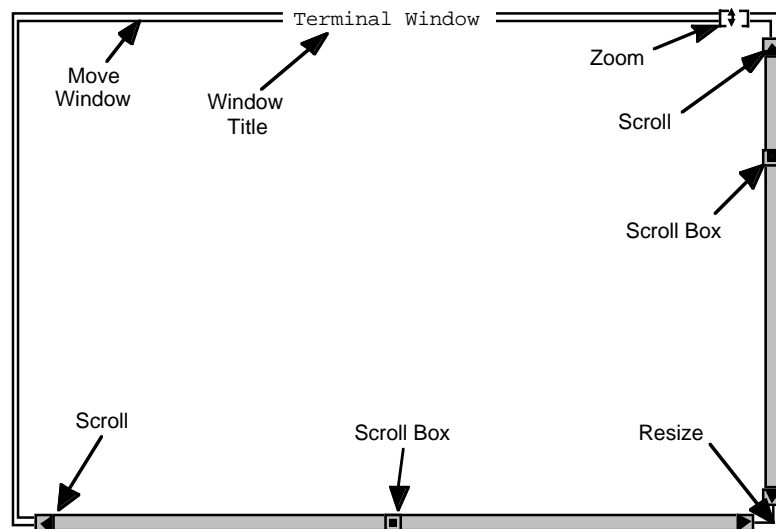


Figure 3-2: Terminal Window Display

Table 3-2: Terminal Window Feature Descriptions

Feature	Description
Title	This is always "Terminal Window" for this window but will vary for the Edit file window.
Zoom	By clicking on the Zoom box, you can toggle the size of the window between full screen and a smaller size.
Move	The Move arrow points to the top frame of the window. You can move the window by clicking and holding anywhere along this top frame and then moving the mouse. The window frame will follow the mouse until you release the button.

Table 3-2: Terminal Window Feature Descriptions (Continued)

Feature	Description
Scroll Arrows	Scroll arrows cause the text in the window to scroll one line in the direction of the arrow. Click and hold on these arrows to scroll continuously.
Scroll Box	You can move more quickly through a document by click-hold-dragging on the Scroll Box. Moving the Scroll Box toward the top moves closer to the start of the document and moving the Scroll Box toward the bottom moves closer to the end of the document. The window does not scroll until the Scroll Box is released.
Resize	The size of the window can be adjusted by click-hold-dragging on the Resize corner of the window frame.
<p>NOTE: The Terminal window has a 16000 character circular buffer. This holds about 8 pages of packed text. When the buffer fills, the oldest characters are overwritten. The scrolling is adjusted to keep the characters in order and you cannot scroll back too far.</p>	

Edit File Window Description

When you open a program file or select “New” from the “File” menu, an Edit File window is displayed (see Figure 3-3, Figure 3-4 and Table 3-3).

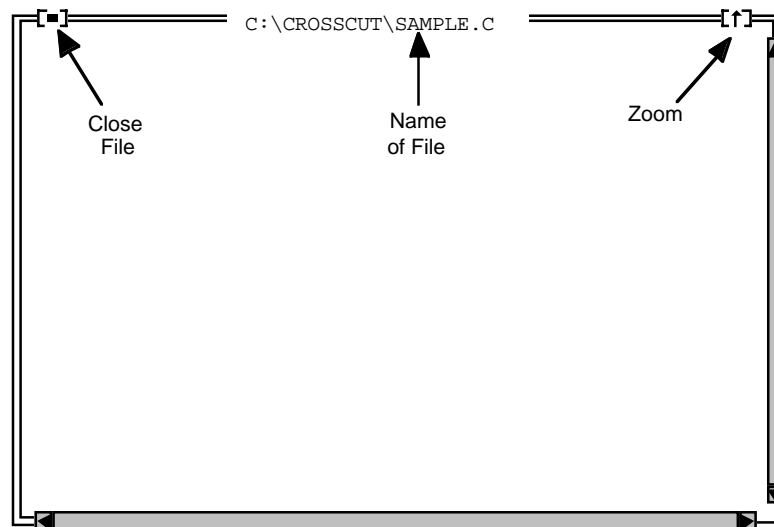


Figure 3-3: Edit File Window

Table 3-3: Edit File Window Feature Descriptions

Feature	Description
Close File	The Close box is in the upper left corner (called Cancel in a dialog window). The Close box attempts to close the DOS file displayed in this window. You will be asked if you want to save the changes if the window text has been modified since the last save. You can agree to save changes, throw out all changes since the last save or cancel the close.
Name of File	Displays the full path name of the DOS file.
Zoom	By clicking on the Zoom box, you can toggle the size of the window between full screen and a smaller size.

NOTE: Characters are entered just before the underline “_” cursor. Backspace removes characters before the cursor and Delete removes characters at the cursor. If a block cursor is displayed, the editor is in overwrite mode and each new character typed will overwrite the one under the cursor. The cursor type and editing mode are selected with the INS and DEL keys.

```

File Edit Search Tattletale CommPort Windows Help
C:\CROSSCUT\SAMPLE.C
/* Tutor2.c - Loading a program to the Model 8 */
#include <stdio.h> /* for printf() */
#include <tt8lib.h> /* for InitTT8() */
int main(void)
{
    InitTT8(NO_WATCHDOG,TT8_TPU); /* SEE
TT8LIB.H */
    printf("Hello, World!\n");
    ResetToMon();
    return(0);
};
F2 Save AltR Run AltL Load Alt Y Syntax AltO Offld Alt P Port

```

Figure 3-4: Typical Window after Opening a CrossCut Program File

To select a block of text, click and hold the mouse button over the beginning of the block and drag the mouse to the end of the block. When you release the mouse button, the block will be selected. You can then cut, copy, paste or clear this block. Moving the cursor causes the selection to be lost, however. This editor also has a limited Undo capability.

NOTE: As soon as you move the cursor, the Undo buffer is cleared.

The windows continue to exist and be displayed even if they are not in the front (active) window. A window is in the background if it has no close box nor a zoom box.

File Menu Option Descriptions

NOTE: Typing the underlined character while the menu is open (pulled down) will execute the command (this applies to all the IBM PC menus).

The File menu contains 10 sub-selections (see Figure 3-5 and Table 3-4).

File	Edit	Search	Tattletale	CommPort	Windows	Help
New						
Open...		F3				
Close						
Save		F2				
Save as...						
Print						
Print selection						
Change dir...						
DOS shell						
Quit		Alt-Q				

Figure 3-5: File Menu Options

Table 3-4: File Menu Option Descriptions

File Menu Option (Alt-F)	Description
<u>N</u> ew	Opens a new "Untitled" window to enter your program into.
<u>O</u> pen (F3)	Opens an existing program file (see Figure 3-6). When you open a program file, the text of the file is viewed in the Edit File window (Figure 3-3). When you select the open option, you are presented with the open file dialog box (see Figure 3-6).
<u>C</u> lose	Closes the currently active program file. If changes have been made in the file since it was opened, you will be asked if you want to save changes before closing.
<u>S</u> ave (F2)	Saves the program file in the active window to disk. If this is a new file, you will be prompted for a file name to save your program as.

Table 3-4: File Menu Option Descriptions (Continued)

File Menu Option (Alt-F)	Description
Save As	Allows you to save the program in the active window to disk under a new name. This is useful when you wish to experiment with your program, but you don't want to make the changes permanent to your original file. If the file already exists, it will ask to confirm before overwriting the file.
Print	Sends the program in the screen buffer to the DOS PRN device for printing. No header or page eject commands are sent to the printer.
Print selection	Sends the currently selected text to the printer. If no text is currently selected, you will be notified and no action will take place.
Change dir	Allows you to change the current DOS working directory. You will be in this new directory when you exit CrossCut.
DOS shell	<p>Suspends CrossCut and launches a new copy of the DOS shell. This also causes the serial port to be closed. You can execute any DOS commands here even other communications programs. When you're done, use the DOS command EXIT to return to CrossCut and the serial port will be opened in the same state it was in before (any characters that were sent to the serial port while running the second DOS session will be lost).</p> <p>NOTE: CrossCut is still in memory so there will be limited memory for other programs.</p>
Quit (Alt-Q)	Exits CrossCut, frees up its memory and returns to the DOS command interpreter.

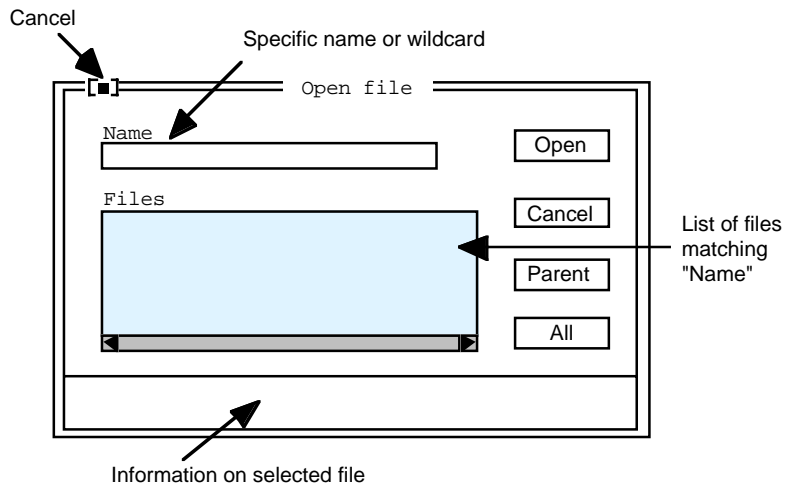


Figure 3-6: Open File Dialog Box

To Cancel this dialog, you have three choices: click on the Cancel button (under the Open button), click on the Cancel box or press the Escape key.

By default, the Name box will contain *.* and display all files and subdirectories in the current directory. You can enter a wild card string (with optional path) here and the Files box will attempt to show all files matching this string.

NOTE: If you type in the name of a file that doesn't exist, a new window will open with that name. Then if you save the window, a file of that name will be created on your DOS disk. There is always a ".." selection (parent directory) in the Files box (but it is easier to use the parent button). There is information about the currently selected file at the bottom of the dialog. To choose a file for opening, click OK (or press the ENTER key) when the file you want is listed in the "Name" field or double click the file name in the "Files" field.

The bottom button in the "Open" file dialog box can be set for up to five extensions of your own (in addition to the "All" and "Dir" options which are always available). Figure 3-7 shows the file name extension dialog box if you held down the shift key while clicking the "All" button. The bottom button is called a rotating button and will cycle through several options when clicked (the default settings are shown in Figure 3-7). To add your own extensions or to modify the existing list, hold down the Shift key while you click the button labeled "All" and enter up to three characters in each box. These extensions are saved in the CFG file when you exit CrossCut. The next time you want to use one of the new extensions, just click on the "All" button several times until the desired extension is displayed.

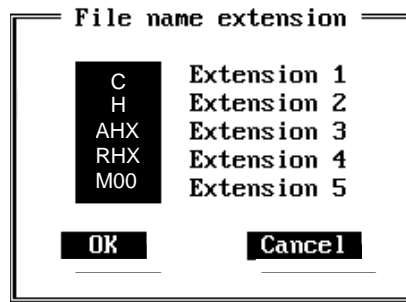


Figure 3-7: File Name Extension Dialog Box

Edit Menu Option Descriptions

The Edit menu contains 7 sub-selections (see Figure 3-8 and Table 3-5).

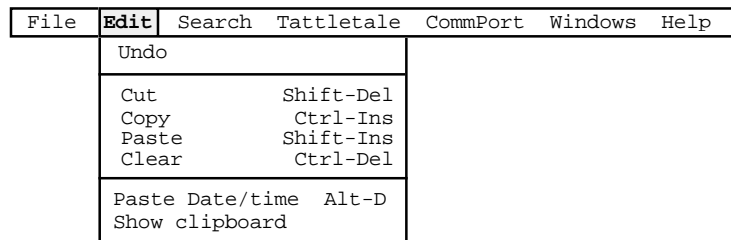


Figure 3-8: Edit Menu Options

Table 3-5: Edit Menu Option Descriptions

Edit Menu Option (Alt-E)	Description
<u>U</u> ndo	Select this item to undo an editing action. NOTE: Once you move the cursor, all Undo information is lost.
<u>C</u> ut (Shift-Del)	Remove the currently selected text and save it on the clipboard for pasting later.
<u>C</u> opy (Ctrl-Ins)	Save a copy of the currently selected text on the clipboard for pasting later. Unlike Cut, this does not remove the selected text.
<u>P</u> aste (Shift-Ins)	Insert whatever is on the clipboard at the cursor. Use Cut or Copy to get text into the clipboard.
<u>C</u> lear (Ctrl-Del)	Remove the currently selected text without saving a copy in the clipboard. You can Undo a Clear but not after the cursor has been moved.

Table 3-5: Edit Menu Option Descriptions (Continued)

Edit Menu Option (Alt-E)	Description
Paste <u>D</u> ate/time (Alt-D)	Inserts a text string showing the current date and time of your PC's clock into the document. If the Terminal window is currently selected, a date and time string will be sent out the serial port to the Tattletale. Selecting this item with no other keys pressed, sends a string of the form: 02/13/94 03:53:52 where 02 is the month, 13 is the day and 94 is the year. If you hold the Shift key down while this is selected, a longer date/time string of this form is used: Friday, February 12, 1994, 03:53 AM. Holding down the Control key while executing this command causes the country-code information in your configuration file to be checked. The date and time will then be pasted in the format normally used in your country (only if you had previously set this in the MS-DOS CONFIG.SYS file).
S <u>h</u> ow clipboard	Opens an editor-type window showing the contents of the Clipboard. You can edit the contents of the Clipboard using the normal editing commands. Only portions of the Clipboard that are selected are available for pasting into other edit windows, so be sure to select that portion of the Clipboard text before exiting this window.

Search Menu Option Descriptions

The Search menu contains 3 sub-selections (see Figure 3-9 and Table 3-6).

File	Edit	Search	Tattletale	CommPort	Windows	Help
Find... Find Again Ctrl-L Replace...						

Figure 3-9: Search Menu Options

Table 3-6: Search Menu Option Descriptions

Search Menu Option (Alt-S)	Description
Find	<p>A dialog box will appear (Figure 3-10) allowing you to search the document for a specific text string. Type the text string to search for in the box labeled "Text to find". You can check either or both of the options "Case sensitive" and "Whole words only" by clicking the mouse between the brackets to the left of the labels or by using the TAB and Arrow keys to work the highlight down to the selection and pressing the SPACE key to toggle the check on and off. Choose "OK" if all selections are correct or "Cancel" to forget the operation and close the dialog box.</p>
Find Again (Ctrl-L)	<p>Once a string has been found with Find, you can continue to look through the document for more occurrences of the same string. It is usually easier to use the Ctrl-L keyboard equivalent for this command.</p>
Replace	<p>A dialog box will appear (Figure 3-11) allowing a text string to be found and replaced with another string. Use of this dialog is similar to that of the Find dialog except that a second text box is available and there are more options.</p> <p>NOTE: Entering nothing in the "New text" box means that found text will be erased.</p> <p>The "Prompt on replace" is normally on (selected with an x). Checking "Prompt on replace" and "Replace all" will automatically look for the next occurrence of "Text to find" after each replace but will query you before replacing text.</p>

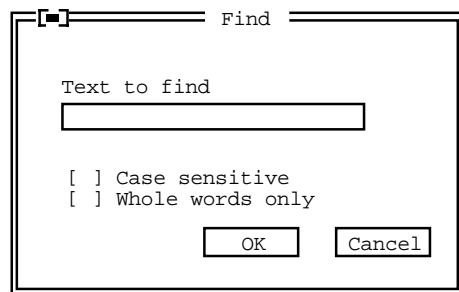


Figure 3-10: Find Option Dialog Box

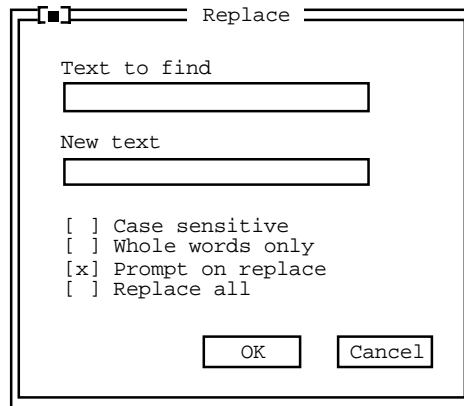


Figure 3-11: Replace Option Dialog Box

Tattletale Menu Option Descriptions

The Tattletale menu option contains 2 sub-selections (see Figure 3-12 and Table 3-7).

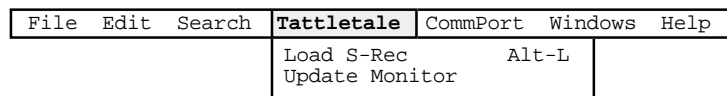


Figure 3-12: Tattletale Menu Options

Table 3-7: Tattletale Menu Option Descriptions

Tattletale Menu Option (Alt-T)	Description
<u>L</u> oad S-record (Alt-L)	This sends the proper commands to the TOM8 monitor to allow a high-speed (57600 baud) load of an S-record (produced by compiling your program with Aztec C or provided by Onset) to the Tattletale. You will be prompted to enter the file name of the S-record to load.
Update Monitor	This allows you to load a new version of the TOM8 monitor over an existing version of the monitor. To use this command you need to know the full path and the exact file name (TOM8Fxxx.HEX) or the update will be cancelled automatically. If you need to load a monitor onto a Model 8 that has no working monitor, you will need to call Onset Computer for a special utility and instructions.

CommPort Menu Option Descriptions

The CommPort menu option contains 8 sub-selections (see Figure 3-13 and Table 3-8).

File	Edit	Search	Tattletale	CommPort	Windows	Help
				Snd file ASCII...		
				Rcv file ASCII...		
				Snd file XMODEM...		
				Rcv file XMODEM...		
				Snd file XMODEM 1K Ctrl-S		
				Rcv file XMODEM 1K Ctrl-R		
				Hex display	Alt-X	
				Capture to file...	Alt-Z	
				Port setup...	Alt-P	
				Options...		

Figure 3-13: CommPort Menu Options

Table 3-8: CommPort Menu Option Descriptions

CommPort Menu Options (Alt-C)	Description
Snd file <u>A</u> SCII	Allows you to send a DOS text file out the serial port to the Tattletale.
Rcv file <u>A</u> SCII	Takes whatever comes in the serial port and stores it in a DOS text file until either an End-of-File character (Ctrl-Z) is received or the Cancel button is pressed. The incoming text will not be displayed in the Terminal Window.
Snd file <u>X</u> MODEM	Allows you to send a DOS text or binary file out the serial port to the Tattletale using the XMODEM protocol.
Rcv file <u>X</u> MODEM	Starts the XMODEM receive process for a text or binary file coming into the serial port. The incoming text will not be displayed in the Terminal Window.
Snd file <u>X</u> MODEM - 1K	Allows you to send a DOS text or binary file out the serial port in 1K blocks to the Tattletale using the XMODEM protocol.
Rcv file <u>X</u> MODEM - 1K	Starts the XMODEM receive process for a text or binary file coming into the serial port as 1K blocks. The incoming text will not be displayed in the Terminal Window.

Table 3-8: CommPort Menu Option Descriptions (Continued)

CommPort Menu Options (Alt-C)	Description
<u>H</u> ex display (Alt-X)	Toggles the Terminal window into and out of hexadecimal display mode. In this mode, any incoming characters are displayed in hexadecimal form in rows of 16 characters on the left side of the screen. The ASCII (printable) equivalent is displayed in 16 character rows on the right side of the screen. Toggling this mode always forces the new mode to start on a new line.
Capture to <u>F</u> ile (Alt-Z)	<p>Allows you to toggle the capture mode on and off. If toggling on, you will be presented with a file selection box to choose a DOS file in which to save all input through the comm port (a default name of CAPTURE.TXT is suggested and can be used by simply pressing the ENTER key). Also, the string "CAPTURE" is written to the lower-right corner of the display. In color mode, the status line background is changed to green as another reminder. When you toggle capture off, the file is closed and the status line returned to normal. The default mode is for overwriting an existing file. You can change this to append by holding down the Shift key while selecting Capture mode. You will be asked if you want to change the Capture mode to append.</p> <p>NOTE: To capture to a printer, use the special DOS file name PRN.</p>
<u>P</u> ort setup (Alt-P)	Allows you to set the comm port parameters as shown in Figure 3-14. These values will be stored in a file called CROSSCUT.CFG when you exit CrossCut. Notice that these items are groups of radio buttons and only one item of a group can be selected at any one time.
<u>O</u> ptions	The only items implemented here are 'Char delay' and 'Line delay' (see Figure 3-15). They only affect ASCII file transfers and program loads. If Char delay is non-zero, it adds this many milliseconds between characters to both ASCII file transfers and program loads. If Line delay is non-zero, it adds this many milliseconds after sending a carriage return for ASCII file transfers only.

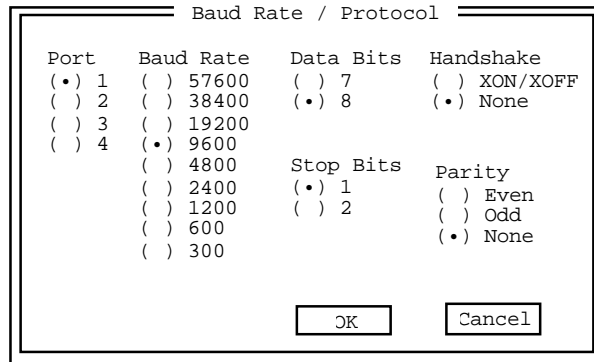


Figure 3-14: Baud Rate / Protocol Option Dialog Box

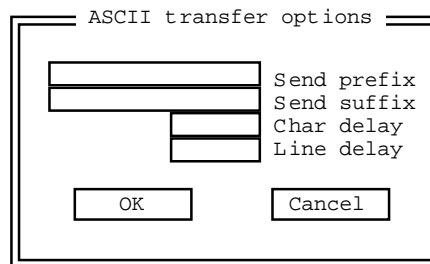


Figure 3-15: ASCII Transfer Option Dialog Box

Windows Menu Option Descriptions

This item contains 7 sub-selections (see Figure 3-16 and Table 3-9).

File	Edit	Search	Tattletale	CommPort	Windows	Help
					Tile	
					Cascade	
					Next	F6
					Previous	Shift-F6
					Screen resolution	
					Color screen	
					Blk/wht screen	

Figure 3-16: Windows Menu Options

Table 3-9: Window Menu Option Descriptions

Window Menu Options (Alt-W)	Description
<u>T</u> ile and <u>C</u> ascade	These allow you to automatically rearrange all the open windows on your screen.
<u>N</u> ext (F6)	These will make different windows the active window in forward or backward order. These are only needed if a particular window is not visible (in which case, the mouse can be used to make it the active window by clicking on it) or if you have no mouse and need to switch windows.
<u>P</u> revious (Shift-F6)	

Table 3-9: Window Menu Option Descriptions (Continued)

Window Menu Options (Alt-W)	Description
<u>S</u> creen resolution	If you have an EGA or VGA screen, this command toggles the screen into and out of a higher resolution mode. EGA is capable of a 43 line per screen mode and VGA is capable of a 50 line per screen mode. This has no effect on CGA screens.
<u>C</u> olor screen	Puts a color-capable screen into color mode. Unnecessary if you have a monochrome screen.
<u>B</u> lk/wht screen	Puts the screen in a black and white mode. This is very useful on LCD screens.

Help Menu Option Descriptions

This item contains 4 sub-selections (see Figure 3-17 and Table 3-10).

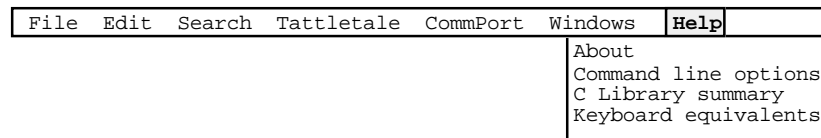


Figure 3-17: Help Menu Options

Table 3-10: Help Menu Option Descriptions

Help Menu Options (Alt-H)	Description
<u>A</u> bout	Displays a dialog showing the version number of CrossCut you are using.
<u>C</u> ommand line options	Shows options you can set when you start CrossCut from the DOS command line.
<u>C</u> Library Summary	Displays a list of C commands and keywords in alphabetical order. To see more of the list, use the arrow keys or use the mouse to move the scroll box. You can get a brief explanation of a command by double clicking on it with the mouse. Keyboard users can step through the commands with the TAB key and select a command by pressing ENTER. If there is a "See also..." highlighted selection, you can change to that topic by selecting it in the same way you selected the original command. When you're done with the help system, either click on the Close box or press the ESCAPE key.

Table 3-10: Help Menu Option Descriptions (Continued)

Help Menu Options (Alt-H)	Description
<u>K</u> eyboard equivalents	A list of CrossCut menus and editor actions and the key or key combinations that trigger them. It is essentially a copy of the next section. When you're done with the help system, either click on the Close box or press the ESCAPE key.

Software Change Information for CrossCut

All software change information is located in the Read Me file that came on the distribution diskettes.

Program Parameters Saved in the Configuration File

The following program parameters are saved in the CROSSCUT.CFG file and will be recalled the next time the program is started:

Comm Port number (1, 2, 3 or 4)

Comm Port baud rate (300, 1200, 2400, 4800, 9600, 19200, 38400 or 57600)

Comm Port parity setting (none, even or odd)

Comm Port stop bits (1 or 2)

Comm Port data bits (7 or 8)

Comm Port flow control setting (XON/XOFF control can be enabled or disabled)

Screen Mode can be low resolution or high resolution (for EGA and VGA)

ASCII file transfer, amount of delay between characters (not normally needed)

ASCII file transfer, amount of delay at end of line (not normally needed)

Whether capture mode overwrites an existing file or appends to it

The five user-defined file extensions used in the 'open file' dialog

The CommPort | Options "Send Prefix" string

The CommPort | Options "Send Suffix" string

Section 4 - C Programming Guide

Introduction

C is the programming language used to control the functions of the Tattletale. All of the commands and procedures shown in this section are entered through the CrossCut program which acts as the development area and then compiled in the Aztec C program. For an explanation of how to use CrossCut, refer to [Section 3 - Operating the CrossCut Program](#). This section and [Section 5 - C Library Reference](#) are to be used primarily for reference. A tutorial showing the procedure of writing a program in C, compiling it in Aztec C and then loading and running the program on the Model 8 is included for clarity.

We assume that you are proficient at programming in C and therefore we will cover only unique aspects of the Aztec and TT8 environment.

TOM8 - Tiny Onset Monitor

Assuming the Model 8 is properly connected to power and connected to a COMM Port at a rate of 9600 baud, when you first power up the Model 8 you are greeted with the following sign on:

```
Tattletale Model 8  
Onset Computer, Pocasset MA USA  
TOM8 Vx.xx, PIC Vx.xx, Copyright 1994
```

```
TOM8>
```

The Tattletale Model 8 TOM8 mini-monitor function is primarily for loading and running programs. It is not a debugger with register manipulation commands or breakpoints. The TOM 8 mini-monitor was deliberately designed this way to keep the memory requirements as small as possible and allow for maximum flexibility.

There are five two letter commands: MD (memory display), MM (memory modify), LO (load s-records), GO (jump to address), and ?, which prints a brief summary of each of the commands and the possible arguments.

Both letters of the command must be typed for it to be recognized (except for the GO command). The letters will always display in upper case, even if entered in lower case. The mini-monitor does a minimal system configuration and does not enable interrupts. The serial port is polled and therefore interrupting a command in process is not possible outside of asserting a reset.

Command Summary

Typing “?” will list the following summary of each of the commands:

```
-----
MD memory display [start addr] [end addr] [ size ;BWL]
MM memory modify [start addr] [ size ;BWL]
LO load s-records [offset addr] [ Go after load ;G]
GO jump to address [start addr]
? prints this listing
-----
```

Memory Display

```
MD [start addr] [end addr] [ size ;BWL]
```

This command displays lines of sixteen bytes of memory in hex and ASCII format. If no arguments are given then it assumes 0 as the starting address. If a start address is specified without an end address then it will display one line starting at the address given. Pressing the return key will display the next line. Pressing any other character terminates the command. If both a start and end address are given it will display all lines from the start to the end. Note that this is NOT interruptable—the only way to terminate the display of a broad range of memory is to reset. The BWL arguments display the hex section in byte, word or long format. The default format is byte.

Memory Modify

```
MM [start addr] [ size ;BWL]
```

This command displays a single byte, word or long from memory in hex format and allows you to change the value if you so desire. If no arguments are given then it assumes 0 as the starting address. Pressing the return key will display the next value. Pressing the . (period key) will go back to the TOM8 monitor. Trying to change the flash memory will have no effect.

Load S-records

```
LO [offset addr] [ Go after load ;G]
```

This command loads Motorola S-Records into memory. If an address is included in the S-Record then no offset need be entered. If an address is included in the S-Record and an offset is supplied then the program will load at the address plus the offset. If the starting address is lower than 0x20000 then you are addressing Flash and you will get a message that asks you if you really want to do this. The auto boot starting address for a program loaded into Flash is location 0x2000. If the address is less than 0x2000 then the command will be aborted because you risk overwriting the mini-monitor. At power up the mini-monitor looks at location 0x2000 and checks for a 0x00 or an 0xFF. If any other value is read at that location then the program immediately jumps to that address to start execution of the user program. To suppress this action, IRQ3 (pin 61 on PR-8 or A5 on the IO-8) needs to be tied to ground at power up. If the mini-monitor sees IRQ3 tied low it will not make the jump.

The ;G argument, if present, will cause a jump to the first location that was loaded as soon as the load is finished.

Jump to address

GO [start addr]

This command makes a jump to the address assuming there is a valid program entry point at the jump address. If not, the program will hang and will need a reset to recover. If you load an S-Record that contains address information (see the flag settings for s-records in the tutorial makefiles), GO will automatically jump to the address designated as the entry point in the S-Record (if no starting address is given).

Print Command summary

?

This command prints out the following summary listing of the commands and their arguments.

```
-----
MD memory display [start addr] [end addr] [ size ;BWL]
MM memory modify [start addr] [ size ;BWL]
LO load s-records [offset addr] [ Go after load ;G]
GO jump to address [start addr]
? prints this listing
-----
```

Learning to Use C on the Model 8

Tutorials

This section assumes that you have installed all of the Model 8 software and have verified the hardware connections to the host computer with the CrossCut program.

There are 3 sections to the tutorial. Each section increases slightly in complexity. Even if you are well versed with the Model 8, reading through the tutorial is helpful because it explains some of the subtle inner workings of the Model 8 that are not covered in any other section of the manual. Keep in mind that the tutorial examples were not designed to push the boundaries of the Model 8's power, they are simply meant to acquaint you with the Model 8 before you start working on your own application. The general information sections provide just that, general information on Model 8 programming practices.

Overview

Tutorial 1	Loading a program into the Model 8 using CrossCut
Tutorial 2	Compiling, linking and generating hex files
Tutorial 3	Simple logger application
General Info	Reducing current drain with the simple logger
General Info	Long term data logging techniques

Tutorial 1 - Loading a program into the Model 8 using CrossCut

Launching CrossCut

Go to the subdirectory "`\tt8\tutorial\tutor1`" and type "**Crosscut**". If you get a DOS message saying something similar to "Bad command or file name", you have not called `aztt8dev.bat`. The call to `AZTT8DEV` is essential because it sets up a path to CrossCut (and the Aztec applications) so it can run in any directory. `AZTT8DEV` also sets up the compiler options and the search paths to the libraries and header files.

Once CrossCut is launched and Terminal window will be displayed. To call `AZTT8DEV` automatically on startup see "**Modifying your Autoexec.bat File**" procedure on page 2-5 in **Section 2 - How to Connect and Setup the Model 8**.

The Model 8 connects to the PC via a serial port (usually COM1 or COM2). COM2 is the default setting for CrossCut, since many users have a serial mouse attached to COM1. To change the default port to COM1 (or any other COM port) use CrossCut's "Port Setup..." option in the CommPort menu. You can also launch CrossCut with a different port directly by setting the port flag to COM1 (type "`CROSSCUT -p1`" for COM1). These new settings are saved when you quit out of CrossCut.

Loading a Motorola S-Record

`TUTOR1.RHX` is the first file you will load into the Model 8. It is a Motorola S-Record, which is an ASCII representation of a compiled program generated by the Aztec linker. More on this in Tutorial 2.

To load the program, first make sure that the Terminal window is shown and the Model 8 is connected and powered. Press return a few times on your PC. You should keep getting the "`TOM8>`" prompt in the Terminal window. Go to the Tattletale menu and choose "**Load S-record**" (or press Alt-L). A file dialog appears showing the contents of the "`\tt8\tutorial\tutor1`" subdirectory (assuming you started in the CrossCut directory). Use the TAB & cursor keys or the mouse to select "`TUTOR1.RHX`" and press ENTER when that file is highlighted. A dialog appears showing the process of the download. After the download you should see the following prompt:

```
TOM8>  
load successful
```

The program is loaded to the first location in RAM, offset by `0x2000` for the stack (see the "**Tattletale Model 8 Memory (256K/256K Version)**" heading on page 4-19). It is not necessary to give a starting address because this S-Record was generated with address information and imbedded in the file during the load. Had an S-Record been generated without address information, CrossCut would prompt for an offset.

Type `GO` at the "`TOM8>`" prompt. This jumps to the starting address of the program (which the TOM8 monitor received from the S-Record) and executes it.

```
TOM8>go
jumping to address 002C2000
Hello, World!

Tattletale Model 8
Onset Computer, Pocasset MA USA
TOM8 Vx.xx, PIC Vx.xx, Copyright 1994
```

The program `tutor1.rhx` prints “Hello, World!” out the serial port and resets back to the TOM8 monitor.

Having used an S-Record to load and run your first program on the Model 8, it is now time to learn how this S-Record was created.

Tutorial 2 - Compiling, linking and generating hex files

In this tutorial we will compile, link and generate the program that was just loaded. This process was done for you by a makefile in Tutorial 1 (which calls the generic `TT8.MAK` makefile in the `\tt8\bin` subdirectory) so you could load and run a program right away on your Model 8. If you understand the process of compiling and linking; or have worked with C on DOS or UNIX systems, most of this tutorial will be a review. To introduce you to putting programs into non-volatile memory, we will load the Application version of the program and burn it to Flash memory after it is compiled.

Compiling -vs- Cross-Compiling

Before learning how to compile and link a source file, it is important to know some of the differences between developing native programs for your host computer (such as writing Windows applications for your PC), and cross-compiling C code on a host system to run on another system such as the Model 8. Though native development has recently evolved into more user friendly packages, cross-development currently resembles the iterative manual edit-compile-debug methodology of older native tools.

The Model 8 lets a host computer compile its programs because it is not designed to hold and manage the hundreds of source files, executables and libraries needed for such a task. It is much more suitable for a desktop computer with a large operating system to do that type of work. If the Model 8 had to compile its own programs (which can be very complex and span many source files), the power requirements for all the peripheral equipment needed to work with those files (hard drive, display, keyboard, etc...) would be overwhelming for a low power logger. The Model 8 does not do its own code generation because it makes more sense to develop a program with an operating system designed for development, and then send that program to the Model 8, designed as a controller / logger.

The Model 8's current mini-operating system, the TOM8 Monitor, takes up less than 8K. Its only responsibility is to load other programs to the Model 8. Once programs are loaded, they become the new operating system unless you reset to the monitor. It must handle all communication with the outside world. The function libraries provided for the Model 8 makes this communication easier.

Below is a comparison between a standard ANSI C program that you could compile for a regular desktop computer and a program modified to run on the Model 8.

A standard (ANSI C) “hello, world” program looks like this:

```
#include <stdio.h>
int main (void)
{
    printf("hello, world\n");
    return (0);
}
```

Here is the same program for the Model 8. You can view this file with CrossCut by choosing open in the File menu and selecting tutor2.c on the disk:

```
/* tutor2.c - Source code for the Tattletale Model 8 tutorial */
#include <stdio.h> /* for printf() */
#include <tt8lib.h> /* for InitTT8() */
int main (void)
{
    InitTT8(NO_WATCHDOG,TT8_TPU); /* SEE TT8LIB.H */
    printf("Hello, World!\n");
    ResetToMon();
    return(0);
}
```

There are a few significant differences between these two files:

- The addition of the **tt8lib.h** header file
- The function call `InitTT8()`
- The function call `ResetToMon()`

tt8lib.h

This header file is included for the two function calls `InitTT8()` and `ResetToMon()` mentioned later. In fact, most (but not all) Model 8 specific function declarations, structures and constants are declared in the **tt8lib.h** header file. It is almost always necessary to include **tt8lib.h** if you want to do anything useful with the Model 8.

The only time you would not need **tt8lib.h** is if you were running a program entirely in RAM (launched from the TOM8 monitor) that didn't have any calls to `STDIO` or Model 8 functions. This is because the TOM8 monitor does some minimal initialization to allow the loading of S-Records.

This is not recommended because eventually you'll probably want to burn the program into Flash memory. Once in Flash, the program is responsible for all of the Model 8's initialization – TOM8 initialization is never called.

When the Model 8 turns on, the first thing the TOM8 monitor does is check the Flash for a burned application (at location 0x2000). If there is one, TOM8 jumps to

it immediately without setting up the ports, baud rates, etc....

It is always best to design the program to run in Flash. That way you won't forget to do the initialization later when you make the switch.

InitTT8() This is the standard initialization function for the Model 8. It sets up the Model 8 with the default baud rates, pin settings, TPU functions etc.... Always call this function before any other Model 8 specific functions until you learn exactly which sections of the Model 8 you will need to initialize (even then you may still have to call InitTT8()).

NOTE: It is possible to use the Model 8 in RAM without calling any of initialization functions, but when you burn to Flash you will run into problems.

ResetToMon() This function will bring you back to the TOM8 monitor even if you burn the program into Flash.

Standard Model 8 Applications need some way to end, because there is no real operating system to which they can quit. An end could be anything from going into a low power stop mode until a pin goes high; to going through the entire program again. For now, this tutorial uses ResetToMon to go back to the monitor.

Now that you know what the added instructions do, you are ready to compile the program. We will go through the process of manually compiling, linking and generating S-Records, so you can understand why we chose the respective compiler flags. The 'make' utility, which greatly facilitates this process, will be introduced after performing the manual compiling process.

Compiling a Source File

To compile your first program, quit out of CrossCut and go to the "\tt8\tutorial\tutor2\" subdirectory. At the DOS prompt type:

```
C68 -PS -MC -MD -SM -O TUTOR2.R TUTOR2.C
```

You should get a prompt like:

```
Aztec C68k/ROM 5.2b Oct 13 1994 Copyright 1994 by Manx Software Systems, Inc.  
C:\TT8\TUTORIAL\TUTOR2>
```

This creates the object file tutor2.r which can then be linked with other object files such as the tt8 library. Below is a description of each of the settings used and why some were chosen.

Compiler Flags

- ps** Makes integers default to 16 bits long.
- mc** Object file uses the large code model. This results in some minor code bloat, but it makes it easier to write interrupt handlers and other more complex functions.
- md** Object code uses the large data model. This allows easy creation of arrays and static variables that are not limited to 32768 bytes with the small data model.
- sm** Adds the `__C_MACROS__` macro
- o file** Creates the `tutor2.r` object file that will be linked later.

Note about Static Arrays

If you create a very large array and are planning to burn its program to Flash, do not initialize the array in the declaration unless you want to use up all of your Flash.

Example: If you create a 100,000 byte array of characters as `char a[100000] = {1};`, the compiler will have to store a 1 and 99,999 bytes of trailing zeros in the object code. These extra 99,999 bytes of zeros are stored in the object file and will be burned to Flash. If you just type `char a[100000];` and initialize the first byte at runtime, space is allocated only during runtime.

Linking an Object File

Now we are ready to link this object file with the libraries needed to make it run on the Model 8. The following is a link for the RAM module (you don't need to type this in, since we will make the Flash version):

```
LN68 -O TUTOR2.RUN +C 2C2000 +J 2C2000 +S 2000 TUTOR2.R -LTT8 -LML16TT8 -LCL16TT8 -LTT8
```

Below is a link to Flash. Type the following at the DOS prompt (don't press enter between lines):

```
LN68 -O TUTOR2.APP +C 2000 +J 2C2000 +D 2C2000 +S 2000 TUTOR2.R -LTT8 -LML16TT8 -LCL16TT8 -LTT8
```

You should get a return message similar to the following:

```
Aztec C68K Rom Linker 5.2b Mar 17 1994 13:58:04
Base:000000 Code: 003c9e Data: 000466 Udata 000670 Total: 004774
```

Here is a description of the flags used:

Linker Flags

- o file** Creates the object file that will be converted to S-Records later.
- +c x** Start of the code segment (trailing x means hex number).
- +j x** Start of the stack pointer (top of stack). As things are added to the stack, this pointer decreases
- +s x** Size of stack.
- +d x** Start of the initialized data segment. Defaults is right after the code segment.
- l lib** Links the portions of the libraries `tt8` and `cl16tt8` needed. The math library is also linked in to show how all included libraries are ordered when they are all linked in, even though `ml16tt8` it is not even called in this program. The `TT8` library must be linked before and after `ml16tt8` and `cl16tt8` or a link error will result.

- t (optional) You can add this flag to one of the links to see which functions and variables are being included in the final module. This is nice to see where things will eventually be loaded.

For more information on how memory is used on the Model 8. See the [“Tattletale Model 8 Memory \(256K/256K Version\)”](#) heading on page 4-19.

Generating S-Records

Now that the tutor2 program is linked, it is time to convert it into something that can be sent to the Model 8.

This is how to generate an S-Record for the RAM module (don't type this in, since we will make the Flash version):

```
SREC68 -P4000 -A3 -B2C2000 TUTOR2.RUN
```

Below is a link to Flash. Type the following at the DOS prompt (don't press enter between lines):

```
SREC68 -P4000 -A3 -B2000 TUTOR2.APP
```

You will see the following return prompt:

```
Aztec SREC68 v5.2a 1-30-92 (C) 1982-1992 by Manx Software Systems, Inc.
```

Here is a description of the flags used:

S-Record Flags

- p n Chip size in kilobytes (n is a decimal number). We use a large size (4 megabytes) since S-Records for the Model 8 are contiguous. The hardware and the TOM8 monitor send S-Records to the appropriate chips. 4 Megs should cover the largest program size.
- a n Size of the address field. 3 bytes allows a 16 Meg S-Record, more than enough for any Model 8 application.
- b n Base offset (where the first byte is loaded) 0x2000 (this leaves space for the TOM 8 monitor, which resides below 0x2000).

The S-Record generated is tutor2.m00. The .m00 stands for the chip number, successive chips files are .m01, .m02, etc... This name doesn't make much sense for a Model 8 S-Record since there is only one file, and the name does not differentiate between S-Records that load to RAM and those that load to Flash. To accomplish this, rename it to tutor2.ahx:

```
rename tutor2.m00 tutor2.ahx
```

In case you are confused by all of the extension names, Table 4-1 shows a list of all of the extensions used during Model 8 development:

Table 4-1: Extension Naming Conventions

Extension	Description
.AHX	Application hex file (renamed S-Record)
.APP	Complete Model 8 Flash program (in binary format)
.C	C Source file
.DBG	Debugger Information
.H	C Header file
.ASM	An assembly file (see the Aztec manual)
.M00	Motorola S-Record Generated by Aztec's srec
.R	Object file generated by the compiler
.LIB	A Library File
.RHX	Run hex file (renamed S-Record)
.RUN	Complete Model 8 RAM program (in binary format)
.SYM	A Symbol Table

The file `tutor2.ahx` is now an S-Record that loads to Flash. It may have seemed like a lot to type for such a simple program, but, fortunately, you can create a makefile (or, better yet, use the supplied makefiles and batch files) that automates the procedures.

Using the Generic Makefile

A generic makefile "`tt8\bin\tt8.mak`" directory has been created for you. It can create both the `.ahx` and `.rhx` files from a single c source file.

NOTE: When editing a makefile, use only TABS to indent commands. If you use the SPACE bar to enter spaces the makefile will not work and an error message will be displayed (a very cryptic "syntax error" message that doesn't even show the line number).

You can use the makefile for your own programs by performing the following:

1. Copy one of the "makefiles" from the tutorial directory to your source file directory.
2. Edit the makefile and set the MAIN variable (near the top of the makefile) equal to your filename (without the `.c` extension). You may want to change the comments to your own.
3. Type "make" at the DOS prompt. The `.ahx` and `.rhx` files will be created automatically.

Don't worry if you get messages saying "*.ahx and *.rhx files were not found" the first time you type "make". This is because they do not yet exist during the first make. The makefile attempts to erase them before generating them with SREC. The second time you type **make**, these messages will not appear.

Now we will load the S-Record to the Model 8's Flash memory.

Loading the Program into Flash Memory

Loading a program to Flash is very similar to loading to RAM.

1. To load the program in Flash, first make sure that the Terminal window is shown and the Model 8 is connected and powered.
2. Press return a few times on your PC. You should keep getting the "TOM8>" prompt in the Terminal window.
3. Go to the Tattletale menu and choose "**Load S-record**" (or press Alt-L). A file dialog appears showing the contents of the "\\tt8\tutorial\tutor2" subdirectory (assuming you started in that directory).
4. Use the TAB & cursor keys or the mouse to select "TUTOR2.AHX" press ENTER when that file is highlighted. A dialog appears showing the process of the download. After the download you should see the following prompt:

```
TOM8>  
load successful
```

```
Target is Flash!  
start addr = 00002000  
end addr = 00006105
```

```
Ok to write flash between above addresses? (Y/N)
```

5. Press the "y" key to burn the program into Flash memory.

The progress of the burn and the Flash ID are shown.

6. At the TOM8> prompt type "**GO**". The "Hello, World!" message is once again displayed, and the program then resets back to the Monitor. To verify that the program is in Flash, turn off power to the Model 8 and then reapply it. You should get the "Hello, World!" message again automatically.

An application will automatically run at power-up or hardware reset if it is burned into Flash. A hardware reset can be performed by briefly connecting –MCLR to DGND.

When the Model 8 is turned on, the TOM8 mini-monitor recognizes if there is a program in Flash and jumps to it immediately. If you wish to jump to the TOM8 monitor rather than the resident Flash program, connect –IRQ3 to DGND during a power-on or reset.

The LED will light up (and stay lit) whenever `-IRQ3` is tied to ground, or after a forced reset. You should never keep `-IRQ3` tied to ground constantly, as it will corrupt timing during running applications and possibly cause the application to halt. Use the process only to get to the TOM8 monitor.

- To clear the program in Flash, there is a hex file called `NOPROG.AHX` file in the `"UTIL"` subdirectory. You can load `NOPROG.AHX` just as any other application hex file, but all it does is erase the portion of the Flash memory where the start of the program is found. When the Model 8 is powered on, the TOM8 monitor notices that there is no longer a program loaded so you get the regular sign-on message.

Now that you can compile and load programs, you can begin to make the Model 8 work for your particular application.

Tutorial 3 - A Simple Logger Application

This tutorial shows you how to create a simple logging program that does the following:

- Takes 100 measurements (1 per second) on A/D channel 1
- Stores them in a dynamically allocated heap.
- When finished, plays back values in a table and gives statistics.
- Allows you to offload the data to Crosscut using Xmodem protocol.

The file `tutor3.c` is in the `tutor3` directory. It is heavily commented, so you know exactly what is going on in the code. Since you already know how to type, compile, link, load and run a program for the Model 8, we will just go through sections of the code (`tutor3.c`) to learn some of the various highlights of the logging program.

Going through the `tutor3.c` source...

You may want to launch CrossCut and open `tutor3.c` so you can have the code in front of you as you read on.

#include & #defines

<code>tt8lib.h</code>	Most (but not all) Model 8 specific function declarations, structures and constants are declared in the <code>tt8lib.h</code> header file. It is almost always necessary to include <code>tt8lib.h</code> if you want to do anything useful with the Model 8.
<code>stdlib.h</code>	Includes the declaration for <code>malloc()</code> which allows for dynamic allocation of memory in the heap.
<code>SAMPLES</code>	The number of samples to make, set to 100.

Variables

<code>TickRate</code>	Long variable signifying the current tick rate of the computer.
-----------------------	---

sample	The current sample number. Incremented after each sample is taken.
*valuePtr	A pointer to a dynamically allocated array of shorts.
ErrorCode	Holds the error code returned by XmodemSendMem.
MaxValue	Maximum sample value.
MinValue	Minimum sample value.
RunningSum	A sum of all of the samples made. It is of type long and is used by AveValue.
AveValue	The average sample value.

Program Initialization

Initialization of the Model 8 is similar to earlier tutorials by the call to InitTT8. In addition, TickRate is set to the current clock rate.

Dynamic Memory Allocation

The standard library malloc() function allocates space for a block of 100 shorts (or 200 bytes) and sets valuePtr to point to this block. This block will hold short values returned by the AtoDReadWord function. The 8 has a 12 bit A/D converter, which easily fits in type short.

Logging Data

Before logging can begin, the sleep counter is first initialized to zero.

The program then goes into a loop that stores 100 samples of channel 1. Within the loop, the following occurs:

```
{
```

Sleep (TickRate)

The program sleeps until the amount of ticks in TickRate has passed. Since TickRate was set to the Model 8's current clock rate, sleep (TickRate) will wait one second since the last sleep() and then proceed with the loop.

The function sleep() is much more useful than a standard delay function since it compensates for the time taken by the function calls between the delays. sleep (Ticks) returns FALSE if the amount of Ticks have already passed before sleep is called. Which means that timing information is corrupted because you used too much processor time with other function calls between sleeps(), or did not sleep() long enough. This return code is useful when debugging.

Since it is very unlikely that sleep() will timeout in our program (we don't do much between sleep() calls), we do not handle timeout.

Use care when using sleep(). For instance, you wouldn't be able to do a sleep(1) (only one clock tick) right after printing out a huge table of data to the serial port. You can't print out a huge table of data in one clock tick.

```
valuePtr[sample] = AtoDReadWord(1);
```

This is where the logging occurs. `AtoDReadWord()` makes an A/D reading of channel one and converts it to a short value. The `sample` variable is the index into the dynamically allocated array pointed to by value pointer `valuePtr[sample]` is the same as `*(valuePtr + sample)`.

The maximum value for `AtoDReadWord()` is 32760 (minimum value 0). There are 12 bits of precision spread out across the positive half of a 16 bit short (or 15 bits). Since there is a three bit discrepancy for now, you may notice that values change in multiples of 8 (2^3 bits). This is normal. It is spread out this way to minimize the code modification for expansion to 16 bit bipolar A/D converters (15 bits + sign).

```
printf(“[%02ld]:%5d ”,sample, valuePtr[sample]);
```

Prints the value to the screen. The `sample` variable is a long decimal (hence the `%ld`); `valuePtr[sample]` is a short decimal.

This section loops back 100 times.

```
}
```

Displaying Data

The program then goes into a loop that shows the 100 samples of channel 1 in a table. Within the loop, the following occurs:

```
{
```

```
RunningSum += (long) valuePtr[sample];
```

A `RunningSum` is kept of all of the values measured. They are cast to long (even though they are short) because the running sum must be large enough to hold the largest possible sum of all of the measurements. This method works unless you are sampling more than 65,535 shorts.

```
MaxValue = (MaxValue>valuePtr[sample] )?MaxValue: valuePtr[sample];
```

```
MinValue = (MinValue<valuePtr[sample] )?MinValue: valuePtr[sample];
```

Initially, `MaxValue` is set to the lowest possible measurement and `MinValue` is set to the highest possible measurement.

The simple (CONDITIONAL) ? IF TRUE : IF FALSE statement is used to store the highest and lowest samples made by the logging loop. The `Max` and `Min` values are compared to each sample. If a sample is greater than `MaxValue`, `MaxValue` is set to that sample. If a sample is less than `MinValue`, `MinValue` is set to that sample. At the end of the loop `MinValue` will be the lowest sample and `MaxValue` will be the highest.

```
printf...
```

The values are displayed as they were in the logging loop (assuming the 8 is hooked up). To print the data in a table, the `mod %` operator is used within an if expression to print a newline after every fifth measurement.

```
}
```


After the loop, the RunningSum is divided by the number of samples made to get the average value of the samples. AveValue, MaxValue and MinValue are printed right after the table of values.

Storing Data to a File

After the data is displayed, the XmodemSendMem function sends the data back to CrossCut. It needs to be passed three things before it can begin the transfer – the pointer to the memory area (ValuePtr), the size of the area to send (100 shorts or 200 bytes), and the timeout (120 seconds).

Running tutor3...

Type “make” to generate the tutor3.rhx file to load to RAM. If you didn’t have a makefile you could alternatively type “8run tutor3” (see ["Using the 8run and 8app Batch Files..."](#) procedure).

Load the TUTOR3.RHX file to the Model 8 as you did before in Tutorials 1 and 2.

Type “GO” as before to run tutor3.

The TOM8 monitor will jump to and run the program logging channel 7 once a second for 100 seconds. If you leave channel 7 floating, the first couple of points will read higher than the rest (due to startup noise). If the line is tied to a proper signal, the voltage reading will be shown as a value ranging from 0 to 32760. When sampling is complete, all of these values will print to the screen in a table. The Maximum, Minimum and Average sample values are also shown.

The TOM8 monitor then prompts you to begin XMODEM-1K receive. CrossCut allows you to do this. Just choose “Rcv File XMODEM-1K...” in the CommPort menu. It prompts you for a filename (defaults to RECEIVE.DAT”) and begins the transfer.

You have just run, logged, stored and offloaded data to your PC.

Using the 8run and 8app Batch Files...

If you examine any of the makefiles in the Tutorial or Examples directory, you may notice calls to two batch files 8run.bat and 8app.bat within the makefiles. They simplify the process of compiling up to nine source files by making calls to the generic makefile “tt8\bin\tt8.mak”. Type 8app or 8run followed by the name of the main source file without the C extension. If there are more source files (even with different path names) they can be added after the main file (separated by spaces).

Usage:

```
8run mainfile [otherfile2 otherfile3 ...] For RAM
8app mainfile [otherfile2 otherfile3 ...] For Flash
```

Examples:

```
8run logger \tt8\source\userio For RAM
8app logger \tt8\source\userio For Flash
```

General Model 8 Logger Techniques

Reducing power in the Simple Logger

Rather than step by step guides on programming, the following will provide general information to help you get the most out of using the Model 8.

The logger program is effective in that it logs data and allows you to store it. Unfortunately, it runs at about 63mA @ 10 Volts during logging when run from Flash. This consumption can be reduced in many ways. In this tutorial we discuss ways to fine tune the logger to reduce power drain.

Lowering System Frequency

One of the easiest things you can do to reduce power consumption in the Model 8 is to lower the system frequency. Table 6-8 on page 6-16 shows all of the available clock frequencies and the baud rate compromises associated with using them.

The system frequency of the TOM8 Monitor is set at 14.72 MHz (a nice frequency for doing ASCII file transfers at 57600). When a program is RUN from Flash (not booted with the TOM8 monitor), its default frequency is 16 MHz. This is a “friendly” frequency that works well with the on-board devices. It is also the fastest frequency available that still allows serial communication via the main UART.

Since we are only making measurements once a second, we will lower the clock frequency to 160 kHz for most of the operation of the Model 8. In fact, the lowest allowable clock frequency (set with SimSetFSys) is 160 kHz. The lowest clock frequency that can still retain acceptable UART communication is 320 kHz (results in 17.6mA current drain during logging). We will change the system clock to 160 kHz between actual logging.

Interrupt Driven Sampling

The AtoD converter only works well above 1MHz so you should keep the system clock at least at that frequency when using it with the AtoD. As an alternative, you can step up the frequency to 1MHz when doing the actual logging; and then lower the frequency to 160 KHz between samples.

Long Term Data Logging Techniques

One of the advantages of the low power consumption of the Model 8 is that it can be left in remote areas for very long periods of time to log data, control other devices, etc. using only battery power. Previous versions of the Tattletale line have been used in very remote areas doing just that. This section will get you started in setting up the software for a long term data logging application.

One of the useful features of the Model 8 is that it has a low power mode. In this state, the 68332 essentially goes to sleep.

It is able to accomplish this because it is essentially a two (some would say more) processor machine. The low-power PIC part acts as a controller for the power hungry 68332 (power hungry in a relative sense; it is low power by every other definition). The 68332 is quite capable of handling regular logging applications, but if you want really low power (for long delays) you may want to shut down the 68332 and let the PIC part wait out the delay.

Example Programs

There are a number of short programs (in the `\tt8\examples` subdirectory) that provide examples for using the Model 8. They are heavily commented and are broken up into sections, so they are easy to read. The best way to master the Model 8 is by looking at these programs, seeing what is done, and incorporating their ideas into new programs. Many of the examples were designed to push the boundaries of the 8, to test its capabilities and to show the end user what the controller engine can do.

If you have any code samples, ideas for examples, application notes, or anything that might be of interest regarding the Model 8 that you are willing to share with other users, please consider posting them to our BBS. Registered BBS users can upload to our Model 8 conference. One of the few problems with a radical new computer design is the lack of a large application base that truly represents the computer's unique aspects. Hopefully, as the BBS conference grows in popularity, this will no longer matter.

Type `"cd \tt8\examples"` to enter the examples directory. Build the samples by typing `"make"` in each of the example sub-directories. Each C file will generate both a run file and a burnable application file with characteristics described below. They use the `8run` and `8app` batch files to create the executables.

Creating and Burning the Example Programs into Flash Memory

As you may have read in the tutorial, you can build two types of programs for the Model 8 using the Aztec C compiler: programs that load into and run from RAM for testing during development (`.RUN`); and programs meant to burn into Flash memory for finished applications (`.APP`). Both types are generated using the same source and library files. Study the makefiles in the examples directory. They are heavily commented and straightforward.

Running Programs in RAM

Programs which load into RAM for testing are created with the extension `".RHX"` (RAM Hex File) to distinguish them from burnable applications, `".AHX"` (Application Hex File).

Run files are loaded into the Model 8 by typing `"LO"` to begin ASCII receive on the Tattletale; then choosing `"Snd file ASCII..."` from CrossCut and selecting one of the `".RHX"` files. On completion of the load, the `".RHX"` file is in Model 8 RAM and can be run by typing `"GO"`. They can also be loaded using the load S-record function under the Tattletale menu in Crosscut.

Startup Applications

An application will automatically run at power-up or reset if it is burned into Flash. When the Model 8 is turned on, the TOM8 mini-monitor recognizes if there is a program in Flash and jumps to it immediately. If you wish to go to the TOM8 monitor rather than the resident Flash program, connect -IRQ3 to DGND during a power-on or reset. The LED will light up (and stay lit) whenever -IRQ3 is tied to ground or after a Reset. To clear the program in Flash, there is a noprog.ahx file in the tt8\tutorial\ subdirectory. You can load noprog.ahx just as any other S-Record, and it will erase the first program sector in the flash so the monitor will not see a program loaded.

Loading your own Programs into the EEPROM

This procedure will show you how to load a program into the Model 8 EEPROM so that it will automatically run when power is turned on. Storing your program in the Model 8's non-volatile flash EEPROM is simple with the following procedure:

1. To load the program in Flash, first make sure that the Terminal window is shown and the Model 8 is connected and powered.
2. Press return a few times on your PC. You should keep getting the "TOM8>" prompt in the Terminal window.
3. Go to the Tattletale menu and choose "**Load S-record**" (or press Alt-L). A file dialog appears showing the contents of whatever directory you are in.
4. Use the TAB & cursor keys or the mouse to select the .AHX version of your program and press the ENTER key. A dialog appears showing the process of the download. After the download you should see the following prompt:

```
TOM8>  
load successful
```

```
Target is Flash!  
start addr = 00002000  
end addr = 0000
```

```
Ok to write flash between above addresses? (Y/N)
```

5. Press the "y" key to burn the program into Flash memory.

The progress of the burn and the Flash ID are shown.

6. To verify that your program is in Flash, turn off power to the Model 8 and then reapply it. Your program should run automatically.

An application will automatically run at power-up or hardware reset if it is burned into Flash. A hardware reset can be performed by briefly connecting -MCLR to DGND.

When the Model 8 is turned on, the TOM8 mini-monitor recognizes if there is a program in Flash and jumps to it immediately.

If you wish to jump to the TOM8 monitor rather than the resident Flash program, connect –IRQ3 to DGND during a power-on or reset. The LED will light up (and stay lit) whenever –IRQ3 is tied to ground, or after a forced reset. You should never keep –IRQ3 tied to ground constantly, as it will corrupt timing during running applications. Use the process only to get to the TOM8 monitor.

How to Erase a Program from the Flash EEPROM Memory

To clear the program in Flash, there is a hex file called NOPROG.AHX file in the "UTIL" subdirectory. You can load NOPROG.AHX just as any other application hex file, but all it does is erase the portion of the Flash memory where the start of the program is found. When the Model 8 is powered on, the TOM8 monitor notices that there is no longer a program so you get the regular sign-on message.

Tattletale Model 8 Memory (256K/256K Version)

RAM always ends at 0x300000 (the highest RAM location is 0x2FFFFFF). The examples below are shown for a Model 8 configured with 256 kilobytes of RAM (ranging from 0x2C0000 to 0x300000) and 256K of flash memory. With other configurations, stack-size (0x2000) and base address (0x2C0000) change.

See the memory map on [page 4-22](#) for a more graphical description of the Model 8's memory.

Memory Map - Load to RAM

A Model 8 program running entirely in RAM looks like this:

Address (hex)	Description
0x000000	Initial Stack Pointer (top of TPU RAM)
0x000004	Program Counter (to TOM8)
0x000008	Start of TOM8 Monitor
0x0017FF	End of TOM8 Monitor
0x001800	Start of custom TPU code
0x001FFF	End of custom TPU code
0x01E000	Start of optional RDB68 monitor
0x03FFFF	End of Flash Memory (256K Flash)
0x2C0000	First RAM location, Bottom of the stack.
0x2C2000	Top of the stack which grows down (stack pointer is equal to this at program start) and the beginning of code segment.
__H0_org	Start of code segment (= 0x2C2000 for 256K configuration).
__H0_end	End of code segment and
__H1_org	Start of initialized data segment.
__H1_end	End of initialized data segment and
__H2_org	Start of uninitialized data segment.
__H2_end	End of uninitialized data segment start of usable heap space.
0x2FEFFF	End of heap.
0x2FFC00	Start of vector table. Vector table is needed for the source level debugger and future expansion.
0x2FFFFFF	End of vector table.
	Highest RAM location.

Memory Map - Load to Flash

A Model 8 program loaded into Flash looks like this:
(shown for 256K RAM configuration):

Address (hex)	Description
0x000000	Stack Pointer
0x000004	Program Counter
0x000008	Start of TOM8 Monitor
0x0017FF	End of TOM8 Monitor
0x001800	Start of custom TPU code
0x001FFF	End of custom TPU code
0x002000	Start of program in ROM
0x03FFFF	End of Flash (256K Flash Memory)
0x200000	First RAM location.
0x2C2000	Top of the stack which grows down (stack pointer is equal to this at program start) and the beginning of the initialized data segment.
__H0_org	Start of code segment = 0x002000.
__H0_end	End of code segment.
__H1_org	Start of initialized data segment (= 0x2C2000 for 256K configuration)
__H1_end	End of initialized data segment and
__H2_org	Start of uninitialized data segment.
__H2_end	End of uninitialized data segment
	Start of usable heap space.
0x2FEFFF	End of heap.
0x2FFC00	Start of vector table.
0x2FFFFF	End of vector table.
	Highest RAM location.

RAM Configurations

There are different size stacks for the different memory configurations offered with the Model 8.

256K RAM

0x2C0000	0x2C2000	Stack area (A7 initially points to 0x2C2000)
0x2C2000	0x2FFFFFF	Code start to end of RAM
Stack is \$2000 or 8192 bytes (8K stack)		

1M RAM

0x200000	0x204000	Stack area (A7 initially points to 0x204000)
0x204000	0x2FFFFFF	Code start to end of RAM
Stack is \$4000 or 16384 bytes (16K stack)		

Stack is different sizes for different memory configurations.

Memory Symbols

The following symbols are pre-defined by the linker and are available by using the "public" directive.

__H0_org	Start address of code segment. If not set with linker option +c, this defaults to 0. To load the program into RAM, use the +c option with the address of the start of RAM plus the size of the stack you desire.
__H0_end	End address of code segment.
__H1_org	Start address of initialized data segment. If not set with linker option +d, it will default immediately after code segment. Therefore, if the code is being loaded into RAM and the +c option used, it can leave off +d so it will follow the code in RAM.
__H1_end	End address of initialized data segment.
__H2_org	Start address of uninitialized data segment. If not set with linker option +u, it defaults to immediately after the initialized data segment.
__H2_end	End address of uninitialized data segment.
__Storg_	Start address of stack. If not set with linker option +j, it defaults to immediately after uninitialized data segment.
STKSIZ	Size of the stack. If not set with linker option +s defaults to 2048 bytes (should be set with every make option in the Model 8).

See the makefiles in the `TT8\Examples` and `TT8\Tutorial` subdirectories to see how the compiler and linker uses these settings. To display the actual values the symbols defined above, use the -t option in the linker which writes the symbol table to a .SYM file.

The following symbols are defined in the start-up code.
Heap Information:

__mbot	points to bottom of heap (lowest usable address)
__mtop	points to top of heap (highest usable address)
__mcur	points to top of allocated heap space (initially __mbot)
__stkbase	points to bottom of stack
_errno	value of last library error code

The **__mbot**, **__mtop** and **__mcur** variables are used by the `brk()` and `sbrk()` functions and once they are initialized in the start-up code, they should only be modified by `brk()` and `sbrk()`.

The **__stkbase** variable is used in the `stkchk()` function to look for stack overflow. Since the stack grows down, this variable should be initialized with the smallest address the stack is expected to use. Notice that when this is initialized, the four characters "MANX" are also in the last four locations of the stack. This is also used in `stkchk()` when checking for stack overflow.

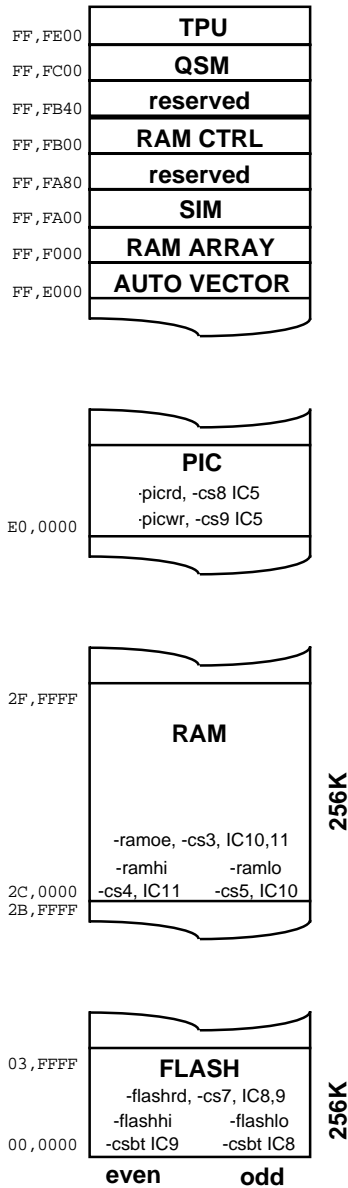
The **_errno** variable should be initialized to zero and must be 2 bytes if the default integer size is short (as it is for the Model 8).

Model 8 Memory Map

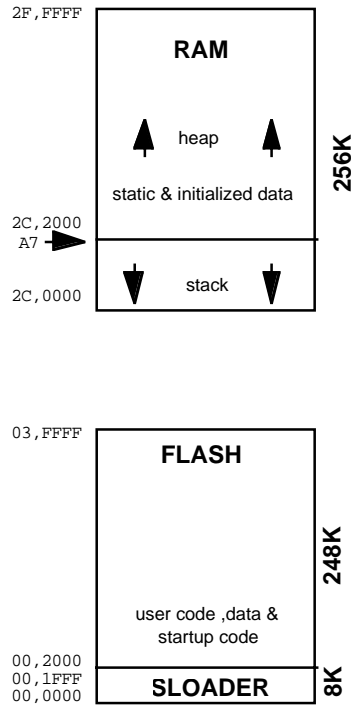
Tattletale Model 8 Memory Map (256K Flash, 256K RAM)

Feb 12, 1995

SYSTEM MAP



Aztec C EXECUTING FROM FLASH



Aztec C EXECUTING FROM RAM

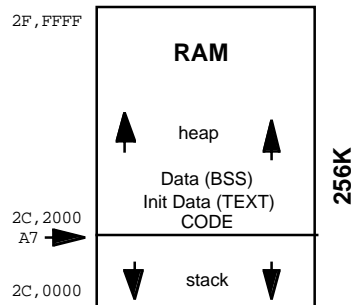


Figure 4-1: Model 8 Memory Map

Exception Handler Info

When your application calls InitTT8() and no other debugger is detected in the system, InitTT8() initializes the VBR and installs a minimal generic exception handler to catch catastrophic exceptions. When the generic exception handler encounters an exception, it breaks to the Monitor and places the above information at the very bottom of the stack (also the first area of RAM, or in this case, 0x2C0000). The TOM8 MD command can then be used to display the exception area for postmortem hints about what caused the crash.

```
002C0000  PC PC PC PC  .. .. SR SR US US US US SS SS SS SS
002C0010  VB VB VB VB SF SF SF SF DF DF DF DF .. .. .. ..
002C0020  D0 D0 D0 D0 D1 D1 D1 D1 D2 D2 D2 D2 D3 D3 D3 D3
002C0030  D4 D4 D4 D4 D5 D5 D5 D5 D6 D6 D6 D6 D7 D7 D7 D7
002C0040  A0 A0 A0 A0 A1 A1 A1 A1 A2 A2 A2 A2 A3 A3 A3 A3
002C0050  A4 A4 A4 A4 A5 A5 A5 A5 A6 A6 A6 A6 A7 A7 A7 A7
002C0060  sr sr pc pc pc pc ft vv s0 s0 s0 s0 s1 s1 s1 s1
002C0070  s2 s2 s2 s2 s3 s3 s3 s3 s4 s4 s4 s4 s5 s5 s5 s5
002C0080  s6 s6 s6 s6 s7 s7 s7 s7 .. .. .. .. .. .. .. ..
```

When the generic exception handler encounters an exception, it breaks to the Monitor and places the above information at the very bottom of the stack (also the first area of RAM, or in this (256K RAM) case, 2C0000).

Key to the abbreviations above:

```
PC          4 Byte Program Counter
..          Reserved for future use
SR          2 Byte Status Register
US          4 Byte User Stack Pointer
SS          4 Byte Supervisor Stack Pointer
VB          4 Byte Vector Base Register
SF          4 Byte SF Register
DF          4 Byte DF Register
D0..D7     8, 4 Byte Data Registers
A0..A7     8, 4 Byte Address Registers
```

General Exception Stack Frame Information as described in the CPU32 Manual.

```
sr          2 Byte Status Register
pc          4 Byte Program Counter
ft*        Frame Format
vv*        Vector Offset
s0..s7     8, 4 Byte System Registers
```

* ft and vv are not on byte boundaries - ft is 4 bits and vv is 12 bits. Refer to the CPU-32 manual for further information.

Section 5 - C Library Reference

How to Use this Section

This section of the manual is to be used for reference for all the C library commands. The commands are listed alphabetically. The structure of each command generally follows this format:

Command Name
 Syntax of the command
 Description of the command
 References to other commands

Tattletale Model 8 Library Functions

This section describes the functions, macros, and header declarations provided for writing C applications for the Tattletale Model 8 which are not covered by descriptions for the standard ANSI C libraries. It provides an overview of the capabilities of the Model 8 libraries, but must be augmented with additional information from the header files and sample programs supplied with the C development packages. The sample programs show how to correctly use many of the functions listed below; and the header files give specifics on the syntax.

Model 8 Header Files

tt8.h	Model 8 specific Hardware Definition
tat332.h	Hardware Definitions common to 68332 based Tattletaletales
sim332.h	68332 System Integration Module Definitions
tpu332.h	68332 Time Processing Unit Definitions
dio332.h	Model 8 Pin I/O Macros and Definitions
qsm332.h	68332 Queued Serial Module Definitions (QSM, QSPI, SCI)
tt8pic.h	Model 8 PIC Parallel Slave Port Definitions
tt8lib.h	C Prototypes for Model 8 specific Library Functions
userio.h	Console input/output functions for TT8

Typedefs used

The following types are used by the Tattletale Model 8 libraries and are defined in the tat332.h header file:

```
typedef unsigned char uchar
typedef unsigned short ushort
typedef unsigned long ulong
typedef char*ptr
typedef void (*vfptr)(void)
```

Library Object Files

There are three object files in the tt8\lib directory: CL16TT8.LIB, ML16TT8.LIB and TT8.LIB.

Model 8 Function Descriptions

PIC

Introduction

The separate PIC microcontroller performs a variety of housekeeping chores for the Tattletale Model 8 which would otherwise require a small handful of LSI and MSI components. Custom Onset code in the PIC manages the startup sequence, performs real-time clock and alarm functions, is the liaison to the serial EEPROM, allows for BDM (Background Debugging Mode) reload of the flash memory via RS-232, and has a variety of potential future capabilities through its hardware connections.

We had hoped to encapsulate all of the functionality that users would require of the PIC functions into high level C calls. We have a full set of primitives in C for talking to the PIC, and believe we have been successful in encapsulating the real-time clock and serial EEPROM functions, but complexity, time and manpower constraints have prevented us from fully wrapping the more esoteric PIC capabilities.

PIC Basics

The PIC and the 68332 both take turns being the master microcontroller. At startup, and when the 332 is sleeping, the PIC is in control. At all other times, your code in the 332 is in charge. Communication takes place using an 8 bit parallel interface with interrupt driven strobes and status. The communication protocol is very reliable at 68332 clock rates above about 1 MHz, but begins to deteriorate below that frequency. Anytime you access the PIC using a related function, make sure you are operating above 1MHZ.

To prevent lockups due to missed communications and acknowledgments, both the 332 and the PIC have built-in timeouts. The 68332 requires a timeout recalibration each time the clock speed changes (if you will be talking to the PIC) which is done with a call to PicInit() specifying the new clock frequency. The library function SimSetFSys() automatically calls PicInit() after changing the clock frequency, making this the safest method to switch speeds.

<tt8pic.h>

The header file <tt8pic.h> in the includes directory contains the definitions and constants needed to use the PIC function calls directly through library calls. This header file contains prototypes for all of the PIC functions though only the few described below are for general consumption. In particular, functions relating to the real time clock and the serial eeprom which have higher level interfaces are excluded from the descriptions.

CL16TT8.LIB

Standard (non-Model 8) C functions and structures compiled to have a 16 bit integer format. Similar to the CL16.LIB described in the Aztec manual, but modified to set up stdio with UART jack 1 (and other minor changes). This will allow, for example, for a printf to 'print' to the terminal of the host computer. This library should always be linked in with your program before loading to the Model 8. See the example make files and the tutorial on how to do this.

- ML16TT8.LIB** Identical to the Aztec ML16.LIB math library. Adds IEEE floating point format and math functions.
- TT8.LIB** Model 8 specific functions (all described below). This library should always be linked in with your program before loading to the Model 8. See the example make files and the tutorial on how to do this.

Time Format Conventions

There are three important time formats used in TT 8 function calls: `time_t`, `struct tm` and `structtime_tt`.

- `time_t` Also called calendar time. This is the number of seconds that have elapsed since some date and time in the past. On the Model 8, this date and time is Jan 1, 1970 at 00:00:00.
- `struct tm` This format is a structure that stores the year, month, day, hour, minute and second (among other information) as separate values.
- `structtime_tt` This is a Tattletale Model 8 typedef that ties a `time_t` element (in seconds) with an element containing the number of clock ticks (the Model 8 clock will produce some number of ticks per second - see `GetTickRate()`) to get fractional seconds.

Short Descriptions

"Header" lists the header files that need to be included in addition to the `<tt8lib.h>` header file.

NOTE: An asterisk (*) in the first column means that the function is PIC related. This is noted because making calls to these functions while operating below 1MHZ will be unreliable. Also note standard C library functions `clock ()` and `time`, have the same restriction.

Table 5-1: C Library Function List

*	Function	Header	Short Description
*	<code>AlarmToCtm ()</code>	<code><time.h></code>	Returns the current setting in the real-time clock's alarm register.
	<code>AtoDMilliVolts ()</code>		<code><macro></code> Converts an A-D reading to millivolts (assuming 4.096VREF).
	<code>AtoDReadMilliVolts ()</code>		<code><macro></code> Takes an A-D reading on a channel and converts to millivolts (assuming 4.096VREF).
	<code>AtoDReadWord ()</code>		Returns result of A-D conversion on specified channel.
	<code>CalcCRC ()</code>		Returns a 16 bit CRC on block of memory.
*	<code>DelayMilliSecs ()</code>		Suspends program execution for specified number of milliseconds.
	<code>FlashError ()</code>		Returns last error encountered in flash EEPROM routines.
	<code>FlashID ()</code>		Returns manufacturer and device ID from flash EEPROM.
	<code>GetFramePtr ()</code>		<code><inline></code> Returns the frame pointer.
	<code>GetInterruptMask ()</code>		<code><inline></code> Returns the interrupt mask.
	<code>GetRamInfo ()</code>		Returns base address, option size ,and speed settings of RAM.

Table 5-1: C Library Function List (Continued)

*	Function	Header	Short Description
	GetStatusReg ()		<inline> Returns the status register.
	GetTickRate ()		Returns current clock tick rate in ticks per second.
	GetVBR ()		<inline > Returns the vector base register.
	InitPorts ()		Initializes the I/O ports.
	InitQSM ()		Initializes the Queued Serial Module (SCI & QSPI).
	InitRam ()		Initializes the RAM (wait states).
	InitSIM ()		Initializes the system integration module registers.
	InitTPU ()		Initializes the Time Processor Unit.
*	InitTT8 ()		Calls all of the other “Init” functions with default settings.
	InitVBR ()		Initializes the Vector Base register and default vectors.
	InputLine ()	<userio.h>	Inputs line from the keyboard with minimal editing features.
	InstallHandler ()		Installs a C function as an interrupt handler.
	kbhit ()	<userio.h>	Returns TRUE if keyboard character available.
	kbflush ()	<userio.h>	Flushes any pending keyboard characters and return TRUE if any were available.
	LMDelay ()		<inline> Loop Mode Delay (wait state independent).
*	Max186PowerUp ()		Turns on the MAX-186 A-D converter.
	Max186PowerDown ()		Turns off the MAX-186 A-D converter.
	Max186Setup ()		Sets up MAX-186 A-D Converter for general use with QSPI.
*	MilliSecs ()		Return a count of milliseconds from the real-time clock.
	NumToHexStr ()		Converts a value into a hexadecimal string.
	PChange ()	<dio332.h>	<macro> Toggles the specified parallel port pin state.
	PClear ()	<dio332.h>	<macro> Clears the specified parallel port pin state to zero.
	PConfBus ()	<dio332.h>	<macro> Configures specified pin for its default bus function.
	PConfInp ()	<dio332.h>	<macro> Configures specified pin for use as digital input line.
	PConfOutp ()	<dio332.h>	<macro> Configures specified pin for use as digital output line.
*	PicAckCmpAlrm ()	<tt8pic.h>	Acknowledges a PIC generated alarm interrupt and releases the interrupt signal.
*	PicAndRF ()	<tt8pic.h>	Requests the PIC to perform a logical AND of the 8 bit data value to the PIC register file at addr.
*	PicGetIrqAddr ()	<tt8pic.h>	Returns the address of the PIC register file which contains the state of the various PIC generated interrupts.
*	PicInit ()	<tt8pic.h>	Initializes the 68332 for operations with PIC parallel slave port
*	PicOrRF ()	<tt8pic.h>	Requests the PIC to perform a logical OR of the 8 bit data value to the PIC register file at addr.
*	PicReadRF ()	<tt8pic.h>	Read one byte from PIC register file memory.
*	PicWriteRF ()	<tt8pic.h>	Write one byte to PIC register file memory.
	Pin ()	<dio332.h>	<macro> Returns state (0 or 1) of the specified parallel port pin.
	PSet ()	<dio332.h>	<macro> Sets the specified parallel port pin state to one.
	PutStr ()		Sends a string to the serial port.

Table 5-1: C Library Function List (Continued)

*	Function	Header	Short Description
	QueryChar ()	<userio.h>	Queries user for a character reply.
	QueryDateTime ()	<userio.h>	Queries user for the Date and Time.
	QueryNum ()	<userio.h>	Queries user for a numeric value.
	QueryYesNo ()	<userio.h>	Queries user for a yes or no reply.
*	Reset ()		<inline> Resets the Model 8.
*	ResetToMon ()		<inline> Resets Model 8 returning control to the monitor.
*	RtcToCtm ()	<time.h>	Returns current real-time clock's time registers as a calendar time.
*	SerActivate ()		Power up the transmit circuitry of the RS-232 driver.
	SerByteAvail ()		Returns non-zero if a byte is available from the serial port.
	SerGetBaud ()		Returns attainable baud rate given a baud rate and clock frequency.
	SerGetByte ()		Returns next byte from the serial port.
	SerInFlush ()		Flushes any buffered serial input characters.
	SerPutByte ()		Sends a byte to the serial port.
	SerSetBaud ()		Sets the serial port baud rate based on input parameters.
	SerSetInBuf ()		Sets up a user allocated buffer for serial input interrupt operation.
*	SerShutDown ()		Shut down the transmit circuitry of the RS-232 driver to save power.
*	SerTimedGetByte ()		Returns next byte form serial port or timeout.
	ServiceWatchdog ()	<sim332.h>	<macro> Services the software watchdog.
*	SetAlarmSecs ()	<time.h>	Sets the real-time clock alarm registers from input calendar time.
*	SetAlarmTM ()	<time.h>	<macro>Sets the real-time clock alarm registers from 'tm' time structure.
	SetInterruptMask ()		<inline> Sets the interrupt mask.
	SetStatusReg ()		<inline> Sets the status register.
	SetTimeSecs ()	<time.h>	Sets the real-time clock registers from input calendar time.
*	SetTimeTM ()	<time.h>	<macro> Sets the real-time clock registers from input 'tm' time structure.
	SetTickRate ()		Sets tick rate (ticks/second) used by Sleep and SleepTill functions.
	SetVBR ()		<inline> Sets the vector base register.
	SimGetFSys ()		Returns current system frequency in MHz.
*	SimSetFSys ()		Sets system clock to specified frequency in Hz. Return actual value.
	SimSetRAMWaits ()		Sets the wait states for the RAM.
	SimSetFlashWaits ()		Sets the wait states for the Flash.
*	Sleep ()		Drops to low power for specified number of ticks relative to last Sleep.
*	SleepTill ()		Drops to low power till time specified in time_tt structure argument.
	Stop ()		Executes processor STOP instruction. SIM subsystems remain active.
	StopLP ()		Executes processor LPSTOP instruction. SIM subsystems shut down.
*	StopWatchStart ()		Starts microsecond stopwatch counter and initialize it to zero.
*	StopWatchTime ()		Returns current value in microsecond stopwatch counter.
*	TensMilliSecs ()		Crude count of tens of milliseconds
*	TPUClearInterrupt ()		<macro> Clears the interrupt status flag for a TPU channel.

Table 5-1: C Library Function List (Continued)

*	Function	Header	Short Description
	TPUGetInterrupt ()		<macro> Polls the TPU to see if an interrupt request is pending for a particular channel.
	TPUGetPin ()		Defines TPU channel as digital input and return current value.
	TPUGetTCR1 ()		Returns the current value of the TPU TCR1 clock in Hz.
	TPUSetPin ()		Defines TPU channel as digital output and set to specified value.
	TPUInterruptDisable ()		<inline> Disables TPU Interrupts.
	TPUInterruptEnable ()		<inline> Enables TPU Interrupts.
	TTMEQ ()		<macro> Time: Equal To
	TTMNE ()		<macro> Time: Not Equal To
	TTMLT ()		<macro> Time: Less Than
	TTMLE ()		<macro> Time: Less Than or Equal To
	TTMGT ()		<macro> Time: Greater Than
	TTMGE ()		<macro> Time: Greater Than or Equal to
	ttmadd ()		Operates on time values in time_tt format.
	ttmcmp ()		Compares time values in time_tt format.
*	ttmnow ()		Returns current real-time clock value in time_tt format.
	TSerByteAvail ()	<tat332.h>	Returns non-zero if a byte is available on a TPU channel UART.
	TSerClose ()	<tat332.h>	Closes a TPU channel set for UART I/O.
	TSerGetByte ()	<tat332.h>	Gets next byte from the input queue of a TPU channel UART.
	TSerInFlush ()	<tat332.h>	Flushes the UART input queue of a TPU channel UART.
	TSerOpen ()	<tat332.h>	Opens a TPU channel for UART I/O.
	TSerPutByte ()	<tat332.h>	Writes a byte to the output queue of a TPU channel UART.
	TSerResetBaud ()	<tat332.h>	Changes the current baud rate for a TPU channel UART.
	UeeCalcCRC ()		Calculates and return CRC on specified portion of user EEPROM.
	UeeClearBit ()		Clears a single bit to zero in user EEPROM.
	UeeErase ()		Erase the entire user section of the EEPROM to all ones.
	UeeError ()		Returns error code for last user EEPROM operation.
	UeeFill ()		Fills entire user section of EEPROM with specified byte.
*	UeeReadBlock ()		Reads a block of data from user EEPROM.
*	UeeReadByte ()		Reads a byte of data from user EEPROM.
*	UeeSetBit ()		Sets a single bit to one in user EEPROM.
*	UeeSize ()		Returns size in bytes of user EEPROM.
*	UeeTestBit ()		Tests a single bit in user EEPROM.
*	UeeWriteBlock ()		Writes a block of data to user EEPROM.
*	UeeWriteByte ()		Writes a byte of data to user EEPROM.
	UpdateCRC ()		Returns next CRC based on input CRC and updated by input byte.
	XmodemSendMem ()		Offloads block to serial port using XMODEM protocol.

Functions by Category (and TypeDefs)

AtoD Converter Functions

```
short AtoDReadWord(short chan);
AtoDtoMilliVolts(adval); /* MACRO */
AtoDReadMilliVolts(chan); /* MACRO */
void Max186PowerUp(void);
void Max186PowerDown(void);
void Max186Setup(short pdMode, short shdnCompMode);
```

Assembly Shortcuts

```
ExcStackFrame *GetFramePtr(void); /* Inline Function */
ushort GetInterruptMask(void); /* Inline Function */
ptr GetRamInfo(long *size, short *waits);
ushort GetStatusReg(void); /* Inline Function */
void *GetVBR(void); /* Inline Function */
ServiceWatchdog(); /* #include <sim332.h> */ /* MACRO */
void SetInterruptMask(ushort mask) /* Inline Function */
void SetStatusReg(ushort mask) /* Inline Function */
void SetVBR(void *vb) /* Inline Function */
short SimSetFlashWaits(short waits);
short SimSetRAMWaits(short waits);
```

Digital I/O macros /* #include <dio332.h> */

```
PChange(port, pin); /* MACRO */
PClear(port, pin); /* MACRO */
PConfBus(port, pin); /* MACRO */
PConfInp(port, pin); /* MACRO */
PConfOutp(port, pin); /* MACRO */
Pin(port, pin); /* MACRO */
PSet(port, pin); /* MACRO */
```

Flash EEPROM Functions

```
FlashErr FlashError(ushort **errorAddr);
FlashErr FlashID(ushort *startAddr, ushort *mfr, ushort *device);
```

Interval Timer Functions

```
void DelayMilliSecs(ulong ms);
void LMDelay (short count); /* Inline Function */
ulong MilliSecs(void);
short Sleep(long ticks);
void StopWatchStart(void);
ulong StopWatchTime(void);
ulong TensMilliSecs(void);
```

Initialization Routines

```
void InitPorts(void);
void InitQSM(void);
ptr InitRam(short waits);
void InitSIM(short watchdog);
void InitTPU(short tpumcr);
void InitTT8(short watchdog, short tpumcr);
void InitVBR(void *vb);
```

Miscellaneous Functions

```
ushort CalcCRC(uchar *tptr, ulong count, ushort crc);
ushort UpdateCRC(uchar b, ushort crc);
```

Real-time functions /* #include <time.h> */

```

time_t AlarmToCtm(void);
void SetAlarmSecs(time_t secs);
void SetAlarmTM(struct tm *tp);
short SleepTill(time_tt wakeup);
time_t RtcToCtm(void);
void SetTimeSecs(time_t secs, vfptr sync);
void SetTimeTM(struct tm *tp, vfptr sync);
time_tt ttmadd(time_tt addendl, time_tt addend2);
long ttmcmp(time_tt t1, time_tt t2);
TTMEQ(t1,t2); /* MACRO */
TTMNE(t1,t2); /* MACRO */
TTMLT(t1,t2); /* MACRO */
TTMLE(t1,t2); /* MACRO */
TTMGT(t1,t2); /* MACRO */
TTMGE(t1,t2); /* MACRO */
time_tt ttmnow(void);

```

Serial I/O Functions

```

short InputLine(ptr linebuf, short linelen);
ptr NumToHexStr(ulong num, ptr str, short digits);
void PutStr(char *str);
short SerByteAvail(void)
long SerGetBaud(long baud, long freq)
uchar SerGetByte(void)
void SerInFlush(void)
void SerPutByte(uchar theByte)
void SerSetInBuf(ptr buffer, long bufsize)
long SerSetBaud(long baud, long freq)
short SerTimedGetByte(long msTimeout);
XmdmErr XmodemSendMem(void *address, long length, ushort timeout)

```

User I/O Extras /* #include <userio.h> */

```

short kbhit(void);
short kbflush(void);
short QueryChar(ptr prompt, short defChar, ptr scanSet, char *reply);
short QueryDateTime(ptr prompt, short defTime, struct tm *tm);
short QueryNum(ptr prompt, ptr defFmt, ptr scanFmt, ulong *value);
short QueryYesNo(ptr prompt, short defYes);

```

System Functions

```

long GetTickRate(void);
vfptr InstallHandler(vfptr funcptr, short vector, ExcCFrame *frame);
void Reset(void); /* Inline Function */
void ResetToMon(void); /* Inline Function */
short SetTickRate(long tickRate);
long SimGetFSys(void);
long SimSetFSys(long freq);
void Stop(ushort statreg);
void StopLP(ushort statreg);

```

TPU Functions

```

int TPUGetPin(short chan);
ulong TPUGetTCR1(void);
void TPUSetPin(short chan, short pinval);
TPUClearInterrupt(chan); /* MACRO */
TPUGetInterrupt(chan); /* MACRO */
void TPUIInterruptDisable(short chan); /* Inline Function */
void TPUIInterruptEnable(short chan); /* Inline Function */

```

```

TPU Serial Functions /* #include <tat332.h> */

```

```

int TSerByteAvail(int chan);
int TSerClose(int chan);
int TSerGetByte(int chan);
void TSerInFlush(int chan);
int TSerOpen(int chan,int priority,int outflag,ptr buffer, long qsize,long baud,int parity,int databits,int stopbits);
void TSerPutByte (int chan, int data);
long TSerResetBaud(int chan, long baud);

```

User EEPROM Functions

```

UeeErr UeeCalcCRC(ushort address, short len, ushort *crc);
UeeErr UeeClearBit(ushort address, uchar bitno);
UeeErr UeeErase(void);
UeeErr UeeError(ushort *errorLocation);
UeeErr UeeFill(uchar data);
UeeErr UeeReadBlock(ushort ueeSrcAddr, uchar *buffer, short len);
UeeErr UeeReadByte(ushort address, uchar *data);
UeeErr UeeSetBit(ushort address, uchar bitno);
ushort UeeSize(void);
UeeErr UeeTestBit(ushort address, uchar bitno, short *ishigh);
UeeErr UeeWriteBlock(ushort ueeDestAddr, uchar *buffer, short len);
UeeErr UeeWriteByte(ushort address, uchar data);

```

Function Descriptions

When using any of the functions listed below, `tt8lib.h` needs to be included in the source file that calls the function. If other include files are needed (such as `dio332.h`) they are listed in the syntax section of the function description.

AlarmToCtm

Syntax

```

#include <time.h>
time_t AlarmToCtm(void);

```

Description

Returns the current setting in the real-time clock's alarm register as a calendar time which is compatible with the other ANSI C time functions.

See Also

`RtcToCtm()`, `SetAlarmSecs()`, `SetAlarmTM()`.

AtoDReadWord

Syntax

```

short AtoDReadWord(short chan);

```

Description

Performs a conversion on the specified input channel *chan* and returns the result as a signed 16 bit integer. The full range returned with this converter is 0x0000 (0) to 0x7FF8 (32760). A return of 0x8000 (-32768) indicates an error occurred while reading the A-D converter.

See Also

MAX-186 data sheet

AtoDtoMilliVolts (*MACRO*)

Syntax

```
AtoDtoMilliVolts(adval);
```

Description

Converts a direct AtoD reading *adval* (short type as returned by AtoDReadWord()) to millivolts.

See Also

AtoDReadWord(), MAX-186 data sheet

AtoDReadMilliVolts (*MACRO*)

Syntax

```
AtoDReadMilliVolts(chan);
```

Description

Reads channel *chan* and converts it to millivolts. The reading has a maximum value of 4096 mV (the internal reference voltage of the MAX-186). It is the same in function as:

```
AtoDtoMilliVolts(AtoDReadWord(chan));
```

See Also

AtoDReadWord(), AtoDMilliVolts(), MAX-186 data sheet

CalcCRC

Syntax

```
ushort CalcCRC(uchar *tptr, ulong count, ushort crc);
```

Description

Returns a 16 bit CRC (Cyclic Redundancy Check) on *count* bytes of the block pointed to by *tptr*, with a starting value of *crc* (usually zero). This is the same CRC used by the XMODEM offload function.

See Also

UpdateCRC(), XmodemSendMem().

DelayMilliSecs

Syntax

```
void DelayMilliSecs(ulong ms);
```

Description

Suspends program execution for *ms* milliseconds using the PIC 40KHz clock via the MilliSecs() function. This routine is provided to allow timing when the other hardware (especially the TPU) may not be available. This function does NOT go into a low power state during the delay. DelayMilliSecs() checks for wrap-around of the return of MilliSecs() whenever the value is larger than the number of milliseconds in a day.

See Also

MilliSecs(), Sleep(), StopWatchStart(), StopWatchTime()

FlashError

Syntax

```
FlashErr FlashError(ushort **errorAddr);
```

Description

Return the last error encountered and where it occurred. The error codes are described in **tt8lib.h**.

See Also

Flash memory manufacturers data sheet, FlashErr enumeration in **tt8lib.h** header file, FlashError(), FlashID().

FlashID

Syntax

```
FlashErr FlashID(ushort *startAddr, ushort *mfr, ushort *device);
```

Description

Returns the manufacturer *mfr* and device ID *device* from the flash EEPROMs which can respond to intelligent identifier commands.

See Also

Flash memory manufacturers data sheet, FlashErr enumeration in **tt8lib.h** header file, FlashError(), FlashID().

GetFramePtr (*Inline Function*)

Syntax

```
ExcStackFrame *GetFramePtr(void);
```

Description

Returns the Frame Pointer.

GetInterruptMask (*Inline Function*)

Syntax

```
ushort GetInterruptMask(void);
```

Description

Returns the interrupt mask.

GetRamInfo

Syntax

```
ptr GetRamInfo(long *size, short *waits);
```

Description

Return base address, option size ,and speed settings.

NULL can be passed for either the size or waits pointers to eliminate the need for temporary dummy pointer variables (for quickly getting base addr).

The Model 8 can have RAM sized from 64K (32K * 2), 256K (128K * 2) or 1M (512K * 2), all with the top of memory at 0x30,0000.

The base address reg. for all configurations starts out at 0x20,0000 and maps into the entire 1 megabyte address space. The standard 128K parts have a second chip enable connected to 68332 address line A18 which maps them into 0x24,0000-0x27,FFFF and 0x2C,0000-0x2F,FFFF (the latter being the assumed map for library standards).

GetStatusReg (*Inline Function*)

Syntax

```
ushort GetStatusReg(void);
```

Description

Returns the status register.

GetTickRate

Syntax

```
long GetTickRate(void);
```

Description

Returns the current clock tick rate (in ticks per second) used by Sleep() and SleepTill() functions. This value gives the units of the 'ticks' member of the time_tt structure. For instance, if this function returns 1000, time_tt.ticks is in units of milliseconds.

NOTE: The default tick rate is 40,000.

See Also

Timing description in hardware manual, SetTickRate(), Sleep(), SleepTill(), ttmadd(), ttmcmp(), ttmnow(), time_tt structure in **tt8lib.h** header file.

GetVBR (*Inline Function*)

Syntax

```
void *GetVBR(void);
```

Description

Returns the vector base register.

InitPorts

Syntax

```
void InitPorts(void);
```

Description

InitTT8() calls this function. For most simple Model 8 applications InitTT8() is the only "Init" function that needs to be called.

Initializes the I/O Ports.

InitQSM

Syntax

```
void InitQSM(void);
```

Description

InitTT8() calls this function. For most simple Model 8 applications InitTT8() is the only “Init” function that needs to be called.

Initializes the Queued Serial Module (SCI & QSPI).

See Also

InitTT8();

InitRam

Syntax

```
ptr InitRam(short waits);
```

Description

InitTT8() calls this function. For most simple Model 8 applications InitTT8() is the only “Init” function that needs to be called.

This assumes the RAM has been minimally setup from startup code (or we wouldn't be here), and that we are just tailoring the size, base address and wait states to fit the RAM on the board. The Model 8 can have RAM sized from 64K (32K * 2), 256K (128K * 2) or 1M (512K * 2), all with the top of memory at 0x30,0000. The base address reg. for all configurations starts out at 0x20,0000 and maps into the entire 1 megabyte address space. The standard 128K parts have a second chip enable connected to 68332 address line A18 which maps them into 0x24,0000-0x27,FFFF and 0x2C,0000-2F,FFFF (the latter being the assumed map for library standards.)

We get the assumed base address from the chip select module, then add in (or) values as shown:

1M = X0,0000 256K = XC,0000 64K = XF,0000

NOTE: This is a one-shot operation. We do not attempt to re-adjust mapping if the base address register is not currently set to 1M or if any of the base address bits below A20 are set.

See Also

InitTT8();

InitSIM

Syntax

```
void InitSIM(short watchdog);
```

Description

InitTT8() calls this function. For most simple Model 8 applications InitTT8() is the only “Init” function that needs to be called.

Initialize the system integration module registers. Program SYPCR with watchdog argument interpreted as follows:

```
>= 0    load lsb into SYPCR (only works if first time)
-1      load default (no watchdog, halt and bus monitor enabled)
< -1    don't write to SYPCR
```

Pass the constant “NO_WATCHDOG” (defined in tt8lib.h) for default Model 8 behavior.

See Also

```
InitTT8();
```

InitTPU (*Inline Function*)

Syntax

```
void InitTPU(short tpumcr);
```

Description

Initializes the Time Processor Unit.

program TMCR with tpumcr argument interpreted as follows:

```
>= 0    load tpumcr into TMCR (only works if first time)
-1      load default (TT8 Custom TPU code from onset)
< -1    don't write to TMCR
```

Pass the constant “TT8_TPU” (defined in tt8lib.h) for default Model 8 behavior.

InitTT8

Syntax

```
void InitTT8(short watchdog, short tpumcr);
```

Description

Calls all of the other “Init” functions at their default settings. This should be at the start of most programs for the Model 8, unless you are an experienced Model 8 programmer and know exactly how you want all of the options set up. For the two initialization functions that need default values InitSIM(watchdog) and InitTPU(tpumcr), two variables are provided.

Pass the constants “NO_WATCHDOG” and “TT8_TPU” (defined in tt8lib.h) for default Model 8 behavior.

See Also

Other “Init” functions. To enable interrupt driven buffered serial input, see the function SerSetInBuf().

InitVBR

Syntax

```
void InitVBR(void *vb);
```

Description

InitTT8() calls this function, for most simple Model 8 applications InitTT8() is the only “Init” function that needs to be called.

Initializes the Vector Base Register, setting up default vectors to jump to the TOM8 monitor.

See Also

InitTT8();

InputLine

Syntax

```
short InputLine(ptr linebuf, short linelen);
```

Description

Input line from keyboard with minimal editing features. Returns a pointer *linebuf* to the string that is at most *linelen* characters long.

Returns TRUE for all input except Ctrl-C and returns NULL terminated string in buffer. Does not echo or include the terminating return character.

Typical Usage:

```
char myline[40];  
if (! InputLine(myline, sizeof(myline)))  
    ... error processing for Ctrl-C ...
```

Edit Commands:

Ctrl-C 0x03	cancel operation, return FALSE
Ctrl-H 0x08	backspace and erase
Ctrl-R 0x12	redisplay current line
Ctrl-U 0x15	clear line and start again

See Also

userio.h, **userio.c**, PutStr()

InstallHandler

Syntax

```
vfptr InstallHandler(vfptr functptr, short vector,
ExcCFrame *frame);
```

Description

This function provides a high-level mechanism to install an interrupt handler written as a C function. The *functptr* parameter points to the new C interrupt handler. The *vector* number (0-255) is multiplied by 4 and offset from the VBR.

The *frame* parameter can either be NULL (for C functions coded entirely in inline assembly which are then responsible for saving any used registers and performing the final RTE (Return From Exception machine); or it can point to a static ExcCFrame (defined in **tt8lib.h**) which handles the interface between C and the interrupt call.

The previous interrupt handler is returned. This can be saved and used as the *functptr* argument in a later call to this function to restore the original interrupt handler.

tt8.h shows the default vector settings. Also many of the examples in **\tt8\examples** show how to use the InstallHandler() routine.

Example

(from tputunes.c)

```
main()
{
    static ExcCFrameframebuf;
    InstallHandler(nextnote, PIT_INT_VECTOR,&framebuf);
    *PITR = 100; /* 10 mS */
} /* main() */
```

See Also

The ExcCFrame structure in the **tt8lib.h** header file, the default vector table in **tt8.h**.

kbflush

Syntax

```
#include <userio.h>
short kbflush(void);
```

Description

Flushes any keyboard characters pending and returns TRUE if any were available.

See Also

userio.h, **userio.c**, kbhit()

kbhit**Syntax**

```
#include <userio.h>
short kbhit(void);
```

Description

Returns TRUE if keyboard character available.

See Also

userio.h, **userio.c**, kbflush()

LMDelay (*Inline function*)**Syntax**

```
void LMDelay(short count);
```

Description

Simple inline function for generating very short, wait state independent delays. This inline function enters CPU-32 loop mode executing NoP op-codes for the short count passed (32767 MAX!). In loop mode, no instruction fetches are made, so this is independent of wait states. Each iteration takes 8 cycles, so:

16MHz = 500nS/iter, 8 MHz = 1uS/iter, ... 160 kHz = 50us/iter

Max186PowerUp**Syntax**

```
void Max186PowerUp(void);
```

Description

Turns on the MAX-186 AtoD converter. Most current C functions that access the MAX-186 part (such as AtoDReadWord()) do not need to call this function. They power up and power down the converter as needed.

See Also

MAX-186 data sheet, AtoDReadWord(), Max186PowerDown()

Max186PowerDown

Syntax

```
void Max186PowerDown(void);
```

Description

Turns off the Max186 AtoD converter. Most current C functions that access the MAX-186 part (such as AtoDReadWord()) does not need to call this function. They power up and power down the converter as needed.

See Also

MAX-186 data sheet, AtoDReadWord(), Max186PowerUp()

Max186Setup

Syntax

```
void Max186Setup(short pdMode, short shdnCompMode);
```

Description

Setup MAX-186 for generic use with QSPI.

Performs generic initialization of QSPI registers for working the MAXIM MAX-186 AtoD converter with the Model 8 QSPI. Routines accessing this part will need to perform additional initialization based on the desired access modes and data rates.

Most current C functions that access the MAX-186 part (such as AtoDReadWord()) does not need to call this function. They power up and power down the converter as needed.

See Also

MAX-186 data sheet, AtoDReadWord(), Max186PowerUp(), Max186PowerDown();

MilliSecs

Syntax

```
ulong MilliSecs(void);
```

Description

Returns a count of milliseconds by multiplying the real-time clock seconds (since Jan. 1, 1970) by one thousand, and adding the PIC 40KHz tick count divided by forty. This value will wrap around in about 49 days.

See Also

Timing description in hardware manual, `TensMilliSecs()`, `StopWatchStart()`, `StopWatchTime()`.

NumToHexStr

Syntax

```
ptr NumToHexStr(ulong num, ptr str, short digits);
```

Description

Converts the value *num* into a hexadecimal string. The value of *digits* on entry determines the number of characters to convert. If the length is less than the number needed to express the value, the returned string will truncate the upper digits. If greater, the string will be left filled with leading zeros.

Len:	Value:	String:
2	0x2345	"45"
9	0x2345	"000002345"

PChange (MACRO)

Syntax

```
#include<dio332.h>
PChange(port, pin);
```

Description

Toggles the specified parallel port *pin*. Assumes the *port* has been configured for output. Argument *pin* must be one of the numbers 0, 1, 2, 3, 4, 5, 6 or 7. Argument *port* must be one of the upper-case characters D, E or F.

Example

```
PConfOut(F,7);/* make it output (only necessary once)*/
PSet(F,7);/* set it high */
PChange(F,7);/* toggle it */
```

See Also

Header file **dio332.h** for a full description of this macro, `PConfOut()`.

PClear (MACRO)

Syntax

```
#include<dio332.h>
PClear(port, pin);
```

Description

Set the specified parallel port *pin* low. Assumes the *port* has been configured for output. Argument *pin* must be one of the numbers 0, 1, 2, 3, 4, 5, 6 or 7. Argument *port* must be one of the upper-case characters D, E or F.

Example

```
PConfOutp(F,7);/* make it output (only necessary once)*/
PClear(F,7);/* set it low */
PChange(F,7);/* toggle it */
```

See Also

Header file **dio332.h** for a full description of this macro, PConfOutp().

PConfBus (MACRO)

Syntax

```
#include<dio332.h>
PConfBus(port, pin);
```

Description

Configures the specified *pin* of the parallel *port* to perform its default bus function. Argument *pin* must be one of the numbers 0, 1, 2, 3, 4, 5, 6 or 7. Argument *port* must be one of the upper-case characters D, E or F.

See Also

Header file **dio332.h** for a full description of this macro.

PConfInp (MACRO)

Syntax

```
#include<dio332.h>
PConfInp(port, pin);
```

Description

Configure the specified *pin* or the parallel *port* for use as digital input line. Argument *pin* must be one of the numbers 0, 1, 2, 3, 4, 5, 6 or 7. Argument *port* must be one of the upper-case characters D, E or F.

See Also

Header file **dio332.h** for a full description of this macro, Pin().

PConfOutp (MACRO)

Syntax

```
#include<dio332.h>
PConfOutp(port, pin);
```

Description

Configures the specified *pin* of the parallel *port* for use as digital output line. Argument *pin* must be one of the numbers 0, 1, 2, 3, 4, 5, 6 or 7. Argument *port* must be one of the upper-case characters D, E or F.

See Also

Header file **dio332.h** for a full description of this macro, PChange(), PClear(), Pset().

PicAckCmpAlrm

Syntax

```
short PicAckCmpAlrm(void);
```

Description

This function acknowledges a PIC generated alarm interrupt and releases the interrupt signal. This function is not guaranteed to work when the system clock is below 1 MHz.

See Also

<tt8.h>, <tt8lib.h>, PIC Introduction, lpsleep example.

PicAndRF

Syntax

```
short PicAndRF(short addr, short data);
```

Description

This function requests the PIC to perform a logical AND of the 8 bit data value to the PIC register file at *addr*. The actual values and addresses that have meaning are contained in Onsets proprietary PIC code. We include the description of this function here because some future advanced examples may reference this function with constant values supplied by the engineers at Onset.

See Also

<tt8.h>, <tt8lib.h>, PIC Introduction.

PicGetIrqAddr

Syntax

```
short PicGetIrqAddr(void);
```

Description

This function returns the address of the PIC register file which contains the state of the various PIC generated interrupts.

This address is then used with one of the PIC register file access functions to return the interrupt values, which are masked with constants described below. A non-zero value indicates active, and zero value indicates inactive.

The address for the corresponding interrupt enable bits is always one more than the address returned by this function, but with the same mask values. A non-zero value enables interrupts and a zero value disables. In normal operation, the PIC generates no interrupts to the 68332.

```
PerAlarmMask0x01
CmpAlarmMask0x02
GlobalAlarmMask0x80
```

Example

```
#include <tt8pic.h>

ushortIrq332F, Irq332E;
Irq332E = (Irq332F = PSPGetIrqAddr()) + 1;
PicWriteRF(Irq332E, CmpAlarmMask | GlobalAlarmMask);
```

See Also

<tt8.h>, <tt8lib.h>, PIC Introduction, lpsleep example.

PicInit

Syntax

```
void PicInit(long sysfreq);
```

Description

This routine initializes the 68332 for operations with PIC parallel slave port. This is called automatically by the library function SimSetFSys() and should only be called by your code if you are making direct changes to the clock.

See Also

<tt8.h>, <tt8lib.h>, PIC Introduction.

PicOrRF

Syntax

```
short PicOrRF(short addr, short data);
```

Description

This function requests the PIC to perform a logical OR of the 8 bit data value to the PIC register file at addr. The actual values and addresses that have meaning are contained in Onset's proprietary PIC code.

We include the description of this function here because some future advanced examples may reference this function with constant values supplied by the engineers at Onset.

See Also

<tt8.h>, <tt8lib.h>, PIC Introduction.

PicReadRF

Syntax

```
short PicReadRF(short addr);
```

Description

Read one byte from PIC register file memory.

See Also

<tt8.h>, <tt8lib.h>, PIC Introduction.

PicWriteRF

Syntax

```
short PicWriteRF(short addr, short data);
```

Description

Write one byte to PIC register file memory.

See Also

<tt8.h>, <tt8lib.h>, PIC Introduction.

Pin (*MACRO*)

Syntax

```
#include<dio332.h>  
Pin(port, pin);
```

Description

Returns zero if the specified *pin* of the parallel *port* is low, or one if the line is high. Assumes the *port* has been configured for input. Argument *pin* must be one of the numbers 0, 1, 2, 3, 4, 5, 6 or 7. Argument *port* must be one of the upper-case characters D, E or F.

See Also

Header file **dio332.h** for a full description of this macro, PChange(), PClear(), PSet(), PConfInp().

PSet (MACRO)

Syntax

```
#include<dio332.h>
PSet(port, pin);
```

Description

Sets the specified parallel port *pin* high. Assumes the *port* has been configured for output. Argument *pin* must be one of the numbers 0, 1, 2, 3, 4, 5, 6 or 7. Argument *port* must be one of the upper-case characters D, E or F.

Example

```
PConfOut(F,7); /* make it output (only necessary once)*/
PHigh(F,7);    /* set it high */
PChange(F,7); /* toggle it */
```

See Also

Header file **dio332.h** for a full description of this macro, PConfOut().

PutStr

Syntax

```
void PutStr(char *str);
```

Description

Sends a string to the serial port (stdout).

See Also

InputLine();

QueryChar

Syntax

```
#include <userio.h>
shortQueryChar(ptr prompt, short defChar, ptr scanSet, char
*reply);
```

Description

Queries user for a character reply.

Returns TRUE for all replies except Ctrl-C. Print prompt string (verbatim) followed by optional default character inside square brackets, followed by a question mark. Return default character if only return is pressed; otherwise scans reply accepting only characters from *scanSet*.

NOTES:

- Carriage return is not echoed.
- If the *scanSet* contains no lower case characters, the input will be converted to upper case.
- If the *scanSet* contains no upper case characters, the input will be converted to lower case.

See Also

userio.h, userio.c

QueryDateTime

Syntax

```
#include <userio.h>
shortQueryDateTime(ptr prompt, short defTime, struct tm
*tm);
```

Description

Queries user for the Date and Time.

Returns TRUE for all replies except Ctrl-C. Print prompt string (verbatim) followed by the default reply (either the current system calender time if defTime is TRUE or the calender time contained in the tm structure. Returns result in struct tm, but does not set the system clock.

NOTES:

- Carriage return is not echoed.
- Ctrl-C always return FALSE.
- Input format is compatible with CrossCut's Paste Time & Date (Cmd-D).

Typical Usage:

```
struct tm t;
if (! QueryDateTime("\nCurrent Date and Time", TRUE, &t))
... error processing for Ctrl-C ...
SetTimeTM(&t, NULL); /* set the system time */
```

See Also

userio.h, userio.c

QueryNum

Syntax

```
#include <userio.h>
shortQueryNum(ptr prompt, ptr defFmt, ptr scanFmt, ulong
*value);
```

Description

Queries user for a numeric value.

Returns TRUE for all replies except Ctrl-C. Prints prompt string (verbatim) followed by optional default value (if defFmt in non-NULL and not a NULL string) inside square brackets, followed by a question mark. Returns default value if only return is pressed, otherwise scan reply using scanFmt string.

NOTES:

- DefFmt and scanFmt are scanf specifiers (e.g. "%ld", "lu", "%lx").
- Carriage return is not echoed.

See Also

userio.h, userio.c

QueryYesNo

Syntax

```
#include <userio.h>
shortQueryYesNo(ptr prompt, short defYes);
```

Description

Queries user for a yes or no reply.

Returns TRUE only for a Yes reply (which may be returned only default if defYes is TRUE). Prints prompt string (verbatim) followed by (Yes/No), followed by default reply character inside square brackets, followed by a question mark.

NOTES:

- Carriage return is not echoed.
- Ctrl-C always returns FALSE.

See Also

userio.h, userio.c

Reset (*Inline Function*)

Syntax

```
void Reset(void);
```

Description

Resets the Model 8 and reboots the program.

See Also

```
ResetToMon();
```

ResetToMon (*Inline Function*)

Syntax

```
void ResetToMon(void);
```

Description

Resets the Model 8 and returns control to the TOM8 Monitor (if the program is being executed from Flash).

See Also

```
Reset();
```

RtcToCtm

Syntax

```
#include <time.h>
time_t RtcToCtm(void);
```

Description

Returns the current setting in the real-time clock's time registers as a calendar time which is compatible with the other ANSI C time functions.

See Also

```
AlarmToCtm(), SetTimeSecs(), SetTimeTM().
```

SerActivate

Syntax

```
void SerActivate(void);
```

Description

Power up the transmit circuitry of the RS-232 driver. This is only needed if you have called `SerShutDown` without specifying `autowake`.

See Also

`<tt8.h>`, `<tt8lib.h>`, `SerShutDown`.

SerByteAvail

Syntax

```
short SerByteAvail(void);
```

Description

Return non-zero if a byte is available from the serial port.

See Also

`SerGetByte()`.

SerGetBaud

Syntax

```
long SerGetBaud(long baud, long freq);
```

Description

Returns the serial port baud rate that is attainable given a requested baud rate and system clock frequency. The returned baud rate may not match the requested baud rate because the system clock frequency can take on only 256 different values and the baud rate divisor can take on only integer values from 1 to 8191. If *baud* = 0, the function uses the current baud rate divisor in the SCI Control Register 0; but if *baud* ≠ 0, *baud* is the requested baud rate for the serial port and the baud rate divisor to achieve this baud rate is returned. If *freq* = 0, the function uses the current system clock; but if *freq* ≠ 0, the function bases its calculation on the value of *freq* instead of the system clock. To get the current serial port baud rate use *baud* = 0 and *freq* = 0.

The actual serial baud rate is derived from this formula:

$$\text{baud rate} = \text{System Clock} / (32 \cdot \text{Baud Rate Divisor})$$

where the baud rate divisor is the value of SCI Control Register 0.

See Also

MC68332 SIM User's Manual, `SerSetBaud()`

SerGetByte

Syntax

```
uchar SerGetByte(void);
```

Description

Returns next byte from the serial port. This function does not time out and will not return until a character is received.

See Also

SerPutByte(), SerTimedGetByte()

SerInFlush

Syntax

```
void SerInFlush(void);
```

Description

Flushes any buffered serial input characters.

See Also

SerByteAvail()

SerPutByte

Syntax

```
void SerPutByte(uchar theByte);
```

Description

Sends *theByte* to the serial port.

See Also

SerGetByte()

SerSetBaud

Syntax

```
long SerSetBaud(long baud, long freq);
```

Description

Sets the serial port baud rate based on the values passed in *freq* and *baud*, or uses the value of the current system clock if *freq* = 0. Return the new baud rate. The returned baud rate may not match the requested baud rate because the system clock frequency can take on only 256 different values and the baud rate divisor can take on only integer values from 1 to 8191. The actual serial baud rate is derived from this formula:

System Clock

$$\text{baud rate} = \text{System Clock} / (32 \cdot \text{Baud Rate Divisor})$$

where the baud rate divisor is the value of SCI Control Register 0.

See Also

MC68332 SIM User's Manual, SerGetBaud()

SerSetInBuf

Syntax

```
void SerSetInBuf(ptr buffer, long bufsize);
```

Description

Sets up a serial port input buffer for interrupt operation. Pointer *buffer* points to user allocated storage and *bufsize* specifies the length of the buffer. If *bufsize* = 0, serial input communications revert to polled operation and *buffer* is ignored. The user is responsible for freeing *buffer* storage.

The Model 8 starts up running the serial controller with polled input operation and serial interrupts disabled. You have to explicitly call this function to start the serial controller running from interrupts. The buffer is a circular buffer. When it is full it does not overwrite, but discards any new characters until old characters are removed.

See Also

SerGetByte()

SerShutDown

Syntax

```
void SerShutDown(short immed, short autowake);
```

Description

Shut down the transmit circuitry of the RS-232 driver to save power. The MAX232 has a control line which allows us to turn off the voltage doubling and inverting circuitry while still accepting incoming characters. The chip draws only tens of microamps in shutdown, but several milliamps when active.

The first parameter, when non-zero, instructs the routine to wait for any character currently being transmitted to complete before shutting down. The second parameter, when non-zero, instructs the serial transmit routine to automatically power up the RS-232 driver before sending a character.

See Also

<tt8.h>, <tt8lib.h>, SerActivate.

SerTimedGetByte

Syntax

```
short SerTimedGetByte(long msTimeout);
```

Description

Returns next byte from the serial port or -1 if timeout *msTimeout* occurs.

See Also

SerGetByte()

ServiceWatchdog (MACRO)

Syntax

```
#include <sim332.h>
ServiceWatchdog();
```

Description

Executes a special service sequence routine that is required by the software watchdog on a periodic basis. If the watchdog timer is enabled (via InitTT8()) and the service sequence is not called periodically, the watchdog issues a reset.

See Also

MC68332 User's Manual(software watchdog)

SetAlarmSecs

Syntax

```
#include <time.h>
void SetAlarmSecs(time_t secs);
```

Description

Sets the real-time clock alarm registers from *secs*, the ANSI C time in seconds. Making use of the alarm timer involves setting up an interrupt handler to be called when the alarm time expires.

See Also

SetAlarmTM(), AlarmToCtm(), example program Lpsleep.c.

SetAlarmTM (MACRO)

Syntax

```
#include <time.h>
void SetAlarmTM(struct tm *tp);
```

Description

Sets the real-time clock alarm registers from the ANSI C time structure. Making use of the alarm timer involves setting up an interrupt handler to be called when the alarm time expires.

See Also

SetAlarmSecs, AlarmToCtm, example program Lpsleep.c.

SetInterruptMask (Inline Function)

Syntax

```
void SetInterruptMask(ushort mask);
```

Description

Sets the interrupt mask equal to variable *mask*.

SetStatusReg (Inline Function)

Syntax

```
void SetStatusReg(ushort mask);
```

Description

Set the status register.

SetVBR (Inline Function)

Syntax

```
void SetVBR(void *vb);
```

Description

Sets the Vector Base Register.

SetTickRate

Syntax

```
short SetTickRate(long tickRate);
```

Description

Sets the current tick rate (in ticks per second) used by `Sleep()` and `SleepTill()` functions. This value defines the units of the 'ticks' member of the `time_tt` structure. For instance, if `tickRate` is 1000, `time_tt.ticks` is in units of milliseconds.

The default (and natural) tick rate for a Model 8 is 40,000 ticks per second, but the rate can be set to any value greater than zero evenly divisible into 40,000.

See Also

Timing description in hardware manual, `GetTickRate()`, `Sleep()`, `SleepTill()`, `ttmadd()`, `ttmcmp()`, `ttmnow()`, `time_tt` structure in **tt8lib.h** header file.

SetTimeSecs

Syntax

```
#include <time.h>
void SetTimeSecs(time_t secs, vfptr sync);
```

Description

Sets the real-time clock registers from the ANSI C calendar time in seconds. The optional function `sync` can be either `NULL` (zero) or a pointer to a user supplied function which will be called just prior to starting the clock. Your `sync` function can then wait for some external event, and simply return when it is time to start the clock.

See Also

`SetTimeTM()`, `RtcToCtm()`.

SetTimeTM (MACRO)

Syntax

```
#include <time.h>
void SetTimeTM(struct tm *tp, vfptr sync);
```

Description

Sets the real-time clock registers from the ANSI C time structure. The optional function `sync` can be either `NULL` (zero) or a pointer to a user supplied function which will be called just prior to starting the clock. Your `sync` function can then wait for some external event, and simply return when it is time to start the clock.

See Also

SetTimeSecs(), RtcToCtm().

SimGetFSys

Syntax

```
long SimGetFSys(void);
```

Description

Returns the current system frequency in MHz.

See Also

MC68332 SIM User's Manual and SimSetFsys()

SimSetFlashWaits

Syntax

```
short SimSetFlashWaits(short waits);
```

Description

Pass: 0 – 13 for normal selection (in wait states)
–1 for fast termination cycles (Moto calls this –1 wait state)

Normally it is not necessary to change this value. InitTT8 sets this value to 1 which always works at 16MHZ. Setting this to 0 should also work at 16MHZ but must be tested in the system. Refer to the CS Access Time column in Table 6-8 on page 6-16 to calculate required wait states for the speed of a particular part. This function always returns the value of the current wait states. To return just the current value pass a number outside the acceptable range i.e. 256.

See Also

SimSetRAMWaits(), InitTT8

SimSetRAMWaits

Syntax

```
short SimSetRAMWaits(short waits);
```

Description

Pass: 0 – 13 for normal selection (in wait states)
–1 for fast termination cycles (Moto calls this –1 wait state)

Normally it is not necessary to change this value. InitTT8 sets this value to 1 which always works at 16MHZ.

Setting this to 0 should also work at 16MHZ but must be tested in the system. Refer to the CS Access Time column in Table 6-8 on page 6-16 to calculate required wait states for the speed of a particular part. This function always returns the value of the current wait states. To return just the current value pass a number outside the acceptable range i.e. 256.

See Also

SimSetFlashWaits(), InitTT8

SimSetFSys

Syntax

```
long SimSetFSys(long freq);;
```

Description

Sets the system clock as close as possible to *freq* (the specified frequency in Hz) and returns the new frequency value in Hz. Returns -1 if *freq* is greater than the maximum possible frequency. A table of frequencies that can be reached exactly can be derived from this equation:

$$\text{freq} = 40000 (4 (Y + 1) 2^{2W+X})$$

where:

Y can be integer values from 0 to 63

W and X can be 0 or 1

the maximum frequency is 16 MHz

Keeps the current W and X settings if possible, unless the new frequency can be obtained by doubling or halving the X bit (instantaneous).

See Also

MC68332 SIM User's Manual and SimGetFsys(). See Table 6-8 on page 6-16 for available clock rates.

Sleep

Syntax

```
short Sleep(long ticks);
```

Description

Sleeps for the specified number of *ticks* relative to the last Sleep() call. This is purely a timing function and does not enter low power mode.

The units for *ticks* depend on the value that GetTickRate() returns. For instance, if GetTickRate() returns 1000, *ticks* is in units of milliseconds. The tick rate can be changed with SetTickRate().

Sleep() returns FALSE if *ticks* have already passed since the last sleep(), TRUE otherwise.

See Also

SetTickRate(), GetTickRate(), SleepTill(), Stop (), StopLP ().

SleepTill

Syntax

```
short SleepTill(time_tt wakeup);
```

Description

Processor idles until the time specified in *wakeup* - a structure including a calendar time element (time_t secs) and a ticks element (long ticks) for the fractional part of a second. This function is called from Sleep ().

The *wakeup.secs* member is compatible with all other time_t calendar time values from the ANSI C functions (like mktime()). The *wakeup.ticks* member can be set to any value from 0 to the tick rate returned by GetTickRate().

See Also

GetTickRate(), ANSI function mktime(), **time.h** header file, Sleep(), time_tt structure in **tt8lib.h** header file. For low power sleep, see example program Lpsleep.c.

Stop

Syntax

```
void Stop(ushort statreg);
```

Description

C interface to MC68332 STOP instruction. Puts the Model 8 into a mode where no instructions are fetched or executed (there is no bus activity) but all SIM subsystems (QSPI, TPU, SCI, the oscillator, etc.) continue to run. The value in *statreg* is loaded into the processor status register (both supervisor and user portions). Of special note is the interrupt priority mask portion of the status word (bits 8, 9 and 10). The only exit from Stop() is a system reset or an interrupt of higher priority than in the interrupt priority mask.

See Also

MC68332 SIM User's Manual, StopLP(), example program Lpsleep.c.

StopLP

Syntax

```
void StopLP(ushort statreg);
```

Description

C interface to the MC68332 LPSTOP instruction. Puts the Model 8 into a mode where no instructions are fetched or executed (there is no bus activity) and all SIM subsystems (QSPI, TPU, SCI, the oscillator, etc.) are shut down. This puts the MC68332 into its lowest power state. Before shutting down, the value in *statreg* is loaded into the processor status register (both supervisor and user portions). Of special note is the interrupt priority mask portion of the status word (bits 8, 9 and 10). The only exit from StopLP() is a system reset or an interrupt of higher priority than the interrupt priority mask. On exit, it restores all SIM subsystems.

See Also

Sleep(), SleepTill(), Stop(), example program Lpsleep.c.

StopWatchStart

Syntax

```
void StopWatchStart(void);
```

Description

Starts the microsecond stopwatch counter and initializes it to zero for subsequent calls by StopWatchTime.

See Also

StopWatchTime

StopWatchTime

Syntax

```
ulong StopWatchTime(void);
```

Description

Returns the current value in the microsecond stopwatch counter. This function has better resolution than either MilliSecs() or TensMilliSecs() but has the same accuracy as those functions because the PIC 40 KHz timer is used by all three functions. The PIC timer has a resolution of 25 microseconds, with about a 1.5millisecs calling overhead at 16 MHz.

See Also

StopWatchStart(), MilliSecs(), TensMilliSecs().

TensMilliSecs

Syntax

```
ulong TensMilliSecs(void);
```

Description

Crudely counts of tens of milliseconds by reading the real-time clock seconds and ticks values. Function `StopWatchTime()` gives more accurate results.

This value will wrap around in about 497 days.

See Also

Timing description in hardware manual, `MilliSecs()`, `StopWatchStart()`, `StopWatchTime()`.

ttmadd

Syntax

```
time_tt ttmadd(time_tt addend1, time_tt addend2);
```

Description

This function is used to operate on time values in `time_tt` format. Argument *addend1* is added to *addend2* and the sum is returned in normalized form. Ticks field may be larger than rate without generating errors.

In the `time_tt` format, the `time_t` calendar time is stored in the `time_tt.secs` member and any fractional seconds are stored in the `time_tt.ticks` member in the ticks per second units set by the `SetTickRate()` command (default is xx ticks per second). In normalized form, the `time_tt.ticks` member is never greater than the clock tick rate set by `SetTickRate()`.

See Also

`ttmnow()`, `ttmcmp()`, `SetTickRate()`, `time_tt` structure in **tt8lib.h** header file.

ttmcmp, TTMEQ, TTMNE, TTMLT, TTMLE, TTMGT, TTMGE

Syntax

```
long ttmcmp(time_tt t1, time_tt t2);
```

Description

Subtracts time *t2* from time *t1* and returns the difference in clock ticks. This value may be higher than the tick rate set by `SetTickRate()`. To normalize this value in a `time_tt` structure, use this method:

```
diff = ttmcmp( time1, time2);
time3.secs = diff / GetTickRate();
time3.ticks = diff % GetTickRate();
```

The macros are also defined in `tt8lib.h`. They simplify comparisons in IF statements. Assume operator in place of comma (`TTMLT(t1,t2) --> t1 LT t2`)

```
#define TTMEQ(t1,t2)    (ttmcmp(t1,t2) == 0)
#define TTMNE(t1,t2)    (ttmcmp(t1,t2) != 0)
#define TTMLT(t1,t2)    (ttmcmp(t1,t2) < 0)
#define TTMLE(t1,t2)    (ttmcmp(t1,t2) <= 0)
#define TTMGT(t1,t2)    (ttmcmp(t1,t2) > 0)
#define TTMGE(t1,t2)    (ttmcmp(t1,t2) >= 0)
```

See Also

`ttmnow()`, `ttmadd()`, `SetTickRate()`, `GetTickRate()`, `time_tt` structure in **tt8lib.h** header file.

ttmnow

Syntax

```
time_tt ttmnow(void);
```

Description

Returns current time as determined by the PIC real-time clock which supplies seconds in `time_tt.secs` and the fractional portion in `time_tt.ticks`. The `time_tt` structure has two members:

- `time_tt.secs` time_t calendar time (in seconds)
- `time_tt.ticks` number of clock ticks since the last second

One potentially confusing aspect to this is that the rate at which `time_tt.ticks` is incremented can be changed (with the `SetTickRate()` function) to any value from 1 to 40,000. If the the tick rate is 1000, for instance, the `time_tt.ticks` value should only take on values from 0 to 999. This is the normalized form of the structure.

See Also

`ttmadd()`, `ttmcmp()`, `GetTickRate()`, `SetTickRate()`, `time_tt` structure in **tt8lib.h** header file.

TPUClearInterrupt (MACRO)

Syntax

```
TPUClearInterrupt ( chan );
```

Description

Clears the interrupt status flag for a TPU channel.

See Also

TPU Reference Manual.

TPUGetInterrupt (*MACRO*)

Syntax

```
TPUGetInterrupt (chan) ;
```

Description

Polls the TPU to see if an interrupt request is pending for a particular channel.

See Also

TPU Reference Manual.

TPUGetPin

Syntax

```
int TPUGetPin(short chan) ;
```

Description

Defines TPU channel as digital input and returns current value.

See Also

PConfInp(), Pin().

TPUGetTCR1

Syntax

```
ulong TPUGetTCR1(void) ;
```

Description

Returns the current value of the TPU TCR1 (Time Control Register 1) clock in Hz.

See Also

TPU Reference Manual

TPUInterruptDisable

Syntax

```
void TPUInterruptDisable(short chan) ;
```

Description

Disables TPU interrupts, on the specified channel.

See Also

TPU Reference Manual

TPUInterruptEnable

Syntax

```
void TPUInterruptEnable(short chan);
```

Description

Enables the TPU Interrupts, on the specified channel.

See Also

TPU Reference Manual

TPUSetPin

Syntax

```
void TPUSetPin(short chan, short pinval);
```

Description

Defines the TPU channel as digital output and sets to specified value.

See Also

PConfOut(), Pset(), Pclear(), Pchange().

TSerByteAvail

Syntax

```
#include <tat332.h>
int TSerByteAvail(int chan);
```

Description

Returns a non-zero if a byte is available on a serial port. The serial port is the TPU channel (0..15) *chan* acting as a UART.

See Also

TSerOpen()

TSerClose

Syntax

```
#include <tat332.h>
int TSerClose(int chan);
```

Description

Closes a serial port. Returns a zero port closes successfully. The serial port is the TPU channel (0..15) *chan* acting as a UART.

See Also

TSerOpen()

TSerGetByte

Syntax

```
#include <tat332.h>
int TSerGetByte(int chan);
```

Description

Gets the next byte from the input queue or waits for and returns next byte if no bytes are available. The serial port is the TPU channel (0..15) *chan* acting as a UART.

See Also

TSerOpen()

TSerInFlush

Syntax

```
#include <tat332.h>
void TSerInFlush(int chan);
```

Description

Flushes the UART input queue. The serial port is the TPU channel (0..15) *chan* acting as a UART.

See Also

TSerOpen()

TSerOpen

Syntax

```
#include <tat332.h>
int TSerOpen(
    int    chan,
    int    priority,
    int    outflag,
    ptr    buffer,
    long   qsize,
    long   baud,
    int    parity,
    int    databits,
    int    stopbits);
```

Description

Opens a TPU channel *chan* for UART I/O with the following settings:

<i>chan</i>	TPU channel (0..15)
<i>priority</i>	TPU: Low Priority, Middle Priority, High Priority
<i>outflag</i>	zero = input, non-zero output
<i>buffer</i>	pointer to user allocated buffer
<i>qsize</i>	memory buffer size (2^{qsize}) Ex: $\text{qsize} = 6$, $\text{buffer} = 2^6 = 64$
<i>baud</i>	requested baud rate
<i>parity</i>	'E' = even, 'O' = odd, else none
<i>databits</i>	1 = 1, 2 = 2,...,8 = 8
<i>stopbits</i>	1 = 1, 2 = 2,...

Memory allocated for the TSerOpen parameter “buffer” combines memory needed for the serial queue (*qsize*) and scratchpad memory needed for use by the TPU serial routines (TSER_MIN_MEM). TSER_MIN_MEM is pre-defined in the header file. When allocated, TSER_MIN_MEM is used exclusively by the TPU serial routines and is not available to the user.

NOTE: For the TSerOpen function to work correctly “qsize” must be a power of two (i.e. 2, 4, 8, 16, 32...) and the total allocation for “buffer” must be exactly the sum of *qsize* + TSER_MIN_MEM. This memory may be appropriated from either static storage (a global or local static array) or allocated dynamically with a call to malloc (see the example).

Example

```
#include <tt8lib.h>
#define TSBUFSIZ 128
ptrbuf;

if ((buf = malloc(TSBUFSIZ+TSER_MIN_MEM))==0)
```

```
    /* error code here */  
    if (TSerOpen(12, HighPrior, 1, buf, TSBUFFSIZ, 9600, 'N', 8, 1) != tsOK)  
        /* error code here */
```

See Also

Motorola TPU reference manual.

TSerPutByte

Syntax

```
#include <tat332.h>  
void TSerPutByte(int chan, int data);
```

Description

Writes byte *data* to the UART output queue. The serial port is the TPU channel (0..15) *chan* acting as a UART.

See Also

TSerOpen()

TSerResetBaud

Syntax

```
#include <tat332.h>  
long TSerResetBaud(int chan, long baud);
```

Description

Attempts to change the current baud rate for a serial port to the requested *baud*. It returns the actual set baud rate. The serial port is the TPU channel (0..15) *chan* acting as a UART.

See Also

TSerOpen()

UeeCalcCRC

Syntax

```
UeeErr UeeCalcCRC(ushort address, short len, ushort *crc);
```

Description

Calculates and returns *crc* (Cyclic Redundancy Check) on specified portion of user serial EEPROM.

See Also

UeeSize(), UeeErr enumeration in **tt8lib.h** header file.

UeeClearBit

Syntax

```
UeeErr UeeClearBit(ushort address, uchar bitno);
```

Description

Clear to zero a single bit in the user serial EEPROM.

See Also

UeeSize(), UeeErr enumeration in **tt8lib.h** header file.

UeeErase

Syntax

```
UeeErr UeeErase(void);
```

Description

Erase to all ones the entire user section of the serial EEPROM.

See Also

UeeErr enumeration in **tt8lib.h** header file.

UeeError

Syntax

```
UeeErr UeeError(ushort *errorLocation);
```

Description

Returns the error code for the last serial EEPROM operation.

See Also

UeeErr enumeration in **tt8lib.h** header file.

UeeFill

Syntax

```
UeeErr UeeFill(uchar data);
```

Description

Fills the entire user section of the serial EEPROM with the specified byte.

See Also

UeeErr enumeration in **tt8lib.h** header file.

UeeReadBlock

Syntax

```
UeeErr UeeReadBlock(ushort ueeSrcAddr, uchar *buffer, short len);
```

Description

Reads a block of data from the user section of the serial EEPROM.

See Also

UeeReadByte(), UeeSize(), UeeErr enumeration in **tt8lib.h** header file.

UeeReadByte

Syntax

```
UeeErr UeeReadByte(ushort address, uchar *data);
```

Description

Reads a byte from the user section of the serial EEPROM.

See Also

UeeReadBlock(), UeeSize(), UeeErr enumeration in **tt8lib.h** header file.

UeeSetBit

Syntax

```
UeeErr UeeSetBit(ushort address, uchar bitno);
```

Description

Sets a single bit in the user serial EEPROM to 1.

See Also

UeeSize(), UeeErr enumeration in **tt8lib.h** header file.

UeeSize

Syntax

```
ushort UeeSize(void);
```

Description

Returns the size in bytes of the user portion of the serial EEPROM.

UeeTestBit

Syntax

```
UeeErr UeeTestBit(ushort address, uchar bitno, short
*ishigh);
```

Description

Tests a single bit in the user serial EEPROM.

See Also

UeeSize(), UeeErr enumeration in **tt8lib.h** header file.

UeeWriteBlock

Syntax

```
UeeErr UeeWriteBlock(ushort ueeDestAddr, uchar *buffer,
short len);
```

Description

Writes a block of data to the user section of the serial EEPROM.

See Also

UeeWriteByte(), UeeSize(), UeeErr enumeration in **tt8lib.h** header file.

UeeWriteByte

Syntax

```
UeeErr UeeWriteByte(ushort address, uchar data);
```

Description

Writes one byte of data to the user section of the serial EEPROM.

See Also

UeeWriteBlock(), UeeSize(), UeeErr enumeration in **tt8lib.h** header file.

UpdateCRC

Syntax

```
ushort UpdateCRC(uchar b, ushort crc);
```

Description

Returns the next CRC (Cyclic Redundancy Check) based on the input *crc* and updated by the byte *b*.

See Also

UpdateCRC() macro in **tt8lib.h** header file, CalcCRC().

XmodemSendMem

Syntax

```
XmdmErr XmodemSendMem(void *address, long length, ushort
timeout);
```

Description

Offloads a block of memory at *address* to the serial port using Xmodem 1K protocol. The memory block is *length* bytes in size. The last record output will be padded with zeros if *length* is not an exact multiple of 1K. The *timeout* argument is in units of seconds. This routine attempts the transfer using CRC (Cyclic Redundancy Check) error checking. If the receiving system does not respond, checksum error checking will be used. When using CrossCut to do get data, choose "Rcv file XMODEM-1K" under the CommPort menu.

The following to be fixed in future releases:

- Timeout does not work - it is always approximately 4 minutes @ 16MHZ.
- Error handling is not complete.
- Will not fall back to Xmodem 128 byte packets - must be used as Xmodem 1K.

See Also

CalcCRC(), UpdateCRC(), XmdmErr enumeration in **tt8lib.h** header file.

Section 6 - Hardware and Interface Specifications

Getting Started

This section shows you the detailed specifications needed for connecting devices to the Tattletale. First time users will generally work with a Model 8 that has an IO-8 prototyping board on top (included in the development kit). If you bought the deluxe development kit you will also have a PR-8 prototyping board in the development kit (it's the 5 x 7 inch prototyping board).

Once you are comfortable using CrossCut and C, you are ready to look at the Model 8 hardware in more detail and start designing your own hardware applications.

Hardware Do's and Don'ts

We have learned through experience that most catastrophic damage occurs during bench test and development. Please take precautions to avoid damaging your Model 8. The following is a list of **Do** and **Don't** rules when getting started:

- Do** use a current limited supply and diode reversal protection during development; a fused, in-line current measuring multimeter can prevent frustration.
- Do** avoid static discharge; the Model 8 uses CMOS components which are highly susceptible to damage from static discharge.
- Do** provide protection circuitry for your finished system.
- Don't** allow the supply voltage to exceed 15 Volts.
- Don't** forget that the factory delivered A/D converter is set up for internal reference (4.096 V).
- Don't** solder to the gold pads; it's virtually impossible to de-solder gold pads.
- Don't** exceed the absolute maximum ratings of the A/D converter's analog inputs: CH0-CH7 to AGND, DGND ... ($V_{ss} - 0.3V$) to ($V_{dd} + 0.3V$).

In addition to the points listed above, some other general precautions should be taken when working with CMOS components. Floating inputs can cause high current drain. Unused digital input pins should therefore be pulled to ground or Vreg through 1 Meg (max) resistors or set to outputs. The digital I/O pins should never be exposed to voltages above the board's positive supply (VREG) or below the ground level, as this could cause latch up. We recommend that all digital interface circuitry be powered from VREG; and that all external circuitry powered from a separate supply be connected with open collector circuits (see Figure 3 in the Hardware Reference) to reduce the likelihood of applying out of range voltages to the board.

Tattletale Model 8 Connectors

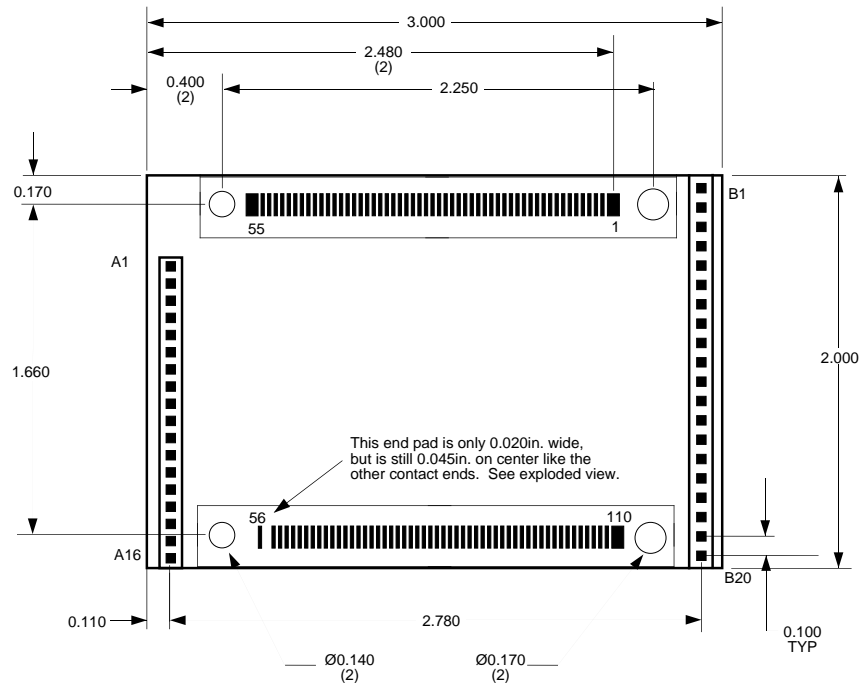
Pin and Socket Connector Specifications

These connectors mate with the connectors on the Model 8. Samtec part numbers are shown.

Mating connectors for the Model 8

16 pin "A" connector	TSW-116-07-SS
20 pin "B" connector	TSW-120-07-SS

Connections A1-A16 are sockets for 0.025in. square pins expressed on the top only. Connections B1-B20 are sockets for 0.025in. square pins expressed on the top only. Square connectors are 0.290 inches high. Pins are on 0.1in. centers. Allow 0.180in. clearance on either side of board for components.



NOTE: All dimensions are in inches.

Figure 6-1: Model 8 Dimensions

SquishyBus Connector Specifications

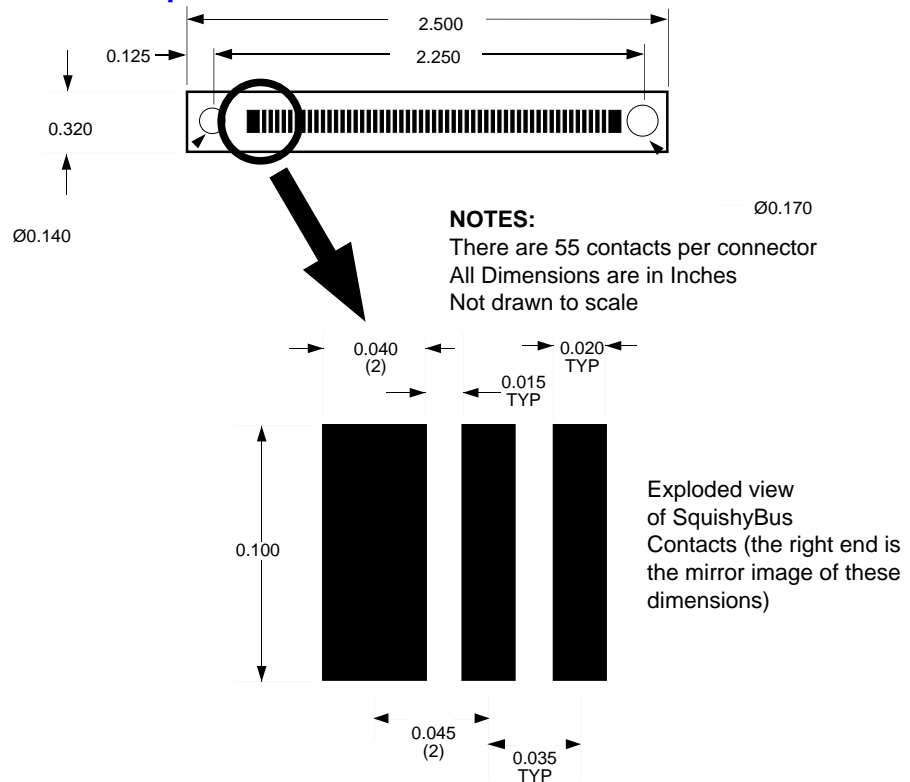


Figure 6-2: SquishyBus Dimensions

Connections 1-110 are surface contacts expressed on both sides of the Tattletale board. There are 55 contacts per SquishyBus connector. The spacing between contacts is 0.015in. Contact widths are 0.020in. except ends which are 0.040in. The connectors are 0.380in. high.

The dimensions of the Model 8 printed circuit board are shown in Figure 6-1. The footprint is 2in. x 3in. Including the socket connectors and ICs, the thickness of the populated Model 8 board is 0.55in. The socket connectors are designed to mate with 0.025in. square pins. While socket connectors are located only on the top side of the board, the gold pads for use with elastomeric expansion connectors (SquishyBus) are located on both sides of the PR-8 board.

Mounting the Model 8 to the Prototyping Boards

Four mounting holes are available for securing the Model 8 in place. Two of the mounting holes have a 0.140in. diameter; and two have a 0.170in. diameter. The differing diameters help to assure proper alignment of Model 8 with its mating boards by keying on the small boss at one end of the elastomeric expansion connectors. The Model 8 must be secured in place through all four holes with machine screws and nuts before putting it into permanent service. The connectors alone do not provide secure connection. Expansion with elastomeric interconnects (SquishyBus) requires no spacers as the elastomeric interconnects themselves serve as spacers.

NOTE: If you are using the pin & socket connectors in your finished system (like with the IO-8 board), you should use spacers between each hole and the relative securing point. Failure to do so may result in twisted or bent pin and socket connectors.

The gold pads are arranged in two rows along each side of the board and are electrically connected to the pads on the opposite side. The end pads have a larger width than the inner pads. The end pads have a width of 0.040in. and the inner pads have a width of 0.020in. Spacing between the end pads and inner pads is 0.045in. center to center. Spacing between inner pads is 0.035in. center to center. Elastomeric interconnects should be selected accordingly. Interconnects are available in various heights to accommodate your external hardware, an example of an elastomeric interconnect is the Fujipoly's part number 0940020 which is 0.250in high. Expansion interconnects are available from Onset.

NOTICE

The Model 8 is a multiple layered board. Do not attempt to cut or lift traces on the board as lower layers may be damaged.

If you purchased the deluxe development kit or the PR-8 board, you should have received a pair of elastomeric interconnects, 4 nylon screws and nuts.



Prototyping Board Details

The IO-8 and PR-8 prototyping boards have convenient connection points for the communication cable (UART connection), and power. Two standard mezzanine-style expansion boards are available for connecting power, communicating with other devices, signal conditioning and adding additional I/O circuits or memory to a Model 8 system. Both the IO-8 and the PR-8 include two 3.5mm UART connectors; one general purpose 8/10-pin modular phone connector, a convenient 14 pin screw-terminal strip (IO-8 only) and one 5.5mm O.D. by 2.1mm I.D. power connector, which mates with readily available AC adapters.

The **IO-8** breadboard has the same 2in. by 3in. footprint as the Model 8, and mates to the Model 8 with standard pin and socket connectors. This economical attachment board offers access to 36 signals: two RS-232 ports, eight A/D channels, nine TPU (time processor unit) digital I/O channels and three SPI (serial peripheral interface) channels. The IO-8 is best suited for data logger or control applications which do not need to access the 68332 bus.


The much larger **PR-8** prototyping board provides access to all 110 Model 8 signals using an elastomeric expansion interface. The PR-8 is well suited to building complex one-of-a-kind logger/controllers.

Power Supply Considerations

NOTICE

The Tattletale can be severely damaged by reversing the power supply or battery connections. **DO NOT** reverse the connections or you will void the warranty.

Before connecting a power supply to the DC power jack, make sure that the center of the tip is negative and the sleeve is positive.



Sleeve + → ● ← - Tip Center

Correct DC Power Jack Polarity

Power supply or battery connection points are available on the Tattletale (BAT+ is at pin B1, GND is at pin B11). The IO-8 and PR-8 prototyping boards bring these lines to a standard DC power jack. The supply voltage should be between 7 and 15 volts.

The Model 8 operates at two levels of VREG. When awake and active, VREG is maintained at 5 Volts. In low power drain mode, Vreg is maintained at 3.3 Volts. In order to take full advantage of these two modes, the Model 8 utilizes two voltage regulators. The LTC1174CS8-5 switching regulator is the relevant regulator in 5V mode. The 1174 has an absolute maximum input supply voltage rating of 15V which must not be exceeded! In low power mode, the 1174 is shutdown. In this shutdown mode, the 1174 draws only 1 μ A. The LT1121CST-3.3 linear supply, which is always active, provides VREG in the low power mode. The 1174 will provide up to 340mA in the 5V mode. The 1121 can provide up to 200mA in the 3.3V mode.

During your initial development you will need a DC power supply with an output in the range of 7 to 15V that can provide a current of about 300mA. *We strongly recommend a current limited supply.* The IO-8 and PR-8 prototyping boards have DC power jacks on the boards.

When no connector is in the DC power jack, the AUX line is connected to the BAT line. When a connector is inserted in the jack, it breaks this connection and supplies power directly to the BAT line. The AUX line is left floating. So, you could have a battery connected to the AUX line. Then when you insert the plug (from some other power source), it will disconnect your battery and supply the Tattletale from the other source. These drawings show how the plug works schematically. Figure 6-3 shows the jack with nothing plugged in. The AUX and BAT lines are connected at the arrow:

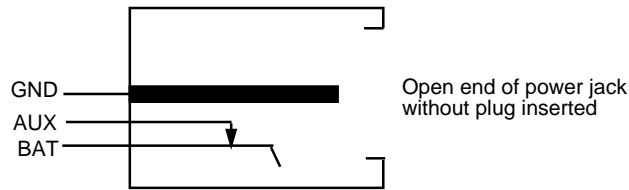


Figure 6-3: DC Power Jack w/o Connector Plugged in

Now, with a plug inserted, the bottom line (which is a spring) is forced away from the AUX line as shown in Figure 6-4:

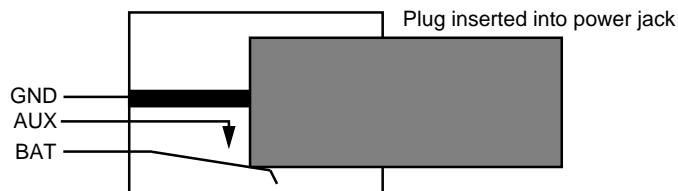


Figure 6-4: DC Power Jack with Connector Plugged in

The UART

The Model 8 has a default baud rate of 9600 baud, eight data bits, no parity and one stop bit. Interface connections are normally made using the PC-3.5 communication cable. The only Tattletale connections to worry about are TXD, RXD and GND; however, your computer may require other connections. Because many computers require it, the PC-3.5 cable ties several pins together, as shown in Figure 6-5.

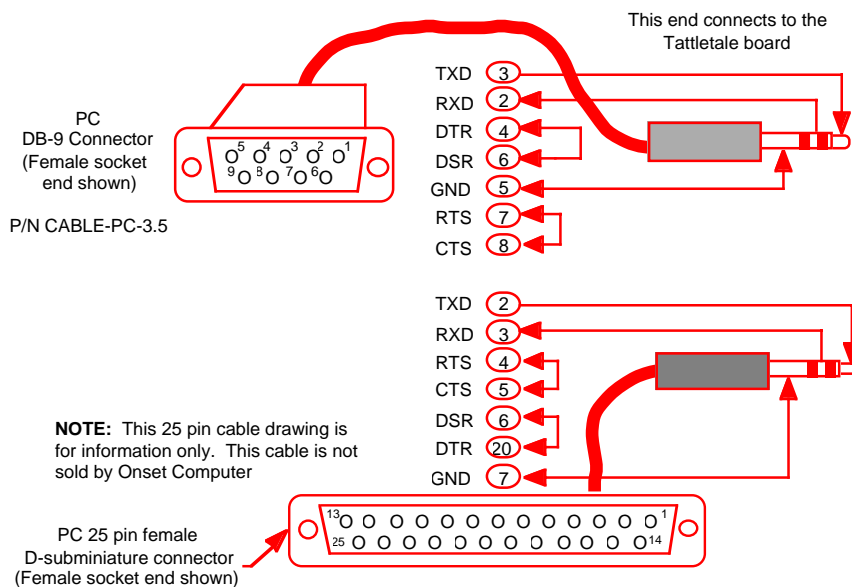


Figure 6-5: Communication Cables

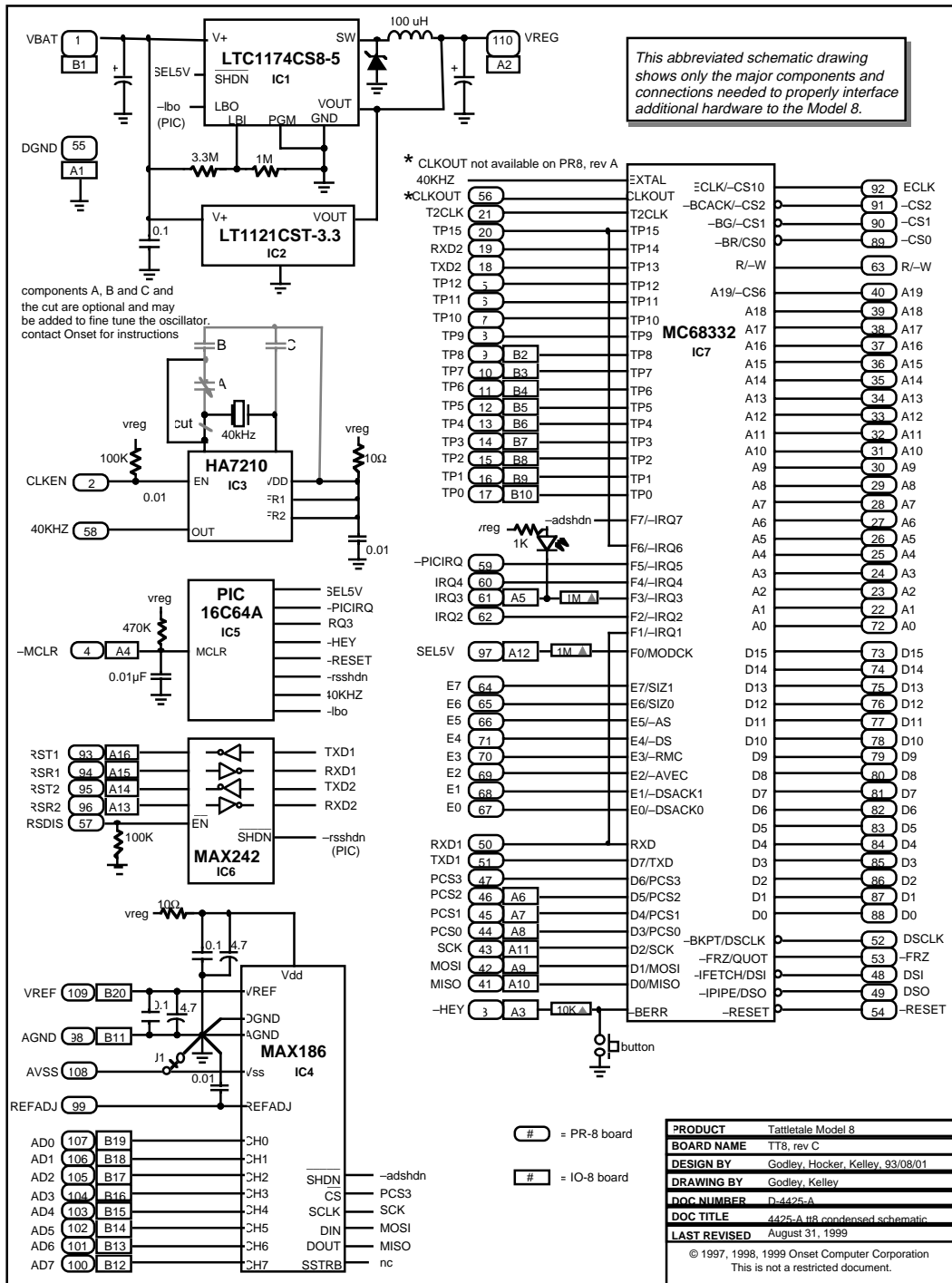


Figure 6-6: Schematic of the Model 8

Tattletale Model 8 Connections and Specifications

A major design goal for the Model 8 was minimum size. To achieve this, parts are mounted on both sides of the board. Table 6-1 shows the Model 8 specifications. Refer to Table 6-2 and Table 6-3 for specific pin functions.

Table 6-1: Model 8 Specifications

Size (inches)	2 x 3 x 0.5
Weight (oz.)	1
Processor	68332
Data capacity (RAM)	256K (or 1M)
Additional capacity	PCMCIA
Flash EEPROM	256K
A-D converter	12-bit
Analog channels	8
Max sampling rate (Hz)	100K
Digital I/O lines	up to 25
Count channels	up to 25
Minimum current	<200 μ A typical
Peak current	150mA
Main UART baud (default) at RS-232 Levels:	9600
TPU UART baud rates (others available):	The 14 TPU lines can be set to any standard rate up to 500K
Serial EEPROM (bytes)	7190
Voltage input	7 to 15V
Battery RAM backup	No
Real-time clock	Hardware
Programming languages	C, TxBASIC
Operating temperature range	-40 to +85° C
Relative humidity range	0 - 95% non-condensing

NOTE: TPU I/O pins may be configured through software as either a UART Rx or Tx. The TPU handles all timing and buffering of serial transmissions and therefore achieves a CPU independence equivalent to a hardware UART.

Operating Temperature Range

Model 8 components are specified to operate over a temperature range of -40°C to +85°C with the following exceptions. The switch used to enter the Background Debugging Mode (BDM) during the power up sequence has an operating range of -20°C to +70°C. The Light Emmiting Diode (LED) used to indicate a low signal on IRQ3 (pin 61 on PR-8, pin A-5 on IO-8) has an operating range of -30°C to +85°C.

Table 6-2: IO-8 Pin Functions

IO-8 Pin#	Signal	Function Description
A1	DGND	The digital ground. This is the negative return for the digital circuitry
A2	VREG	Positive supply
A3	-HEY	I/O with Interrupt on change pin (PIC)
A4	-MCLR	Master clear (active low reset); use to reset the Model 8
A5	-IRQ3	Interrupt request (level 3) PIC ACR
A6	PCS2	Peripheral chip select
A7	PCS1	Peripheral chip select
A8	PCS0	Peripheral chip select
A9	MOSI	Master-Out Slave-In
A10	MISO	Master-In Slave-Out
A11	SCK	QSPI serial clock
A12	SEL5V	Clock mode select
A13	RSR2	RS-232 receive (Jack 2)
A14	RST2	RS-232 transmit (Jack 2)
A15	RSR1	RS-232 receive (Jack 1)
A16	RST1	RS-232 transmit (Jack 1)
B1	VBAT	The main battery supply
B2	TP8	TPU channel 8
B3	TP7	TPU channel 7
B4	TP6	TPU channel 6
B5	TP5	TPU channel 5
B6	TP4	TPU channel 4
B7	TP3	TPU channel 3
B8	TP2	TPU channel 2
B9	TP1	TPU channel 1
B10	TP0	TPU channel 0
B11	AGND	The analog ground
B12	AD7	A/D channel 7
B13	AD6	A/D channel 6
B14	AD5	A/D channel 5
B15	AD4	A/D channel 4
B16	AD3	A/D channel 3
B17	AD2	A/D channel 2
B18	AD1	A/D channel 1
B19	AD0	A/D channel 0
B20	VREF	The reference voltage for the A/D, or output of the reference buffer amplifier

Table 6-3: PR-8 Pin Functions

PR-8 Pin#	Signal	Function Description
1	VBAT	The main battery supply
2	CLKEN	Clock enable
3	-HEY	Interrupt on change pin (PIC); bus error signal to 332
4	-MCLR	Master clear (reset) input/prog voltage input to PIC; active low reset
5	TP12	TPU channel 12
6	TP11	TPU channel 11
7	TP10	TPU channel 10
8	TP9	TPU channel 9
9	TP8	TPU channel 8
10	TP7	TPU channel 7
11	TP6	TPU channel 6
12	TP5	TPU channel 5
13	TP4	TPU channel 4
14	TP3	TPU channel 3
15	TP2	TPU channel 2
16	TP1	TPU channel 1
17	TP0	TPU channel 0
18	TXD2	TPU UART transmit
19	RXD2	TPU UART receive
20	TP15	TPU channel 15
21	T2CLK	TPU clock in
22	A1	CPU Address Line (A1)
23	A2	CPU Address Line (A2)
24	A3	CPU Address Line (A3)
25	A4	CPU Address Line (A4)
26	A5	CPU Address Line (A5)
27	A6	CPU Address Line (A6)
28	A7	CPU Address Line (A7)
29	A8	CPU Address Line (A8)
30	A9	CPU Address Line (A9)
31	A10	CPU Address Line (A10)
32	A11	CPU Address Line (A11)
33	A12	CPU Address Line (A12)
34	A13	CPU Address Line (A13)
35	A14	CPU Address Line (A14)
36	A15	CPU Address Line (A15)
37	A16	CPU Address Line (A16)
38	A17	CPU Address Line (A17)
39	A18	CPU Address Line (A18)
40	A19	CPU Address Line (A19)
41	MISO	Master-In Slave-Out (QSPI)
42	MOSI	Master-Out Slave-In (QSPI)
43	SCK	QSPI serial clock
44	PCS0	QSPI chip select
45	PCS1	QSPI chip select
46	PCS2	QSPI chip select
47	PCS3	QSPI chip select
48	DSI	Development serial in
49	DSO	Development serial out
50	RXD1	SCI receive data
51	TXD1	SCI transmit data
52	DSCLK	Development serial clock
53	-FRZ	Freeze
54	-RESET	Reset
55	DGND	Digital ground

Table 6-3: PR-8 Pin Functions (Continued)

PR-8 Pin#	Signal	Function Description
56	NC	No connection
57	RSDIS	RS-232 driver disable
58	40KHZ	40KHz clock out
59	-PICIRQ	Interrupt request
60	-IRQ4	Interrupt request (level 4)
61	-IRQ3	Interrupt request (level 3)
62	-IRQ2	Interrupt request (level 2)
63	R/-W	Read/Write
64	E7	E ports 0-7 can be used for communicating and controlling peripherals and correspond to various specific functions. When not in specific use, they can serve as extra digital I/O lines.
65	E6	
66	E5	
67	E0	
68	E1	
69	E2	
70	E3	
71	E4	
72	A0	CPU Address Line (A0)
73	D15	CPU Data Line (D15)
74	D14	CPU Data Line (D14)
75	D13	CPU Data Line (D13)
76	D12	CPU Data Line (D12)
77	D11	CPU Data Line (D11)
78	D10	CPU Data Line (D10)
79	D9	CPU Data Line (D9)
80	D8	CPU Data Line (D8)
81	D7	CPU Data Line (D7)
82	D6	CPU Data Line (D6)
83	D5	CPU Data Line (D5)
84	D4	CPU Data Line (D4)
85	D3	CPU Data Line (D3)
86	D2	CPU Data Line (D2)
87	D1	CPU Data Line (D1)
88	D0	CPU Data Line (D0)
89	-CS0	Chip select
90	-CS1	Chip select
91	-CS2	Chip select
92	ECLK	Outputs 1/8 system clock (can also be used as chip select)
93	RST1	RS-232 transmit (jack 1)
94	RSR1	RS-232 receive (jack 1)
95	RST2	RS-232 transmit (jack 2)
96	RSR2	RS-232 receive (jack 2)
97	SEL5V	Clock mode select
98	AGND	The analog ground
99	REFADJ	A/D Reference Adjust
100	AD7	A/D channel 7
101	AD6	A/D channel 6
102	AD5	A/D channel 5
103	AD4	A/D channel 4
104	AD3	A/D channel 3
105	AD2	A/D channel 2
106	AD1	A/D channel 1
107	AD0	A/D channel 0
108	AVSS	Can be negative reference for A/D in bipolar mode or GND
109	VREF	The reference voltage for the A/D, or 4.096 V out
110	VREG	Positive supply

Model 8 Components

The CPU32 instruction processing core executes a superset of the industry standard MC68000 instructions. The CPU32 accepts most 68020 instructions, missing only bit field and CALLM, CAS, CAS2, PACK, RTM, UNPK instructions, the memory indirect program counter and memory indirect with postincrement / predecrement addressing modes $\{([bd,An],Xn,od), ([bd,An,Xn],od), ([bd,PC],Xn,od), ([bd,PC,Xn],od)\}$.

The CPU32 also offers several new instructions. LPSTOP drops power consumption by turning off the system clock. TBLS, TBLU, TBLSN and TBLUN perform highly optimized table lookup and interpolate functions. BGND allows the CPU32 to yield control to an external BDM controller.

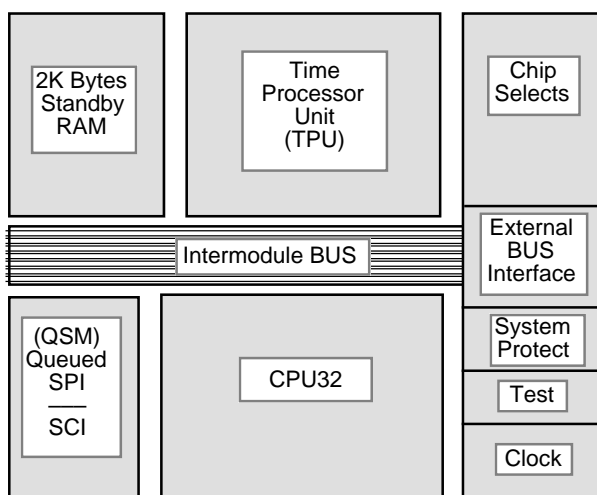


Figure 6-7: MC68332 Block Diagram

TPU Functions

The 68332 includes a time processor unit (TPU) which can perform match and capture operations on time, thus freeing up the CPU for other tasks. The TPU is essentially a special purpose slave-processor that controls two timers and sixteen I/O lines. Fourteen of the TPU channels are available for development on the Model 8: TP0-TP12. However, the IO-8 board only provides access to TP0-TP8. The following standard functions are available for each TPU channel:

NOTE: The TPU TCR2 pin is a floating input in the current Model 8 boards. The 68332 has an effective weak pulldown on this pin, so it does not contribute to static current drain, but it does interfere with the use of TCR2 as a time base for the TPU without an external pullup, which is only available from the SquishyBus.

Table 6-4: TPU Channel Functions

TPU Channel Function	Description
Input Capture	Latches counter value at rising or falling edge of input
Input Transition Counter	counts the number of transitions on an input
Output Compare	sets the output line high or low on counter match
Pulse-Width Modulation	toggles output at specified frequency and duty cycle
Synchronized Pulse-Width Modulation	low to high transitions have a time relationship to transitions on a second channel
Stepper Motor and Period	drives output for slewing a stepper (half or full step)
Period Measurement with Additional Transition Detect	
Period measurement with Missing Transition Detect	
Position-Synchronized Pulse Generator	
Pulse-Width Accumulator	

Consult the TPU reference manual and the Model 8 software manual for further details on standard TPU functions.

In addition to the standard functions, a selection of custom functions are available for your Model 8. The TPU UART is reached through Jack 2 (RS-232(2)). The custom functions are:

Table 6-5: Custom Functions for Jack #2

asynchronous I/O	UART I/O, 300 to 500K baud
microsecond stopwatch	real-time clock synchronization

Fourteen pins are available for use as general purpose I/O or as dedicated bus control signals. Running at 16 MHz and using simple C macros defined in DIO332.H, you can read, set, clear and toggle the port pins in under 1 μ S. The I/O lines operate at CMOS levels, sinking 1.6mA and sourcing 0.8mA. The thirteen available pins and their alternate functions are listed in Table 6-6.

Table 6-6: Thirteen I/O Line Alternate Functions

Port	BUS	IO8	PR8	Alternate Function
E0	DSACK0	N/A	67	Data and Size Acknowledge
E1	DSACK1	N/A	68	Data and Size Acknowledge
E2	AVEC	N/A	69	Interrupt ack auto vector request
E3	RMC	N/A	70	Read modify write cycle
E4	DS	N/A	71	Data strobe
E5	AS	N/A	66	Address strobe
E6	SIZ0	N/A	65	Data transfer size
E7	SIZ1	N/A	64	Data transfer size
D3	PCS0	A8	44	QSPI peripheral chip select
D4	PCS1	A7	45	QSPI peripheral chip select
D5	PCS2	A6	46	QSPI peripheral chip select
F2	IRQ2	N/A	62	External interrupt request
F4	IRQ4	N/A	60	External interrupt request

Analog Input Connections and Specifications

12-Bit, 8-Channel A-D Converter

Connecting to the Analog Inputs:

The Model 8 includes a MAX186 A/D converter for handling analog inputs. The MAX186 is a 12 bit, 8 channel device which can operate in unipolar or bipolar mode. The MAX186 can also operate with either external or internal reference. The factory standard configuration in which the Model 8 is supplied is internal reference. The internal reference is 4.096V unipolar full scale and $\pm 2.048V$ in bipolar mode.

NOTE: Bipolar mode is not available while using the IO-8 prototyping board; however, it is available while using the PR-8 board.

To use an external reference, simply provide your reference voltage to REFADJ or VREF. Using external reference, full scale can be as high as $V_{dd} + 50mV$. Sensors which are particularly appropriate to such ratiometric operation include strain gauges, thermistor bridges and other resistive divider circuits where the output voltage is proportional to the input voltage. None of the Model 8's channels are connected to a sensor (unless you soldered the thermistor and resistor to the board in **Section 2 -How to Connect and Setup the Model 8** and did not remove the components after testing the Tattletale). Refer to the Appendix for MAX186 data sheet.

Analog Inputs

The Model 8 has eight analog inputs to a 12-bit A-D converter. The data sheets on major Model 8 IC's are in the appendix of this manual. Table 6-7 shows the pins on the converter strip used by the IO-8 and the PR-8 boards.

Table 6-7: Analog Input Pin Specifications

IO-8 Pin #	PR-8 Pin #	Analog Input Pin Function
---	108	Negative reference for the A-D in bipolar mode (AVSS) Tied to AGND via jumper J1
B20	109	VREF (should be between 2.5V and 5V)
B19	107	A-D chan 0
B18	106	A-D chan 1
B17	105	A-D chan 2
B16	104	A-D chan 3
B15	103	A-D chan 4
B14	102	A-D chan 5
B13	101	A-D chan 6
B12	100	A-D chan 7
---	99	REFADJ
B11	98	Analog Ground (AGND)

Main UART

At power-up the hardware UART (at RS-232 levels) is programmed for 9600 baud, eight data bits, one stop bit and no parity. The 14 TPU lines can be set to any standard baud rate up to 500K.

The maximum software UART baud rates available at TTL levels are 300-500K. TPU I/O pins may be configured through software as either a UART Rx or Tx. The TPU handles all timing and buffering of serial transmissions and therefore achieves a CPU independence equivalent to a hardware UART.

Changing the Baud Rate

You can change the UART's baud rate by using the information in Table 6-8.

Table 6-8 is a table of clock rates for the 40 kHz crystal. The system clock can be set up to run at any of the frequencies given below (depending the power requirements and/or chip ratings of your particular system).

Change the baud rate by using the SerSetBaud command. Refer to [Section 5 - C Library Reference](#) for further information on using the SerSetBaud command.

Watchdog Timeouts:

12.800mS 51.200mS 204.800mS 819.200mS 6.554S 26.214S 104.858S 419.430S

Periodic Interrupt Periods:

\$001 = 100.00 uS \$0FF = 25.50 mS \$101 = 51.200 mS \$1FF = 13.056 S

Table 6-8: 68332 Clock Rates for the 40000 Crystal

System Frequency	/---- SYNCR ----\				/-- CS Access nS -\				/- Baud Error % -\				TMCR uS
	w0x0	w0x1	w1x0	w1x1	wt	0wt	1wt	2wt	9600	19.2	38.4	57.6	
0.160 MHz	0000	----	----	----	>1mS	>1mS	>1mS	>1mS	----	----	----	----	25.000
0.320 MHz	0100	4000	----	----	>1mS	>1mS	>1mS	>1mS	4.0	----	----	----	12.500
0.480 MHz	0200	----	----	----	>1mS	>1mS	>1mS	>1mS	----	----	----	----	8.3333
0.640 MHz	0300	4100	8000	----	>1mS	>1mS	>1mS	>1mS	4.0	4.0	----	----	6.2500
0.800 MHz	0400	----	----	----	>1mS	>1mS	>1mS	>1mS	----	----	----	----	5.0000
0.960 MHz	0500	4200	----	----	>1mS	>1mS	>1mS	>1mS	4.0	----	----	----	4.1667
1.120 MHz	0600	----	----	----	858	>1mS	>1mS	>1mS	----	----	----	----	3.5714
1.280 MHz	0700	4300	8100	C000	746	>1mS	>1mS	>1mS	4.0	4.0	4.0	----	3.1250
1.440 MHz	0800	----	----	----	659	>1mS	>1mS	>1mS	----	----	----	----	2.7778
1.600 MHz	0900	4400	----	----	590	>1mS	>1mS	>1mS	4.0	----	----	----	2.5000
1.760 MHz	0A00	----	----	----	533	>1mS	>1mS	>1mS	----	----	----	----	2.2727
1.920 MHz	0B00	4500	8200	----	486	>1mS	>1mS	>1mS	4.0	4.0	----	4.0	2.0833
2.080 MHz	0C00	----	----	----	446	927	>1mS	>1mS	----	----	----	----	1.9231
2.240 MHz	0D00	4600	----	----	411	858	>1mS	>1mS	4.0	----	----	----	1.7857
2.400 MHz	0E00	----	----	----	382	798	>1mS	>1mS	----	----	----	----	1.6667
2.560 MHz	0F00	4700	8300	C100	356	746	>1mS	>1mS	4.0	4.0	4.0	----	1.5625
2.720 MHz	1000	----	----	----	333	700	>1mS	>1mS	9.6	9.6	9.6	----	1.4706
2.880 MHz	1100	4800	----	----	312	659	>1mS	>1mS	4.0	----	----	----	1.3889
3.040 MHz	1200	----	----	----	294	623	952	>1mS	1.1	1.1	----	----	1.3158
3.200 MHz	1300	4900	8400	----	278	590	902	>1mS	4.0	4.0	----	----	1.2500
3.360 MHz	1400	----	----	----	263	560	858	>1mS	0.6	8.6	----	----	1.1905
3.520 MHz	1500	4A00	----	----	249	533	817	>1mS	4.0	----	----	----	1.1364
3.680 MHz	1600	----	----	----	237	508	780	>1mS	0.2	0.2	0.2	0.2	1.0870
3.840 MHz	1700	4B00	8500	C200	225	486	746	>1mS	4.0	4.0	4.0	4.0	1.0417
4.000 MHz	1800	----	----	----	215	465	715	965	0.2	7.8	7.8	7.8	1.0000
4.160 MHz	1900	4C00	----	----	205	446	686	927	4.0	----	----	----	0.9615
4.320 MHz	1A00	----	----	----	196	428	659	891	0.4	0.4	----	----	0.9259
4.480 MHz	1B00	4D00	8600	----	188	411	635	858	4.0	4.0	----	----	0.8929
4.640 MHz	1C00	----	----	----	181	396	612	827	0.7	7.3	----	----	0.8621
4.800 MHz	1D00	4E00	----	----	173	382	590	798	4.0	----	----	----	0.8333
4.960 MHz	1E00	----	----	----	167	368	570	771	0.9	0.9	0.9	----	0.8065
5.120 MHz	1F00	4F00	8700	C300	160	356	551	746	4.0	4.0	4.0	----	0.7812
5.280 MHz	2000	----	----	----	154	344	533	723	1.1	6.9	6.9	----	0.7576
5.440 MHz	2100	5000	----	----	149	333	516	700	4.0	9.6	9.6	----	0.7353
5.600 MHz	2200	----	----	----	144	322	501	679	1.3	1.3	----	1.3	0.7143
5.760 MHz	2300	5100	8800	----	139	312	486	659	1.3	4.0	----	4.0	0.6944
5.920 MHz	2400	----	----	----	134	303	472	641	1.4	6.6	----	6.6	0.6757
6.080 MHz	2500	5200	----	----	129	294	458	623	1.1	1.1	1.1	9.1	0.6579
6.240 MHz	2600	----	----	----	125	286	446	606	1.5	1.5	1.5	----	0.6410
6.400 MHz	2700	5300	8900	C400	121	278	434	590	0.8	4.0	4.0	----	0.6250
6.560 MHz	2800	----	----	----	117	270	422	575	1.6	6.3	6.3	----	0.6098
6.720 MHz	2900	5400	----	----	114	263	411	560	0.6	0.6	8.6	----	0.5952
6.880 MHz	2A00	----	----	----	110	256	401	546	1.8	1.8	----	----	0.5814
7.040 MHz	2B00	5500	8A00	----	107	249	391	533	0.4	4.0	----	----	0.5682
7.200 MHz	2C00	----	----	----	104	243	382	521	1.9	6.1	----	----	0.5556
7.360 MHz	2D00	5600	----	----	101	237	373	508	0.2	0.2	0.2	0.2	0.5435
7.520 MHz	2E00	----	----	----	98	231	364	497	2.0	2.0	2.0	2.0	0.5319
7.680 MHz	2F00	5700	8B00	C500	95	225	356	486	0.0	4.0	4.0	4.0	0.5208
7.840 MHz	3000	----	----	----	93	220	348	475	2.0	6.0	6.0	6.0	0.5102
8.000 MHz	3100	5800	----	----	90	215	340	465	0.2	0.2	7.8	7.8	0.5000
8.160 MHz	3200	----	----	----	88	210	333	455	2.1	2.1	9.6	9.6	0.4902
8.320 MHz	3300	5900	8C00	----	85	205	326	446	0.3	4.0	----	----	0.4808
8.480 MHz	3400	----	----	----	83	201	319	437	1.4	1.4	1.4	----	0.4717
8.640 MHz	3500	5A00	----	----	81	196	312	428	0.4	0.4	0.4	----	0.4630
8.800 MHz	3600	----	----	----	79	192	306	420	1.2	2.3	2.3	----	0.4545
8.960 MHz	3700	5B00	8D00	C600	77	188	300	411	0.6	4.0	4.0	----	0.4464
9.120 MHz	3800	----	----	----	75	184	294	404	1.1	1.1	5.7	1.1	0.4386

Table 6-8: 68332 Clock Rates for the 40000 Crystal (Continued)

System Frequency	SYNCR				CS Access nS				Baud Error %				TMCR us
	w0x0	w0x1	w1x0	w1x1	wt	Owt	1wt	2wt	9600	19.2	38.4	57.6	
9.280 MHz	3900	5C00	----	----	73	181	288	396	0.7	0.7	7.3	0.7	0.4310
9.440 MHz	3A00	----	----	----	71	177	283	389	0.9	2.4	8.9	2.4	0.4237
9.600 MHz	3B00	5D00	8E00	----	69	173	278	382	0.8	4.0	----	4.0	0.4167
9.760 MHz	3C00	----	----	----	67	170	272	375	0.7	0.7	0.7	5.6	0.4098
9.920 MHz	3D00	5E00	----	----	66	167	267	368	0.9	0.9	0.9	7.1	0.4032
10.080 MHz	3E00	----	----	----	64	163	263	362	0.6	2.5	2.5	8.6	0.3968
10.240 MHz	3F00	5F00	8F00	C700	63	160	258	356	1.0	4.0	4.0	----	0.3906
10.560 MHz	----	6000	----	----	60	154	249	344	1.1	1.1	6.9	----	0.3788
10.880 MHz	----	6100	9000	----	57	149	241	333	1.2	4.0	9.6	----	0.3676
11.200 MHz	----	6200	----	----	54	144	233	322	1.5	1.3	1.3	1.3	0.3571
11.520 MHz	----	6300	9100	C800	52	139	225	312	1.3	1.3	4.0	4.0	0.3472
11.840 MHz	----	6400	----	----	49	134	218	303	1.2	1.4	6.6	6.6	0.3378
12.160 MHz	----	6500	9200	----	47	129	212	294	1.1	1.1	1.1	9.1	0.3289
12.480 MHz	----	6600	----	----	45	125	205	286	0.9	1.5	1.5	----	0.3205
12.800 MHz	----	6700	9300	C900	43	121	199	278	0.8	0.8	4.0	0.8	0.3125
13.120 MHz	----	6800	----	----	41	117	194	270	0.7	1.7	6.3	1.7	0.3049
13.440 MHz	----	6900	9400	----	39	114	188	263	0.6	0.6	0.6	4.0	0.2976
13.760 MHz	----	6A00	----	----	38	110	183	256	0.5	1.8	1.8	6.2	0.2907
14.080 MHz	----	6B00	9500	CA00	36	107	178	249	0.4	0.4	4.0	8.4	0.2841
14.400 MHz	----	6C00	----	----	34	104	173	243	0.3	1.9	6.1	----	0.2778
14.720 MHz	----	6D00	9600	----	33	101	169	237	0.2	0.2	0.2	0.2	0.2717
15.040 MHz	----	6E00	----	----	31	98	164	231	0.1	2.0	2.0	2.0	0.2660
15.360 MHz	----	6F00	9700	CB00	30	95	160	225	0.0	0.0	4.0	4.0	0.2604
15.680 MHz	----	7000	----	----	29	93	156	220	0.1	2.0	6.0	6.0	0.2551
16.000 MHz	----	7100	9800	----	28	90	152	215	0.2	0.2	0.2	7.8	0.2500
16.320 MHz	----	7200	----	----	26	88	149	210	0.2	2.1	2.1	9.6	0.2451
16.640 MHz	----	7300	9900	CC00	25	85	145	205	1.5	0.3	4.0	0.3	0.2404
16.960 MHz	----	7400	----	----	24	83	142	201	1.4	1.4	1.4	2.2	0.2358
17.280 MHz	----	7500	9A00	----	23	81	139	196	1.3	0.4	0.4	4.0	0.2315
17.600 MHz	----	7600	----	----	22	79	135	192	1.2	1.2	2.3	5.7	0.2273
17.920 MHz	----	7700	9B00	CD00	21	77	132	188	1.1	0.6	4.0	7.4	0.2232
18.240 MHz	----	7800	----	----	20	75	129	184	1.1	1.1	1.1	1.1	0.2193
18.560 MHz	----	7900	9C00	----	19	73	127	181	1.0	0.7	0.7	0.7	0.2155
18.880 MHz	----	7A00	----	----	18	71	124	177	0.9	0.9	2.4	2.4	0.2119
19.200 MHz	----	7B00	9D00	CE00	17	69	121	173	0.8	0.8	4.0	4.0	0.2083
19.520 MHz	----	7C00	----	----	16	67	119	170	0.7	0.7	0.7	5.6	0.2049
19.840 MHz	----	7D00	9E00	----	15	66	116	167	0.7	0.9	0.9	7.1	0.2016
20.160 MHz	----	7E00	----	----	15	64	114	163	0.6	0.6	2.5	0.6	0.1984
20.480 MHz	----	7F00	9F00	CF00	14	63	111	160	0.5	1.0	4.0	1.0	0.1953
21.120 MHz	----	----	A000	----	12	60	107	154	0.4	1.1	1.1	4.0	0.1894
21.760 MHz	----	----	A100	D000	11	57	103	149	0.2	1.2	4.0	6.8	0.1838
22.400 MHz	----	----	A200	----	10	54	99	144	1.5	1.5	1.3	1.3	0.1786
23.040 MHz	----	----	A300	D100	8	52	95	139	1.3	1.3	1.3	4.0	0.1736
23.680 MHz	----	----	A400	----	7	49	92	134	1.2	1.2	1.4	4.2	0.1689
24.320 MHz	----	----	A500	D200	6	47	88	129	1.1	1.1	1.1	1.5	0.1645
24.960 MHz	----	----	A600	----	5	45	85	125	0.9	0.9	1.5	4.0	0.1603
25.600 MHz	----	----	A700	D300	4	43	82	121	0.8	0.8	0.8	0.8	0.1562
26.240 MHz	----	----	A800	----	3	41	79	117	0.7	0.7	1.7	1.7	0.1524
26.880 MHz	----	----	A900	D400	2	39	77	114	0.6	0.6	0.6	4.0	0.1488
27.520 MHz	----	----	AA00	----	1	38	74	110	1.6	0.5	1.8	0.5	0.1453
28.160 MHz	----	----	AB00	D500	1	36	72	107	1.5	0.4	0.4	1.8	0.1420
28.800 MHz	----	----	AC00	----	0	34	69	104	1.3	0.3	1.9	4.0	0.1389
29.440 MHz	----	----	AD00	D600	-1	33	67	101	1.2	0.2	0.2	0.2	0.1359
30.080 MHz	----	----	AE00	----	-2	31	65	98	1.1	0.1	2.0	2.0	0.1330
30.720 MHz	----	----	AF00	D700	-2	30	63	95	1.0	0.0	0.0	4.0	0.1302
31.360 MHz	----	----	B000	----	-3	29	61	93	0.9	0.1	2.0	0.1	0.1276
32.000 MHz	----	----	B100	D800	-4	28	59	90	0.8	0.2	0.2	2.1	0.1250

Current Drain

In high voltage mode, the bulk of the current drain will come from the MC68332. The 68332 will draw about 1mA per MHz, but may draw upwards of 70mA when performing tasks which are I/O intensive. By way of comparison, in the 5V mode the PIC and other components should draw less than 10mA. In 3.3V mode, the 68332 will draw about 20-30µA. Plan accordingly if you intend to use VREG for peripheral devices.

The most important point to remember is that the current drain is largely dependent on functions you have the Model 8 perform, i.e., it is software dependent. For this reason, the only practical way to determine whether you can power your peripherals from VREG is through experimentation with your application and direct measurement.

The Tattletales typical current drain is a strong function of the program it is running. The current drain is low while waiting for a character from the UART. Be sure to disconnect the RS-232 cable from the Tattletale when it is not in use as the interface drivers take an additional 1 to 2mA.

Table 6-9: Current Drain Due while using Low Power Modes

Current Modes on the Tattletale Model 8			
SimSetFsys	A-D on	A-D off	System Clock
640000	17.7mA	15.9mA	640 kHz
3680000	26.2mA	24.3mA	3.68 MHz
6080000	32.8mA	31.0mA	6.08 MHz
7360000	36.3mA	34.4mA	7.36 MHz
9120000	41.4mA	39.5mA	9.12 MHz
11200000	46.8mA	45.0mA	11.2 MHz
12800000	50.9mA	49.0mA	12.8 MHz
14720000	56.0mA	54.2mA	14.72 MHz
16000000	60.0mA	58.1mA	16.0 MHz

NOTE 1: After using the SimSetFsys command you must use the SerSetBaud command or the baud rate will be incorrect.

NOTE 2: With the system clock at 640 kHz, the baud rate can only be as high as 19200.

NOTE 3: These frequencies were chosen so that the most baud rates (with the least error) would be available, but if you do not need to communicate you may run at any frequency.

Table 6-10: Digital I/O Line Specifications (from Motorola)

Item	Test Condition	Min.	Max.	Units
Input "High" Voltage	—	2.0	5.0	Volts
Input "Low" Voltage	—	0	0.8	Volts
Output "High" Voltage	Sourcing 200 μ A	2.4	—	Volts
Output "High" Voltage	Sourcing 10 μ A	4.3	—	Volts
Output "Low" Voltage	Sinking 1.6mA	—	0.4	Volts
Input Capacitance	25°C, f=1 MHz	—	12.5	pF

A-D Converter Circuit

Figure 6-8 is a schematic of the A-D converter on the Model 8.

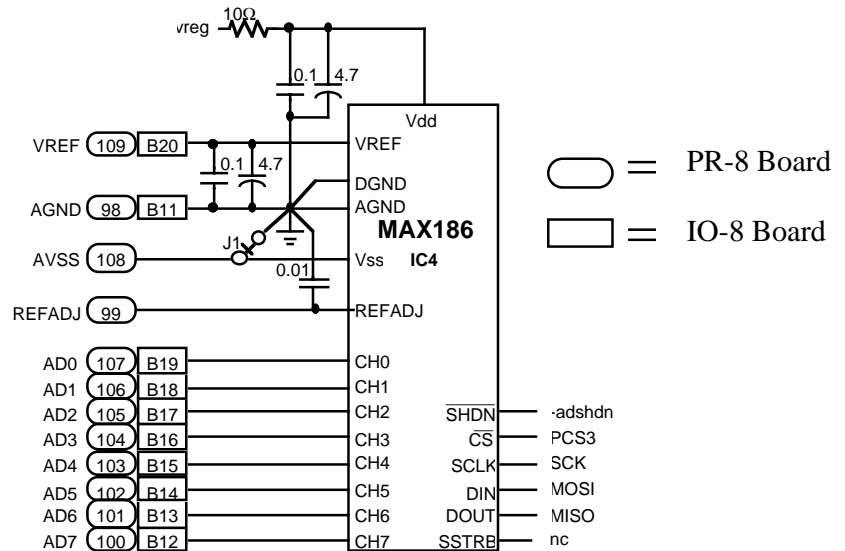


Figure 6-8: Diagram of A-D Regulator Circuit

Using the Model 8 in Bipolar Operation

Refer to the [“Converting the MAX186 A-D Converter to Bipolar Operation”](#) procedure on page 7-1.

RESET

The Model 8's reset is on pin 54 of the PR-8 board (it is not available on the IO-8 board). $\overline{\text{RESET}}$ is active low, but is controlled by the PIC. Do not reset using this line. If you need an external reset use $\overline{\text{MCLR}}$, pin 4 on the PR-8 and A4 on the IO-8.

Memory

RAM

Depending on the configuration of your Model 8, you probably have 256K or 1M of

static RAM. For larger data storage needs, we have an adapter that accepts PCMCIA memory cards.

Flash Memory

Depending on the configuration, your Model 8 may have 256K or 1M of Flash EEPROM (not available at the time of this writing). The Flash parts can be programmed and erased quickly. Further, they have a near zero current drain while inactive.

QSM - Queued Serial Module

SCI - Serial Communications Interface

The SCI can be used to communicate with external devices and can be reached on both the IO-8 and PR-8 through UART jack 1 (RS-232(1)). The SCI UART operates at rates from 64 baud to 500 KBaud. It features advanced error detection, full duplex operation, selectable word length, receiver wakeup and parity generation and detection.

QSPI - Queued Serial Peripheral Interface

The 68332 also has a versatile Queued Serial Peripheral Interface which operates with a minimum of processor overhead at rates up to 4 MBaud and features four separate chip selects. One of these chip select lines (PCS3) controls the Model 8's A/D converter. The other three may be multiplexed to eight and used along with MOSI, MISO and serial clock (SCK) to connect other serial devices.

SIM - System Integration Module

System Clock

The 68332 derives its system clock from a 40 KHZ crystal which is frequency multiplied to any rate from 160 KHz to 16 MHz, in steps of 160 KHz. This frequency can be changed at any time under software control and will settle within about one millisecond, thus allowing you to dial up just the amount of power you need.

The SIM includes a software watchdog timer. The watchdog can be used to prevent the software from running away or becoming trapped in an endless loop. The watchdog will timeout and issue a reset if not periodically serviced. The Model 8's watchdog timer is defaulted to a timeout period of approximately 62msec. User developed programs must service the watchdog to avoid sending the Model 8 into a continuous reset loop. The watchdog can be disabled; however it can only be enabled on reset. See example program watchdog.c.

The SIM also includes a periodic interrupt timer (PITR) which is clocked by a signal derived from the buffered crystal oscillator (EXTAL) input pin. The periodic timer can be set up as a real time clock by configuring it to generate a 1 second interrupt. Consult the Motorola SIM manual for further details.

Peripheral Chip Selects

The 68332 has 12 programmable chip selects that assign memory and peripheral base addresses, block sizes, 8/16 bit port size, read/write permission, address/data strobe timing and wait state generation. Four of these chip selects are available for

development: CS0, CS1, CS2 and CS10. Unlike the QSPI peripheral chip selects, the SIM chip selects cannot be MUXed.

BDM (Background Debug Mode)

The 68332 features a background debug mode which, in the Model 8, is used to institute emergency procedures when the flash EEPROM (program source) is lost or the 68332 hangs up. Under such conditions, the watchdog will timeout and send the 68332 into a reset loop. The 332's reset line is connected to the RTCC line on the PIC part. The PIC can be programmed to count resets and, after a predetermined number of consecutive resets, initiate BDM by pulling DSCLK low.

PIC 16C64

The 16C64 is an EPROM based 8 bit CMOS microcontroller. One of the many features of this microcontroller is low current drain. The Model 8 is designed to utilize the PIC to survive extended periods of near inactivity while monitoring for the start of an event.

In the low power mode, the PIC takes full control of the Model 8 while the 68332 is stopped. While the 68332 is stopped, the PIC can be programmed to wakeup at fixed intervals and check the sensor inputs for measurements which would indicate the need of the 332. Alternatively, the PIC can be tipped off by an interrupt to wake the 332.

The 16C64 includes a Real Time Clock/Counter (RTCC) module. The RTCC features an 8-bit timer/counter, read/write, 8 bit software programmable prescaler, internal or external clock select, interrupt on overflow from FFh to 00h and edge select for external clock. In the Model 8, the RTCC is used to monitor the 332 through the reset line. RTCC can thus be configured to start BDM emergency procedures on the 332. Consult the Microchip data sheet for further details (data sheet not provided by Onset).

Oscillator

The Model 8 uses a Harris HA7210; low power crystal oscillator with a Statek 40 KHz tuning fork quartz crystal. The crystal has an initial accuracy of $\pm 0.003\%$ at 25°C . , and will operate from -40°C to $+85^{\circ}\text{C}$ with a stability of about $-0.035 \text{ ppm}/^{\circ}\text{C}^2$. This crystal also has excellent aging characteristics; however, like any crystal it will change over time. Refer to Figure 6-9 for the oscillator error curve.

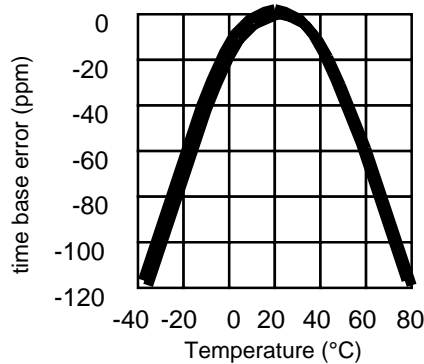


Figure 6-9: Graph of Oscillator Error vs Temperature

For those applications requiring more accuracy, the Model 8 includes a provision for adding an external time base. To add an external time base, first disable the crystal oscillator by pulling pin 2 on the PR-8 (CLKEN) low. The external time base can then be added on pin 58 of the PR-8 (40KHz).

Model 8 Accessories

IO-8 Prototyping Board

The IO-8 provides a place to build signal conditioning circuits. More importantly, the IO-8 provides an easy place to connect the DC power source to.

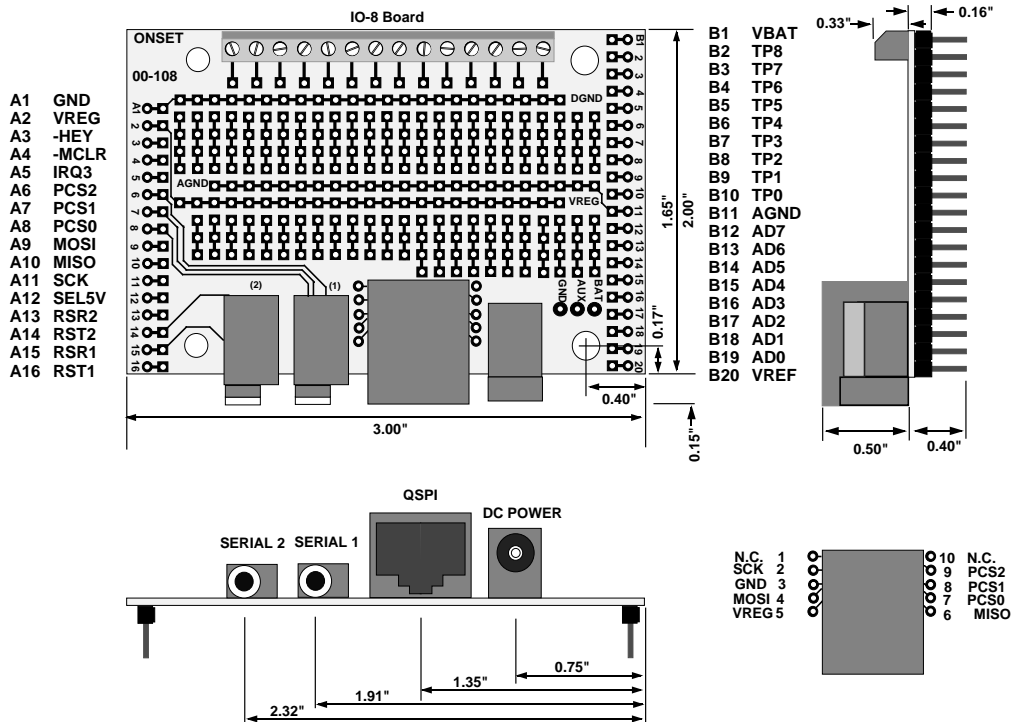


Figure 6-10: IO-8 Prototyping Board

PR-8 Prototyping Board

The PR-8 has all the benefits of the IO-8 prototyping board plus many more. This board connects to the Model 8 using two elastomeric connectors (SquishyBus) and gives control to the user for 110 pins as opposed to only 55 pins on the IO-8 board.

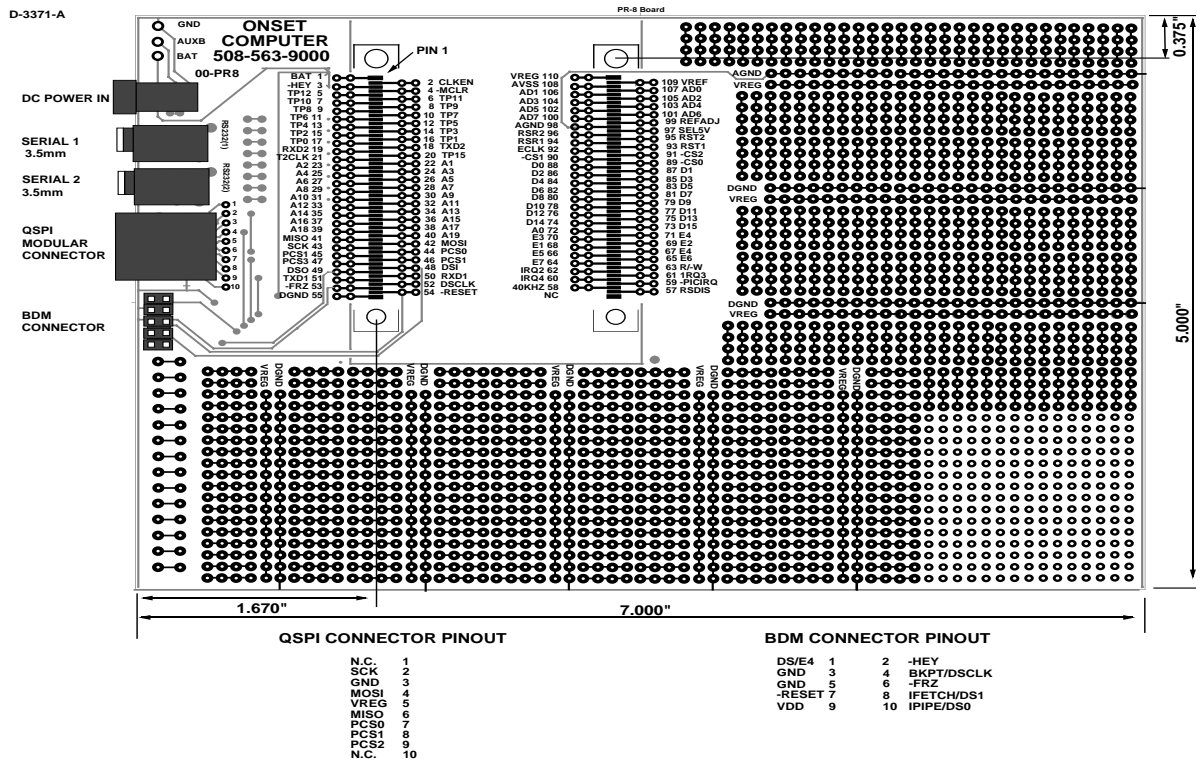


Figure 6-11: PR-8 Prototyping Board

Aztec C Compiler for DOS

This will allow you to write programs for the Model 8 in C language. If you plan on using any of the functionality of the BYOB (Build Your Own Basic) with TxBASIC, you will need a C compiler.

Aztec C Source Level Remote Debugger

This source level debugger will aid you in locating and correcting programming errors in C.

Tattletale 8 C Libraries for DOS

This package includes all the functions that are specific to the Tattletale Model 8, using the C language.

SquishyBus Connectors

These elastomeric connectors are used to connect the Tattletale Model 8 to the PR-8 prototyping board. These connectors feature a solderless connection.

PCMCIA Card Adapter (Obsolete, contact Onset for alternate part.)

This adapter will allow you to purchase PCMCIA cards of whatever size is available on the market and be used with the Tattletale Model 8. We have tested individual software routines that will read and write the following cards throughout their address range. We will add more to this list as we test them.

Flash:

Intel Series 2+	20MB	P/N IMC020FLSP-15/20
Intel Series 2+	4MB	P/N IMC004FLSP-15/20
AMD	1MB	P/N AmC001BFLKA

SRAM:

FUJITSU	256K	P/N 90811-20
MITSUBISHI	2MB	P/N MF32MI-L6DAT-00

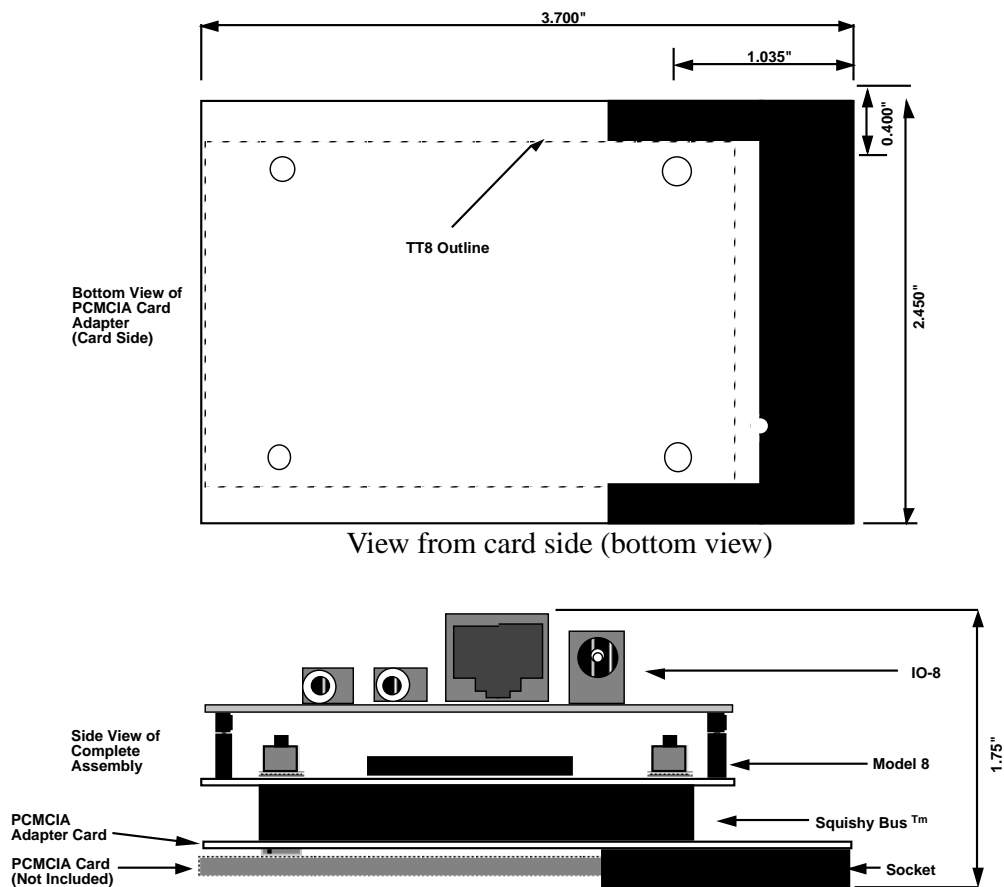


Figure 6-12: Side View of IO-8, TT8 and PCMCIA Adapter Assembly

Section 7 - Application Notes for the Model 8

Converting the MAX186 A-D Converter to Bipolar Operation

Hardware Modification

To allow bipolar operation **AVSS** must be disconnected from digital ground and connected to a $-5V$ supply. To disconnect from digital ground, jumper J1 on the Model 8 must be cut. This jumper is located right next to a thru-hole on the top of the board near the A-D converter and very close to the socket strip (see Figure 7-1).

NOTE: Early releases of the *General Release Schematic* do not show a segment of the direct connection between AVSS and Vss, but it does exist.

It is not masked and therefore is bright gold. To help verify you have identified the correct jumper trace it directly back to pin 9 on the converter. Cut the trace taking care not to cut any adjacent traces and make sure it is completely cut. When the jumper is cut the circuit in Figure 7-2 can be built to provide the $-5V$ supply.

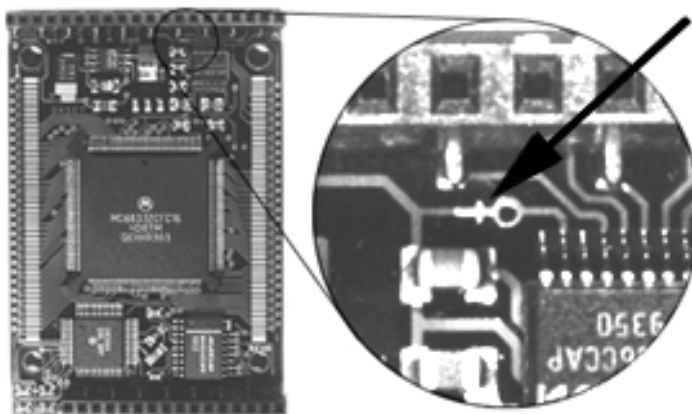


Figure 7-1: Location of the Trace for Bipolar A-D Operation

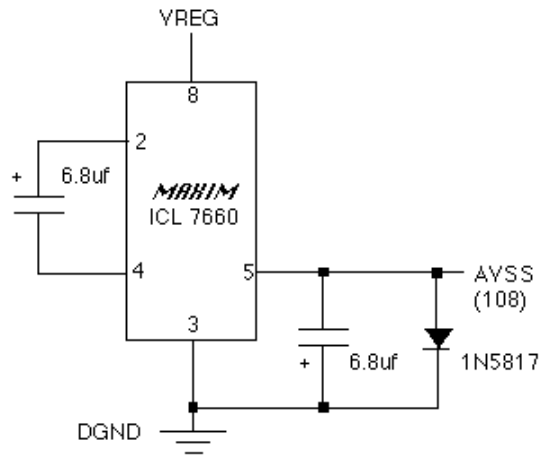


Figure 7-2: -5V Circuit for Bipolar A-D Operation

Software Modification

To enable bipolar mode operation you need to make two software modifications to the TT8LIB.H header file.

1. Modify the definition of AtoDtoMillivolts() to read as follows:

```
#define AtoDtoMilliVolts(adval)  ( ((adval) <<1) >> 4)
```

The MAX186 always leaves a 0 in the MSB of a valid returned value. This macro definition with the shift operators receives the sign bit in the MSB so a sign extend can be done on the raw value. (see the MAX186 data sheet in the Appendix).

2. Add the following prototype to TT8LIB.H:

```
short MAX186ReadChan(short chan, short bipolar, short diff, short pmode);
```

After the above changes are made, to make a bipolar conversion in a program, use the following code fragment:

```
unsigned char ADsel[] = {0, 4, 1, 5, 2, 6, 3, 7};
MAX186PowerUp();
MAX186ReadChan(ADsel[chan], 1, 0, pdExtClk);
```

Convert Bipolar Input to Unipolar

Signals that go both Positive and Negative

We have been asked how to interface a bipolar signal to a Tattletale. Some Tattletaes (like the Model 4A and 6) can be made to run in bipolar mode (make sure to add a Schottky diode between V- and ground), others won't. This note shows a simple way to convert a bipolar input to a positive only signal.

The simple schematic in Figure 7-3 shows an operational amplifier connected in an inverting configuration. Note that this design assumes a relatively low impedance signal is driving V_{in} .

The only tricky thing we have done is to give a positive bias to the non-inverting input to the amplifier. Remembering that op-amps adjust V_{out} so that the inverting and non-inverting inputs have the same voltage. The equation for this configuration is:

$$V_{set} = V_{in} + (V_{out} - V_{in}) * R1 / (R1 + R2),$$

or, rearranged a little: $V_{out} = ((R1 + R2) * (V_s - V_{in}) / R1) + R1$

You can change this to two equations, one for gain, another for offset:

$$\text{Gain} = \Delta V_{out} / \Delta V_{in} = -R2 / R1, \quad \text{and offset (} V_{out} \text{ for } V_{in} = 0) \quad \text{Offset} = V_{set} * (R1 + R2) / R1.$$

$$\text{or } R2 / R1 = -\text{Gain}, \quad \text{and } V_{set} = \text{Offset} * R1 / (R1 + R2)$$

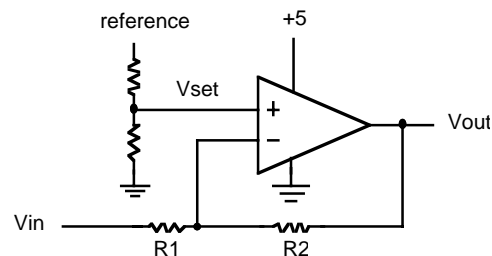


Figure 7-3: Circuit for Converting Bipolar to Unipolar

Example Application:

Suppose we had a signal that ranged from $-0.5V$ to $+0.5V$, and wanted to translate that to a voltage we could read with our positive signal only A-D. We first add a $2.5V$ reference to the A-D and translate our input to a signal ranging from 0 to $2.5V$. For this we would need a gain of 2.5 , and an offset of $1.25V$. Our circuit actually gives a negative gain so that $-0.5V$ would translate to $2.5V$, and $+0.5V$ will translate to $0V$. From the gain and offset equations we can find the ratio $R2/R1$ (2.5) and V_{set} ($0.357V$). If we use standard 1% resistors and the op-amp from Linear Technology we get the schematic shown in Figure 7-4.

We show a $2.5V$ reference, also from Linear Tech, fed by a $3K$ resistor and divided down to reach the $0.357V$. The resistors are not the exact values that we need to give the gain and offset we wished for, but come very close. The LT1077's input offset current is $0.25nA$ max., giving an offset of about $25\mu V$ at the input. This, combined with the input offset voltage of $40\mu V$ max. is still less than 1 LSB of a 12-bit converter.

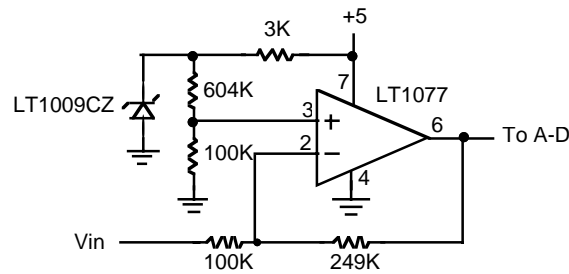


Figure 7-4: Another Circuit for Converting Bipolar to Unipolar

Operational Amplifiers and Instrumentation Amplifiers

For those of you that are not familiar with the terms “operational amplifier” and “instrumentation amplifier”, this very brief explanation should be enough to get you started.

Operational Amplifiers

The op amp (as they are commonly called) is an amplifier with both an inverting and non-inverting input, and a single output. For our purposes we will treat it as having an infinite gain, and no current flows through either input. Two commonly used circuits for op amps are shown in Figure 7-5, with the ‘-’ designating the inverting and the ‘+’ designating the non-inverting inputs.

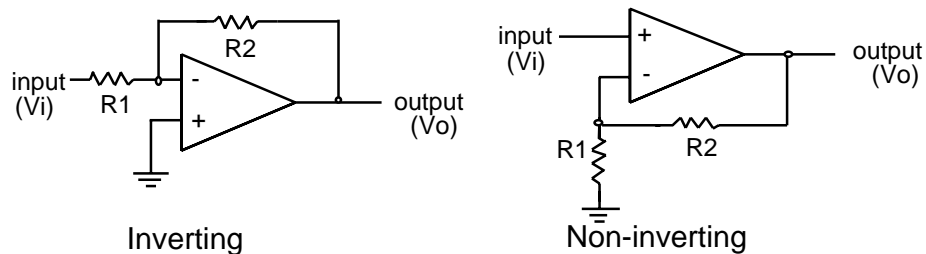


Figure 7-5: Operational Amplifiers

The Inverting Configuration

The inverting circuit has a gain of $-R2/R1$. This makes sense since the op amp will do what is needed to keep the + and - inputs at the same voltage, and any current flowing into the input resistor (R1) will also flow through the ‘feedback resistor’ (R2). An input voltage V_i will cause a current $V_i/R1$, and cause an output voltage V_o of $-R2(V_i/R1)$. Note the minus sign: the current flows through R2 away from ground if the current through R1 is flowing toward ground.

An op amp is not magic and it cannot give an output that is larger than its positive supply or lower than its negative supply. The Tattletale has a regulated 5V, but its negative supply is ground, so most applications will supply the op amp with ground and +5V. This makes it impossible to use the inverting circuit as drawn, unless the input is a negative voltage.

Advantages:

R1 protects input from external voltage.

Disadvantages:

To get a large gain, R1 must be small, loading the input.

Requires negative input (in Tattletale application, without adding a negative supply).

The Non-inverting Configuration

Following the same analysis we did for the inverting configuration, we find that the gain of the circuit is $(R1+R2)/R1$ for the non-inverting circuit. In this case we are limited to positive inputs, since the Tattletale has no negative supply.

Advantages:

High input impedance (will not load down your signal source).

Disadvantages:

Requires positive input (in Tattletale application, without adding a negative supply).

Op amps are not perfect. Some limitations are described in Table 7-1.

Table 7-1: Op Amp Limitation Data

Item	Limitation
Current drain:	In your Tattletale application you won't want to use any more power than needed. Check the current drain specs on the part you choose.
Supply voltage:	Not all op amps work with supply voltages of +5 and ground. In some applications you may even want your circuit to work while the Tattletale is running from 3 volts.
Max output swing:	Most op amps can only drive to within about a volt of their positive supply, others to only about a volt of their negative supply.
Input voltage range:	Not all op amps are pleased with inputs close to the supply lines; some will work with input voltages <u>below</u> ground. Don't be confused by the inverting configuration described above. In that circuit the input voltage is always at ground.
Input offset voltage:	The input offset voltage can be understood as the voltage difference between the positive and negative inputs needed to make the output voltage zero. Ideally this should be zero; in reality many op amps have input offsets of 10mV or more. This offset will appear as an error at the output equal to the input offset times the gain of the circuit.
Input offset current:	Many op amps have FET inputs that take almost no current at the input, but some can have sizable currents flowing (nanoamps) into or out of the inputs. This can lead to sizable errors if your input resistor is a large value.

That's not all, but should be enough to give you some idea of what to look for in op amp data sheets.

The Op Amps we use:

We will show some of our biases by giving you short descriptions of three op amps we like. Note that each has its own strengths.

LT1077, 1078, 1079

Single, dual and quad amplifier from Linear Technology (408) 432-1900. This part has lovely current drain, input voltage range and input offset voltage specs.

Current drain:	40 μ A typical for each amplifier (μ A = microamp, a millionth of an amp)
Supply voltage:	3 volts minimum
Max output swing:	ground +6mV to positive supply -1 Volt (mV= millivolt, 1/1000V)
Input voltage range:	ground -5V to positive supply
Input offset voltage:	30 μ V typical (μ V = microvolt, a millionth of a volt)
Input offset current:	0.25nA max. (nA = nanoamp = a milli-micro amp)

TLC1078

Dual amplifier from Texas Instruments (800) 232-3200. Note the superb input offset current specifications.

Current drain:	15 μ A typical for each amplifier
Supply voltage:	1.4V minimum
Max output swing:	ground to positive supply -1 Volt
Input voltage range:	ground -0.2V to positive supply -1 Volt
Input offset voltage:	160 μ V typical
Input offset current:	0.1pA max. (pA = picoamp = a micro micro amp)

ALD1704, 1701, 2701, 4701

Single, dual and quad amplifier from Advanced Linear Devices (408) 720-8737. This part drives line to line.

Current drain:	120 μ A typical for each amplifier
Supply voltage:	2.0V minimum
Max output swing:	ground to positive supply -1 Volt
Input voltage range:	ground -0.2V to positive supply -1 Volt
Input offset voltage:	2mV <u>for the premium version</u>
Input offset current:	25pA max. (pA = picoamp = a micro micro amp)

NOTE: If you need the specifications for gain-bandwidth product, noise characteristics, stability, offset drift, temperature coefficients or output drive characteristics, refer to the data sheets from the manufactures. We do not supply them in this manual. These and other parameters are important in some applications, but we don't have the space to cover everything.

Instrumentation Amplifiers

What if your sensor provides two outputs (like a bridge circuit)? An operational amplifier has both an inverting and a non-inverting input, but one is needed to set the gain of the circuit. You can build an instrumentation amplifier from two operational amplifiers and a small pile of precision resistors using the circuit below:

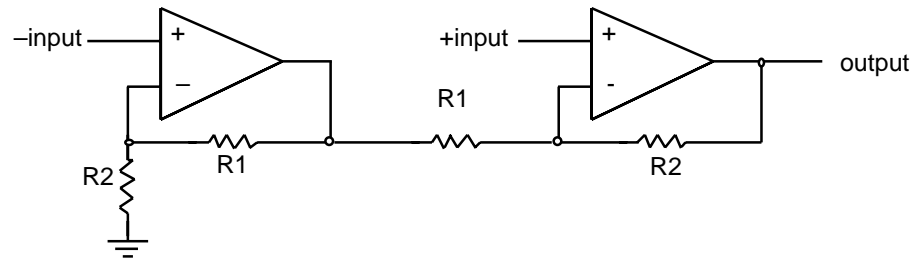


Figure 7-6: Instrumentation Amplifier Circuit

This circuit has a gain $G = (R1+R2)/R1$, and works over a range of input voltages that go from ground to 'Max positive output' * $R2/(R1+R2)$.

LT1101 Instrumentation Amplifier:

Linear Technology's LT1101 packages this all in one 8-pin part and can be set to a gain of 10 or 100 with no external parts.

Current drain:	75 μ A typical
Supply voltage:	2V minimum
Max output swing:	ground +4mV to positive supply -1V
Input voltage range:	ground +70mV to positive supply -2V
Input offset voltage:	50 μ V typical
Input offset current:	8nA max.

Digital Output Protection

Digital outputs are just as vulnerable as inputs, and they cannot be driven, even transiently, with a signal larger than the 'V' supply or lower than ground. They also have relatively low drive capability and should be buffered if they are expected to drive substantial loads. Some examples of output buffering are given in Figure 7-7.

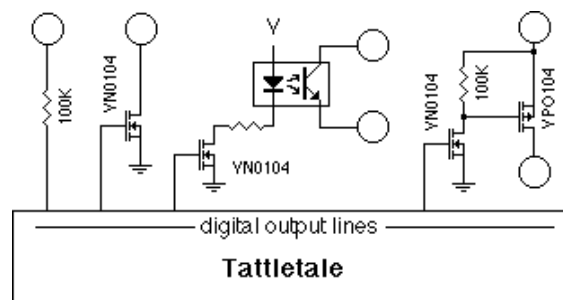


Figure 7-7: Digital Output Protection Circuit

100K resistor The resistor in the first example will provide protection but very little drive current. It is suitable for connecting to CMOS inputs driven from a separate supply (where there is a possibility that the supplies won't track).

VFET driver The VN0104 is an N-channel VMOS transistor with an on-resistance of about 4 ohms when the gate is +5V and lots of megohms when the gate is grounded. In the configuration shown, it is suitable for driving small relays or opto-isolators. The VN0104 has a V_{ds} of 40V, so the solenoid can be powered directly from V_{bat} or any other convenient source. The VN0104 and VP0104 parts are available from Supertex, 1225 Bordeaux Dr., Sunnyvale CA 94088-3607, phone (408) 744-0100.

Power switch The last example shows how to use two VFETs (one P-channel and one N-channel) to, for example, turn on and off a separate voltage to enable V_{bat} to a separate regulator.

Digital Input Protection

The digital inputs should be protected from signals that exceed the Tattletale's internal bus supply levels. Figure 7-8 shows five techniques that protect the inputs from large current spikes which may cause latch-up.

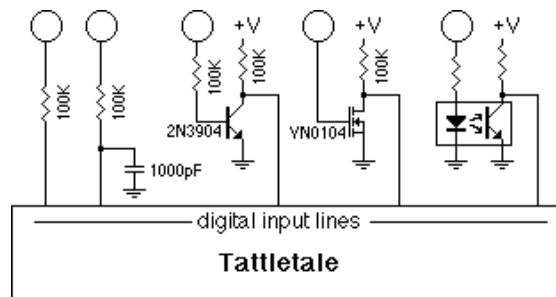


Figure 7-8: Digital Input Protection Circuit

- Resistor** The resistor protects the input from surges by limiting the amount of current that can be injected.
- R - C** Adding a capacitor helps protect inputs from high voltage spikes caused by electrostatic discharge.
- Transistor** This solution isolates the Tattletale's inputs completely from the source, placing the transistor at hazard (the transistor is a lot easier to replace than the 64-pin microprocessor that the inputs are connected to).
- VFET** Same as the transistor, but takes less current.
- Opto-isolation** This solution has the advantage of total isolation of both the supply and the ground of the source from that of the microprocessor. However, it comes at the expense of a substantial current drain.

Input Protection

If sensors are mounted external to the Tattletale, and particularly if they are made so that they can be disconnected, you should provide some protection so that static discharge will not damage the Tattletale's A-D converter or your signal conditioning circuitry. Some input protection circuits are shown in Figure 7-9, Figure 7-10 and Figure 7-11.

Using just a Resistor for Protection

This is minimal protection for a line that goes out of the Tattletale box. For A-D converter inputs the resistor (plus impedance of the source) should not be larger than about 3K since the converter has a capacitor that must fill to 12-bit accuracy during its sampling time. Digital inputs can have values of 100K; the only limitation is input leakage and stray leakage on the board. Op amps will see an input offset equal to the product of the resistance and the input offset current.

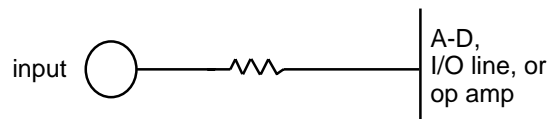


Figure 7-9: Input Protection with just a Resistor

Using a Resistor and a Capacitor for Protection

Figure 7-10 is slightly more complex, and helps protect the input from momentary voltage transients such as mild static discharge, but can significantly distort AC signals at the input. Make sure that the time constant of the circuit ($t = RC$) is short compared to the expected change rate of your input signal. Typical resistor and capacitor values of 100K and 0.1 μ F are fine for digital lines but analog lines must have very low resistor and capacitor values. Analog line resistors (plus source impedance) should never go above 3K and capacitors shouldn't go above 0.001 μ F.

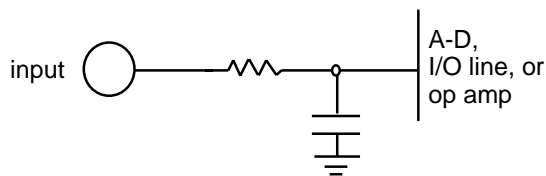


Figure 7-10: Input Protection with a Resistor and Capacitor

Using a Resistor, a Capacitor and a Zener Diode for Protection

Add a 6.3V zener diode and you will have very good input protection. The zener makes sure your input never gets far outside the supply lines, and the resistor limits the current into the zener. See the text above for the analog input lines. For the digital I/O lines, a 100K resistor is not inappropriate for both resistors. This circuit is very effective against static discharge, but is not adequate protection against lightning.

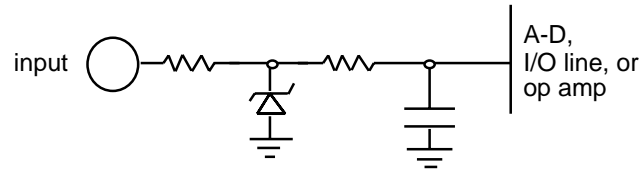
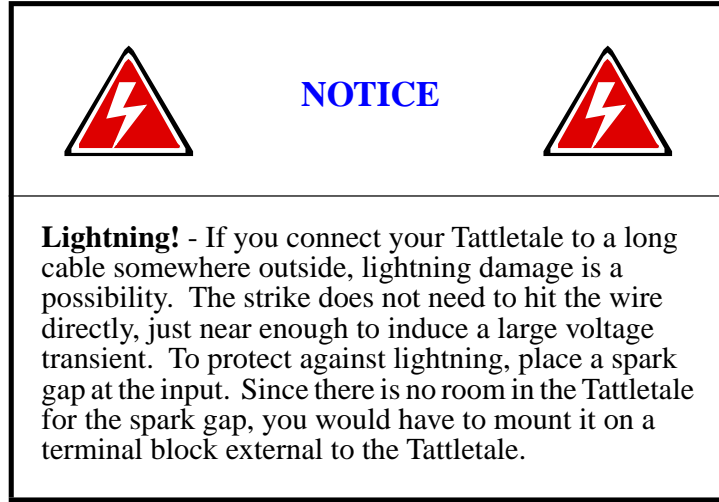


Figure 7-11: Input Protection with a Zener Diode



Using Flash Memory for Non-Volatile Data Storage

This application note provides generic flash programming routines that allow the flash to be burned when running from flash. This is done by moving the actual burn routine into RAM at 2F0000H, and copying/programming the 256 bytes pointed to in RAM to flash. Make sure the move to address (2F0000H) is not a problem with your system. As it is a define, (RAMMOVESTART), it may be readily changed. Note that RMSHI and RMSLO must also be changed.

This code will not allow blocks below 2000H to be written in order to avoid clobbering the TOM8 Monitor. Remove this protection at your own risk. For that matter, use this code at your own risk. The program is loaded at 2000H and therefore is fair game to be overwritten - there is no protection for your code!

The program can be located in the EXAMPLES directory.

```

/*****
**
**   flashbrn.c       Tattletale Model-8 code for reading and writing flash
**
**                   for non-volatile data storage

PLEASE NOTE!
These routines have the potential for causing great distress -
use them carefully!!

FLASH HAS A LIMITED NUMBER OF WRITES CYCLES - 10,000

```


- BE CAREFUL NOT TO EXCEED THIS LIMIT OR DATA MAY BE LOST!
This can be easy to do during development!

These are generic flash programming routines and allow the flash to be burned when running from flash. This is done by moving the actual burn routine into RAM at 2f0000H, and copying/programming the 256 bytes pointed to in RAM to flash. Make sure the moved to address (2f0000H) is not a problem with your system. As it is a define, (RAMMOVESTART), it may be readily changed. Note that RMSHI and RMSLO must also be changed.

This code will not allow blocks below 2000H to be written in order to avoid clobbering the TOM8 Monitor. Remove this protection at your own risk. For that matter, use this code at your own risk. The program is loaded at 2000H and therefore is fair game to be overwritten - there is no protection for your code!

If you find these routines useful, tell us so.
If you have some ideas about routines that would be useful to yourself and others please tell us too.
If you make any useful enhancements please share them with us and others via our BBS.

```
**      -tam

**      Copyright (C) 1994 ONSET Computer Corp.  All rights reserved.
**
**
*****/
#include      <tt8lib.h>
#include      <tpu332.h>
#include      <tat332.h>
#include      <sim332.h>
#include      <qsm332.h>
#include      <dio332.h>
#include      <stdio.h>
#include      <userio.h>

/*
**      AIMEL COMMAND DEFINITIONS AND FLASH DEFINES
*/

#define      FLASH_MFR_OFS      0
#define      FLASH_DEV_OFS      2

#define      FEE_QUERY          0x9090
#define      FEE_PROG           0x0505
#define      FEE_RESET          0xF0F0

#define      FEE_PROT_DIS1      0x8080
#define      FEE_PROT_DIS2      0x2020

#define      CR0                ((uspv) (0xAAAA))      /* 5555 << 1 */
#define      CR1                ((uspv) (0x5554))      /* 2AAA << 1 */
#define      CMDHDR(x)          *CR0=0xAAAA; *CR1=0x5555; *CR0=x;

#define      WORDS_PER_SECT      128
#define      BYTES_PER_SECT      (WORDS_PER_SECT * 2)

#define      RAMMOVESTART      0x2f0000
#define      RMSHI              0x002f
#define      RMSLO              0x0000

typedef enum
{
    FlashOk = 0,
```

```

FlashBadAddress,
flashBadID,
NotErased,
NoMatch
} Flasherr;

FlashErr      FlashID(ushort *startAddr, ushort *mfr, ushort *device)={
0x207c,RMSHI,RMSLO,0x4e90 };

FlashErr      FBurnBlock(register ushort *srcAddr, register ushort
*flashAddr)={ 0x207c,RMSHI,RMSLO,0x4e90 };
FlashErr      FBurnBlockMove(register ushort *srcAddr, register long block);

FlashErr      GetFlashID(void);
FlashErr      BlockErase(long block);
FlashErr      BlockErased(long block);
FlashErr      BlockWrite(long block, ushort *RamAddress);
FlashErr      BlockVerify(long block, ushort *RamAddress);

int           BlockDisplay(long block);
void          DisplayFlashMap(void);

// globals

static ushort RamArray[WORDS_PER_SECT];
static ushort *RamAddress;

/*****
**      MAIN
**
*****/
main()
{
int index;
ushort* pointer = (ushort*) 0x30000;
ulong  blocknum=1000;
char   cmd = 'M';

for (index = 0; index < WORDS_PER_SECT; index++)
    RamArray[index] = index | (index<<8);

RamAddress = RamArray;

InitTT8(NO_WATCHDOG,TT8_TPU);

loop:  QueryChar(  "\n\n\t<S>elect Block\
                \n\t<W>rite  Block\
                \n\t<E>rase  Block\
                \n\t<C>heck  Block\
                \n\t<R>ead   Block\
                \n\t<M>ap    Flash\
                \n\t<F>lash  ID\
                \n\t<Q>uit\
                \n\t default =" , cmd, "SWECRMFO", &cmd);

printf("\n");
switch (cmd)
{
case 'S' :
    if (!QueryNum("\nEnter Block Number","%ld","%ld",&blocknum))
        Reset();
    break;

case 'W' :
    if (BlockErased(blocknum))
    {
        printf("\nblock %ld not erased",blocknum);
    }
}
}

```

```

        if (!QueryYesNo("\nYou will overwrite existing data - proceed?", 'N'))
            break;
        }
        (BlockWrite(blocknum,RamAddress)) ?
        printf("\nblock %ld block not written",blocknum):
        printf("\nblock %ld written",blocknum);
        break;
    case 'E' :
        if (BlockErased(blocknum))
        {
            printf("\nblock %ld not erased",blocknum);
            if (!QueryYesNo("\nYou will destroy existing data - proceed?", 'N'))
                break;
            (BlockErase(blocknum)) ?
            printf("\nblock %ld cannot be erased",blocknum):
            printf("\nblock %ld erase complete",blocknum);
        }
        else
            printf("\nblock %ld already erased",blocknum);
        break;

    case 'C' :
        (BlockErased(blocknum)) ?
        printf("\nblock %ld contains data",blocknum):
        printf("\nblock %ld erased",blocknum);
        break;

    case 'R' :
        BlockDisplay(blocknum);
        break;

    case 'M' :
        DisplayFlashMap();
        break;

    case 'F' :
        GetFlashID();
        break;

    case 'Q' :
        Reset();
    }

goto loop;

(BlockVerify(blocknum,RamAddress)) ?
printf("\nblock %ld not verified",blocknum):
printf("\nblock %ld verified",blocknum);

(BlockErase(blocknum)) ?
printf("\nblock %ld will not erase",blocknum):
printf("\nblock %ld erase done",blocknum);

printf(" \n");
fflush(stdout);

} // end main

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

```

```

*****/
FlashErr GetFlashID(void)
{
    char*          dest = (char*) RAMMOVESTART;

    void (*source)  () = FlashID;
    void (*endsource) () = BlockErased;

    unsigned int   csorbt;
    unsigned int   csor7;

    ushort        mfr, device;
    ptr           FlashStart = (ptr) BASEADDR(*CSBARBT);

    //SET UP CHIP SELECTS TO ALLOW WRITES TO FLASH
    csorbt = *CSORBT;
    *CSORBT= 0x7cf0;      // 3 wait states for slowest flash

    printf("\nFLASH ID = ");
    //move FlashID routine into ram to run

    while (source != endsource)
        *dest++ = *((char*)source)++;

    FlashID((ushort *) FlashStart, &mfr, &device);

    //SET UP CHIP SELECTS TO DISALLOW WRITES TO FLASH
    *CSORBT = csorbt;

    if ( (mfr == 0x1F1F) /* ATMEL */ &&
        ((device == 0xDCDC) /* 29C256 */ ||
         (device == 0x5D5D) /* 29C512 */ ||
         (device == 0xD5D5)) /* 29C010 */ )
    {
        printf("ATMEL ");
        if (device == 0xDCDC)
            printf("29C256");

        if (device == 0x5D5D)
            printf("29C512");

        if (device == 0xD5D5)
            printf("29C010");
        return(flashOk);
    }
    else
    {
        printf("bad device ID");
        return(flashBadID);
    }
}

/*****
**
**
**
*****/
FlashErr FBurnBlockMove(    register ushort *srcAddr,
                            register long block)
{
    unsigned int   csorbt;
    unsigned int   csor7;

```

```

char*          dest = (char*) RAMMOVESTART;
void (*source)  () = FBurnBlock;
void (*endsource)  () = FlashID;
ushort *flashAddr;

if (block < 321)
    return(FlashBadAddress);

flashAddr = (ushort*) (block*256);

//SET UP CHIP SELECTS TO ALLOW WRITES TO FLASH
csorbtc = *CSORBT;
*CSORBT= 0x7cf0;          // 3 wait states for slowest flash

//move flash burn routine into ram
dest = (char*) RAMMOVESTART;
while (source != endsource)
    *dest++ = *((char*)source)++;

//Burn Flash
FBurnBlock(srcAddr, flashAddr);

//SET UP CHIP SELECTS TO DISALLOW WRITES TO FLASH
*CSORBT = csorbtc;
return(flashOk);
}

/*****
**      FBurnBlock
**
**      Assumes:
**          16.0 MHz CPU-32
**          srcAddr is valid even RAM memory address.
**          Interrupts will not occur.
**
**      *****/
FlashErr FBurnBlock(    register ushort *srcAddr,
                      register ushort *flashAddr)
{
    register volatile ushort    x, y;
    long                        j;

    for (j = 0; j < WORDS_PER_SECT; j++)
        *flashAddr++ = *srcAddr++;
    do
    {
        x = *flashAddr & 0x4040;    /* bit 6 */
        y = *flashAddr & 0x4040;
        }while (y != x);          /* wait for toggle to end */

    return (flashOk);

}    /* FBurnBlock() */

/*****
**      Do Not Move - used as end address for move to upper memory
**      FlashID
**
**      *****/
FlashErr FlashID(ushort *startAddr, ushort *mfr, ushort *device)
{
    register short    count;

```

```

long    i;
int     index;

CMDHDR(FEE_QUERY);
for (i = 0; i < 1000; i++)
{
    for (count = 0; count < 3; count++) // Delay10uS()
        ;
}

*mfr = *((ushort *) ((ptr) startAddr + FLASH_MFR_OFS));
*device = *((ushort *) ((ptr) startAddr + FLASH_DEV_OFS));

CMDHDR(FEE_RESET);
for (i = 0; i < 1000; i++)
{
    for (count = 0; count < 3; count++) // Delay10uS()
        ;
}
return (flashOk);

} /* FlashID() */

/*****
**      Do Not Move - used as end address for move to upper memory
**      BlockErased()
**
**      Checks to see if block is all FFs
*****/
FlashErr BlockErased(long block)
{
    int     j;
    int     errors=0;
    ushort *flashAddr;

    flashAddr = (ushort*) (block*256);
    for (j = 0; j < WORDS_PER_SECT; j++)
        if (*flashAddr++ != 0xFFFF)
            errors++;

    if (errors)
        return (NotErased);
    else
        return (flashOk);
} // BlockErased

/*****
**      BlockErase()
**
**      writes block to all FFs
*****/
FlashErr BlockErase(long block)
{
    int     j;
    ushort RamArray[WORDS_PER_SECT];
    ushort *RamAddress;

    for (j = 0; j < WORDS_PER_SECT; j++)
        RamArray[j] = 0xFFFF;

    RamAddress = RamArray;

    return(FBurnBlockMove(RamAddress,block));

} // BlockErase

```

```

/*****
**      BlockWrite()
**
**      writes RAM contents to Flash block
*****/
FlashErr BlockWrite(long block, ushort *RamAddress)
{
    return(FBurnBlockMove(RamAddress,block));
} // BlockWrite

/*****
**      BlockVerify()
**
**      Checks to see if flash block matches block pointed to by RamAddress
*****/
FlashErr BlockVerify(long block, ushort *RamAddress)
{
    int    j;
    int    errors=0;
    ushort *flashAddr;

    flashAddr = (ushort*) (block*256);

    for (j = 0; j < WORDS_PER_SECT; j++)
        if (*flashAddr++ != *RamAddress++)
            errors++;

    if (errors)
        return (NoMatch);
    else
        return (flashOk);
} // BlockErased

/*****
**      BlockDisplay()
**
**      prints out flash block
*****/
int BlockDisplay(long block)
{
    int j;
    ushort *flashAddr;

    flashAddr = (ushort*) (block*256);

    printf("\n\nBLOCK #%ld",block);
    printf("\nADDRESS = %08lX", (long)flashAddr);

    for (j = 0; j < WORDS_PER_SECT; j++)
        printf("%c%04X ",((j%8)? ' ':'\n'),*flashAddr++);

    printf("\n");
    return(0);
}

/*****
**      DisplayFlashMap()
**
**      prints out flash blocks written and erased
*****/
void DisplayFlashMap(void)
{
    int j;
    int block;

```

```

for (j = 0; j <1024; j++)
{
  if ((j%50)==0)
  {
    printf("\n%04d  ",j);
  }
  printf("%s%c",((j%10)?(" "):(" ")),((BlockErased(j))? 'X': '.'));
}
printf("\n");
}

```

Adding Additional Flash Memory

This application note shows you how to connect additional 5V flash memory. This memory can be used for non-volatile data storage and can be written to and read from about 10,000 times before data will be lost. As Figure 7-12 shows, two flash memory chips are required and can extend the flash memory up to 1 megabyte (both chips must be the same size). The OE, WE and the A0 - A18 signals are common to both chips (The pin numbers are in []). The program can be located in the EXAMPLES directory.

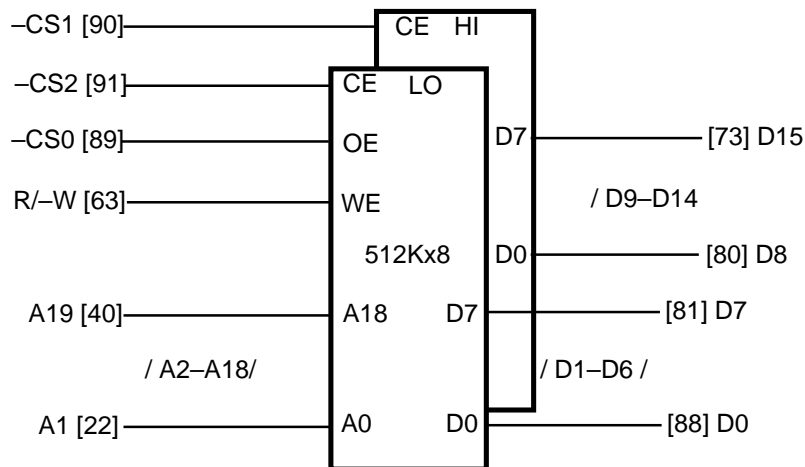


Figure 7-12: Adding One Megabyte of Flash Memory

```

/*****
**      addflash.c          Tattletale Model 8 chip select example for
**                          adding additional flash

**      Tattletale Model-8 code for reading and writing added 5v flash
**                          for non-volatile data storage

**
**      Copyright 1994 ONSET Computer Corp. All rights reserved.
**

PLEASE NOTE!
These routines have the potential for causing great distress -
use them carefully!!

FLASH HAS A LIMITED NUMBER OF WRITES CYCLES - 10,000
- BE CAREFUL NOT TO EXCEED THIS LIMIT OR DATA MAY BE LOST!
This can be easy to do during development!

```


These are generic flash programming routines and allow the added flash to be burned when running from flash.

If you find these routines useful, tell us so.

If you have some ideas about routines that would be useful to yourself and others please tell us too.

If you make any useful enhancements please share them with us and others via our BBS.

```

**      -tam
**      Copyright (C) 1994 ONSET Computer Corp. All rights reserved.
**
**
*****/
#ifndef PRECOMPHDRS
#include <TT8.h>          /* Tattletale Model 8 Definitions */
#include <tat332.h>       /* 68332 Tattletale (7,8) Hardware Definitions */
#include <sim332.h>       /* 68332 System Integration Module Definitions */

#include <tt8lib.h>       /* definitions and prototypes for Model 8 library */

#include <stdio.h>
#include <stdlib.h>
#endif
#include <tt8lib.h>
#include <tpu332.h>
#include <tat332.h>
#include <sim332.h>
#include <qsm332.h>
#include <dio332.h>
#include <stdio.h>
#include <userio.h>

#define XMemBaseAddr    0x400000    /* anywhere from 300000-CFFFFFF is ok */
#define XMemOEPar       _CSPAR0->CS0 /* pin assignment for devices OE */
#define XMemOEBase      CSBAR0      /* base address register for OE */
#define XMemOEOpts      CSOR0       /* options register for OE */
#define XMemCSHiPar     _CSPAR0->CS1 /* pin assignment for high device CS */
#define XMemHiBase      CSBAR1      /* base address register for high CS */
#define XMemHiOpts      CSOR1       /* options register for high CS */
#define XMemCSLoPar     _CSPAR0->CS2 /* pin assignment for low device CS */
#define XMemLoBase      CSBAR2      /* base address register for low CS */
#define XMemLoOpts      CSOR2       /* options register for low CS */

/*
**      ATMEL COMMAND DEFINITIONS AND FLASH DEFINES
*/

#define FLASH_MFR_OFS   0
#define FLASH_DEV_OFS   2

#define FEE_QUERY       0x9090
#define FEE_PROG        0x0505
#define FEE_RESET       0xF0F0

#define FEE_PROT_DIS1   0x8080
#define FEE_PROT_DIS2   0x2020

#define CR0              (uspv) (XMemBaseAddr + (0xAAAA)) /* 5555 << 1 */
#define CR1              (uspv) (XMemBaseAddr + (0x5554)) /* AAAA << 1 */
#define CMDHDR(x)        *CR0=0xAAAA; *CR1=0x5555; *CR0=x;

#define WORDS_PER_SECT  128

```

```

#define          BYTES_PER_SECT  (WORDS_PER_SECT * 2)

typedef enum
{
    FlashOk = 0,
    FlashBadAddress,
    flashBadID,
    NotErased,
    NoMatch
} Flasherr;

// The Flash burning routine - ATMEL specific
FlashErr      FBurnBlock(register ushort *srcAddr, register ulong block);

// These are the functions you may call

void          SetupExtFlashMem(void);
FlashErr      GetFlashID(void);
FlashErr      BlockErase(ulong block);
FlashErr      BlockErased(ulong block);
FlashErr      BlockWrite(ulong block, ushort *WriteAddress);
FlashErr      BlockVerify(ulong block, ushort *VerifyAddress);

int           BlockDisplay(ulong block);
void          DisplayFlashMap(void);

// globals
static ushort RamArray[WORDS_PER_SECT];
static ushort *RamAddress;

/* UP TO 1 MEG OF FLASH MEMORY (2 32K-512K x 8's)
**
**      -CS1 [90]-----| CE  HI |
**
**      -CS2 [91]-----| CE  LO |
**
**      -CS0 [89]-----| OE    | D7 -----[73] D15
**
**      R/-W [63]-----| WE    | / D9-D14 /
**
**                      | 512Kx8 | D0 -----[80] D8
**
**      A19 [40]-----| A18  D7 | -----[81] D7
**
**                      / A2-A18/ | / D1-D6 /
**
**      A1 [22]-----| A0   D0 | -----[88] D0
**
**
**
*/

void SetupExtFlashMem(void);

/*****
**      main
*****/
main()
{
    int    index;
    ulong  blocknum = 0 ;
    char   cmd      = 'M';

    InitTP8(NO_WATCHDOG, TP8_TPU); /* setup Model 8 for running C progXMems */
    printf("\nTattletale Model 8 Chip Select Example for Adding Flash Memory\n");

    SetupExtFlashMem();

```

```

for (index = 0; index < WORDS_PER_SECT; index++)
    RamArray[index] = index | (index<<8);

RamAddress = RamArray;

loop:  QueryChar(  "\n\n\t<S>elect  Block\
                 \n\t<W>rite  Block\
                 \n\t<E>rase  Block\
                 \n\t<C>heck  Block\
                 \n\t<D>isplay Block\
                 \n\t<V>erify  Block\
                 \n\t<M>ap    Flash\
                 \n\t<F>lash  ID\
                 \n\t<Q>uit\
                 \n\t default = " , cmd, "SWECDVMFQ", &cmd);

printf("\n");
switch (cmd)
{
    case 'S' :
        if (!QueryNum("\nEnter Block Number", "%ld", "%ld", &blocknum))
            Reset();;
        break;

    case 'W' :
        if (BlockErased(blocknum))
        {
            printf("\nblock %ld not erased", blocknum);
            if (!QueryYesNo("\nYou will overwrite existing data - proceed?", 'N'))
                break;
        }
        (BlockWrite(blocknum, RamAddress)) ?
            printf("\nblock %ld block not written", blocknum):
            printf("\nblock %ld written", blocknum);
        break;

    case 'E' :
        if (BlockErased(blocknum))
        {
            printf("\nblock %ld not erased", blocknum);
            if (!QueryYesNo("\nYou will destroy existing data - proceed?", 'N'))
                break;
        }
        (BlockErase(blocknum)) ?
            printf("\nblock %ld cannot be erased", blocknum):
            printf("\nblock %ld erase complete", blocknum);
        }
        else
            printf("\nblock %ld already erased", blocknum);
        break;

    case 'C' :
        (BlockErased(blocknum)) ?
            printf("\nblock %ld contains data", blocknum):
            printf("\nblock %ld erased", blocknum);
        break;

    case 'V' :
        (BlockVerify(blocknum, RamAddress)) ?
            printf("\nblock %ld data not verified", blocknum):
            printf("\nblock %ld data verified", blocknum);
        break;

    case 'D' :
        BlockDisplay(blocknum);
        break;

    case 'M' :

```

```

        DisplayFlashMap();
        break;

    case 'F' :
        GetFlashID();
        break;

    case 'Q' :
        ResetToMon();
    }
goto loop;

} // end main

/*****
**      SetupExtFlashMem          Setup external memory
**      *****/
void SetupExtFlashMem(void)
{
    XMemOEPar = XMemCSHiPar = XMemCSLoPar = PORT16;

    *XMemOEBase = (XMemBaseAddr >> 8) | SZ1M;
    *XMemOEOpts =
        M_MDSYNC      &    CLR    /* 0 = async, 1 = synchronous */
        M_BYTEU      &    SET    /* upper byte */
        M_BYTEL      &    SET    /* lower byte */
        M_CSWRITE    &    CLR    /* write */
        M_CSREAD     &    SET    /* read */
        M_DSSTRB     &    CLR    /* 0 = address, 1 = data strobe */
        M_WAIT6      &    CLR    /* wait states */
        M_SPCS       &    SET    /* supervisor */
        M_SPCU       &    SET    /* user */
        M_AVEC       &    CLR    /* 0 = ext rupt, 1 = autovector */
    ;

    *XMemHiBase = (XMemBaseAddr >> 8) | SZ1M;
    *XMemHiOpts =
        M_MDSYNC      &    CLR    /* 0 = async, 1 = synchronous */
        M_BYTEU      &    SET    /* upper byte */
        M_BYTEL      &    CLR    /* lower byte */
        M_CSWRITE    &    SET    /* write */
        M_CSREAD     &    SET    /* read */
        M_DSSTRB     &    CLR    /* 0 = address, 1 = data strobe */
        M_WAIT6      &    CLR    /* wait states */
        M_SPCS       &    SET    /* supervisor */
        M_SPCU       &    SET    /* user */
        M_AVEC       &    CLR    /* 0 = ext rupt, 1 = autovector */
    ;

    *XMemLoBase = (XMemBaseAddr >> 8) | SZ1M;
    *XMemLoOpts =
        M_MDSYNC      &    CLR    /* 0 = async, 1 = synchronous */
        M_BYTEU      &    CLR    /* upper byte */
        M_BYTEL      &    SET    /* lower byte */
        M_CSWRITE    &    SET    /* write */
        M_CSREAD     &    SET    /* read */
        M_DSSTRB     &    CLR    /* 0 = address, 1 = data strobe */
        M_WAIT6      &    CLR    /* wait states */
        M_SPCS       &    SET    /* supervisor */
        M_SPCU       &    SET    /* user */
        M_AVEC       &    CLR    /* 0 = ext rupt, 1 = autovector */
    ;

} /* SetupExtFlashMem() */

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

```

```

FlashErr GetFlashID(void)
{
    register short    count;
    ulong            i;
    int              index;
    ushort           mfr, device;

    CMDHDR(FEE_QUERY);
    Sleep(0);
    Sleep(2);
    mfr = *(ushort *) (XMemBaseAddr + FLASH_MFR_OFS);
    device = *(ushort *) (XMemBaseAddr + FLASH_DEV_OFS);

    printf("\nmfr %04x device = %04x\n",mfr,device);
    printf("\n address %08lx ",XMemBaseAddr + FLASH_MFR_OFS);
    printf("\n address %08lx ",XMemBaseAddr + FLASH_DEV_OFS);

    CMDHDR(FEE_RESET);
    Sleep(0);
    Sleep(2);
    printf("\nFLASH ID = ");

    if ( (mfr == 0x1F1F) /* ATMEL */ &&
        ((device == 0xDCDC) /* 29C256 */ ||
         (device == 0x5D5D) /* 29C512 */ ||
         (device == 0xD5D5) /* 29C010 */ )
        {
            printf("ATMEL ");
            if (device == 0xDCDC)
                printf("29C256");

            if (device == 0x5D5D)
                printf("29C512");

            if (device == 0xD5D5)
                printf("29C010");
            return(flashOk);
        }
    else
        {
            printf("bad device ID");
            return(flashBadID);
        }
    }

/*****
**
**
**
*****/
FlashErr FBurnBlock(    register ushort *srcAddr,
                        register ulong block)
{
    register volatile ushort    x, y;
    ulong                        j;
    ushort *flashAddr;

    flashAddr = (ushort*) XMemBaseAddr+(block*128);

    for (j = 0; j < WORDS_PER_SECT; j++)
        *flashAddr++ = *srcAddr++;
    flashAddr--; // back up to last address written for next test
    do
    {
        x = *flashAddr & 0x4040; /* bit 6 */

```

```

        y = *flashAddr & 0x4040;
        }while (y != x);          /* wait for toggle to end */

return (flashOk);

}          /* FBurnBlock() */

/*****
**      FlashID
**
**
**
*****/
FlashErr FlashID(ushort *startAddr, ushort *mfr, ushort *device)
{
    register short count;
    ulong i;
    int index;

    CMDHDR(FEE_QUERY);
    Sleep(0);
    Sleep(2);
    *mfr = *((ushort *) ((ptr) startAddr + FLASH_MFR_OFS));
    *device = *((ushort *) ((ptr) startAddr + FLASH_DEV_OFS));

    CMDHDR(FEE_RESET);
    Sleep(0);
    Sleep(2);
    return (flashOk);

}          /* FlashID() */

/*****
**      BlockErased()
**
**      Checks to see if block is all FFs
*****/
FlashErr BlockErased(ulong block)
{
    int j;
    int errors=0;
    ushort *flashAddr;

    flashAddr = (ushort*) XMemBaseAddr+(block*128);
    for (j = 0; j < WORDS_PER_SECT; j++)
        if (*flashAddr++ != 0xFFFF)
            errors++;

    if (errors)
        return (NotErased);
    else
        return (flashOk);
} // BlockErased

/*****
**      BlockErase()
**
**      writes block to all FFs
*****/
FlashErr BlockErase(ulong block)
{
    int j;
    ushort RamEraseArray[WORDS_PER_SECT];
    ushort *RamEraseAddress;

```

```

    for (j = 0; j < WORDS_PER_SECT; j++)
        RamEraseArray[j] = 0xFFFF;

    RamEraseAddress = RamEraseArray;

    return(FBurnBlock(RamEraseAddress,block));

} // BlockErase

/*****
**   BlockWrite()
**
**   writes RAM contents to Flash block
*****/
FlashErr BlockWrite(ulong block, ushort *WriteAddress)
{
    return(FBurnBlock(WriteAddress,block));

} // BlockWrite

/*****
**   BlockVerify()
**
**   Checks to see if flash block matches block pointed to by RamAddress
*****/
FlashErr BlockVerify(ulong block, ushort *VerifyAddress)
{
    int    j;
    int    errors=0;
    ushort *flashAddr;

    flashAddr = (ushort*) XMemBaseAddr+(block*128);

    for (j = 0; j < WORDS_PER_SECT; j++)
        if (*flashAddr++ != *VerifyAddress++)
            errors++;

    if (errors)
        return (NoMatch);
    else
        return (flashOk);
} // BlockErased

/*****
**   BlockDisplay()
**
**   prints out flash block
*****/
int BlockDisplay(ulong block)
{
    int j;
    ushort *flashAddr;

    flashAddr = (ushort*) XMemBaseAddr+(block*128);

    printf("\n\nBLOCK #%ld",block);
    printf("\nADDRESS = %08lX", (ulong)flashAddr);

    for (j = 0; j < WORDS_PER_SECT; j++)
        printf("%c%04X ",((j%8)? ' ':'\n'),*flashAddr++);

    printf("\n");
    return(0);
}

/*****

```

```

**      DisplayFlashMap()
**
**      prints out flash blocks written and erased
*****/
void    DisplayFlashMap(void)
    {
        ulong j;

        printf("\n(X = data : (.) = erased)\n\n");

        for (j = 0; j <1024; j++)
            {
                if ((j%50)==0)
                    {
                        printf("\n%04ld  ",j);
                    }
                printf("%s%c",((j%10)?(" "):(" ")),((BlockErased(j))?'X':'.'));
            }
        printf("\n");
    }

```

Adding Additional RAM

This application note shows you how to connect additional RAM memory. As Figure 7-13 shows, two RAM memory chips are required and can extend the memory up to 1 megabyte (both chips must be the same size). The OE, RW and the A0 - A18 signals are common to both chips (The pin numbers are in []). The program can be located in the EXAMPLES directory.

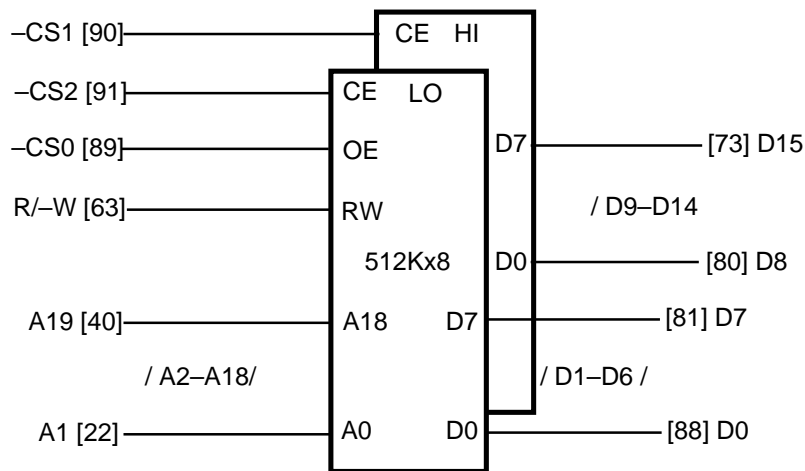


Figure 7-13: One Megabyte of Static RAM (Two 512K x 8's)

```

/*****
**      addmem.c          Tattletale Model 8 chip select example for additional RAM
**
**      Copyright 1994 ONSET Computer Corp. All rights reserved.
**
**      Wednesday, May 4, 1994    --jhg
*****/

#include    <TT8.h>          /* Tattletale Model 8 Definitions */

```



```

#include <tat332.h> /* 68332 Tattletale (7,8) Hardware Definitions */
#include <sim332.h> /* 68332 System Integration Module Definitions */

#include <tt8lib.h> /* definitions and prototypes for Model 8 library */

#include <stdio.h>
#include <stdlib.h>

/*
**                                     1 MEG OF STATIC RAM (2 512K x 8's)
**
**
**  -CS1 [90]-----|-----| CE HI
**
**  -CS2 [91]-----|-----| CE LO
**
**  -CS0 [89]-----|-----| OE      D7 -----[73] D15
**
**  R/-W [63]-----|-----| RW      / D9-D14 /
**
**                                     512Kx8
**                                     DO -----[80] D8
**
**  A19 [40]-----|-----| A18  D7 -----[81] D7
**
**      / A2-A18/
**      / D1-D6 /
**
**  A1 [22]-----|-----| A0   D0 -----[88] D0
**
**
*/

#define XMemBaseAddr      0x400000 /* anywhere from 300000-CFFFFFFF is ok */
#define XMemOEPar         _CSPAR0->CS0 /* pin assignment for devices OE */
#define XMemOEBase        CSBAR0 /* base address register for OE */
#define XMemOEOpts        CSOR0 /* options register for OE */
#define XMemCSHiPar       _CSPAR0->CS1 /* pin assignment for high device CS */
#define XMemHiBase        CSBAR1 /* base address register for high CS */
#define XMemHiOpts        CSOR1 /* options register for high CS */
#define XMemCSLoPar       _CSPAR0->CS2 /* pin assignment for low device CS */
#define XMemLoBase        CSBAR2 /* base address register for low CS */
#define XMemLoOpts        CSOR2 /* options register for low CS */

void SetupXMem(void);
void TestXMem(void);

/*****
**      main
**      *****/
main()
{
    InitTT8(NO_WATCHDOG, TT8_TPU); /* setup Model 8 for running C progXMems */

    printf("\nTattletale Model 8 Chip Select Example for Addition Memory\n");

    SetupXMem();

    TestXMem();

} /* main() */

/*****
**      SetupXMem          Setup external memory
**      *****/
void SetupXMem(void)
{
    XMemOEPar = XMemCSHiPar = XMemCSLoPar = PORT16;
}

```

```

*XMemOEBase = (XMemBaseAddr >> 8) | SZ1M;
*XMemOEOpts =
    M_MDSYNC      &    CLR    /* 0 = async, 1 = synchronous */
    M_BYTEU      &    SET    /* upper byte */
    M_BYTEL      &    SET    /* lower byte */
    M_CSWRITE    &    SET    /* write */
    M_CSREAD     &    SET    /* read */
    M_DSSTRB     &    CLR    /* 0 = address, 1 = data strobe */
    M_WAITO     &    CLR    /* wait states */
    M_SPCS       &    SET    /* supervisor */
    M_SPCU       &    SET    /* user */
    M_AVEC       &    CLR    /* 0 = ext rupt, 1 = autovector */
;

*XMemHiBase = (XMemBaseAddr >> 8) | SZ1M;
*XMemHiOpts =
    M_MDSYNC      &    CLR    /* 0 = async, 1 = synchronous */
    M_BYTEU      &    SET    /* upper byte */
    M_BYTEL      &    CLR    /* lower byte */
    M_CSWRITE    &    SET    /* write */
    M_CSREAD     &    SET    /* read */
    M_DSSTRB     &    CLR    /* 0 = address, 1 = data strobe */
    M_WAITO     &    CLR    /* wait states */
    M_SPCS       &    SET    /* supervisor */
    M_SPCU       &    SET    /* user */
    M_AVEC       &    CLR    /* 0 = ext rupt, 1 = autovector */
;

*XMemLoBase = (XMemBaseAddr >> 8) | SZ1M;
*XMemLoOpts =
    M_MDSYNC      &    CLR    /* 0 = async, 1 = synchronous */
    M_BYTEU      &    CLR    /* upper byte */
    M_BYTEL      &    SET    /* lower byte */
    M_CSWRITE    &    SET    /* write */
    M_CSREAD     &    SET    /* read */
    M_DSSTRB     &    CLR    /* 0 = address, 1 = data strobe */
    M_WAITO     &    CLR    /* wait states */
    M_SPCS       &    SET    /* supervisor */
    M_SPCU       &    SET    /* user */
    M_AVEC       &    CLR    /* 0 = ext rupt, 1 = autovector */
;

} /* SetupXMem() */

/*****
**      TestXMem          Test external memory
*****/
void TestXMem(void)
{
    ulong    memBase = BASEADDR(*CSBAR0);
    long     memSize = BLOCKSIZE(*CSBAR0);
    long     errcount;
    ushort   *moving0, *moving1;
    ulong    mask;
    ushort   testvalue;

/*
**      ADDRESS TEST
*/

    printf("\nAddress Test %lX:%lX ...", memBase, memSize);
    fflush(stdout);
/*
**      FILL FIRST
*/
    for (testvalue = 0x0101, mask = 1; mask < memSize; mask <= 1)
    {
        moving1 = (ushort *) ((memBase + mask) & 0xffffffe);
        moving0 = (ushort *) (((memBase + memSize - sizeof(ushort)) & ~mask) & 0xffffffe);
        *moving1 = testvalue;
        testvalue += 0x0103;
        *moving0 = testvalue;
        testvalue += 0x0103;
    }
}

```

```

    }

    /*
    **     THEN CHECK
    */
    for (errcount = 0, testvalue = 0x0101, mask = 1; mask < memSize; mask <<= 1)
    {
        moving1 = (ushort *) ((memBase + mask) & 0xfffffffffe);
        moving0 = (ushort *) ((memBase + memSize - sizeof(ushort)) & ~mask) & 0xfffffffffe);

        if (*moving1 != testvalue)
        {
            printf("\n %08lX %04X %04X", moving1, *moving1, testvalue);
            errcount++;
        }
        testvalue += 0x0103;

        if (*moving0 != testvalue)
        {
            printf("\n %08lX %04X %04X", moving0, *moving0, testvalue);
            errcount++;
        }
        testvalue += 0x0103;
    }

    if (errcount)
        printf(" %ld errors\n", errcount);
    else
        printf(" OK\n");
} /* TestXMem() */

```

Low Power Sleep Mode Examples

This program can be located in the EXAMPLES directory. It shows you how to use a couple of variations of the Sleep() and SleepTil() functions which drop the Model 8 into low power modes. The Model 8 can be woken early by using IRQ1 and keystrokes entering the serial port.

Watchdog Reset Test Program

This application note shows how to enable the watchdog timer to reset a Model 8 program gone awry.

NOTE: This requires the addition of two 10K pulldown resistors to data lines 8 (SB-80) and 9 (SB-79) to work. Also be aware that the maximum watchdog rate of approx 14mS is too fast to guarantee InitTT8() completion in time for your first ServiceWatchdog() call. The next fastest rate of approx. 50mS should be more than enough for current and future versions of InitTT8(). The program can be located in the EXAMPLES directory.

```

/*****
**     watchdog.c                               Tattletale Model 8 Watchdog Reset Test Program
**
**     Copyright 1995 ONSET Computer Corp. All rights reserved.
**

```

```

**      Thursday, February 16, 1995   --jhg
**
**
**      This example shows how to enable the watchdog timer to reset a Model 8
**      program gone awry.  NOTE! this requires the addition of two 10K
**      pulldown resistors to data lines 8 (SB-80) and 9 (SB-79) to work.  Also
**      be aware that the maximum watchdog rate of approx 14 mS is too fast to
**      guarantee InitTT8() completion in time for your first ServiceWatchdog()
**      call.  The next fastest rate of approx. 50 mS should be more than enough
**      for current and future versions of InitTT8().
*****/

#ifdef PRECOMPHDRS
#include      <stdio.h>
#include      <tt8lib.h>          /* definitions and prototypes for Model 8 library */
#include      <sim332.h>
#endif

/*****
**      main
*****/
main()
{
    ServiceWatchdog();

    //Set up the watchdog timer here before InitTT8 since at
    // the default interval it will expire before it can be serviced.

    //Intervals are set using SYPCR bits 6-4, (SWP, SWT1, SWT0)
    //
    //      6 | 5 | 4
    //      ---
    //      0 | 0 | 0          512 / xtal freq 30.6 uS @ 16.718 MHz
    //      0 | 0 | 1          2048 / xtal freq
    //      0 | 1 | 0          8196 / xtal freq
    //      0 | 1 | 1          32768 / xtal freq 1.96 mS @ 16.718 MHz.
    //      1 | 0 | 0          262144 / xtal freq
    //      1 | 0 | 1          1048576 / xtal freq
    //      1 | 1 | 0          4194304 / xtal freq
    //      1 | 1 | 1          16777216 / xtal freq 1 second @ 16.718 MHz
    //

    *SYPCR =
        |      M_SWE      &      SET      /* enable watchdog          */
        |      M_SWP      &      CLR      /* watchdog prescale      CLR=none, SET= /512 */
        |      M_SWT1     &      SET      /* \ watchdog timing divide count bits */
        |      M_SWT0     &      SET      /* /
        |      M_HME      &      SET      /* enable halt monitor    */
        |      M_BME      &      SET      /* enable bus monitor     */
        |      M_BMT1     &      CLR      /* \ bus monitor timing   */
        |      M_BMT0     &      CLR      /* /

;
    InitTT8(-2, TT8_TPU);          /* don't touch watchdog */
    ServiceWatchdog();
    printf("\nWatchdog Reset Test, *RSR = %02X, *SYPCR = %02X\n", *RSR, *SYPCR);
    ServiceWatchdog();
    printf("\nKeep typing characters to extend the on cycle\n");
    ServiceWatchdog();
    while (SerGetByte())          /* give a chance to extend */
        ServiceWatchdog();
}
/* main() */

```

Section 8 - Troubleshooting

Introduction

This section will help you diagnose problems with the Tattletale and either correct them or instruct you to call Onset Computer Product Support for additional help.

NOTE: If you use the Tattletale Model 8 with the Aztec C compiler and you need support for problems with Aztec C, contact the Technical support at Manx. The Model 8 is the only Tattletale that can use the Aztec C compiler.

Common Model 8 Problems and Possible Solutions

NOTE: If for some reason you need to force the Model 8 to clear the flash memory, load and run the file called CLRFLASH located in the BIN directory.

Problems in Aztec C

NOTE: Also check the compiler linker section in the separate Aztec Manual.

Problem: “*Illegal Integer to Pointer conversion*” error when no illegal conversion is found in the program.

Solution: It is possible that you did not initialize the Model 8 using the InitTT8 function. Programs that run from the TOM8 monitor (by typing GO at the TOM8> prompt) have the advantage of retaining all of the initialization that was done by the Model 8. It is possible to run a program in RAM without the InitTT8() function but there will be problems when burning to Flash.

Problem: “*Error in Make: Syntax Error*” is displayed while using the makefile.

Solution: The makefiles will not work if the SPACE bar was used to indent before commands. Anytime an indent is used before a command you MUST use the TAB key. The Aztec C manuals incorrectly state that the space bar can be used when in reality it can't.

Problems with the Model 8

Problem: “*The burned flash program doesn't act like the same program run from the TOM8 monitor.*”

Solution: It is possible that you did not initialize the Model 8 using the InitTT8 function. Programs that run from the TOM8 monitor (by typing GO at the TOM8> prompt) have the advantage of retaining all of the initialization that was done by the Model 8. It is possible to run a program in RAM without the InitTT8() function but there will be problems when burning to Flash.

Problem: “*Flash memory won't reprogram.*” We have seen the Atmel flash refuse to program when powered by some of our inexpensive bench power supplies. Powering up the Model 8 by switching the power supply would sometimes result in the flash ignoring the program and flash I.D. commands, but powering up by disconnecting and reconnecting the power cable never exhibits this behavior.

Reading the Atmel data sheets, and also calling the engineers at Atmel gives no explanation for this phenomenon, but competitor AMD's data sheets allude to a built in write protection feature which activates when the supply voltage behaves strangely and stays active until the device is fully powered down.

NOTE: We cannot be sure but we have not noted this problem with the 29C010 parts.

Solution: Since this problem only appears to occur when using bench supplies and probably relates to the ramp up speed of the supply voltage, you can simply disconnect and reconnect the positive supply lead, or install a switch on the DC line.

Problem: *“Time and serial EEPROM functions don’t seem to work”* If the time and serial EEPROM functions appear to stop working, or work unreliably, check the LED on the Model 8 which should be off, but which will be turned on when a Model 8 application aborts abnormally.

Solution: Turning the Model 8 off and then back on should clear the problem.

Problem: *“Where do the serial EEPROM addresses start from?”*

Solution: The address parameters in the serial EEPROM routines UeeXxxx() are not really addresses, but are instead zero based offsets from the start of the EEPROM. Address zero corresponds to the first byte of user eeprom memory, address one corresponds to the next byte, and the last address may determined by subtracting one from the length returned by UeeSize().

Problems with CrossCut

Problem: *“After an XMODEM receive, the status dialog shows an error message for an instant before the dialog goes away.”*

Solution: The file after the receive is okay, it is just reported as an error.

Problem: *“I don’t see the TOM8 > Prompt when I power up the Model 8.”*

Solution: CrossCut is a terminal program (like Procomm, Microphone, etc...). If you are having trouble communicating, you may want to try talking to the Model 8 with another terminal program (at the standard default settings). You can also attempt to use CrossCut to talk to another serial device—this way, you can see which end is at fault. It is also possible that you have a program loaded into flash that does not print anything out the serial UART. See “How to Erase a Program from the Flash EEPROM Memory” on page 4-19 for information on clearing the flash EEPROM.

Problems with the TOM8 Monitor

Problem: *“Bad Device ID”*, when burning to Flash.

Solution: At power-up the voltage transition between off and on must be rapid and clean. Some current limited supplies power up slowly (it takes a while for the voltage to reach full potential. If you see Bad Device ID when burning to Flash, try disconnecting and reconnecting the power supply positive lead while the power supply is on.

Troubleshooting Tattletale Problems

Use the flow chart in Figure 8-1 to help determine the problem with the Tattletale. Refer to Table 8-1 while performing any of the test procedures.

Table 8-1: Tattletale Communication Default Settings

Tattletale Model	Protocol	Setting
Model 8	Baud Rate	9600
All Models	Data Bits	8
All Models	Stop Bits	1
All Models	Handshake	None
All Models	Parity	None

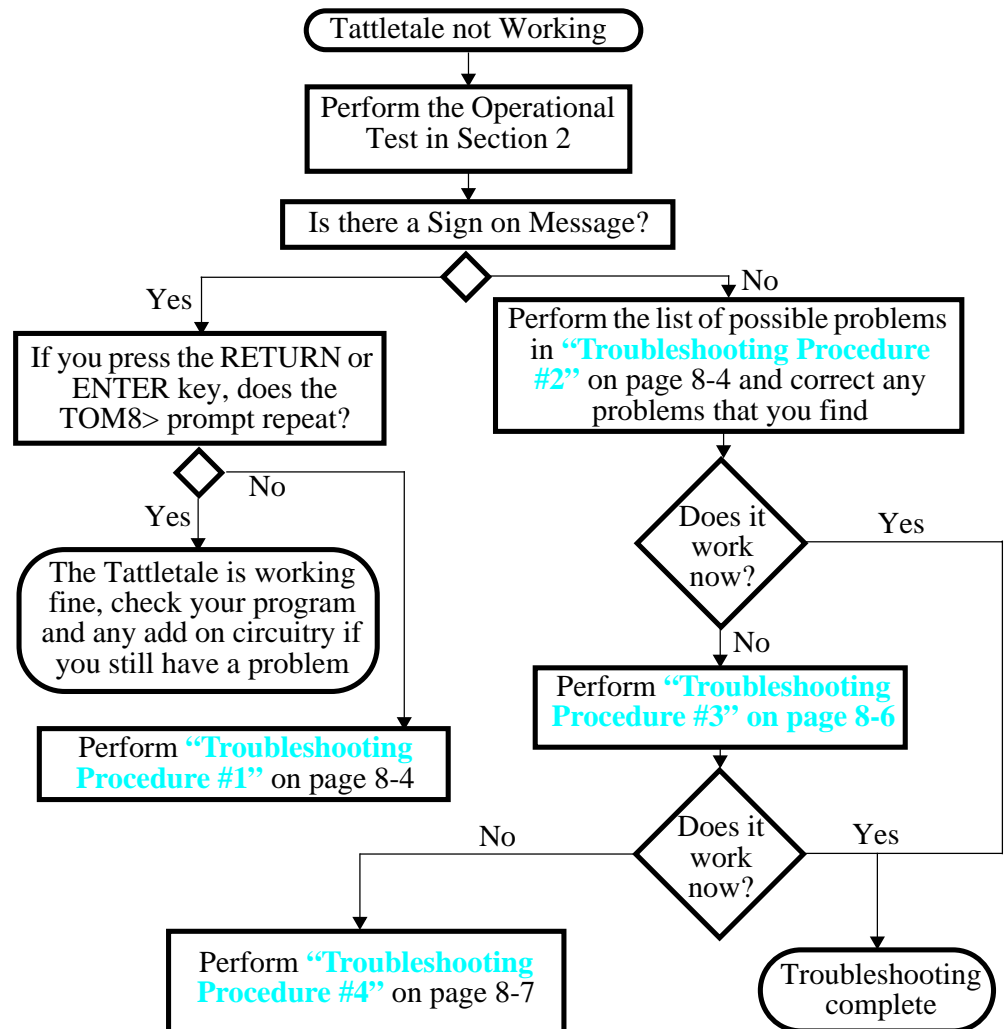


Figure 8-1: Troubleshooting Flow Chart

What to do if the Operation Test Fails

NOTE: Refer to Section 2 for the **Testing the Operation of the Tattletale** procedure.

Troubleshooting Procedure #1

If you get the sign-on message but cannot type any characters including the RETURN or ENTER key, perform the following:

1. Select QUIT from the File menu.
2. Disconnect the power source from the Tattletale.
3. Restart the CrossCut application.
4. Connect the power source to the Tattletale. The screen should display the startup data again.
5. Press the RETURN or ENTER key and see if you get the TOM8> prompt to repeat.
6. If you still can't enter any characters, check the following:

NOTE: Try eliminating any circuits you added on to get back to the basic Tattletale, communication cable and computer.

- The battery or power supply may be weak under load. Check the battery with a 1K resistor across it.
- Check that the power supply current limit is not set to <30mA.
- The power supply may be set to the wrong voltage. Set the voltage to 7 to 15 volts for the initial tests.
- The battery or power supply connection may be bad or intermittent. Wiggle connector and check VREG on the Tattletale.
- The communication cable may be bad. Check for a signal at the UART out line with an oscilloscope while powering up the Tattletale.

Troubleshooting Procedure #2

If you do NOT get the sign-on message any of the following may be the problem:

NOTE: Most of these can be tested by performing **“Troubleshooting Procedure #3” on page 8-6.**

- The wrong baud rate may be selected in the communications setup window (see Table 8-1).
- The default communication settings have been changed (see Table 8-1).
- The communication cable may be in the wrong port of the computer.

- The communication port may be set up for a mouse (IBM PC only).
- If a serial switch box is in use, disconnect it and connect the Tattletale communications cable directly to the serial port.
- The CrossCut version may be outdated.
- The wrong port may be selected in the communications setup window.
- The Tattletale may not be aligned with the pins of the prototyping board.
- The battery or power supply may be dead or weak under load.
- The power supply may be set to the wrong voltage.
- The battery or power supply connection may be bad or intermittent.
- The communication cable may be bad.
- The serial port on the computer may be bad. Check by shorting pins 2 & 3 of the Comm Port on the IBM PC and typing characters on the keyboard. If the characters appear in the Terminal Window, the COMM Port is working correctly.
- Try eliminating any circuits you added on to get back to the basic Tattletale, communication cable and computer.

NOTE: Make sure that you are using the latest version of CrossCut. If you power up the Model 8 and a TXB# prompt is displayed, TxBASIC has been burned into the EEPROM and will need to be erased from the EEPROM before your C programs will work correctly.

If none of these suggestions work, try another computer or terminal if you can. If you still can't get the Tattletale to work, perform the [“Troubleshooting Procedure #4”](#) procedure on page 8-7.

NOTE: If you need to send anything back to Onset Computer, you will need to call Customer Support and get an RMA number.

After correcting the problem press the RETURN or ENTER key. The Tattletale should respond with another TOM8> prompt.

This simple test proves that the serial interface can send as well as receive. Once the TOM8> prompt repeats, you can proceed with:

[Section 3 - Operating the CrossCut Program](#) for additional information on the CrossCut software.

To write your own programs in C refer to [Section 4 - C Programming Guide](#) and [Section 5 - C Library Reference](#). These sections show the details of all the commands available to operate the Tattletale.

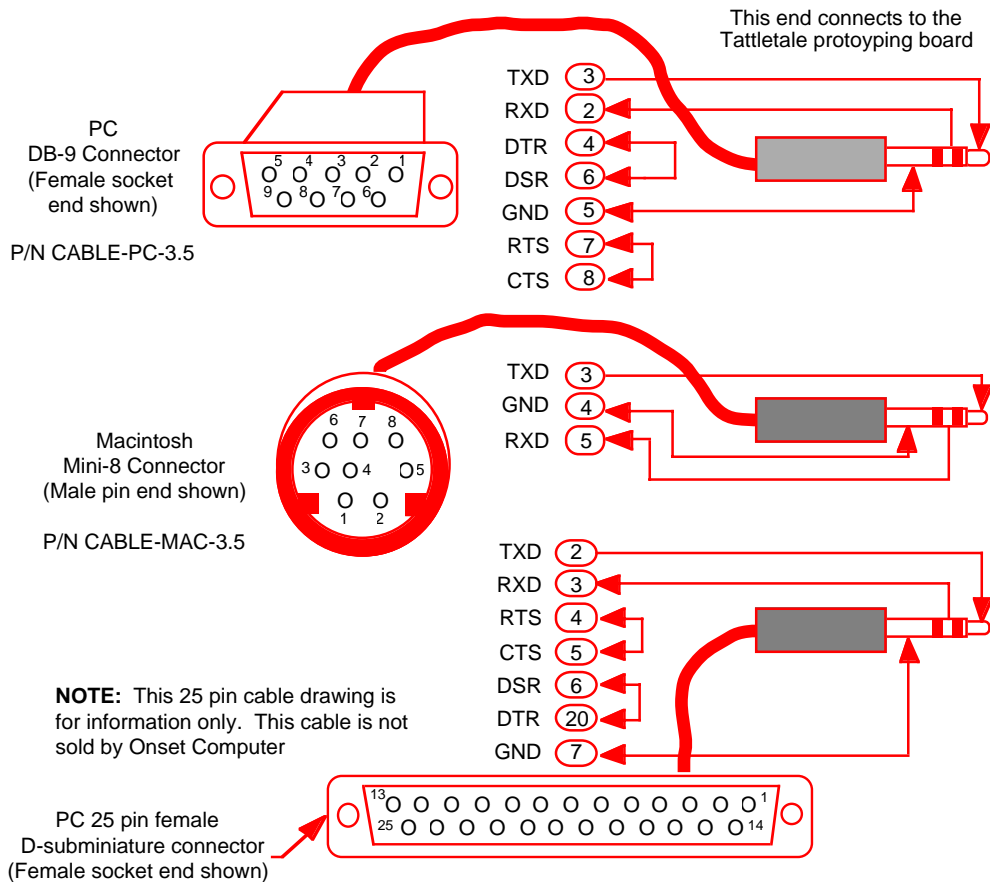


Figure 8-2: Communication Cable Pin Layouts

Troubleshooting Procedure #3

This procedure will verify the proper operation of the software, the computer’s serial port and the communications cable without the Tattletale attached. It will help if someone else can hold the communications cable steady during this procedure.

NOTE: If this procedure does not work for either of the serial ports, then remove the communications cable from the back of the computer and short the TXD and RXD pins together (of the port selected in CrossCut). Refer to Figure 8-2 for the locations of those pins.

1. Verify that the communications cable is connected to one of the serial ports on the computer (COM1 or COM2) and if the port is marked, record which one you are connected to.
2. Start the computer and start the CrossCut software.

3. Select the “**CommPort**” menu and then select the “**Port Setup**” option.

Verify that the port selected is the same port the communication cable is connected to. Also verify that the correct baud rate was selected (see Table 8-1) The values of the other parameters are not critical, but it is best if they are left at the default values. If you are not sure which port you are connected to then repeat the following steps for both of the serial ports.

NOTE: Be sure you’re not trying to use the serial port used by a serial mouse. If you select the port your mouse is connected to you may lock up the computer.

- a. While using alligator clips, a piece of wire or even a bent paper clip, short the chrome tip of the phone plug to the next closest chrome section (see Figure 8-3).
- b. While shorting the plug, type characters on the keyboard. If everything is working correctly the characters typed should be displayed on the computer screen. If not, go back and select the other serial port and try again.
- c. If this procedure does not work for either of the serial ports, then the problem is either in the cable or the computer’s serial port. If your computer has another serial port to connect to, repeat this procedure using it.
- d. If this procedure does display the characters typed on the screen and the Tattletale still will not communicate after reconnecting the Tattletale, perform “**Troubleshooting Procedure #4**” on page 8-7.

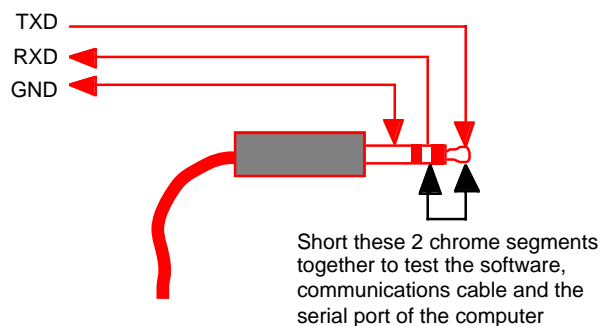


Figure 8-3: Testing the Communication Cable

Troubleshooting Procedure #4

NOTE: An oscilloscope and a DVM are required for this procedure.

The following is a very detailed procedure for measuring voltages and signals at specific pins on the Tattletale. This should only be performed if the previous 3 troubleshooting procedures have failed to solve your problem.

1. Check the baud rate. The baud rate should be set to 9600 for the Model 8.
2. Check the communication cable connections by disconnecting both ends of the communications cable and then insert the cable completely into the Tattletale and into the computer.

3. Make sure that the communication cable is connected to either COMM Port 1 or COMM Port 2.

NOTE: Be sure you're not trying to use the serial port used by a serial mouse. If you select the port your mouse is connected to you may lock up the computer.

4. Make sure that pin 1 of the Model 8 is aligned with pin 1 of the prototyping board.
5. Test the battery or power supply for the proper voltage (7 to 15V). If you are using a battery, test it under a load (with it connected to the Tattletale or put a 1K resistor across it) or you may think the battery is okay but in reality it is bad and supplying little or no current.
6. With a DVM, measure between VREG (pin A2) and DGND (pin A1). There should be a 5V signal there.
7. With a DVM, measure between VREF (pin B20) and DGND (pin A1). There should be a 5V signal there also.

NOTE: VREF is off when the Model 8 is in low power mode. VREF is on when the Model 8 is first powered up before running a program.

8. Using an oscilloscope, measure the UART Data Out at RS-232 levels (pins A16 and A14) and perform the following:
 - a. Turn off power to the Tattletale.
 - b. Turn on power to the Tattletale and look for any signal output at all. Each time you power up the Tattletale, it will send the sign-on message out on this pin and you will see a digital signal on this pin switching between $-4.5V$ and $+4.5V$. If you don't see a signal on this pin, the Tattletale may have a problem.

If you have performed all of these steps and still can't get your Tattletale to work, call Onset Computer customer service for an RMA # to send the Tattletale back for repairs.

Contacting Onset Computer Product Support

Please use the following methods to contact Onset Computer for support (they are in our preferred order to be contacted):

1. Send a FAX
2. Send E-mail to Product Support over the Internet
3. Call Onset Computer Product Support

Regardless of which method you use to contact us, we will need the following information to help you:

- Tattletale model number
- Software version (CrossCut and the version numbers displayed when the Model 8 is powered up)
- Type of computer the Tattletale is connected to (IBM PC, Macintosh Laptop etc.)
- Detailed information about the problem you are having
- Information about any added circuitry to the Tattletale or prototyping board

Sending a FAX to Product Support

If you have access to a FAX machine, you can easily contact us at (508) 563-9477. Please include as much detail about your problem and system configuration as possible.

Sending E-mail to Product Support over the Internet

If you have access to the internet you can send us E-mail at:

sales@onsetcomp.com
info@onsetcomp.com
onset@onsetcomp.com
Tech_Support@onsetcomp.com

You can also visit our WORLD WIDE WEB page at: <http://www.onsetcomp.com>
or our anonymous FTP site at: <ftp://ftp.onsetcomp.com>

Calling Onset Computer Product Support

You can also call our Product Support Specialists at (508) 563-9000 between the hours of 8:30am - 5pm EST.

Using the Onset Computer BBS

If you have a modem, you can reach us easily by calling our BBS. The BBS has the latest versions of released software, technical notes, technical specifications and sales information, all of which can be downloaded free of charge. Certain software on the BBS is not free and requires a key to unlock the program after downloading.

NOTES:

- You can connect to the BBS at baud rates up to 14,400.
 - All you can do on the BBS is download messages and files.
 - There is an easy to use graphical interface available for Macintosh and Windows users (the graphical interface can be downloaded from the BBS). IBM PC DOS users will find a regular text interface (there is not a DOS version of the graphical interface).
1. Before calling the BBS, set up your modem for 8 data bits, 1 stop bit, no handshake and no parity.
 2. Call the BBS at (508) 563-2269 with your computer.

3. Once you are connected to the BBS, follow the screen prompts for logging into the BBS as a guest.
4. After logging into the BBS the following menu will be displayed (The menu titles and content change frequently). The following is the text version of the BBS software (download the BBS graphical interface from the Useful Utilities directory for a user friendly graphical BBS interface):

Home: 6 Conferences, 1 Folder.

1 MailBox

* 2 Conferences

3 News

4 Help Folder

5 Testerion

6 New Age

* 7 Cirtronics Files

* 8 Reception Desk

Type an item's name or number to open it.

Commands: Help,Logout,Scan.

These are the main selections to choose from. To select one, just enter the number and press ENTER or RETURN.

NOTE: Any items with an "*" in front of the number means it has new information.

If you select the option for Help, the following will be displayed:

Help: 17 Text files.

1 Composing

3K 3/9/94 6:43 PM Composing a message

2 Conferencing

3K 3/9/94 6:43 PM Participating in
Conferences

3 Home

2K 3/9/94 6:43 PM The Home Directory

4 General

1K 3/9/94 6:43 PM FirstClass Help

5 News

1K 3/9/94 6:43 PM Reading the News

6 Summary

3K 3/9/94 6:44 PM List of Commands

7 Welcome

2K 3/9/94 6:44 PM Login Help

8 Editor

2K 3/9/94 6:43 PM How to use the editor

9 Message Lists

2K 3/9/94 6:43 PM Message List Summary

10 Sending Files

3K 3/9/94 6:44 PM How to attach & send files

11 Reply & Forward

2K 3/9/94 6:43 PM Replying to & forwarding
mail

12 Message History

1K 3/9/94 6:43 PM Who's read my message

13 Resumes

2K 3/9/94 6:43 PM About Resumes

14 Advanced Features

3K 3/9/94 6:43 PM Advanced Commands &
Hints

15 Preferences

2K 3/9/94 6:43 PM Changing User Options

16 Chat

3K 3/9/94 6:43 PM Online Discussions

17 Search

2K 3/9/94 6:42 PM Searching for file and mail

Type an item's name or number to open it, or EXIT to exit.

Commands: Help, Logout, Exit, Home.

NOTE: The BBS files are constantly being updated and changed, so they may not match these listings exactly.

You are free to explore our BBS and download information (some files are password protected). As a guest you can't delete or damage any of our files.

Appendix A - Manufacturer Contact Numbers

The following information is provided as an aid for helping you locate data sheets and other manufacturer specific information to design and build your product. This information was accurate at the time of printing, however; it is subject to change frequently.

Hitachi Corporation
Phone (800) 448-2244

Linear Technology
1630 McCarthy Blvd.
Milpitas, CA 95035-7417
Phone (408) 432-1900
Fax (408) 434-0507

Maxim Integrated Products, Inc.
120 San Gabriel Drive
Sunnyvale, CA 94086
Phone (800) 998-8800
Phone (408) 737-7600
Fax (408) 737-7194

Microchip Technology Inc.
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Phone (602) 786-7200
Fax (602) 899-9210

Motorola Literature Distribution
P.O. Box 20912
Phoenix, AZ 85036
Phone (800) 441-2447
Fax (303) 675-2150

Supertex Inc.
1350 Bordeaux Drive
Sunnyvale, CA 94089
Phone (408) 744-0100
Fax (408) 734-5247

Appendix B - Data Sheets

The following data sheets are provided to aid you in designing and building your products. Contact the specific manufacturer on the previous page for the latest information regarding any of these data sheets.

Glossary

Absolute reference	A voltage reference that does not vary with battery voltage.
A-D converter	Analog -to- digital converter - converts an analog signal to a digital code.
AGND	Analog ground reference - AGND should be tied directly to the analog ground plane.
Amplifier	A circuit used to increase signal strength.
Analog inputs	The Tattletale inputs that accept data from analog sources.
Argument	A type of variable whose value is not a direct function of another variable. It can represent the location of a number in a mathematical operation, or the number with which a function works to produce its results.
Array	A set of related variables that are accessed by a common name with an index number.
Aux UART input	Auxiliary Universal Asynchronous Receiver Transmitter input
Aux UART output	Auxiliary Universal Asynchronous Receiver Transmitter output
Background task	A task that takes place simultaneously while a foreground task is being completed. Background task is rigorously timed. Foreground only executes when background is sleeping.
Baud rate	A unit of measurement of data communication speed. The speed in bauds is the number of signal elements per second. Since a signal element can represent more than one bit, baud is not synonymous with bits-per-second. Typical baud rates are 300, 1200, 2400, 4800, 9600 and 14,400.
Bipolar mode	An A-D mode where analog inputs can accept signals that may be positive or negative.
Block	An amount of storage space or data, arbitrary length, usually contiguous, and often composed of several similar records, all of which are handled as a unit.
Boot	Runs, usually at power up whatever program is burned into the ROM, EPROM or EEPROM.
Breakout Box	A device to put in a communications line that brings the signals out to test points without interrupting the signal.
COM	Common - The common pin defines the zero reference point for all single ended inputs. It must be free of noise and is usually tied to the analog ground plane.
Current drain	The amount of current a device uses to operate.

Data Bits	A logical 0, represented by 0V or a logical 1, represented by 5V. Eight bits together make up 1 byte.
EEPROM	Electrically Erasable Programmable Read Only Memory - can be erased and reprogrammed with a power supply of the correct voltage and the host computer. Refer to EPROM and ROM for additional information.
EPROM	Erasable Programmable Read Only Memory - can be erased by a special EPROM ultraviolet eraser and then reprogrammed with the host computer. Refer to EEPROM and ROM for additional information.
FCB	Form Constant Byte - An assembler directive for initializing memory.
FCC	Form Constant Characters - An assembler directive for initializing memory.
FDB	Form Double Byte - An assembler directive for initializing memory.
Floating point arithmetic	A method of calculation in which the computer or program automatically records, and accounts for, the location of the radix point. The programmer need not consider the location of the radix point.
Gain error	The error in an amplifier that causes the output to be wrong by a constant multiplier.
Handshake	The exchange of signals between communicating entities. Refer to protocol for additional information.
I/O lines	The digital input / output lines connected to the Tattletale.
Input impedance	The resistance an electronic device shows to the outside world. Impedance is really a complex combination of resistance, capacitance and inductance.
Input protection	Protects the Tattletale from voltages that would damage the Tattletale
Instrumentation amplifier	A very high gain amplifier with differential inputs. Has very low noise and high common mode rejection.
Interrupt	An asynchronous event that causes a computer program to interrupt what it is doing to service the event.
Level shifting	To change the voltage of a signal for a specific purpose. It's almost like an amplifier. RS-232 level shifter we talk about converts 0V input to +5V and +5V input to -5V.
Linearity error	An error in an amplifier that causes the gain to vary over the range of inputs.
Logical operators	The Tattletale supports the two logical operators, AND (&) and OR (), which are used for both bit-wise operations and logical connectives.
Loss of precision error	A floating point error that results when a number with more significant digits than the floating point format can handle.

MAC-3.5	The cable used by the Macintosh to connect to the Tattletale.
Marking State	The marking state, also known as the idle state, is +5V (non-RS-232) and the negative voltage RS-232.
Memory map	A map showing the locations in memory for all the parameters, ROM and RAM data, and buffers etc.
Monitor	The low level program that sends commands directly to the Tattletale (not accessible by the user).
Operator	A symbol indicating an operation and itself the subject of the operation. It indicates the process that is being performed.
Output protection	A circuit used to protect the components connected to the output of the Tattletale.
PC-3.5	The cable used to connect most IBM PC's to the Tattletale.
Parity	An extra bit used to detect errors in data sent over communication lines by making the sum of the active bits in a data word either an odd or an even number.
Ping-pong buffering	Storing data to one part of a buffer while it's being read from another part.
Protocol	A formal set of conventions governing the format and relative timing of message exchange between two communicating processes. Refer to handshake for additional information.
Radix	A number that is made the fundamental number of a system of numbers; a base. Humans use a radix of 10, computers prefer 2.
RAM	Random Access Memory - A data storage device that can retain and produce on demand any data placed in it.
Ratiometric Measurement	To make a measurement as a percentage of its full-scale value (as opposed to an absolute measurement). Consistent readings are available even if the full-scale reference varies.
Real-time clock	A clock that keeps time in year, month, day, hour, minute, second format and updates it at some regular rate (usually one second).
Reference diode	Usually a Zener diode used to supply a voltage reference.
Reference input voltage (Voltage reference)	A fixed voltage used for comparing with other unknown or varying voltages.
Registers	Storage locations internal to the microprocessor that are used as operands for all actions.
Regulator, Voltage, REG5V	Keeps a steady amount of voltage over a range of current drains being output from the voltage regulator so long as the input voltage to the voltage regulator does not drop below some cut-off voltage.

Resistivity	How resistant something is to allow electricity to flow.
ROM	Read Only Memory - A data storage device that permanently retains all data placed in it and produces that data on demand. Once the data is placed into it, it can't be changed. Refer to EEPROM and EPROM for additional information.
Schottky diode	A very fast switching diode with a low turn-on threshold (0.4V or less versus the normal 0.6V of other diodes).
SCR latching	If the input (or output) is driven beyond the supply momentarily.
Serial transmission	A system in which the data bits occur serially in time.
Setup time	The amount of time for a system, device or program to reach a required initial condition.
Soldering pencil	A soldering iron with a very fine point for soldering in tight areas.
Square wave	A signal with only two stable states that alternates between these states so that it spends equal time in each state.
Start Bit	This is the first bit issued when a UART wants to send a data byte. The start bit will unambiguously tell the receiver that a set of data bits is to follow. This is normally a space, or 0 bit, which is 0V (non-RS-232) and a positive voltage for RS-232.
Stop Bits	This bit is normally issued by a UART to follow the data byte. This bit is usually a mark, or a 1 bit, and is used to separate consecutive characters and to help with resynchronization.
String operators	Symbols representing the various operations that can be performed on strings - comparing, copying, appending etc.
Symbol table	A list of all the variables, arrays and labels used in a program. Usually, addresses and sizes of the symbols are included.
Tabs	A group of ASCII tab characters. Also called the horizontal tab or HT. The ASCII character value is 9.
Thermistor	An electronic component used to measure temperature. It is also much easier to interface to than a thermocouple.
Thermocouple	A bi-metal device used to measure temperatures of a much greater range than a thermistor.
UART	Universal Asynchronous Receiver Transmitter - A UART is normally a single chip dedicated to converting bytes to and from a serial data stream that can travel on RS-232 data lines.
Waiting for NAK	A state of the XMODEM file transfer protocol where it is waiting for the character (ASCII NAK = 15H) signaling to start.
XMODEM	A simple file transfer protocol used to send and receive files.

Index

Section Number – Page Number

A

ABS	5-1
A-D converter	1-3, 6-14
A-D converter circuit	6-19
Adding flash memory	7-26
AGND	6-15
AlarmToCtm	5-3, 5-9
ALD1704, 1701, 2701, 4701	7-6
Alt-C	3-4, 3-15
Alt-D	3-4, 3-12
Alt-E	3-4, 3-11
Alt-F	3-4, 3-8
Alt-L	3-4, 3-14
Alt-P	3-4, 3-16
Alt-Q	3-4, 3-9
Alt-S	3-4, 3-13
Alt-T	3-4, 3-14
Analog input specifications	6-14
Analog inputs	6-15
AND	G-2
Assembly shortcuts	5-7
AtoD converter functions	5-7
AtoDMilliVolts	5-3
AtoDReadMilliVolts	5-3, 5-10
AtoDReadWord	5-3, 5-9
AtoDtoMilliVolts	5-10
Aztec C compiler	6-23

B

Background Debug Mode	6-21
Battery	2-1, 8-4
Baud rate	6-6, 6-8, 6-15, 6-16, 6-18
Bipolar mode	7-2
Bipolar operation	7-1
Block	7-10

C

C Library command list	5-3
CalcCRC	5-3, 5-10
Compiler flags	4-8
Configuration byte	6-22
Connectors	6-2
CrossCut	3-1

Current drain	6-17
---------------------	------

D

DelayMilliSecs	5-3, 5-11
Digital I/O macros	5-7

E

Editing	3-4
Erasing memory	4-19
Exception handler	4-23

F

Flash EEPROM functions	5-7
Flash memory	6-20
FlashError	5-3, 5-11
FlashID	5-3, 5-11
Function descriptions	5-9

G

GetFramePtr	5-3, 5-12
GetInterruptMask	5-3, 5-12
GetRamInfo	5-3, 5-12
GetStatusReg	5-4, 5-12
GetTickRate	5-4, 5-13
GetVBR	5-4, 5-13

H

Header files	5-1
Hex files	4-5

I

Initialization routines	5-7
InitPorts	5-4, 5-13
InitQSM	5-4, 5-14
InitRam	5-4, 5-14
InitSIM	5-4, 5-15
InitTPU	5-4, 5-15
InitTT8	5-4, 5-15
InitVBR	5-4, 5-16
Input protection	7-7, 7-8
Input, analog	7-9
Input, digital	7-9
InputLine	5-4, 5-16
InstallHandler	5-4, 5-17
Instrumentation amplifier	7-4, 7-6
Interval timer functions	5-7
IO-8 prototyping board	6-1, 6-22

- K**
- kbflush 5-4, 5-17
 - kbhit 5-4, 5-18
 - Keyboard shortcuts 3-3
- L**
- Library functions 5-1
 - Library object files 5-1
 - Lightning damage 7-10
 - Linking object files 4-8
 - LMDelay 5-4, 5-18
 - Low power sleep 7-29
 - LT1077, 1078, 1079 7-6
 - LT1101 Instrumentation Amplifier 7-7
- M**
- Makefile 4-10
 - Max186PowerDown 5-4, 5-19
 - Max186PowerUp 5-4, 5-18
 - Max186Setup 5-4, 5-19
 - Memory 6-19
 - Memory map 4-19
 - Memory symbols 4-21
 - MilliSecs 5-4, 5-19
 - Miscellaneous functions 5-7
 - Model 8 components 6-12
- N**
- NumToHexStr 5-4, 5-20
- O**
- Op amps 7-6
 - OR G-2
 - Oscillator 6-21
 - Output protection 7-7
- P**
- PC-3.5 6-6
 - PChange 5-4, 5-20
 - PClear 5-4, 5-20
 - PCMCIA card adapter 6-24
 - PConfBus 5-4, 5-21
 - PConfInp 5-4, 5-21
 - PConfOutp 5-4, 5-21
 - Peripheral chip selects 6-20
 - PIC 16C64 6-21
 - PicAckCmpAlrm 5-4, 5-22
 - PicAndRF 5-4, 5-22
 - PicGetIrqAddr 5-4, 5-22
 - PicInit 5-4, 5-23
 - PicOrRF 5-4, 5-23
 - PicReadRF 5-4, 5-24
 - PicWriteRF 5-4, 5-24
 - Pin 5-4, 5-24
 - Pin and socket specifications 6-2
 - Power supply 6-5
 - Prototyping boards (I/O-8, PR-8) 6-4
 - PSet 5-4, 5-25
 - PutStr 5-4, 5-25
- Q**
- QueryChar 5-5, 5-25
 - QueryDateTime 5-5, 5-26
 - QueryNum 5-5, 5-27
 - QueryYesNo 5-5, 5-27
 - Queued Serial Module 6-20
 - Queued Serial Peripheral Interface 6-20
- R**
- RAM 6-19
 - RAM configurations 4-20
 - Real-time functions 5-8
 - Reducing power 4-16
 - Reference 7-3
 - RESET 6-19
 - Reset 5-5, 5-28
 - ResetToMon 5-5, 5-28
 - RtcToCtm 5-5, 5-28
- S**
- Schottky diode 7-2
 - SerActivate 5-5, 5-28
 - SerByteAvail 5-5, 5-29
 - SerGetBaud 5-5, 5-29
 - SerGetByte 5-5, 5-30
 - Serial Communications Interface 6-20
 - Serial I/O functions 5-8
 - SerInFlush 5-5, 5-30
 - SerPutByte 5-5, 5-30
 - SerSetBaud 5-5, 5-30
 - SerSetInBuf 5-5, 5-31
 - SerShutDown 5-5, 5-31
 - SerTimedGetByte 5-5, 5-32
 - ServiceWatchdog 5-5, 5-32

SetAlarmSecs	5-5, 5-32	TSerInFlush	5-6, 5-43
SetAlarmTM	5-5, 5-33	TSerOpen	5-6, 5-44
SetInterruptMask	5-5, 5-33	TSerPutByte	5-6, 5-45
SetStatusReg	5-5, 5-33	TSerResetBaud	5-6, 5-45
SetTickRate	5-5, 5-34	ttmadd	5-6, 5-39
SetTimeSecs	5-5, 5-34	ttmcmp	5-6, 5-39
SetTimeTM	5-5, 5-34	TTMEQ	5-6, 5-39
SetVBR	5-5, 5-33	TTMGE	5-6, 5-39
SimGetFSys	5-5, 5-35	TTMGT	5-6, 5-39
SimSetFlashWaits	5-5, 5-35	TTMLE	5-6, 5-39
SimSetFSys	5-5, 5-36	TTMLT	5-6
SimSetRAMWaits	5-5, 5-35	TTMNE	5-6
Sleep	5-5, 5-36	TTMNE TTMLT	5-39
SleepTill	5-5, 5-37	ttmnow	5-6, 5-40
Specifications	6-8	Typedefs used	5-1
SquishyBus	6-3, 6-24	U	
S-Record	4-4	UART	6-6, 6-15
S-Record flags	4-9	UeeCalcCRC	5-6, 5-45
S-Records	4-9	UeeClearBit	5-6, 5-46
Static arrays	4-8	UeeErase	5-6, 5-46
Stop	5-5, 5-37	UeeError	5-6, 5-46
StopLP	5-5, 5-37	UeeFill	5-6, 5-46
StopWatchStart	5-5, 5-38	UeeReadBlock	5-6, 5-47
StopWatchTime	5-5, 5-38	UeeReadByte	5-6, 5-47
System Clock	6-20	UeeSetBit	5-6, 5-47
System clock	6-20	UeeSize	5-6, 5-47
System functions	5-8	UeeTestBit	5-6, 5-48
System Integration Module	6-20	UeeWriteBlock	5-6, 5-48
T		UeeWriteByte	5-6, 5-48
TensMilliSecs	5-5, 5-39	UpdateCRC	5-6, 5-48
Thermistor	1-10, 2-1	User EEPROM functions	5-9
Time format conventions	5-3	User I/O extras	5-8
TLC1078	7-6	W	
TOM8	4-1	Watchdog	7-29
TOM8 commands	4-2	X	
TPU functions	5-8, 6-12	XmodemSendMem	5-6, 5-49
TPUClearInterrupt	5-5, 5-40	Z	
TPUGetInterrupt	5-6, 5-41	Zener diode	7-10
TPUGetPin	5-6, 5-41		
TPUGetTCR1	5-6, 5-41		
TPUInterruptDisable	5-6, 5-41		
TPUInterruptEnable	5-6, 5-42		
TPUSetPin	5-6, 5-42		
TSerByteAvail	5-6, 5-42		
TSerClose	5-6, 5-43		
TSerGetByte	5-6, 5-43		

Tattletale Model 8-C

Installation and Operation Manual

Revision History

<u>Revision</u>	<u>Release Date</u>	<u>Description of revision or changes</u>
D-EAS-00006 Rev. A	October 1995	Initial release of the new Model 8 manual for C users. The entire manual has been revised since the preliminary version in June of 94.
D-EAS-00006 Rev. B	December 1995	Replaced the LTC-1174 data sheets in the appendix and added information for the 20MB PCMCIA card on page 6-24.
D-EAS-00006 Rev. C	July 1997	Pages 1-2, 6-2 and 6-8 were updated by adding new information on the Operating Temperature range and updating the specifications in Table 1-3 and Table 6-1. Page 6-2 had a correction to the pin layout for the SquishyBus.
D-3285-A	July 1998	Updated for modifications in the Aztec-C libraries version 3.12.
D-3258-B	February 1999	Updated line: Interconnects are available in various heights to accommodate your external hardware, an example of an elastomeric interconnect is the Fujipoly's part number 0940020 which is 0.250in high.
D-3258-C	September 1999	Updated schematic Fig 6-6, Page 6-7 to Rev C. Page 6-13, change lower baud rate Table 6-5 from 1200 to 300. Page 6-24, add note PCMCIA Card Adapter obsolete

